Single-Server Private Information Retrieval with Sublinear Amortized Time

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

[1] MIT
[2] Stanford

January 20, 2022

Abstract. We construct new private-information-retrieval protocols in the single-server setting. Our schemes allow a client to privately fetch a sequence of database records from a server, while the server answers each query in average time sublinear in the database size. Specifically, we introduce the first single-server private-information-retrieval schemes that have sublinear amortized server time, require sublinear additional storage, and allow the client to make her queries adaptively. Our protocols rely only on standard cryptographic assumptions (decision Diffie-Hellman, quadratic residuosity, learning with errors, etc.). They work by having the client first fetch a small "hint" about the database contents from the server. Generating this hint requires server time linear in the database size. Thereafter, the client can use the hint to make a bounded number of adaptive queries to the server, which the server answers in sublinear time—yielding sublinear amortized cost. Finally, we give a lower bound proving that our most efficient scheme is optimal with respect to the trade-off it achieves between server online time and client storage.

1 Introduction

A private-information-retrieval protocol [34, 35] allows a client to fetch a record from a database server without revealing which record she has fetched. In the simplest setting of private information retrieval, the server holds an n-bit database, the client holds an index $i \in \{1, \ldots, n\}$, and the client's goal is to recover the i-th database bit while hiding her index i from the server.

Fast protocols for private information retrieval (PIR) would have an array of applications. Using PIR, a student could fetch a book from a digital library without revealing to the library which book she fetched. Or, she could stream a movie without revealing which movie she streamed. Or, she could read an online news article without revealing which article she read. More broadly, PIR is at the heart of a number of systems for metadata-hiding messaging [7,32], privacy-preserving advertising [8,54,64,82], private file-sharing [37], private e-commerce [60], private media-consumption [56], and privacy-friendly web browsing [66].

Unfortunately, the *computational cost* of private information retrieval is a barrier to its use in practice. In particular, to respond to each client's query,

Beimel, Ishai, and Malkin [14] showed that the running time of a PIR server must be at least linear in the size of the database. This linear-server-time lower bound holds even if the client communicates with many non-colluding database replicas. So, for a client to privately fetch a single book from a digital library, the library's servers would have to do work proportional to the total length of all of the books in the library, which is costly both in theory and in practice.

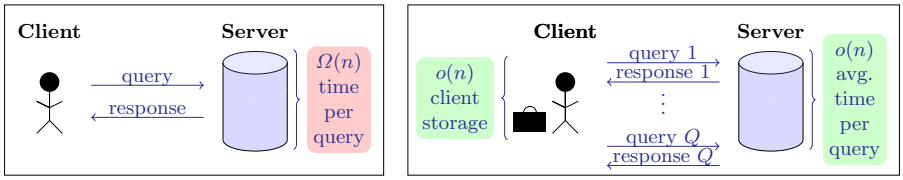
Towards reducing the server-side cost of PIR, a number of prior works [7,36, 58,62,73] observe that clients in many applications of PIR will make a sequence of queries to the same database. For example, a student may browse many books in a library; a web browser makes many domain name system (DNS) queries on each page load [74]; a mail client may check all incoming URLs against a database of known phishing websites [16,66]; or, an antivirus software may check the hashes of executed files against known malware [66]. The lower bound of Beimel, Ishai, and Malkin [14] only implies that a PIR server will take linear time to respond to the client's very first PIR query. This leaves open the possibility of reducing the server-side cost for subsequent queries. In other words, in the multi-query setting, we can hope for the *amortized* server-side time per query to be sublinear in the database size.

Indeed, there exist an array of techniques for constructing PIR schemes with sublinear amortized server-side cost. Yet, prior PIR schemes achieving sublinear amortized time come with limitations that make them cumbersome to use in practice. Schemes that require multiple non-colluding servers [36, 66, 83] demand careful coordination between many business entities, which is a major practical annoyance [4, 15, 75, 84]. In addition, the security of these schemes is relatively brittle, since it relies on an adversary not being able to compromise multiple servers, rather than on cryptographic hardness. Recent offline/online PIR schemes [36, 66, 83] require, in the single-server setting, the server to perform a linear-time preprocessing step *for each query*. Thus, these schemes cannot have sublinear amortized time. Batch-PIR schemes [7, 58, 62, 73], which require the client to make all of her queries at once, in a single non-adaptive batch, do not apply to many natural applications (e.g., digital library, web browsing), in which the client decides over time which elements she wants to query.

The world of private-information-retrieval is thus in an undesirable state: the practical applications are compelling, but existing schemes cannot satisfy the deployment demands (single server, adaptive queries, small storage, based on implementable primitives) while avoiding very large server-side costs.

1.1 Our results

This paper aims to advance the state of the art in private information retrieval by introducing the first PIR schemes that simultaneously offer a number of important properties for use in practice: they require only a single database server, they have sublinear amortized server time, they allow the client to issue its database queries adaptively, and they require extra storage sublinear in the database size (Figure 1). Our schemes further rely only on standard cryptographic primitives and incur no additional server-side (per client) storage, making them attractive



(a) Standard single-server PIR [68].

(b) *This work*: Single-server, many-query PIR with sublinear amortized time and storage.

Fig. 1: Comparison of single-server PIR models, on database size n.

even when many clients query a single database. One limitation of our schemes is that they require more client-side storage and computation than standard PIR schemes, though we give a lower bound showing that some of these costs are inherent to achieving sublinear amortized server time. While the schemes in this paper may not yet be concretely efficient enough to use in practice, they demonstrate that sublinear-amortized-time single-server PIR is theoretically feasible. We hope that future work pushes PIR even closer to practice.

Specifically, in this paper we construct two new families of PIR schemes:

Single-server PIR with sublinear amortized time from linearly homomorphic encryption. First, we show in Theorem 4.1 that any one of a variety of standard assumptions—including quadratic residuosity, decision Diffie-Hellman, decision composite residuosity, and learning with errors—suffices to construct single-server PIR schemes with sublinear amortized time. In particular, on database size n, if the client makes at least $n^{1/4}$ adaptive queries, our schemes have: amortized server time $n^{3/4}$, amortized communication complexity $n^{1/2}$, client storage $n^{3/4}$, and amortized client time $n^{1/2}$. (When describing protocol costs in this section, we hide both $\log n$ factors and polynomials in the security parameter.) More generally, the existence of linearly homomorphic encryption with sufficiently compact ciphertexts and standard single-server PIR with polylogarithmic communication together imply the existence of our PIR schemes. Our client-side costs are much larger than those required for standard stateless PIR—which needs no client storage and requires client time polylogarithmic in the database size. Our schemes thus reduce server-side costs at some expense to the client.

Single-server PIR with sublinear amortized time and an optimal storage/online-time trade-off from fully homomorphic encryption. Next, we show in Theorem 5.1 that under the stronger assumption that fully homomorphic encryption exists, we can construct PIR schemes with even lower amortized server time and client storage. In particular, we construct a PIR scheme that on database size n, and as long as the client makes at least $n^{1/2}$ queries, has amortized server time $n^{1/2}$, amortized communication complexity $n^{1/2}$, client storage $n^{1/2}$, and amortized client time $n^{1/2}$. (In contrast, from linearly homomorphic encryption, we get schemes with larger server time and client storage $n^{3/4}$.)

Lower bound on multi-query PIR. Finally, we give a new lower bound on PIR schemes in the amortized (i.e., multi-query) setting. In the *adaptive* setting, we

show in Theorem 6.2 that any multi-query PIR scheme on database size n in which: the client stores S bits between queries, the server stores the database in its original form, and the server runs in amortized online time T, it must be that $ST \geq n$. This lower bound implies that our fully-homomorphic-encryption-based PIR scheme achieves the optimal trade-off (up to $\log n$ factors and polynomials in the security parameter) between online server time and client storage, when the servers store the database in unmodified form.

1.2 Overview of techniques

We construct our new PIR schemes in two steps. First, we construct a new sort of *two*-server PIR scheme. Second, we use cryptographic assumptions to "compile" the two-server scheme into a single-server scheme.

Step 1: Two-server offline/online PIR with a single-server online phase. In the first step (Section 3), we design a new type of *two*-server offline/online PIR scheme [36]. The communication pattern of the two-server schemes we construct is as follows:

1. *Offline phase.* In a setup phase, the client sends a setup request to the first server (the "offline server"). The offline server runs in time at least linear in the database size and returns to the client a "hint" about the database state. The hint has size sublinear in the length of the database.
2. *Online phases (runs once for each of Q queries).* Whenever the client wants to make a PIR query, it uses its hint to issue a query to the second server (the "online server"). The online server produces an answer to the query in time sublinear in the database size and returns its answer to the client. The total communication in this step is sublinear in the database size.

   The client can run the online phase Q times—for some parameter Q determined by the PIR scheme—using the same hint and without communicating with the offline server. After Q queries, the client discards its hint and reruns the offline setup phase from scratch.

Prior offline/online PIR schemes [36] require the client to communicate with *both* servers in the online phases, whenever the client makes multiple queries with the same hint. (If the client only ever makes a single query, the client can communicate with only one server in the online phase, but then the scheme cannot achieve sublinear amortized time.) In contrast, our schemes crucially allow the client to only communicate with a single server (the online server) in the online phase. Unlike schemes for private stateful information retrieval [77], the online phase in our scheme runs in sublinear time.

To build our two-server offline/online PIR scheme, we give a generic technique for "compiling" a two-server PIR scheme that supports a *single* query with sublinear online time into one that supports *multiple* queries with sublinear online time. Plugging the existing single-query offline/online PIR schemes with sublinear online time [36, 83] into this compiler completes the two-server construction.

Provided that the offline server time is $\widetilde{O}(n)$ and the number of supported queries is at least n^ϵ, for constant $\epsilon > 0$, this two-server scheme already allows adaptive queries and has sublinear total amortized time and sublinear client storage. The only limitation is that it requires two non-colluding servers.

Step 2: Converting a two-server scheme to a one-server scheme. The last step (Sections 4 and 5) is to convert the two-server PIR scheme into a one-server scheme. Following Corrigan-Gibbs and Kogan [36], we have the client encrypt the hint request that she sends to the offline server using a fully homomorphic encryption scheme. (As we discuss in Section 4, Aiello, Bhatt, Ostrovsky, and Rajagopalan [2] proposed a similar technique for converting multi-prover proof systems to single-prover proof systems, formalizing the approach of Biehl, Meyer, and Wetzel [18].) The offline server can then homomorphically answer the client's hint request in the offline phase while learning nothing about it. At this point, the client can execute both the offline and online phases with the same server, which completes the construction.

To construct the PIR schemes from weaker assumptions (linearly homomorphic encryption), we exploit the linearity of the underlying two-server PIR scheme. In particular, we show that the hint that the client downloads from the offline server corresponds to a client-specified linear function applied to the database. With a careful balancing of parameters and application of linearly homomorphic encryption and standard single-server PIR, we show that the client can obtain this linear function without revealing it to the database server.

The construction of our most asymptotically efficient PIR scheme, which appears in Section 5, implicitly follows essentially the same two-step strategy. The only difference is that achieving the improved efficiency requires us to design a new two-server offline/online PIR scheme for multiple queries from scratch. The offline phase of this scheme requires the server to compute non-linear functions of the client query—and thus requires fully homomorphic encryption—but the online time of the scheme is lower, which is the source of efficiency improvements.

Lower bound. Our lower bound (Theorem 6.2) relates the number S of bits of information the client stores between queries and the amortized online time T of the PIR server, for PIR schemes in which the server stores the database in unmodified form. In particular, we show that $ST = \widetilde{\Omega}(n)$. To prove this lower bound, we show that if there is a single-server PIR scheme with client storage S and amortized online T, there exists a two-server offline/online PIR scheme for a single query with hint size S and online time T. Then, applying existing lower bounds on such schemes [36] completes the proof.

1.3 Related work

Multi-server PIR. Chor, Goldreich, Kushilevitz, and Sudan [35] introduced private information retrieval and gave the first protocols, which were in the multi-server information-theoretic setting and achieved communication $O(n^{1/3})$.

Table 2: A comparison of single-server, many-query PIR schemes. We present the per-query, asymptotic costs of each scheme, on a database of size n, where each of m clients, of which at most $\overline{m}$ may be corrupted, makes an unbounded sequence of queries. We omit poly-logarithmic factors in n and m, along with polynomial factors in the security parameter. For lower bounds, we denote the extra client storage by S. We use ϵ as an arbitrarily small, positive constant. We amortize the costs over the number of queries that minimizes the per-query costs. For each scheme, the table indicates:

- the additional cryptographic assumptions made beyond single-server PIR with poly-logarithmic communication,
- the number of queries (per client) over which we amortize,
- whether the client makes her queries adaptively or as a batch,
- the *amortized* number of bits communicated per query,
- the *amortized* client and server time per query, and
- the additional number of bits stored by the client and the server between queries.

For schemes in the offline/online model, the communication and computation costs are taken to be the sum of the offline costs, amortized over the number of queries supported by a single offline phase, and the online costs. The extra server storage does not include the n-bit database, stored by the server. The extra client storage does not include the indices queried, even if these indices are queried as a batch.

Scheme (extra assumptions)	*Per-client queries*	*Adaptive?*	*Per-query comm.*	Per-query time		Extra storage	
				Client	Server	Client	Server
Batch PIR [6, 58, 62]	Q	×	1	1	$\frac{n}{Q}$	0	0
Stateful PIR [77]	$n^{1/2}$	✓	$n^{1/2}$	n	n †	$n^{1/2}$	0
Single-query single-server PIR							
Standard [28, 68]	1	✓	1	1	n	0	0
Offline/online [36]	1	✓	$n^{2/3}$	$n^{2/3}$	n	$n^{2/3}$	0
Offline/online [36] (FHE)	1	✓	$n^{1/2}$	$n^{1/2}$	n	$n^{1/2}$	0
Download entire DB	$n^{1-\epsilon}$	✓	n^{ϵ}	n^{ϵ}	n^{ϵ}	n	0
Doubly-efficient PIR							
Secret key (OLDC) [25, 29]	$n^{1-\epsilon}$	✓	n^{ϵ}	n^{ϵ}	n^{ϵ}	1	mn
Public key (OLDC+**VBB**) [25]	1 *	✓	n^{ϵ}	n^{ϵ}	n^{ϵ}	0	n
Private anonymous data access							
Read-only [57] (FHE)	1 *	✓	$\overline{m}$	$\overline{m}$	$\overline{m}$	$\overline{m}$	$\overline{m}n^{1+\epsilon}$
This work							
Theorem 4.1 (LHE)	$n^{1/4}$	✓	$n^{1/2}$	$n^{1/2}$	$n^{3/4}$	$n^{3/4}$	0
Theorem 5.1 (FHE)	$n^{1/2}$	✓	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	0
Lower bounds, for Q queries, *on schemes storing the database in its original form*							
Standard PIR [14]		×	–	–	$\geq \frac{n}{Q}$	–	–
This work (Theorem 6.2)		✓	–	–	$\geq \frac{n}{S}$	S	0

† The number of public-key operations is $n^{1/2}$.

* This number of per-client queries assumes that the total number of clients, m, grows sufficently large.

A sequence of works [5,11,12,21,22,33,40,43,48,88] then improved the communication complexity of PIR, and today's PIR schemes can achieve sub-polynomial communication complexity in the information-theoretic setting [40] and logarithmic communication complexity in the computational setting [22]. Multi-server PIR schemes are more efficient, both in terms of communication and computation, than single-server schemes. However, the security of multi-server PIR relies on non-collusion between the servers, which can be hard to guarantee in practice.

Single-server PIR. Kushilevitz and Ostrovsky [68] presented the first single-server PIR schemes, based on linearly homomorphic encryption. A sequence of works then improved the communication complexity of single-server PIR, and showed how to construct PIR schemes with polylogarithmic communication from a wide range of public-key assumptions, such as the ϕ-hiding assumption [28,46], the decisional composite-residuosity assumption [30, 71], the decisional Diffie-Hellman assumption [39], and the quadratic-residuosity assumption [39].

Recent works [1, 4, 6, 45, 75] have used lattice-based encryption schemes to improve the concrete efficiency of single-server PIR, in terms of both communication and computation. The goal is to get the most efficient single-server PIR schemes subject to the linear-server-time lower bound. These techniques are complementary to ours, and applying lattice-based optimizations to our setting could improve the concrete efficiency of our protocols.

Computational overhead of PIR. All early PIR protocols required the servers to perform work linear in the database size when responding to a query. Beimel, Ishai, and Malkin [14] showed that this is inherent, giving an $\Omega(n)$ lower bound on the server time. Their lower bound applies to both multi-server and single-server schemes with either information-theoretic or computational security.

Many lines of work have sought to construct PIR schemes with lower computational costs, which circumvent the above linear lower bound:

- *PIR with preprocessing* denotes a class of schemes in which the server(s) store the database in encoded form [13, 14, 87], which allows them to respond to queries in time sublinear in the database size. The first such schemes targeted the multi-server setting. Recent work [25, 29] applies oblivious locally decodable codes [19, 23, 24] to construct single-server PIR schemes with sublinear server time, after a one-time database preprocessing step. However, these schemes require extra server-side storage per client that is linear in the database size. While an idealized form of program obfuscation [9] can be used to drastically reduce this storage [25], the lack of concretely efficient candidate constructions for program obfuscation rules out the use of these schemes for the time being. In contrast, the single-server schemes in this paper require only standard assumptions.

  "Offline/online PIR" schemes use a different type of preprocessing: the client and server run a one-time linear-complexity offline setup process, during which the client downloads and stores information about the database. After that, the client can make queries to the database, and the server can respond in sublinear time. Previous works [36, 66, 83] mostly focus on the two-server

setting, where they achieve sublinear amortized time. In the single-server setting, previous offline/online PIR schemes [36] allow for only a single online query after each execution of the offline phase. As a result, in the single-server setting, the cost of each query is still linear in the database size.

Finally, Lipmaa [72] constructs single-server PIR with slighly sublinear time by encoding the database as a branching program that is obliviously evaluated in $O(\frac{n}{\log n})$ operations. The schemes in this work achieve significantly lower amortized time, yet require the client to make multiple queries.

- *Make queries in a non-adaptive batch:* When the client knows the entire sequence of database queries she will make in advance, the client and server can use "batch PIR" schemes [6,7,31,55,58,59,62] to achieve sublinear amortized server time. The multi-server scheme of Lueks and Goldberg [73] allows the servers to simultaneously process a batch of queries from different clients, and achieves sublinear per-query time. Our schemes require only one server and achieve sublinear amortized time, even given a single client making her queries in an adaptive sequence.
- *Download and store the entire database:* If the client has enough storage space, she can keep a local copy of the entire database. The server pays a linear cost to ship the database to the client, but the client can answer subsequent database queries on her own with no server work. In contrast, the schemes in this paper avoid having to store the entire database at the client.
- *Settle on a sublinear number of public-key operations:* Private stateful information retrieval [77] schemes improve the concrete efficiency of single-server PIR by having the server do a sublinear number of public-key operations for each query. Such schemes [75, 77] still require a linear number of symmetric key and plaintext operations for each query. In contrast, the schemes in this paper require sublinear amortized work of any kind, per query.

Communication lower bounds on PIR. A series of works give bounds on the communication required for multi-server PIR [49,86]. Single-server PIR constructions match the trivial $\log n$ lower bound (up to polylogarithmic factors).

Lower bounds for PIR with preprocessing. Beimel, Ishai, and Malkin [13] proved that if a server can store an S-bit hint and run in amortized time T, then it must hold that $ST \geq n$. Persiano and Yeo [78] recently improved this lower bound to $ST \geq n \log n$ in the single-server case. In this paper, we are interested in offline/online PIR schemes, in which the client store the hint, and the server stores the database in unmodified form.

Lower bounds on oblivious RAM. Recent work proves strong limits on the performance of oblivious-RAM [51] schemes [26, 63, 67, 69, 70]. These schemes allow the server to maintain per-client state; in our setting of PIR, the server is stateless. The PIR setting thus requires different lower-bound approaches [13].

2 Background

Notation. We write the set of positive integers as $\mathbb{N}$. For an integer $n \in \mathbb{N}$, we write $[n] = \{1, \ldots, n\}$ and we write the empty set as $\emptyset$. We ignore issues of integrality, and treat numbers such as $n^{1/2}$ and n/k as integers. We use $\text{poly}(\cdot)$ to denote a fixed polynomial in its argument. We use the standard Landau notation $O(\cdot)$ and $\Omega(\cdot)$ for asymptotics. When the big-O contains multiple variables, such as $f(n) = O(n/S)$, all variables other than n are implicit functions of n (which is the database size when it is not made explicit). The notation $\widetilde{O}(f(n))$ hides polylogarithmic factors in the parameter n, and $\widetilde{O}_\lambda(\cdot)$ hides $\text{poly}(\log n, \lambda)$ factors. For a finite set $\mathcal{X}$, $x \xleftarrow{R} \mathcal{X}$ denotes an independent and uniformly random draw from $\mathcal{X}$. When unspecified, we take all logarithms base two.

We work in the RAM model, with word size logarithmic in the input length (i.e., database size n) and polynomial in the security parameter λ. We give running times up to $\text{poly}(\log n, \lambda)$ factors, which makes our results relatively independent of the specifics of the computational model. An "efficient algorithm" is one that runs in probabilistic polynomial time in its inputs and in λ.

2.1 Standard definitions

We begin by defining the standard cryptographic primitives that this work uses.

Pseudorandom permutations. We use the standard notion of pseudorandom permutations [50]. On security parameter $\lambda \in \mathbb{N}$, a domain size $n \in \mathbb{N}$, and a key space $\mathcal{K}_\lambda$, we denote a pseudorandom permutation by $\mathsf{PRP} : \mathcal{K}_\lambda \times [n] \to [n]$.

Definition 2.1 (Linearly homomorphic encryption). Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. The scheme is *linearly homomorphic* if, for every keypair $(\mathsf{sk}, \mathsf{pk})$ that Gen outputs,

- the message space is a group $(\mathcal{M}_{\mathsf{pk}}, +)$,
- the ciphertext space is a group $(\mathcal{C}_{\mathsf{pk}}, \cdot)$, and
- for every pair of messages $m_0, m_1 \in \mathcal{M}_{\mathsf{pk}}$, it holds that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m_0) \cdot \mathsf{Enc}(\mathsf{pk}, m_1) \in \mathcal{C}_{\mathsf{pk}}) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m_0 + m_1 \in \mathcal{M}_{\mathsf{pk}})).$$

Definition 2.2 (Gate-by-gate fully homomorphic encryption). We use $(\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ to denote a symmetric-key fully homomorphic encryption scheme [44]. We say a scheme is a *gate-by-gate* fully homomorphic encryption scheme if the homomorphic evaluation routine $\mathsf{FHE.Eval}$ on a circuit of size $|C|$ and security parameter λ runs in time $|C| \cdot \text{poly}(\log |C|, \lambda)$. Standard fully homomorphic encryption schemes are gate-by-gate [27, 44, 47].

2.2 Definition of offline/online PIR

Throughout, we present our new single-server PIR schemes in an offline/online model [36,77]. That is, the client first interacts with the server in an offline phase

to obtain a succinct "hint" about the database contents. This hint allows the client to make many queries in a subsequent online phase. Provided that the server-side cost is low enough in both phases, the server's total amortized time (including the cost of both phases) will be sublinear in the database size.

We now give definitions for one- and two-server offline/online PIR schemes that support many adaptive queries. Our definition of offline/online PIR differs from that of prior work in one important way [36, 66]. In our definition, in the two-server setting, the client may only communicate with a *single* server in the online phase. Prior two-server offline/online PIR schemes [36,66] allow the client to communicate with *both* servers in the online phase.

Definition 2.3 (Offline/online PIR for adaptive queries). An *offline/online PIR scheme for adaptive queries* is a tuple of polynomial-time algorithms:

- $\mathsf{HintQuery}(1^\lambda, n) \to (\mathsf{ck}, q)$, a randomized algorithm that takes in a security parameter λ and a database length $n \in \mathbb{N}$, and outputs a client key ck and a hint request q,
- $\mathsf{HintAnswer}(D, q) \to a$, a deterministic algorithm that takes in a database $D \in \{0,1\}^n$ and a hint request q, and outputs a hint answer a,
- $\mathsf{HintReconstruct}(\mathsf{ck}, a) \to h$, a deterministic algorithm that takes in a client key ck and a hint answer a, and outputs a hint h,
- $\mathsf{Query}(\mathsf{ck}, i) \to (\mathsf{ck}', \mathsf{st}, q)$, a randomized algorithm that takes in a client key ck and an index $i \in [n]$, and outputs an updated client key ck', a client query state st, and a query q,
- $\mathsf{Answer}^D(q) \to a$, a deterministic algorithm that takes in 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 $D_j \in \{0,1\}$,

  and outputs an answer string a, and
- $\mathsf{Reconstruct}(\mathsf{st}, h, a) \to (h', D_i)$, a deterministic algorithm that takes in a query state st, a hint h, and an answer string a, and outputs an updated hint h' and a database bit D_i.

In a deployment, ($\mathsf{HintQuery}, \mathsf{HintAnswer}, \mathsf{HintReconstruct}$) are executed in the offline phase, while ($\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct}$) are executed in each online phase. Furthermore, we say that the PIR scheme *supports Q adaptive queries* if it satisfies the following notions of (1) correctness and (2) security for Q queries:

Correctness for Q queries. We require that if a client and a server correctly execute the protocol, the client can recover any Q database records of its choosing, even if the client chooses these records adaptively. Formally, a multi-query offline/online PIR scheme Π satisfies *correctness for Q queries* if for every $\lambda, n \in \mathbb{N}$, $D \in \{0,1\}^n$, and every $(i_1, \ldots, i_Q) \in [n]^Q$, Experiment 2.1 outputs "1" with probability $1 - \mathrm{negl}(\lambda)$.

Security for Q queries. We require that an adversarial (malicious) server "learns nothing" about which sequence of database records the client is fetch-

Experiment 2.1 (Correctness). Parameterized by a PIR scheme Π, security parameter $\lambda \in \mathbb{N}$, number of queries $Q \in \mathbb{N}$, database size $n \in \mathbb{N}$, database $D \in \{0,1\}^n$, and query sequence $(i_1, \ldots, i_Q) \in [n]^Q$.

- Compute:

$$(\mathsf{ck}, q) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, n)$$
$$a \leftarrow \Pi.\mathsf{HintAnswer}(D, q)$$
$$h \leftarrow \Pi.\mathsf{HintReconstruct}(\mathsf{ck}, a)$$

- For $t = 1, \ldots, Q$, compute:

$$(\mathsf{ck}, \mathsf{st}, q) \leftarrow \Pi.\mathsf{Query}(\mathsf{ck}, i_t)$$
$$a \leftarrow \Pi.\mathsf{Answer}^D(q)$$
$$(h, v_i) \leftarrow \Pi.\mathsf{Reconstruct}(\mathsf{st}, h, a)$$

- Output "1" if $v_t = D_{i_t}$ for all $t \in [Q]$. Output "0" otherwise.

Experiment 2.2 (Security). Parameterized by an adversary $\mathcal{A}$, PIR scheme Π, number of servers $k \in \{1, 2\}$, security parameter $\lambda \in \mathbb{N}$, number of queries $Q \in \mathbb{N}$, database size $n \in \mathbb{N}$, and bit $b \in \{0, 1\}$.

- Compute:

$$(\mathsf{ck}, q) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, n)$$
If $k = 1$: *// Single-server security*
$$\mathsf{st} \leftarrow \mathcal{A}(1^\lambda, q)$$
Else: *// Two-server security*
$$\mathsf{st} \leftarrow \mathcal{A}(1^\lambda)$$

- For $t = 1, \ldots, Q$, compute:

$$(\mathsf{st}, i_0, i_1) \leftarrow \mathcal{A}(\mathsf{st})$$
$$(\mathsf{ck}, \_, q) \leftarrow \Pi.\mathsf{Query}(\mathsf{ck}, i_b)$$
$$\mathsf{st} \leftarrow \mathcal{A}(\mathsf{st}, q)$$

- Output $b' \leftarrow \mathcal{A}(\mathsf{st})$.

ing, even if the adversary can adaptively choose these indices. In the single-server setting, where the same server runs both the offline and online phase, the adversary is first given the hint request. In the two-server setting, where a separate server runs the offline phase, the adversary only sees the online queries. (This is sufficient, as an adversarial offline server trivially learns nothing about the client's queries since the hint request does not depend on these queries.)

Formally, for an adversary $\mathcal{A}$, multi-query offline/online PIR scheme Π, number of servers $k \in \{1, 2\}$, security parameter $\lambda \in \mathbb{N}$, database size $n \in \mathbb{N}$, and bit $b \in \{0, 1\}$, let $W_{\mathcal{A},k,\lambda,Q,n,b}$ be the event that Experiment 2.2 outputs "1" when parameterized with these values. We define the Q-query PIR advantage of $\mathcal{A}$:

$$\mathsf{PIRAdv}_k[\mathcal{A}, \Pi](\lambda, n) := \left|\Pr[W_{\mathcal{A},k,\lambda,Q,n,0}] - \Pr[W_{\mathcal{A},k,\lambda,Q,n,1}]\right|.$$

We say that a multi-query offline/online PIR scheme Π is *k-server secure* if, for all efficient algorithms $\mathcal{A}$, all polynomially bounded functions $n(\lambda)$, and all $\lambda \in \mathbb{N}$, $\mathsf{PIRAdv}_k[\mathcal{A}, \Pi](\lambda, n(\lambda)) \leq \mathrm{negl}(\lambda)$.

Definition 2.4 (Sublinear amortized time). We say that an offline/online PIR scheme has *sublinear amortized time* if the there exists a number of queries $Q \in \mathbb{N}$ such that the total server time required to run the offline and online phases for Q queries on a database of size n is $o(Qn)$. More formally, for every choice of the security parameter $\lambda \in \mathbb{N}$, database size $n \in \mathbb{N}$, and query sequence $(i_1, \ldots, i_Q) \in [n]^Q$, the total running time of $\mathsf{HintAnswer}$ (executed once) and Answer (executed Q times) in Experiment 2.1 must be $o(Qn)$.

Remark 2.5 (Handling an unbounded number of queries). A scheme with sublinear amortized time for some number of queries $Q \in \mathbb{N}$ immediately implies a scheme with sublinear amortized time for any larger number of queries, including a number that is a-priori unbounded. One can obtain such a scheme by "restarting" the scheme every Q queries and rerunning the offline phase from scratch. The amortized costs remain the same.

Remark 2.6 (Malicious security). In our definition (Definition 2.3), following prior work [36], the client's queries do not depend on the server's answers to prior queries. In this way, our PIR schemes naturally protect client privacy against a malicious server—the server learns the same information about the client's queries whether or not the server executes the protocol faithfully.

Remark 2.7 (Correctness failures). Our definition does not require that correctness holds if the client makes a sequence of queries that is correlated with the randomness it used to generate the hint request. A stronger correctness definition would guarantee correctness in all cases (i.e., with probability one). Strengthening our PIR schemes to provide this form of correctness represents an interesting challenge for future work.

Remark 2.8 (Handling database changes). In many natural applications of private information retrieval, the database contents change often. Naïvely, whenever the database contents change, the client and server would need to rerun the costly hint-generation process. In the limit—when the entire contents of the database changes between a client's queries—rerunning the hint-generation step is inherently required. When the database changes more slowly, prior work on offline/online PIR [66], building on much earlier work in dynamic data structures [17], shows how to update the client's hint at modest cost. In particular, when a constant number of database rows change between each pair of client queries, the scheme's costs do not change, up to factors in the security parameter and logarithmic in the database size. These techniques from prior work apply directly to our setting, so we do not discuss them further.

3 Two-server PIR with a single-server online phase and sublinear amortized time

In this section, we give a generic construction that converts a two-server offline/online PIR scheme that supports *a single query* into a two-server offline/online PIR scheme that supports *any number of adaptive queries*. The transformation has three useful properties:

1. If the original PIR scheme has linear offline server time, then the resulting multi-query scheme has linear offline server time as well.
2. If the original PIR scheme has sublinear online server time, then the resulting multi-query scheme has sublinear online server time as well.

3. During the online phase—when the client is making its sequence of adaptive queries—the client only communicates with one of the servers. (In contrast, prior two-server PIR schemes with sublinear amortized time [36, 66] require the client to communicate with *both* servers in the online phase.)

After presenting the generic transformation (Lemma 3.1) in this section, we instantiate this transformation in Section 4 and use it to construct single-server PIR schemes with sublinear amortized time.

Lemma 3.1 (The Compiler Lemma). *Let Π be a two-server offline/online PIR scheme that supports a single query. Then, for any database size $n \in \mathbb{N}$, security parameter $\lambda \in \mathbb{N}$, and number of queries $Q < n$, Construction 3.5, when instantiated with a secure pseudorandom permutation, is a two-server offline/online PIR scheme that supports Q* adaptive queries *and whose offline and online phases have communication, computation, and client storage costs dominated by running $O(\lambda Q)$ instances of Π, each on a database of size n/Q.*

To prove the lemma, we must show that the scheme of Construction 3.5 satisfies the claimed efficiency properties, along with correctness and security. Efficiency follows by construction. We give the full correctness and security arguments in Appendix A.

Remark 3.2. In the PIR scheme implied by Lemma 3.1, the online-phase upload communication (from the client to server) is in fact only as large as the upload communication required for running a *single* instance of the underlying PIR scheme Π on a database of size n/Q.

Before giving the construction that proves Lemma 3.1, we describe the idea behind our approach. We take inspiration from the work of Ishai, Kushilevitz, Ostrovsky, and Sahai [62], who construct "batch" PIR schemes, in which the client can issue a batch of Q queries at once, and the server can respond to all Q queries in time $\widetilde{O}(n)$. (In contrast, answering Q queries using a non-batch PIR scheme requires server time $\Omega(Qn)$.) The crucial difference between our PIR schemes and prior work on batch PIR is that our schemes allow the client to make its Q queries adaptively, rather than in a single batch all at once.

Our idea is to first permute the database according to a pseudorandom permutation and then partition the n database records into Q chunks, each of size n/Q. The key observation is that, if the client makes Q adaptive queries, it is extremely unlikely that the client will ever need to query any chunk more than λ times. In particular, by a balls-in-bins argument, the probability, taken over the random key of the pseudorandom permutation, that any chunk receives more than λ queries is negligible in λ.

Then, given a two-server offline/online PIR scheme Π for a *single query*, we construct a two-server offline/online PIR scheme for many queries as follows:

- *Offline phase.* The client and the offline server run the offline phase of Π on each of the Q database chunks λ times. For each of the Q database chunks, the client then holds λ client keys and hints.

– *Online phase.* Whenever the client wants to make a database query, it identifies the chunk in which its desired database record falls. The client finds an unused client key for that chunk and runs the online phase of Π for that chunk to produce a query. The client sends the query to the online server, who answers that query with respect to each of the Q database chunks. Using the online server's answers, the client can reconstruct its database record of interest. Crucially, the client's query does not reveal to the server the chunk in which its desired database record falls. Finally, the client then deletes the client key and hint that it used for this query.

The formal description of our protocol appears in Construction 3.5.

Remark 3.3. Construction 3.5 uses a pseudorandom permutation (PRP) to permute and partition the database. The client then reveals the PRP key it used for this partitioning to the server. Crucially, the security of our construction does *not* rely on the pseudorandomness of the PRP. The PRP security property only appears in the correctness argument of our scheme (Appendix A). So, revealing the PRP key to the server in this way has no effect on the security of the scheme.

Remark 3.4 (Reducing online download). In the online phase of Construction 3.5, the online server's answer to the client consists of a vector of Q answers $a = ((a)_1, \ldots, (a)_Q)$. The client uses only one of these answers $(a)_{j^*}$. To reduce download cost, the client and server can run a single-server PIR protocol, where the server's input is the database a of Q answers and the client's input is the index $j^* \in [Q]$ of it's desired answer. This reduces the client's online download cost by a factor of Q, at the cost of requiring the server to perform $O_\lambda(Q)$ public-key operations in the online phase.

4 Single-server PIR with sublinear amortized time from DCR, QR, DDH, or LWE

In this section, we use the general transformation of Section 3 to construct the first single-server PIR schemes with sublinear amortized total time and sublinear client storage, relying on only standard cryptographic assumptions.

These constructions work in two steps:

– First, we use the Compiler Lemma (Lemma 3.1) to convert a two-server offline/online PIR scheme for a single query into a two-server offline/online PIR scheme for *multiple adaptive queries*, in which the client only communicates with a single server in the online phase.
– Next, we use linearly homomorphic encryption and single-server PIR to allow the client and server to run the offline phase of the two-server scheme without leaking any information to the server. At this point, we can execute the functionality of both servers in the two-server scheme using just a single server. In other words, we have constructed a single-server offline/online PIR scheme that supports multiple adaptive queries.

Construction 3.5 (Two-server offline/online PIR for Q adaptive queries with a single-server online phase). The scheme uses a single-query two-server offline/online PIR scheme Π and a pseudorandom permutation $\mathsf{PRP} : \mathcal{K}_\lambda \times [n] \to [n]$. The scheme is parameterized by a maximum number of queries $Q = Q(n) < n$.

I. Offline phase.

$\mathsf{HintQuery}(1^\lambda, n) \to (\mathsf{ck}, q)$.

1. For $j \in [Q]$ and $\ell \in [\lambda]$: $((\hat{\mathsf{ck}})_{j\ell}, (\hat{q})_{j\ell}) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, n/Q)$.
2. Sample $k \xleftarrow{R} \mathcal{K}_\lambda$, set $\mathsf{ck} \leftarrow (k, \hat{\mathsf{ck}}, \emptyset)$, and set $q \leftarrow (k, \hat{q})$.
3. Return (ck, q).

$\mathsf{HintAnswer}(D, q) \to a$.

1. Parse $(k, \hat{q}) \leftarrow q$.
2. *// Permute the database according to* $\mathsf{PRP}(k, \cdot)$ *and divide it into* Q *chunks.* For $j \in [Q]$: $C_j \leftarrow (D_{\mathsf{PRP}(k,(j-1)(n/Q)+1)} \| \ldots \| D_{\mathsf{PRP}(k,(j+1)(n/Q))}) \in \{0,1\}^{n/Q}$.
3. For $j \in [Q]$ and $\ell \in [\lambda]$: $(a)_{j\ell} \leftarrow \Pi.\mathsf{HintAnswer}(C_i, (\hat{q})_{j\ell})$.
4. Return a.

$\mathsf{HintReconstruct}(\mathsf{ck}, a) \to h$.

1. For $j \in [Q]$ and $\ell \in [\lambda]$: $(\hat{h})_{j\ell} \leftarrow \Pi.\mathsf{HintReconstruct}((\mathsf{ck})_{j\ell}, (a)_{j\ell})$.
2. Set $\mathsf{cache} \leftarrow \{\}$. *// An empty map (associative array) data structure.*
3. Return $h = (\hat{h}, \mathsf{cache})$.

II. Online phase.

$\mathsf{Query}(\mathsf{ck}, i) \to (\mathsf{ck}', \mathsf{st}, q)$.

1. Parse $(k, \hat{\mathsf{ck}}, \mathsf{queried}) \leftarrow \mathsf{ck}$.
2. Find (the unique) integers $i^* \in [n/Q]$ and $j^* \in [Q]$ such that $\mathsf{PRP}(k, i) = (j^* - 1)(n/Q) + i^*$.
3. Find $\ell^* \in [\lambda]$ such that $(\mathsf{ck})_{j^*\ell^*} \neq \perp$.
   - If no such ℓ^* exists or $i \in \mathsf{queried}$, sample $i^* \xleftarrow{R} [n/Q]$ and choose a random $j^* \in [Q]$ and $\ell^* \in [\lambda]$ out of those for which $(\mathsf{ck})_{j^*\ell^*} \neq \perp$.
4. Let $(\_, \mathsf{st}', q') \leftarrow \Pi.\mathsf{Query}((\hat{\mathsf{ck}})_{j^*\ell^*}, i^*)$.
5. Let $(\hat{\mathsf{ck}})_{j^*\ell^*} \leftarrow \perp$, let $\mathsf{st} \leftarrow (\mathsf{st}', i, j^*, \ell^*)$, let $q \leftarrow (k, q')$, and let $\mathsf{ck}' \leftarrow (k, \hat{\mathsf{ck}}, \mathsf{queried} \cup \{i\})$.
6. Return $(\mathsf{ck}', \mathsf{st}, q)$.

$\mathsf{Answer}^D(q) \to a$.

1. Parse $(k, q') \leftarrow q$.
2. For $j \in [Q]$: $(a)_j \leftarrow \Pi.\mathsf{Answer}^{\mathcal{O}_j}(q')$, where $\mathcal{O}_j(x) := D_{\mathsf{PRP}(k,(j-1)(n/Q)+x)}$.
3. Return a.

$\mathsf{Reconstruct}(\mathsf{st}, h, a) \to (h', D_i)$.

1. Parse $(\mathsf{st}', i, j^*, \ell^*) \leftarrow \mathsf{st}$ and parse $(\hat{h}, \mathsf{cache}) \leftarrow h$.
2. If $\mathsf{cache}[i]$ is not set, let $\mathsf{cache}[i] \leftarrow \Pi.\mathsf{Reconstruct}(\mathsf{st}', (\hat{h})_{j^*\ell^*}, (a)_{j^*})$.
3. Set $D_i \leftarrow \mathsf{cache}[i]$. Set $h' \leftarrow (\hat{h}, \mathsf{cache})$.
4. Return (h', D_i).

The idea of using homomorphic encryption to run a two-server protocol on a single server arose first, to our knowledge, in the domain of multi-prover interactive proofs. Aiello, Bhatt, Ostrovsky, and Rajagopalan [2] formalized this general approach, which was initially proposed by Biehl, Meyer, and Wetzel [18]. Subsequent work demonstrated that compiling multi-prover proof systems to single-prover systems requires care [38,41,42,65,85] (in particular it requires the underlying proof system to be sound against "no-signaling" provers [85]). Corrigan-Gibbs and Kogan [36] used homomorphic encryption to convert a two-server PIR scheme to a single-server offline/online PIR scheme that supports a *single* query in sublinear online time. Our contribution is to construct a single-server PIR scheme that supports multiple, adaptive queries and that thus achieves *sublinear amortized total time*.

We now show that any one of a variety of cryptographic assumptions—the Decision Composite Residuosity assumption [71, 76], the Quadratic Residuosity assumption [52], the Decision Diffie-Hellman assumption [20], or the Learning with Errors assumption [81]—suffices for constructing single-server PIR with sublinear amortized time:

Theorem 4.1 (Single-server PIR with sublinear amortized time). *Under the DCR, LWE, QR, or DDH assumptions, there exists a single-server offline/online PIR scheme that, on database size n, security parameter λ, and as long as the client makes at least $n^{1/4}$ adaptive queries, has*

- *amortized communication $\widetilde{O}_\lambda(n^{1/2})$,*
- *amortized server time $\widetilde{O}_\lambda(n^{3/4})$,*
- *amortized client time $\widetilde{O}_\lambda(n^{1/2})$, and*
- *client storage $\widetilde{O}_\lambda(n^{3/4})$.*

The proof of Theorem 4.1 will make use of the following two-server offline/online PIR scheme which is implicit in prior work.

Lemma 4.2 (Implicit in Theorem 20 of CK20 [36]). *There is a two-server offline/online PIR scheme (with information-theoretic security) that supports a single query on database size n such that, in the offline phase:*

- *the client uploads a vector $q \in \{0,1\}^n$ to the offline server,*
- *the offline server computes the inner product of the database with all n cyclic shifts of the query vector q (in $\widetilde{O}(n)$ time using a fast Fourier transform),*
- *the client downloads $\widetilde{O}(\sqrt{n})$ bits of the resulting matrix-vector product*

and, in the online phase:

- *the client uploads $\widetilde{O}(\sqrt{n})$ bits to the online server,*
- *the online server runs in time $\widetilde{O}(\sqrt{n})$, and*
- *the client downloads one bit.*

Proof of Theorem 4.1. The proof works in two main steps. First, we use Lemma 3.1 to "compile" the single-query two-server PIR scheme of Lemma 4.2 into a multi-query two-server PIR scheme. Second, we use linearly homomorphic encryption—

following the work of Corrigan-Gibbs and Kogan [36] in the single-query setting—to allow a single server to implement the role of both servers.

Step 1: A stepping-stone two-server scheme. Our first step is to construct a two-server offline/online PIR scheme that: (a) supports multiple queries, (b) has sublinear online time, and (c) requires only one server in the online phase. To complete this step, we use the Compiler Lemma (Lemma 3.1) to convert the two-server PIR scheme of Lemma 4.2 into a two-server PIR scheme that satisfies these three goals.

In particular, Lemma 3.1 and Lemma 4.2 together imply a two-server offline/online PIR scheme that supports any number of queries $Q < n$, and whose offline and online phases consist of running $O(\lambda Q)$ instances of the PIR scheme of Lemma 4.2 on databases of size n/Q. The resulting scheme then has the following structure in the offline phase:

- the client uploads $\widetilde{O}_\lambda(Q)$ bit vectors to the offline server, each of size n/Q,
- the offline server applies a length-preserving linear function to each vector (in quasi-linear time, as in the Lemma 4.2 scheme),
- the client downloads a total of $\widetilde{O}_\lambda(\sqrt{Qn})$ bits from the vectors that the server computes.

And in the online phase,

- the client uploads $\widetilde{O}_\lambda(\sqrt{Qn})$ bits to the online server,
- the online server runs in time $\widetilde{O}_\lambda(\sqrt{Qn})$, and
- the client downloads $\widetilde{O}_\lambda(Q)$ bits.

This scheme requires the existence of one-way functions.

As desired, this scheme supports multiple queries, has sublinear online time (whenever $Q \ll n$), and requires only one server in the online phase. The offline upload cost and the client time of the scheme are $\widetilde{\Omega}_\lambda(n)$—linear in the database size, but we remove this limitation later on.

Step 2: Using homomorphic encryption to run the two-server scheme on one server. Next, we show that the client can fetch the information it needs to complete the offline phase of the Step-1 scheme without revealing any information to the server. In the Step-1 scheme, the offline server's work consists of evaluating a client-supplied linear function over the database and can thus be performed under linearly homomorphic encryption. For this step, we will need a linearly homomorphic encryption scheme with ciphertexts of size $\widetilde{O}_\lambda(1)$, along with a single-server PIR scheme with communication cost and client time $\widetilde{O}_\lambda(1)$. The existence of both primitives follows from the Decision Composite Residue (DCR) assumptions [71, 76] and the Learning with Errors (LWE) assumption [81]. Recent work of Döttling, Garg, Ishai, Malavolta, Mour, and Ostrovsky [39] shows that the Quadratic Residuosity (QR) assumption [52] and decision Diffie-Hellman (DDH) assumption [20] also imply these primitives.

In particular, the client first samples a random encryption key for a linearly homomorphic encryption scheme. Then the client executes the offline phase as follows:

– The client encrypts each component of its $\widetilde{O}_\lambda(Q)$ bit vectors using the linearly homomorphic encryption scheme. The client sends these vectors to the server.
– Under encryption, the server applies the length-preserving linear function to each encrypted vector. As in the Step-1 scheme, this computation takes $\widetilde{O}_\lambda(n)$ time using an FFT on the encrypted values.
– The client uses a single-server PIR scheme [68], to fetch a total of $\widetilde{O}_\lambda(\sqrt{Qn})$ components of the ciphertext vectors that the server has computed. Since modern single-server PIR schemes have communication cost $\widetilde{O}_\lambda(1)$, this step requires communication and client time $\widetilde{O}_\lambda(\sqrt{Qn})$. Using batch PIR [7,58,62], the server can answer this set of queries in time $\widetilde{O}_\lambda(n)$.

Finally, the client decrypts the resulting ciphertexts to recover exactly the same information that it obtained at the end of the offline phase of the two-server scheme. At this point, the offline phase has upload $\widetilde{O}_\lambda(n)$, server time $\widetilde{O}_\lambda(n)$, client time $\widetilde{O}_\lambda(n)$, and download $\widetilde{O}_\lambda(\sqrt{Qn})$ and the online phase has upload $\widetilde{O}_\lambda(\sqrt{Qn})$ bits, server time $\widetilde{O}(\sqrt{Qn})$, client time $\widetilde{O}_\lambda(\sqrt{Qn}+Q)$, and download $\widetilde{O}_\lambda(Q)$.

Final rebalancing. We complete the proof by reducing the offline upload cost using the standard rebalancing idea [34, Section 4.3]. In particular, we divide the database into k chunks, of size $n' = n/k$, for a parameter k chosen later.

Now, the offline phase has upload $\widetilde{O}_\lambda(n/k)$, server time $\widetilde{O}_\lambda(n)$, client time $\widetilde{O}_\lambda(n/k+\sqrt{Qnk})$, and download $k\cdot\widetilde{O}_\lambda(\sqrt{Qn/k})$ and the online phase has upload $\widetilde{O}_\lambda(\sqrt{Qn/k})$ bits, server time $k\cdot\widetilde{O}(\sqrt{Qn/k})$, client time $\widetilde{O}_\lambda(\sqrt{Qn/k}+Qk)$ and download $k\cdot\widetilde{O}_\lambda(Q)$. We choose Q and k to balance the following costs, ignoring $\mathrm{poly}(\lambda,\log n)$ factors:

– the amortized offline time: n/Q, and
– the online server time: $\sqrt{kQn}$.

To do so, we choose $k = \frac{n}{Q^3}$ and $Q \le n^{1/3}$. This yields a PIR scheme with amortized server time $\widetilde{O}_\lambda(n/Q)$, amortized client time $\widetilde{O}_\lambda(Q^2 + n/Q^2)$ and amortized communication $\widetilde{O}_\lambda(Q^2 + n/Q^2)$. The client storage is equal to the (non-amortized) offline download cost, which is $\widetilde{O}_\lambda(n/Q)$.

Finally, to construct the scheme of Theorem 4.1, we chose $Q = n^{1/4}$ to minimize the offline upload. This causes the amortized server time and the client storage to become $\widetilde{O}_\lambda(n^{3/4})$, while the amortized client time and the amortized communication are both $\widetilde{O}_\lambda(n^{1/2})$.

Efficiency. The efficiency claims of Theorem 4.1 follow immediately from the construction.

Security. The security argument closely follows that of prior work on single-server offline/online PIR [36]. More formally, the server's view in an interaction with a client consists of (1) the client's encrypted bit vectors sent in the offline phase, (2) the client's standard single-server PIR queries sent in the offline phase, (3) the messages that the client sends in the online phase. To prove security, we can construct a sequence of hybrid distributions that move from the world in which

the client queries a sequence of database indexes $I_0 = (i_{0,1}, i_{0,1}, \ldots, i_{0,Q})$ to the world in which the client queries a different sequence $I_1 = (i_{1,1}, i_{1,1}, \ldots, i_{1,Q})$. The steps of the argument are:

- replace the encrypted bit vectors with encryptions of zeros, using the semantic security of the encryption scheme,
- replace the client's standard single-server PIR query with a query to a fixed database row, using the security of the underlying single-server PIR scheme,
- swap query sequence I_0 with query sequence I_1, using the security of the underlying two-server offline/online PIR scheme,
- swap the client's standard single-server PIR query and encrypted bit vectors back again, using the security of these primitives.

□

Remark 4.3 (Single-server PIR with $\widetilde{O}_\lambda(n^{2/3})$ amortized time and communication). With an alternate rebalancing (taking Q to be $n^{1/3}$), we can build a single-server offline/online PIR scheme that, as long as the client makes at least $n^{1/3}$ adaptive queries, has amortized communication $\widetilde{O}_\lambda(n^{2/3})$, amortized server time $\widetilde{O}_\lambda(n^{2/3})$, amortized client time $\widetilde{O}_\lambda(n^{2/3})$, and client storage $\widetilde{O}_\lambda(n^{2/3})$. This PIR scheme has better amortized server time than that of Theorem 4.1, at the cost of requiring a client upload linear in n in the offline phase. (However, the *amortized* communication of this scheme is still sublinear in n.)

5 Single-server PIR with optimal amortized time and storage from fully homomorphic encryption

In this section, we construct a single-server many-query offline/online PIR scheme directly, rather than through a generic transformation. Assuming fully homomorphic encryption (Definition 2.2), our scheme achieves the optimal tradeoff between amortized server time and client storage, up to polylogarithmic factors. This fills a gap left open by the protocols of Section 4 and demonstrates that the lower bound we give in Section 6 is tight. We prove the following result:

Theorem 5.1 (Single-server PIR with optimal amortized time and storage from fully homomorphic encryption). *Assuming gate-by-gate fully homomorphic encryption (Definition 2.2), there exists a single-server offline/online PIR scheme that, on security parameter $\lambda \in \mathbb{N}$, database size $n \in \mathbb{N}$, and maximum number of queries $Q < n$, supports Q adaptive queries with:*

- *amortized server time $\tilde{O}_\lambda(n/Q)$,*
- *client-side storage $\tilde{O}_\lambda(Q)$,*
- *amortized communication $\tilde{O}_\lambda(n/Q)$, and*
- *amortized client time $\tilde{O}_\lambda(Q + n/Q)$.*

This new scheme achieves amortized server time better than we could expect from any protocol derived from the generic compiler of Section 3, given current

state-of-the-art offline/online PIR protocols. To answer each query, that compiler executes the online phase a PIR scheme on Q database chunks, each of size n/Q. Similar to the compiler of Section 3, the PIR scheme here works by splitting the database into random chunks, so that the client's distinct adaptive queries fall into distinct chunks with high probability. However, the new PIR scheme in this section keeps the mapping of database rows to chunks *secret* from the server. (In contrast, in the scheme of Section 3, the client reveals to the server the mapping of database rows to chunks.) By keeping the mapping of database rows to chunks secret, in the online phase of this scheme, the server only has to compute over the contents a single chunk. In this way, we achieve lower computation than the schemes of Section 4, which execute an online phase for each database chunk.

In the remainder of this section, we sketch the ideas behind the PIR scheme that proves Theorem 5.1; a complete proof appears in Appendix B.

Proof idea for Theorem 5.1. At a very high level, the PIR scheme that we construct works as follows:

1. In an offline phase, the client chooses small, random subsets $S_1, \ldots, S_m \subseteq [n]$. For each subset, the client privately fetches from the server the parity of the database bits indexed by the set.
2. When the client wants to fetch database record i in the online phase, it finds a subset $S \in \{S_1, \ldots, S_m\}$ such that $i \in S$. Then, the client *usually* asks the server for the parity of the database bits indexed by $S \setminus \{i\}$. The parity of the database bits indexed by S and $S \setminus \{i\}$ give the client enough information to recover the value of the ith database record, D_i. Then, the client re-randomizes the set S it just used.

In more detail, our PIR scheme operates as follows: in the offline phase, the client samples $(\lambda + 1) \cdot Q$ random subsets of $[n]$, each of size n/Q. We call the first λQ sets the "primary" sets and the remaining Q sets the "backup" sets. For each set S, the client retrieves the parity of the database bits the set indexes, i.e., $\sum_{j \in S} D_j \mod 2$, from the server, while keeping the set contents hidden using encryption. For each backup set S, the client additionally chooses a random member of the set S and privately retrieves the database value indexed by that element, via a batch PIR protocol [7, 58, 62].

With high probability over the client's random choice of sets, whenever the client wants to fetch the i-th database record, the client holds a primary set that contains i. Again with good probability, the client then asks the server for the parity of the database bits indexed by the punctured set $S \setminus \{i\}$, with which she can reconstruct the desired database value D_i. Finally, the client must refresh her state, as using the same S to query for another index i' could leak (i, i') to the server and thus break security. To achieve this, the client discards S and promotes the next available backup set, S_b, to become a new primary set. If S_b does not already contain i, the client modifies S_b by deleting the set element whose database value she knows and inserting i; the client recomputes this new set's parity using the value of D_i she just retrieved. With this mechanism, the

distribution of the client's primary sets remains random, ensuring that her online queries are independent.

There are two failure events in this scheme: it is possible that (a) none of the primary sets contain the index queried, i, or that (b) the client sends the server a set other than $S \setminus \{i\}$, as decided by a coin flip (to avoid always sending a query set that does not contain i). We drive down the probability of either failure event to $\mathrm{negl}(\lambda)$, by repeating the offline and online phases λ times. Then, by construction, this scheme satisfies correctness for Q queries. Intuitively, the scheme is secure because (a) the use of encryption and batch PIR in the offline phase prevents the server from learning the contents of the presampled sets, and (b) the client's online queries are indistinguishable from uniformly random subsets of $[n]$ of size $n/Q - 1$, as proved in Appendix B.3. □

We now discuss the PIR scheme's efficiency.

Communication and storage. The client can succinctly represent her presampled sets with only logarithmic-size keys by leveraging pseudorandomness. Then, in the offline phase, she exchanges only $\widetilde{O}_\lambda(Q)$ bits with the server to communicate the descriptions and parities of $O_\lambda(Q)$ randomly sampled sets. The client additionally retrieves the database values of Q indices—one from each backup set—in $\widetilde{O}_\lambda(Q)$ communication with batch PIR. The client stores her presampled sets and her state between queries in $\widetilde{O}_\lambda(Q)$ bits. In each online phase, the client must however hide whether she inserted an index into her query set (and, if so, which index she inserted). Therefore, the client explicitly lists all elements in the punctured set she is querying for (instead of using pseudoradnomness) and thus exchanges $\widetilde{O}_\lambda(n/Q)$ bits with the server.

Computation. In the offline phase, the client must retrieve the encrypted parities of the database bits indexed by each of $O_\lambda(Q)$ encrypted sets of size n/Q. In Lemma B.2, we present a Boolean circuit that computes the parities of the database bits of s subsets of $[n]$, each of size ℓ, in $\widetilde{O}(s \cdot \ell + n)$ gates. Our circuit is inspired by circuits for private set intersection [61, 79, 80] and makes use of sorting networks [10]. The server can execute the offline phase in $\widetilde{O}_\lambda(n)$ time by running the above circuit under a gate-by-fate fully homomorphic encryption scheme. Further, the offline server can respond to the client's batch PIR query in $\widetilde{O}_\lambda(n)$ time. In each online phase, the server must complete $O_\lambda(n/Q)$ work per query, as it computes the parity of a punctured set containing $n/Q - 1$ elements. Thus, each query requires $\widetilde{O}_\lambda(n/Q)$ amortized total server time.

As for the client, in the offline phase, she generates $\mathcal{O}_\lambda(Q)$ random sets. Using pseudorandomness to represent each set, the time to generate these sets without expanding them is $\widetilde{O}_\lambda(Q)$. Also in the offline phase, the client runs a batch PIR protocol with the server to recover Q database values, requiring at most $\widetilde{O}_\lambda(Q)$ client time. In the online phase, the client first has to find a primary set that contains the index $i \in [n]$ she wants to read. By generating each set using a pseudorandom permutation, she can efficiently test whether each set contains i by inverting the permutation in time $\widetilde{O}_\lambda(1)$. Testing all $\mathcal{O}_\lambda(Q)$ primary sets takes the client time $\widetilde{O}_\lambda(Q)$. When she finds a succinctly-represented primary

set that contains i, the client expands the set in time $\widetilde{O}_\lambda(n/Q)$ to build her online query. Finally, promoting a backup set to become a new primary set and, if necessary, replacing a set element by i takes time $\widetilde{O}_\lambda(1)$. We conclude that the client's amortized, per-query time is $\widetilde{O}_\lambda(Q + n/Q)$.

Remark 5.2 (Two-server offline/online PIR with reduced server time amortized over many adaptive queries). The PIR scheme of Theorem 5.1 immediately gives a two-server, many-query PIR scheme with $\widetilde{O}_\lambda(n/Q)$ amortized server time from one-way functions. When the client makes many queries (i.e., $Q \gg \sqrt{n}$), this result improves upon the $\widetilde{O}_\lambda(n/Q + \sqrt{n})$ time achieved by prior work [36]. The construction that proves the remark works as follows: rather than sending her offline hint request encrypted to the single server, the client sends her offline hint request in plaintext to one server and sends her online query to a second server that does not collude with the first.

6 Lower bound

In this section, we present a lower bound for multi-query offline/online PIR schemes in which the server stores the database in its original form—that is, the server does not preprocess or encode the database. (If preprocessing is allowed, candidate single-server PIR schemes using program obfuscation can circumvent our lower bound [25].) Our result is a lower bound on the product of the (a) client storage and (b) online time of any offline/online PIR scheme for many adaptive queries. Specifically, we show that in any adaptive multi-query offline/online PIR scheme, where the client stores S bits between queries and the server responds to each query in amortized time T, it must hold that $ST = \widetilde{\Omega}(n)$. This new lower bound matches the best adaptive multi-query scheme in the two-server setting [36, Section 4] and it matches our new scheme (Section 5) in the single-server setting, up to polylogarithmic factors.

We thus rule out PIR schemes with small client storage and small amortized server online time in the adaptive setting.

Remark 6.1 (Generalization to multi-server PIR). While we present and prove this lower bound in the single-server setting, it also holds for protocols with any constant number of servers. With multiple servers, T bounds the database bits probed per query by any online server.

Theorem 6.2 (Lower bound for adaptive queries). *Consider a computationally secure single-server offline/online PIR scheme for many adaptive queries, such that, on security parameter $\lambda \in \mathbb{N}$ and database size $n \in \mathbb{N}$,*

- *the server stores the database in its original form,*
- *the client stores at most S bits between consecutive queries, and*
- *the server probes T database bits per query on average,*

Then, for polynomially bounded $n = n(\lambda)$, holds that $(S + 1) \cdot (T + 1) \geq \widetilde{\Omega}(n)$.

To prove Theorem 6.2, we invoke the following lower bound from prior work on the offline communication and online server time of *single-query* PIR schemes. With a reduction, we then relate the offline communication of a *single-query* scheme to the client storage of a *many-query* scheme, giving the desired bound.

Theorem 6.3 ([36, Section 6]). *Consider a computationally secure single-query offline/online PIR scheme such that, on security parameter $\lambda \in \mathbb{N}$ and database size $n \in \mathbb{N}$,*

- *the server stores the database in its original form,*
- *the client downloads C bits in the offline phase, and*
- *the server probes T bits of the database while processing each online query.*

Then, for polynomially bounded $n = n(\lambda)$, it holds that $(C+1) \cdot (T+1) \geq \widetilde{\Omega}(n)$.

We now give a proof of Theorem 6.2.

Proof of Theorem 6.2. Let Π be a computationally secure single-server offline/online PIR scheme for Q adaptive queries, as in the theorem statement. For an integer $Q' \in \{0, \ldots, Q-1\}$ and a sequence of indices $(i_1, \ldots, i_{Q'}, i) \in [n]^{Q'+1}$, denote by $T(i_1, i_2, ..., i_{Q'}; i)$ the number of database bits that the server probes when processing the client's query on input i, after having previously processed the client's queries on inputs $i_1, \ldots, i_{Q'}$.

Claim 6.4. *There exists an integer $Q' \in \{0, \ldots, Q-1\}$ and a sequence of indices $(i_1, \ldots, i_{Q'}) \in [n]^{Q'}$ such that, for every index $i \in [n]$, it holds that $T(i_1, \ldots, i_{Q'}; i) \leq T$.*

Proof. Consider the following procedure.

- For $Q' \coloneqq 1, \ldots, Q$ do:
  - Set $\mathsf{foundBad} \leftarrow \mathsf{false}$.
  - For $i \coloneqq 1, \ldots, n$:
    - If $T(i_1, \ldots, i_{Q'-1}; i) > T$, set $i_{Q'} \leftarrow i$, set $\mathsf{foundBad} \leftarrow \mathsf{true}$, and break of the inner loop.
  - If $\mathsf{foundBad} = \mathsf{false}$, output ok and $i_1, \ldots, i_{Q'-1}$ and halt.
- Output fail and $i_1, \ldots, i_Q$.

When the above procedure does not fail, then by construction it outputs a sequence $i_1, \ldots, i_{Q'}$ such that for every $i \in [n]$, it holds that $T(i_1, \ldots, i_{Q'-1}; i) \leq T$. We only need to show that the procedure never fails. Suppose for the sake of contradiction that the procedure fails and outputs $i_1, \ldots, i_Q$. Then, when the client reads the sequence $i_1, \ldots, i_Q$, the server probes more than T database bits when processing each query, which contradicts our assumption that the (worst-case) amortized number of bits that the server probes is at most T. □

Returning to the proof of Theorem 6.2, we now build the following single-query two-server offline/online PIR scheme Π'.

Offline phase. The first server proceeds as follows:

- Run the offline phase of scheme Π, playing the part of both the client and the offline server of Π, to generate a client key ck and a hint h.
- Compute indices $i_1, \ldots, i_{Q'}$ as in Claim 6.4.
- Run the online phase of scheme Π on indices $i_1, \ldots, i_{Q'}$, playing the part of the client and the online server of Π. When playing the part of the client, use the above ck and h as the initial state of the client.
- Send the updated client key ck and hint h to the client.

To complete the offline phase, the client stores the client key ck and hint h that the first server sends to it.

Online phase. To read the database bit at index i, the client runs the online phase of Π with the online server, using its local client key ck and hint h.

The resulting scheme Π' is a secure single-query offline/online PIR scheme. Correctness holds by construction, from the correctness of Π. Security follows from the security of Π, since the online server's view in Π', when the client is reading index i, is contained in the server's view in Π, when the client is reading indices $i_1, \ldots, i_{Q'}, i$.

Finally, the offline communication C in Π' is equal to the size S of the client storage between consecutive queries in Π. Moreover, by the choice of $(i_1, \ldots, i_{Q'})$, the number of database bits that the online server probes in Π' is at most T, the amortized number of bits probed by each server in scheme Π.

Therefore, by Theorem 6.3, we conclude that $(S+1)(T+1) \geq \widetilde{\Omega}(n)$. □

7 Conclusion

We construct new single-server PIR schemes that have sublinear amortized total server time. A number of related problems remain open:

- Is it possible to match the performance of our PIR scheme based on fully homomoprhic encryption (Section 5) while using simpler assumptions?
- Can we construct single-server PIR schemes for many adaptive queries that achieve optimal $\widetilde{O}_\lambda(1)$ communication, $\widetilde{O}_\lambda(n^{1/2})$ amortized server time, and $\widetilde{O}_\lambda(n^{1/2})$ client storage? Our scheme from Section 5 has larger communication $\widetilde{O}_\lambda(n^{1/2})$. One approach would be to design puncturable pseudorandom sets [36,83] with short descriptions that support both insertions and deletions.
- Our lower bound in Section 6 only applies to PIR schemes in which the server stores the database in unencoded form. Can we beat this bound by having the server store the database in some encoded form [14]?

Acknowledgements. We thank David Wu and Yuval Ishai for reading an early draft of this work and for their helpful suggestions on how to improve it. We thank Yevgeniy Dodis, Siyao Guo, and Sandro Coretti for answering questions about presampling. We deeply appreciate the support and technical advice that Dan Boneh gave on this project from the very start. This work was supported in part by the National Science Foundation (Award CNS-2054869), a gift from

Google, a Facebook Research Award, and the Fintech@CSAIL Initiative, as well as the National Science Foundation Graduate Research Fellowship under Grant No. 1745302 and an EECS Great Educators Fellowship.

References

[1] Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.O.: XPIR: Private information retrieval for everyone. PoPETs **2016**(2), 155–174 (2016)
[2] Aiello, W., Bhatt, S., Ostrovsky, R., Rajagopalan, S.R.: Fast verification of any remote procedure call: Short witness-indistinguishable one-round proofs for NP. In: ICALP (2000)
[3] Ajtai, M., Komlós, J., Szemerédi, E.: An O(N log N) sorting network. In: STOC (1983)
[4] Ali, A., Lepoint, T., Patel, S., Raykova, M., Schoppmann, P., Seth, K., Yeo, K.: Communication–computation trade-offs in PIR. In: USENIX Security (2021)
[5] Ambainis, A.: Upper bound on communication complexity of private information retrieval. In: ICALP (1997)
[6] Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: IEEE Security and Privacy (2018)
[7] Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: SOSP (2016)
[8] Backes, M., Kate, A., Maffei, M., Pecina, K.: ObliviAd: provably secure and practical online behavioral advertising. In: IEEE Security and Privacy (2012)
[9] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: CRYPTO (2001)
[10] Batcher, K.E.: Sorting networks and their applications. In: AFIPS (1968)
[11] Beimel, A., Ishai, Y.: Information-theoretic private information retrieval: A unified construction. In: ICALP (2001)
[12] Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.: Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In: FOCS (2002)
[13] Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers computation in private information retrieval: PIR with preprocessing. In: CRYPTO (2000)
[14] Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers' computation in private information retrieval: PIR with preprocessing. J. Cryptol. **17**(2), 125–151 (2004)
[15] Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure single-server aggregation with (poly) logarithmic overhead. In: CCS (2020)
[16] Bell, S., Komisarczuk, P.: An analysis of phishing blacklists: Google Safe Browsing, OpenPhish, and PhishTank. In: ACSW (2020)

[17] Bentley, J.L., Saxe, J.B.: Decomposable searching problems I: static-to-dynamic transformation. J. Algorithms **1**(4), 301–358 (1980). https://doi.org/10.1016/0196-6774(80)90015-2
[18] Biehl, I., Meyer, B., Wetzel, S.: Ensuring the integrity of agent-based computations by short proofs. In: Mobile Agents (1998)
[19] Blackwell, K., Wootters, M.: A note on the permuted puzzles toy conjecture. arXiv preprint arXiv:2108.07885 (2021)
[20] Boneh, D.: The decision Diffie-Hellman problem. In: International Algorithmic Number Theory Symposium (1998)
[21] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT (2015)
[22] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS (2016)
[23] Boyle, E., Holmgren, J., Ma, F., Weiss, M.: On the security of doubly efficient PIR. Cryptology ePrint Archive, Report 2021/1113 (2021)
[24] Boyle, E., Holmgren, J., Weiss, M.: Permuted puzzles and cryptographic hardness. In: TCC (2019)
[25] Boyle, E., Ishai, Y., Pass, R., Wootters, M.: Can we access a database both locally and privately? In: TCC (2017)
[26] Boyle, E., Naor, M.: Is there an oblivious RAM lower bound? In: ITCS (2016)
[27] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: CRYPTO (2011)
[28] Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: EUROCRYPT (1999)
[29] Canetti, R., Holmgren, J., Richelson, S.: Towards doubly efficient private information retrieval. In: TCC (2017)
[30] Chang, Y.: Single database private information retrieval with logarithmic communication. In: ACISP (2004)
[31] Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: CCS (2018)
[32] Cheng, R., Scott, W., Masserova, E., Zhang, I., Goyal, V., Anderson, T.E., Krishnamurthy, A., Parno, B.: Talek: Private group messaging with hidden access patterns. In: ACSAC (2020)
[33] Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: STOC (1997)
[34] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: FOCS (1995)
[35] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–982 (1998)
[36] Corrigan-Gibbs, H., Kogan, D.: Private information retrieval with sublinear online time. In: EUROCRYPT (2020)
[37] Dauterman, E., Feng, E., Luo, E., Popa, R.A., Stoica, I.: DORY: an encrypted search system with distributed trust. In: OSDI (2020)
[38] Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: CRYPTO (2016)

[39] Döttling, N., Garg, S., Ishai, Y., Malavolta, G., Mour, T., Ostrovsky, R.: Trapdoor hash functions and their applications. In: CRYPTO (2019)
[40] Dvir, Z., Gopi, S.: 2-server PIR with subpolynomial communication. J. ACM **63**(4), 39:1–39:15 (2016)
[41] Dwork, C., Langberg, M., Naor, M., Nissim, K., Reingold, O.: Succinct proofs for NP and Spooky interactions (2004)
[42] Dwork, C., Naor, M., Rothblum, G.N.: Spooky interaction and its discontents: Compilers for succinct two-message argument systems. In: CRYPTO (2016)
[43] Efremenko, K.: 3-query locally decodable codes of subexponential length. SIAM J. Comput. **41**(6), 1694–1703 (2012)
[44] Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009)
[45] Gentry, C., Halevi, S.: Compressible FHE with applications to PIR. In: TCC (2) (2019)
[46] Gentry, C., Ramzan, Z.: Single-database private information retrieval with constant communication rate. In: ICALP (2005)
[47] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO (2013)
[48] Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: EUROCRYPT (2014)
[49] Goldreich, O., Karloff, H., Schulman, L., Trevisan, L.: Lower bounds for linear locally decodable codes and private information retrieval. In: CCC (2002)
[50] Goldreich, O.: Foundations of Cryptography. Cambridge University Press (2001)
[51] Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. J. ACM **43**(3), 431–473 (1996)
[52] Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of computer and system sciences **28**(2), 270–299 (1984)
[53] Goodrich, M.T.: Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In: STOC (2014)
[54] Green, M., Ladd, W., Miers, I.: A protocol for privately reporting ad impressions at scale. In: CCS (2016)
[55] Groth, J., Kiayias, A., Lipmaa, H.: Multi-query computationally-private information retrieval with constant communication rate. In: PKC (2010)
[56] Gupta, T., Crooks, N., Mulhern, W., Setty, S., Alvisi, L., Walfish, M.: Scalable and private media consumption with Popcorn. In: NSDI (2016)
[57] Hamlin, A., Ostrovsky, R., Weiss, M., Wichs, D.: Private anonymous data access. Cryptology ePrint Archive, Report 2018/363 (2018)
[58] Henry, R.: Polynomial batch codes for efficient IT-PIR. PoPETs **2016**(4), 202–218 (2016)
[59] Henry, R., Huang, Y., Goldberg, I.: One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In: NDSS (2013)

[60] Henry, R., Olumofin, F.G., Goldberg, I.: Practical PIR for electronic commerce. In: CCS (2011)
[61] Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS (2012)
[62] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Batch codes and their applications. In: STOC (2004)
[63] Jacob, R., Larsen, K.G., Nielsen, J.B.: Lower bounds for oblivious data structures. In: SODA (2019)
[64] Juels, A.: Targeted advertising ... and privacy too. In: CT-RSA (2001)
[65] Kalai, Y.T., Raz, R., Rothblum, R.D.: How to delegate computations: the power of no-signaling proofs. In: STOC (2014)
[66] Kogan, D., Corrigan-Gibbs, H.: Private blocklist lookups with Checklist. In: USENIX Security (2021)
[67] Komargodski, I., Lin, W.K.: A logarithmic lower bound for oblivious RAM (for all parameters). In: CRYPTO (2021)
[68] Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: FOCS (1997)
[69] Larsen, K.G., Nielsen, J.B.: Yes, there is an oblivious RAM lower bound! In: CRYPTO (2018)
[70] Larsen, K.G., Simkin, M., Yeo, K.: Lower bounds for multi-server oblivious RAM. In: TCC (2020)
[71] Lipmaa, H.: An oblivious transfer protocol with log-squared communication. In: International Conference on Information Security (2005)
[72] Lipmaa, H.: First cpir protocol with data-dependent computation. In: ICISC (2009)
[73] Lueks, W., Goldberg, I.: Sublinear scaling for multi-client private information retrieval. In: Financial Cryptography (2015)
[74] Mockapetris, P.: Domain names - concepts and facilities. RFC 1034 (1987), `http://www.rfc-editor.org/rfc/rfc1034.txt`
[75] Mughees, M.H., Chen, H., Ren, L.: OnionPIR: Response efficient single-server PIR. Cryptology ePrint Archive, Report 2021/1081 (2021)
[76] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT (1999)
[77] Patel, S., Persiano, G., Yeo, K.: Private stateful information retrieval. In: CCS (2018)
[78] Persiano, G., Yeo, K.: Limits of preprocessing for single-server pir. In: SODA (2022). https://doi.org/10.1137/1.9781611977073.99
[79] Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: USENIX Security (2014)
[80] Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Transactions on Privacy and Security **21**(2) (2018)
[81] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 1–40 (2009)
[82] Servan-Schreiber, S., Hogan, K., Devadas, S.: AdVeil: A private targeted-advertising ecosystem (2021)

[83] Shi, E., Aqeel, W., Chandrasekaran, B., Maggs, B.: Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In: CRYPTO (2021)
[84] Stark, E.M.: Splitting up trust. `https://emilymstark.com/2021/09/14/splitting-up-trust.html` (September 14, 2021)
[85] Tauman Kalai, Y., Raz, R., Rothblum, R.D.: Delegation for bounded space. In: STOC (2013)
[86] Wehner, S., de Wolf, R.: Improved lower bounds for locally decodable codes and private information retrieval. In: ICALP (2005)
[87] Woodruff, D., Yekhanin, S.: A geometric approach to information-theoretic private information retrieval. In: CCC (2005)
[88] Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. J. ACM **55**(1), 1:1–1:16 (2008)

A Deferred material from Section 3

Proof of Lemma 3.1. Construction 3.5 gives the PIR scheme that proves the lemma. The bounds on the communication cost and running time follow immediately from the construction.

To complete the proof, we need only show that Construction 3.5 satisfies correctness and security for Q queries.

Correctness. Fix $\lambda, n, Q \in \mathbb{N}$, database $D \in \{0,1\}^n$, and input sequence $(i_1, \ldots, i_Q) \in [n]^Q$. When sequentially reading database records $i_1, \ldots, i_Q$ using scheme Π as in Experiment 2.1, the experiment outputs "0" (i.e., correctness fails), when reading one of the input indices i_{bad}, it holds that $i_{\mathsf{bad}} \notin \mathsf{queried}$, and one of the following two failure events happen:

F1. The client, when running Step 3 of the Query algorithm, fails to find an index $\ell^* \in [\lambda]$ such that $(\mathsf{ck})_{j^*\ell^*} \neq \perp$.

F2. The client, when running Step 2 of the $\mathsf{Reconstruct}$ algorithm, obtains an incorrect value from the $\mathsf{Reconstruct}$ function of the single-query scheme Π.

The correctness of the underlying single-query scheme Π implies that failure event F2 happens with probability negligible in the security parameter.

In the remainder, we show that failure event F1 also happens with negligible probability. In particular, for $j \in [Q]$, we say that the client *uses* chunk j during an execution of the Query algorithm, if it calls $\Pi.\mathsf{Query}$ on $(\mathsf{ck})_{jl}$ for some $\ell \in [\lambda Q]$. When this happens the client also sets $(\mathsf{ck})_{jl} \leftarrow \perp$. A necessary condition for the event F1 to happen is that there exists an index $j \in [Q]$ such that the client uses chunk j the maximal number of times λ.

We use the following combinatorial claim, which bounds the probability that failure event F1 occurs:

Claim A.1. *For any sequence of Q query indexes, the probability, taken over the choice of the pseudorandom permutation key, that there exists a chunk $j \in [Q]$ that the client uses more than λ times is* $\mathrm{negl}(\lambda)$.

Proof of claim. We have that $Q < n$. We may also assume that $\lambda < Q$ since otherwise the claim is vacuously true. In addition, replace the pseudorandom permutation used in the construction with a truly random permutation. By the security of the pseudorandom permutation, this can only increase the probability of failure by a quantity negligible in the security parameter λ.

Next, we use a standard balls-into-bins argument. Fix a chunk $j \in [Q]$ and sequence of query indexes $(i_1, \ldots, i_Q) \in [n]^Q$. Recall that when querying an index $i \in \mathsf{queried}$, the client chooses a random chunk and makes a dummy query. (In this case, it later recovers the value of database bit i from the cache, rather than from the server's response.) Therefore, we may assume that the query indexes are distinct, since this can only increase the probability of an overloaded chunk. We may also assume that $\lambda < n/Q$, since the number of times the client uses each chunk is at most the number of distinct indices in a chunk (i.e., the size of a chunk), which is n/Q.

Consider a subset $I \subseteq \{i_1, \ldots, i_Q\}$ of size λ. Let C_I be the event that all queries in I use chunk j. Then

$$\Pr[C_I] = \left(\frac{(n/Q)}{n}\right)\left(\frac{(n/Q)-1}{n-1}\right)\cdots\left(\frac{(n/Q)-\lambda}{n-\lambda}\right),$$

where the probability is over the choice of the random permutation. Each term in this product is at most

$$\frac{n/Q}{n-\lambda} \leq \frac{n/Q}{n-n/Q} = \frac{1}{Q-1}.$$

Then $\Pr[C_I] \leq (Q-1)^{-\lambda}$. There are at most

$$\binom{Q}{\lambda} \leq \left(\frac{eQ}{\lambda}\right)^{\lambda}$$

choices of the index set I, so by the union bound, the probability that there exists a bad set is

$$\left(\frac{1}{Q-1}\cdot\frac{eQ}{\lambda}\right)^{\lambda} \leq \left(\frac{6}{\lambda}\right)^{\lambda},$$

which is negligible in λ. Taking a union bound over all Q chunks completes the proof of the claim. □

We now return to the proof of Lemma 3.1.

Security. Consider any security parameter $\lambda \in \mathbb{N}$, efficient adversary $\mathcal{A}$, and polynomially bounded $n = n(\lambda) \in \mathbb{N}$ and $Q = Q(n) \in \mathbb{N}$. We design a sequence of $Q+1$ hybrid games, named Game 0 up to Game Q:

Game t, for $t \in \{0, \ldots, Q\}$. Parameterized by an adversary $\mathcal{A}$, PIR scheme Π, security parameter $\lambda \in \mathbb{N}$, number of queries $Q \in \mathbb{N}$, and database size $n \in \mathbb{N}$.

1. Compute:
   $(\mathsf{ck}, q) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, n)$
   $\mathsf{st} \leftarrow \mathcal{A}(1^\lambda)$
2. For $l = 1, \ldots, t$, compute:
   $(\mathsf{st}, i_0, i_1) \leftarrow \mathcal{A}(\mathsf{st})$
   $(\mathsf{ck}, \_, q) \leftarrow \Pi.\mathsf{Query}(\mathsf{ck}, i_0)$
   $\mathsf{st} \leftarrow \mathcal{A}(\mathsf{st}, q)$
3. For $l = t+1, \ldots, Q$, compute:
   $(\mathsf{st}, i_0, i_1) \leftarrow \mathcal{A}(\mathsf{st})$
   $(\mathsf{ck}, \_, q) \leftarrow \Pi.\mathsf{Query}(\mathsf{ck}, i_1)$
   $\mathsf{st} \leftarrow \mathcal{A}(\mathsf{st}, q)$
4. Output $b \leftarrow \mathcal{A}(\mathsf{st})$

Game 0 corresponds to Experiment 2.2 with $b = 1$, while Game Q corresponds to Experiment 2.2 with $b = 0$. For $0 \leq t \leq Q$, let $\mathcal{G}_{\mathcal{A},\lambda,Q,n,t}$ be the event that Game t outputs "1" when parametrized by these values, and let View_t denote the adversary's view in Game t. The construction is secure if it holds that

$$|\Pr[\mathcal{G}_{\mathcal{A},\lambda,Q,n,0}] - \Pr[\mathcal{G}_{\mathcal{A},\lambda,Q,n,Q}]| \leq \mathrm{negl}(\lambda).$$

Equivalently, we prove that the adversary $\mathcal{A}$ has a negligibly small advantage in distinguishing adjacent games. As the total number of games is polynomially bounded, this completes the argument.

Consider any two adjacent games, t and $t+1$ (for $0 \leq t < Q$). We show that $\mathcal{A}$'s advantage in distinguishing Games t and $t+1$ is negligible, as $\mathcal{A}$'s views in both games are computationally indistinguishable. View_t consists of

1. the security parameter λ, and
2. Q online queries, $q_1, \cdots, q_Q$, of which the first t are to an index i_0, chosen by $\mathcal{A}$, and the remaining $(Q - t)$ are to an index i_1, chosen by $\mathcal{A}$.

By a hybrid argument:

- We begin with View_t.
- We replace query q_Q in View_t by the corresponding query in View_{t+1}.
  In the offline phase, the client constructs her keys for each database chunk following the same key generation procedure. Therefore, all client keys she holds are distributed identically—regardless of which database chunk they map to. By the security of the underlying single-query scheme Π, using any client key, the output of $\Pi.\mathsf{Query}$ on any index is computationally indistinguishable from its output on any other index in $[n/Q]$. Thus, query q_Q is computationally indistinguishable from a query to any other index in $[n]$.
  As Construction 3.5 never re-uses the same client key for more than one online query, and the client keys are generated independently, each of the online queries $q_1, \cdots, q_Q$ is distributed independently.
  We conclude that this new distribution is computationally indistinguishable from View_t.
- We repeat the above step for each query from q_{Q-1} until query q_{t+1}, one by one. By the same argument, each pair of consecutive distributions is computationally indistinguishable.
- The resulting distribution is exactly View_{t+1}, as required.

This completes the proof of Lemma 3.1. □

B Deferred material from Section 5

In this section, we present a PIR scheme that proves Theorem 5.1. We first give the required definitions of batch PIR (Appendix B.1). Then, we show that it is possible to privately retrieve the parities of many subsets of a database in quasi-linear time, assuming fully homomorphic encryption (Appendix B.2). We build upon this result to construct the first single-server, many-query, adaptive PIR scheme with optimal work and storage (Appendix B.3).

Experiment B.1 (Correctness). Parameterized by a PIR scheme $\Pi = (\mathsf{Batch.Query}, \mathsf{Batch.Answer}, \mathsf{Batch.Reconstruct})$, database size $n \in \mathbb{N}$, batch size $Q \in [n]$, database $D \in \{0,1\}^n$, and query sequence $(i_1, \ldots, i_Q) \in [n]^Q$.

– Compute:

$$\begin{aligned}(\mathsf{st}, q) &\leftarrow \Pi.\mathsf{Batch.Query}(1^\lambda, n, i_1, \ldots, i_Q)\\ a &\leftarrow \Pi.\mathsf{Batch.Answer}(D, q)\\ (v_{i_1}, \ldots, v_{1_Q}) &\leftarrow \Pi.\mathsf{Batch.Reconstruct}(\mathsf{st}, a)\end{aligned}$$

– Output "1" if $v_t = D_{i_t}$ for all $t \in [Q]$. Output "0" otherwise.

Experiment B.2 (Security). Parameterized by an adversary $\mathcal{A}$, a security parameter $\lambda \in \mathbb{N}$, PIR scheme $\Pi = (\mathsf{Batch.Query}, \mathsf{Batch.Answer}, \mathsf{Batch.Reconstruct})$, database size $n \in \mathbb{N}$, batch size $Q \in [n]$, and bit $b \in \{0,1\}$.

– Compute:

$$\begin{aligned}(\mathsf{st}, i_{0,1}, \ldots, i_{0,Q}, i_{1,1}, \cdots, i_{1,Q}) &\leftarrow \mathcal{A}(1^\lambda)\\ (\_, q) &\leftarrow \Pi.\mathsf{Batch.Query}(1^\lambda, n, i_{b,1}, \ldots, i_{b,Q})\\ \mathsf{st} &\leftarrow \mathcal{A}(\mathsf{st}, q)\end{aligned}$$

– Output $b' \leftarrow \mathcal{A}(\mathsf{st})$.

B.1 Standard definitions of batch PIR

Definition B.1 (Single-server batch PIR). A single-server *batch PIR* scheme, on a security parameter $\lambda \in \mathbb{N}$, a database size $n \in \mathbb{N}$, and a batch size $Q \in [n]$, is a tuple of polynomial-time algorithms:

– $\mathsf{Batch.Query}(1^\lambda, n, i_1, \cdots, i_Q) \to (\mathsf{st}, q)$, a randomized algorithm that takes as input a security parameter $\lambda \in \mathbb{N}$, a database length $n \in \mathbb{N}$ and Q indices in $[n]$, $i_1, \cdots, i_Q$, and outputs a query state st and a query q,

– $\mathsf{Batch.Answer}(D, q) \to a$, a deterministic algorithm that takes in a database $D \in \{0,1\}^n$ and a query q, and outputs an answer a, and

– $\mathsf{Batch.Reconstruct}(\mathsf{st}, a) \to (D_{i_1}, \cdots, D_{i_Q})$, a deterministic algorithm that takes as input the query state st and the server's answer a, and outputs Q database bits, $D_{i_1}, \cdots, D_{i_Q}$.

The protocol must satisfy both correctness and security for Q queries:

1. **Correctness for Q queries**: If a client and a server correctly execute the protocol, the client can recover any Q database records of its choosing. Formally, a batch PIR scheme on database size $n \in \mathbb{N}$ and batch size $Q \in [n]$ satisfies correctness if, for every $D \in \{0,1\}^n$ and every $(i_1, \cdots, i_Q) \in [n]^Q$, Experiment B.1 outputs "1" with probability $1 - \mathrm{negl}(\lambda)$.
2. **Security for Q queries:** An adversarial server "learns nothing" about which sequence of database indices the client is fetching, even if the adversary

can choose these indices. Formally, let $W_{\mathcal{A},\lambda,n,Q,b}$ be the event that Experiment B.2 outputs "1" when parametrized by a batch PIR scheme Π, on security parameter $\lambda \in \mathbb{N}$, database size $n \in \mathbb{N}$ and batch size $Q \in [n]$, and by a bit $b \in \{0,1\}$. Protocol Π satisfies security for Q queries if, for all efficient algorithms $\mathcal{A}$,

$$|\Pr[W_{\mathcal{A},\lambda,n,Q,0}] - \Pr[W_{\mathcal{A},\lambda,n,Q,1}]| \leq \mathrm{negl}(n).$$

Using batch codes [62] on top of a state-of-the-art single-server PIR scheme [28], prior work constructs correct and secure batch PIR protocols where:

- $\mathsf{Batch.Query}$ runs in time $\widetilde{O}_\lambda(Q)$ and produces a query q of length $\widetilde{O}_\lambda(Q)$ bits,
- $\mathsf{Batch.Answer}$ runs in time $\widetilde{O}_\lambda(n)$ and produces an answer a of length $\widetilde{O}_\lambda(Q)$ bits, and
- $\mathsf{Batch.Reconstruct}$ runs time time $\widetilde{O}_\lambda(Q)$.

B.2 A new scheme for batch parity retrieval

In this section, we present a key building block for the PIR scheme that proves Theorem 5.1. We construct a family of Boolean circuits for the *batch parity retrieval* problem that, parametrized by an n-bit database,

1. take as input a batch of m length-l lists of elements in $[n]$, and
2. output the m parities of the database bits indexed by each list.

Each circuit in this family has size $\widetilde{O}(l \cdot m + n)$.

In our PIR scheme, the server holds a database and constructs the corresponding batch parity retrieval circuit. (We explain how the server picks m and l in Appendix B.3.) In the offline phase, the server evaluates this circuit under encryption, using gate-by-gate fully homomorphic encryption (Definition 2.2). By our definition of gate-by-gate fully homomorphic encryption, evaluating this circuit under encryption preserves its asymptotic runtime. This gives a solution to the problem of *privately* retrieving the parities of many subsets of a database in quasi-linear time, referred to in prior work as *batch PIR-for-parities* or *batch private sum retrieval* [36, 75, 77].

Lemma B.2 (Batch parity retrieval in quasi-linear time). *For all $n \in \mathbb{N}$, all $m \in \mathbb{N}$, all $l \in \mathbb{N}$, and any n-bit database D, there is a Boolean circuit of size $\widetilde{O}(l \cdot m + n)$ over the standard basis that:*

- *takes m lists $S_1, \ldots, S_m \in [n]^l$, each represented as l $(\log_2 n)$-bit values, and*
- *outputs the parities of the database bits indexed by the m lists:*

$$\sum_{j \in S_1} D_j \bmod 2, \ldots, \sum_{j \in S_m} D_j \bmod 2.$$

Proof. Let $n \in \mathbb{N}$ be a database size, $m \in \mathbb{N}$ be a number of lists, and $l \in \mathbb{N}$ be a list length. We first build the intermediate circuit $C_{n,l,m}$ that takes as input (1) m lists, $S_1, \ldots, S_m$, each containing l elements in $[n]$, and (2) an n-bit database D, and outputs the m parities of the database bits indexed by each list. The circuit $C_{n,l,m}$ operates as follows:

1. *Join the input lists and the input database.* The circuit builds $(l \cdot m + n)$ tuples in $[n] \times \{0, \ldots, m\} \times \{0, 1\}$, each consisting of (a) a database index, (b) a list index, and (c) a database value.
   - The first $l \cdot m$ tuples are of the form $(i, j, 0)$, where $j \in [m]$ and $i \in S_j$. The circuit can produce each such tuple using a polylog(n, m)-size gadget that takes the $\log_2 n$ input bits that correspond to i and has j and 0 hardcoded.
   - The other n tuples are of the form $(i, 0, D_i)$ where $i \in [n]$. The circuit can produce each such tuple using a polylog(n)-size gadget that takes the input bit that corresponds to D_i and has i and 0 hardcoded.

   As this step requires $(l \cdot m + n)$ gadgets of size polylog(n, m) gates each, it requires $\widetilde{O}(l \cdot m + n)$ gates.
2. *Sort by database index.* With a sorting network, the circuit sorts the tuples first by database index, and secondarily by list index. This sorting network operates on $(l \cdot m + n)$ elements of size $(\log n + \log(m + 1) + 1)$ each; therefore, it requires $\widetilde{O}(l \cdot m + n)$ gates [3, 53].
3. *Propagate the database values for each database index.* The circuit now computes the correct database value for each tuple. To achieve this, the circuit compares every pair of consecutive tuples, from left to right. If two consecutive tuples have the same database index, the circuit propagates the first tuple's database value to the second tuple. Concretely, the circuit takes $(l \cdot m + n)$ input tuples $(i_1, j_1, v_1), \ldots, (i_{l \cdot m+n}, j_{l \cdot m+n}, v_{l \cdot m+n})$ and produces $(l \cdot m + n)$ output tuples $(i'_1, j'_1, v'_1), \ldots, (i'_{l \cdot m+n}, j'_{l \cdot m+n}, v'_{l \cdot m+n})$ using a sequence of the following gadgets.

   The first gadget sets (i'_1, j'_1, v'_1) to be (i_1, j_1, v_1). Then for $t \coloneqq 2, \ldots, l \cdot m + n$, the tth gadget takes as input the input tuple (i_t, b_t, v_t) and the output of the previous gadget $(i'_{t-1}, j'_{t-1}, v'_{t-1})$ and compares i_t and i'_{t-1}. If $i'_{t-1} = i_t$ holds, then the gadget sets its output $(i'_t, j'_t, v'_t) \coloneqq (i_t, j_t, v'_{t-1})$. Otherwise it outputs $(i'_t, j'_t, v'_t) \coloneqq (i_t, j_t, v_t)$. As it operates on $(l \cdot m + n)$ tuples of $(\log n + \log(m + 1) + 1)$ bits each, this step requires $\widetilde{O}(l \cdot m + n)$ gates.
4. *Sort by input list.* With a sorting network, the circuit sorts the tuples by list index. This sorting network again requires $\widetilde{O}(l \cdot m + n)$ gates.
5. *Sum the database values in each input list.* The circuit ignores the first n tuples. (These tuples were constructed from the input database, rather than from the input sets.) For each group of l consecutive tuples, the circuit sums their database values. The circuit outputs the m resulting sums.
   As it sums a total of $(l \cdot m)$ one-bit values, this step requires $\widetilde{O}(l \cdot m)$ gates.

By construction, $C_{n,l,m}$ correctly outputs the m parities of the database bits indexed by each of its m input lists, with respect to its input database. By summing the number of gates in each step, we conclude that circuit $C_{n,l,m}$ has size $\widetilde{O}(l \cdot m + n)$.

Consider any database $D \in \{0, 1\}^n$. We now build the circuit $C_{D,l,m}$ that is identical to $C_{n,l,m}$, except it hardcodes D's database values into the circuit (instead of taking them as inputs). As required, $C_{D,l,m}$ takes as input m lists in

$[n]^l$ and retrieves the parity of the database bits indexed by each list, relative to D. As $C_{D,l,m}$ is no larger than $C_{n,l,m}$, $C_{D,l,m}$ contains at most $\widetilde{O}(l \cdot m + n)$ gates, completing the proof. □

B.3 Proof of Theorem 5.1

Consider any security parameter $\lambda \in \mathbb{N}$, polynomially bounded database size $n = n(\lambda) \in \mathbb{N}$, and maximum number of online queries $Q = Q(n)$, where $Q < n$. Using our circuits for batch parity retrieval (Lemma B.2) as a building block, we construct a PIR scheme $\Pi = (\mathsf{HintQuery}, \mathsf{HintAnswer}, \mathsf{HintReconstruct}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ that proves Theorem 5.1. We present a formal specification of Π in Construction B.3 and we analyze its correctness, security, and efficiency.

Constructing pseudorandom sets that support a random deletion and a single insertion. Our construction makes use of sets of size n/Q, which the client randomly samples in the offline phase and then optionally modifies in the online phase, by deleting a random element and inserting a chosen element. Inspired by prior work on puncturable pseudorandom sets [36, 83], our scheme minimizes communication by succinctly representing these sets using pseudorandomness. Given a pseudorandom permutation $\mathsf{PRP} : \mathcal{K}_\lambda \times [n] \to [n]$, we represent each set by (1) a PRP key $k \in \mathcal{K}_\lambda$, and (2) a point $p \in [n]$. We define the (unordered) set contents to be $\{\mathsf{PRP}(k, 1), \mathsf{PRP}(k, 2), \cdots, \mathsf{PRP}(k, n/Q - 1), p\}$. As long as $\mathsf{PRP}^{-1}(k, p) \geq n/Q$, this set has size n/Q; as long as p is chosen randomly to satisfy this condition, this set is pseudorandom. Then, we can remove an element from the set by setting $p \leftarrow \perp$. After removing this element, it is possible to insert any element $i \in [n]$ into the set by setting $p \leftarrow i$.

In the offline phase, the client uses the succinct representation of the set as a pair (k, p) for PRP key $k \in \mathcal{K}_\lambda$ and point $p \in [n]$. In the online phase, the client must hide which points have been added to or removed from the set, so the client represents the set by explicitly listing its elements.

We begin by showing that sets sampled in this way using a PRP and a random point are indeed pseudorandom (Fact B.4). Therefore, the probability that any index $i \in [n]$ is not present in at least one of λQ such sets, each generated independently, is negligible in λ (Fact B.5).

Fact B.4. *For any security parameter $\lambda \in \mathbb{N}$, universe size $n = n(\lambda) \in \mathbb{N}$, set size $s = s(n) \leq n$, and pseudorandom permutation $\mathsf{PRP} : \mathcal{K}_\lambda \times [n] \to [n]$, let S be the set of size s constructed as*

$$S \leftarrow \{\mathsf{PRP}(k, 1), \cdots, \mathsf{PRP}(k, s)\} \text{ for } k \xleftarrow{R} \mathcal{K}_\lambda.$$

If PRP is computationally secure, then, for any $i \in [n]$,

$$s/n - \mathrm{negl}(\lambda) \leq \Pr[i \in S] \leq s/n + \mathrm{negl}(\lambda).$$

Proof. For any $i \in [n]$, we define $\epsilon_i = \Pr[i \in S]$. We build an efficient algorithm $\mathcal{A}_i$ that distinguishes between

$$\mathcal{D}_{\lambda,s,0} := \{S : k \xleftarrow{R} \mathcal{K}_\lambda, S \leftarrow \{\mathsf{PRP}(k, 1), \cdots, \mathsf{PRP}(k, s)\}\}$$

Construction B.3 (Single-server offline/online PIR with $\tilde{O}(n/Q)$ amortized time from fully homomorphic encryption). The scheme is parameterized by a security parameter $\lambda \in \mathbb{N}$, database size $n \in \mathbb{N}$, and maximum number of online queries $Q = Q(n)$ and uses (1) a pseudorandom permutation $\mathsf{PRP} : \mathcal{K}_\lambda \times [n] \to [n]$, (2) a gate-by-gate fully homomorphic encryption scheme ($\mathsf{FHE.Gen}$, $\mathsf{FHE.Enc}$, $\mathsf{FHE.Dec}$, $\mathsf{FHE.Eval}$), (3) a single-server batch PIR scheme ($\mathsf{Batch.Query}, \mathsf{Batch.Answer}, \mathsf{Batch.Reconstruct}$) with database size n and batch size Q, and (4) the circuit $C_{D,(\lambda+1)\cdot Q, n/Q}$ for the batch parity retrieval of $(\lambda+1)\cdot Q$ subsets of $[n]$, each of size n/Q, with respect to D (constructed in the proof of Lemma B.2). We define $m = (\lambda+1)\cdot Q$. The final scheme runs λ instances of each phase in parallel.

I. Offline phase.

$\mathsf{HintQuery}(\mathsf{ck}, n) \to (\mathsf{ck}, q)$.

- Sample $sk \leftarrow \mathsf{FHE.Gen}(1^\lambda)$.
- *// The PRP keys determine m pseudorandom sets of database indexes: λQ primary sets, followed by Q backup sets.*
  For $j \in [m]$, let $k_j \xleftarrow{\text{R}} \mathcal{K}_\lambda$, $\hat{k}_j \leftarrow \mathsf{FHE.Enc}(sk, k_j)$, and $l_j \leftarrow \mathsf{PRP}(k_j, n/Q)$.
- *// Fetch the value of one database element indexed by each backup set.*
  Compute $q_b \leftarrow \mathsf{Batch.Query}(1^\lambda, n, l_{\lambda Q+1}, \cdots, l_m)$.
- Let $\mathsf{ck} \leftarrow (sk, (k_1, l_1), \cdots, (k_m, l_m))$ and $q \leftarrow (\hat{k}_1, \cdots, \hat{k}_m, q_b)$.

$\mathsf{HintAnswer}(D, q) \to a$.

- Parse $(\hat{k}_1, \cdots, \hat{k}_m, q_b) \leftarrow q$.
- *// Generate m pseudorandom sets $\hat{S}_j \subseteq [n]$ under encryption.*
  For $j \in [m]$, let $\hat{S}_j \leftarrow \bigcup_{i \in [n/Q]} \mathsf{FHE.Eval}(\mathsf{PRP}(\cdot, i), \hat{k}_j)$.
- *// Compute (under encryption) the parity of the database bits these sets index.*
  Compute $(\hat{p}_1, \cdots, \hat{p}_m) \leftarrow \mathsf{FHE.Eval}\left(C_{D,m,n/Q}(\cdot), \hat{S}_1, \cdots, \hat{S}_m\right)$.
- *// Return the value of one database element indexed by each backup set.*
  Compute $a_b \leftarrow \mathsf{Batch.Answer}(D, q_b)$.
- Let $a \leftarrow (\hat{p}_1, \cdots, \hat{p}_m, a_b)$.

$\mathsf{HintReconstruct}(\mathsf{ck}, a) \to h$.

- Parse $(sk, (k_1, l_1), \cdots, (k_m, l_m)) \leftarrow \mathsf{ck}$. Parse $(\hat{p}_1, \cdots, \hat{p}_m, a_b) \leftarrow a$.
- *// Recover the m parities of database bits indexed by the pseudorandom sets.*
  For $j \in [m]$, let $p_j \leftarrow \mathsf{FHE.Dec}(sk, \hat{p}_j)$.
- *// Recover one database element in each backup set.*
  Let $(b_1, \cdots, b_Q) \leftarrow \mathsf{Batch.Reconstruct}(a_b)$.
- Let $h \leftarrow (p_1, \cdots, p_m, b_1, \cdots, b_Q)$.

Continued on Page 38...

... continued from Page 37.

II. Online phase.

$\mathsf{Query}(\mathsf{ck}, i) \to (\mathsf{ck}', \mathsf{st}, q)$.

- Parse $(sk, (k_1, l_1), \cdots, (k_m, l_m)) \leftarrow \mathsf{ck}$.
- *// Toss a coin to see if i should be in the query set.*
  Sample $r \xleftarrow{R} \mathsf{Bernoulli}\left(\frac{1}{Q} - \frac{1}{n}\right)$.
- If $r = 0$: *// Build a query set that looks random and does not contain i.*
  - *// Find a primary set that contains i.*
    If $\exists\ j \in [\lambda Q]$ such that $l_j = i$ or $\mathsf{PRP}^{-1}(k_j, i) < n/Q$:
    - *// Generate the contents of this primary set, and remove i.*
      Initialize $q \leftarrow \left(\bigcup_{i \in [n/Q-1]} \mathsf{PRP}(k_j, i)\right) \cup l_j \setminus \{i\}$.
    - *// Promote the next available backup set to be a new primary set.*
      Find the smallest $j' > \lambda Q$ such that $k_{j'} \neq \perp$. Set $k_j \leftarrow k_{j'}$ and $k_{j'} \leftarrow \perp$.
      - *// If the new primary set already contains i, do not modify it.*
        If $\mathsf{PRP}^{-1}(k_j, i) < n/Q$, set $i_r \leftarrow \perp$ and $l_j \leftarrow l_{j'}$.
      - *// If the new primary set does not contain i, puncture it and insert i.*
        Else, set $i_r \leftarrow l_j$ and $l_j \leftarrow i$.
    - Set $\mathsf{st} \leftarrow (i, j, j', i_r)$.
  - Else: *// No primary set contains i.*
    - Sample $S \xleftarrow{R} \binom{[n] \setminus \{i\}}{n/Q-1}$ and set $q \leftarrow S$.
    - Set $\mathsf{st} \leftarrow (i, \perp, \perp, \perp)$.
- Else: *// $r = 1$, so build a query set that looks random and contains i.*
  - Sample $S \xleftarrow{R} \binom{[n] \setminus \{i\}}{n/Q-2}$ and set $q \leftarrow S \cup \{i\}$.
  - Set $\mathsf{st} \leftarrow (i, \perp, \perp, \perp)$.
- Let $\mathsf{ck}' \leftarrow (sk, (k_1, l_1), \cdots, (k_m, l_m))$.

$\mathsf{Answer}^D(q) \to a$.

- Parse q as a set of $n/Q - 1$ elements.
- Let $a \leftarrow \sum_{i \in q} D_i \bmod 2$.

$\mathsf{Reconstruct}(\mathsf{st}, h, a) \to (h', D_i)$.

- Parse $(i, j, j', i_r) \leftarrow \mathsf{st}$ and parse $(p_1, \cdots, p_m, b_1, \cdots, b_Q) \leftarrow h$.
- *// Reconstruct the database bit at index i.*
  If $j \neq \perp$, $D_i \leftarrow a \oplus p_j$.
  Otherwise, $D_i \xleftarrow{R} \{0, 1\}$.
- If $j \neq \perp$ and $j' \neq \perp$: *// Update the parity of the new primary set.*
  - *// If i was inserted into the set, compute its new parity based on the database values at i and at the index that was replaced.*
    If $i_r \neq \perp$, set $p_j \leftarrow p_{j'} \oplus b_{i_r - \lambda Q} \oplus D_i$.
    Else, set $p_j \leftarrow p_{j'}$.
- Let $h' \leftarrow (p_1, \cdots, p_m, b_1, \cdots, b_Q)$.

and

$$\mathcal{D}_{\lambda,s,1} := \left\{ S : S \leftarrow \binom{[n]}{s} \right\}.$$

On input $S \in \binom{[n]}{s}$, $\mathcal{A}_i$ outputs 1 iff $i \in S$. Then,

$$\mathsf{DistAdv}[\mathcal{A}_i, \mathcal{D}_{\lambda,s,0}, \mathcal{D}_{\lambda,s,1}] = |s/n - \epsilon_i|.$$

As PRP is computationally secure and $s \leq n$ is polynomially bounded, it must hold that $\mathsf{DistAdv}[\mathcal{A}_i, \mathcal{D}_{\lambda,n,0}, \mathcal{D}_{\lambda,n,1}] \leq \mathrm{negl}(\lambda)$. It follows that:

$$\begin{aligned} s/n - \epsilon_i \leq \mathrm{negl}(\lambda) \quad &\text{and} \quad \epsilon_i - s/n \leq \mathrm{negl}(\lambda) \\ \epsilon_i \geq s/n - \mathrm{negl}(\lambda) \quad &\text{and} \quad \epsilon_i \leq s/n + \mathrm{negl}(\lambda) \end{aligned}$$

□

Fact B.5. *For any security parameter $\lambda \in \mathbb{N}$, universe size $n = n(\lambda) \in \mathbb{N}$, $Q = Q(n) \in \mathbb{N}$ where $Q < n$, and computationally secure pseudorandom permutation $\mathsf{PRP} : \mathcal{K}_\lambda \times [n] \to [n]$, let $S_1, \cdots, S_{\lambda Q}$ be λQ sets, each of size n/Q, generated independently as follows:*

$$S_j \leftarrow \{\mathsf{PRP}(k_j, 1), \cdots, \mathsf{PRP}(k_j, n/Q)\} \text{ where } k_j \xleftarrow{R} \mathcal{K}_\lambda.$$

The probability that any index $i \in [n]$ does not occur in at least one of the λQ sets is negligibly small in λ.

Proof. We construct λQ sets independently as above. Then, from Fact B.4,

$$\begin{aligned} \Pr\left[i \notin \bigcup_{j \in [\lambda Q]} S_j \right] &\leq (1 - 1/Q + \mathrm{negl}(\lambda))^{\lambda Q} \\ &\leq e^{-(1/Q - \mathrm{negl}(\lambda)) \cdot \lambda Q} \\ &= e^{-\lambda + \mathrm{negl}(\lambda)} \\ &= \mathrm{negl}(\lambda). \end{aligned}$$

□

We now prove a sequence of useful claims about the PIR protocol of Construction B.3. Throughout, we use the term "primary sets" to denote the first λQ sets sampled by the client in the offline phase and the term "backup sets" to denote the latter Q sets. First, following the proof technique of Corrigan-Gibbs and Kogan [36, Lemma 45], we show that the distribution of primary sets that the client holds remains identical as she makes queries, even conditioned on her past queries (Claim B.6). Using this fact, we prove that the online queries made by the client are indistinguishable from queries to any other index (Claim B.7) and that they are independent of all prior queries (Claim B.8).

Claim B.6 (Primary set distribution). *As the client makes adaptive queries using the PIR scheme of Construction B.3, the distribution of primary sets she holds remains statistically identical and is distributed independently of prior queries. Concretely, for any security parameter $\lambda \in \mathbb{N}$, database size $n = n(\lambda) \in \mathbb{N}$, and any index $i \in [n]$,*

$$\left\{\begin{array}{l}(\mathsf{ck}, \_) \leftarrow \mathsf{HintQuery}(1^\lambda, n) \\ (\_, \_, q) \leftarrow \mathsf{Query}(\mathsf{ck}, i) \\ (\mathsf{ck}, \_) \leftarrow \mathsf{HintQuery}(1^\lambda, n) \\ \text{Output } \text{(the primary sets in } \mathsf{ck}, q)\end{array}\right\} \overset{s}{\approx} \left\{\begin{array}{l}(\mathsf{ck}, \_) \leftarrow \mathsf{HintQuery}(1^\lambda, n) \\ (\mathsf{ck}, \_, q) \leftarrow \mathsf{Query}(\mathsf{ck}, i) \\ \text{Output } \text{(the primary sets in } \mathsf{ck}, q)\end{array}\right\}$$

Proof. As query q is constructed identically in the left-hand side and the right-hand side of the above equation, we know that q is distributed identically in both cases. We must show that the distribution of primary sets held by the client is statistically identical before and after she queries for any index $i \in [n]$, even conditioned on her query q for i. By cases:

- If bit r is sampled to be 1, or if i does not appear in any of the primary sets, then the client does not modify her primary sets. The client samples her query q to be a random set of the appropriate size, containing i iff $r = 0$. The claim trivially holds.
- Otherwise, the client replaces a primary set S with a backup set, S_b. Both S and S_b are pseudorandom sets, sampled independently following the same procedure. S necessarily contains i. Further, the client ensures that S_b also contains i (by removing a random element and inserting i if it isn't already in the set). Thus, S and S_b are identically distributed and independent of all other primary sets. Further, the client derives q from only the contents of S and the index i. After replacing S with S_b, the joint distribution of all primary sets thus remains the same, and independent of q.

We conclude that the distribution of primary sets held by the client is distributed identically before and after she makes query q, even when conditioned on q.

□

Claim B.7 (Online query indistinguishability). *Consider any security parameter $\lambda \in \mathbb{N}$, database size $n = n(\lambda) \in \mathbb{N}$, maximal number of queries $Q = Q(n) < n$, and query sequence $i_1, \cdots, i_t \in [n]^t$ for any $0 \le t < Q$. Then, for any $i_{t+1}, i'_{t+1} \in [n]$,*

$$\left\{\begin{array}{ll}q: & (\mathsf{ck}_0, \_) \leftarrow \mathsf{HintQuery}(1^\lambda, n) \\ & (\mathsf{ck}_1, \_, \_) \leftarrow \mathsf{Query}(\mathsf{ck}_0, i_1) \\ & \cdots \\ & (\mathsf{ck}_t, \_, \_) \leftarrow \mathsf{Query}(\mathsf{ck}_{t-1}, i_t) \\ & (\_, \_, q) \leftarrow \mathsf{Query}(\mathsf{ck}_t, i_{t+1})\end{array}\right\} \overset{c}{\approx} \left\{\begin{array}{ll}q: & (\mathsf{ck}_0, \_) \leftarrow \mathsf{HintQuery}(1^\lambda, n) \\ & (\mathsf{ck}_1, \_, \_) \leftarrow \mathsf{Query}(\mathsf{ck}_0, i_1) \\ & \cdots \\ & (\mathsf{ck}_t, \_, \_) \leftarrow \mathsf{Query}(\mathsf{ck}_{t-1}, i_t) \\ & (\_, \_, q) \leftarrow \mathsf{Query}(\mathsf{ck}_t, i'_{t+1})\end{array}\right\}$$

Proof. By applying Claim B.6 inductively, the distribution of primary sets in ck_t is statistically identical to the distribution of primary sets in ck_0, after only the

offline phase. Therefore, we know that the left-hand side in the equation above is statistically identical to $\mathcal{D}_{\lambda,n,i_{t+1}}$, while the right-hand side is statistically identical to $\mathcal{D}_{\lambda,n,i'_{t+1}}$, where, for any $i \in [n]$, we define the following distribution:

$$\mathcal{D}_{\lambda,n,i} = \left\{ \begin{array}{l} \text{For } j \in [\lambda Q],\ k_j \xleftarrow{R} \mathcal{K}_\lambda \text{ and } l_j \leftarrow \mathsf{PRP}(k_j, n/Q) \\ \text{Sample } r \xleftarrow{R} \mathsf{Bernoulli}\left(\frac{1}{Q} - \frac{1}{n}\right) \\ \text{If } r = 0: \\ \text{ - If } \exists\ j \in [\lambda Q] \text{ such that } l_j = i \text{ or } \mathsf{PRP}^{-1}(k_j, i) < n/Q, \\ \quad \text{initialize } q \leftarrow \left(\bigcup_{i \in [n/Q-1]} \mathsf{PRP}(k_j, i)\right) \cup l_j \setminus \{i\} \\ \text{ - Else, sample } S \xleftarrow{R} \binom{[n] \setminus \{i\}}{n/Q-1} \text{ and set } q \leftarrow S \\ \text{Else:} \\ \text{ - Sample } S \xleftarrow{R} \binom{[n] \setminus \{i\}}{n/Q-2} \text{ and set } q \leftarrow S \cup \{i\} \\ \text{Output } q \end{array} \right\}.$$

We now analyze the distribution $\mathcal{D}_{\lambda,n,i}$ for any $i \in [n]$. By Fact B.4, the sets $\{(k_1, l_1), \cdots, (k_{\lambda Q}, l_{\lambda Q})\}$ sampled as in $\mathcal{D}_{\lambda,n,i}$ are pseudorandom. Then, by Fact B.5, the condition in the inner if statement (checking whether there exists a primary set that contains i) evaluates to true with probability $1 - \mathrm{negl}(\lambda)$. When this is the case, the selected jth set (k_j, l_j) is computationally indistinguishable from a random set that contains i. Thus, $\mathcal{D}_{\lambda,n,i}$ is computationally indistinguishable from the distribution $\mathcal{D}'_{\lambda,n,i}$, which we define as follows:

$$\mathcal{D}'_{\lambda,n,i} = \left\{ \begin{array}{l} \text{Sample } r \xleftarrow{R} \mathsf{Bernoulli}\left(\frac{1}{Q} - \frac{1}{n}\right) \\ \text{If } r = 0: \\ \text{ - Sample } S \xleftarrow{R} \binom{[n] \setminus \{i\}}{n/Q-1} \text{ and output } q \leftarrow S \\ \text{Else:} \\ \text{ - Sample } S \xleftarrow{R} \binom{[n] \setminus \{i\}}{n/Q-2} \text{ and output } q \leftarrow S \cup \{i\} \end{array} \right\}.$$

Since the probability that $r = 1$ above is exactly the probability that a random set of size $n/Q - 1$ contains i, we can again rewrite the distribution as follows:

$$\mathcal{D}''_{\lambda,n,i} = \left\{ \text{Output } q \xleftarrow{R} \binom{[n]}{n/Q-1} \right\}.$$

The distribution $\mathcal{D}''_{\lambda,n,i}$ is independent of i, and therefore $\mathcal{D}''_{\lambda,n,i_{t+1}} \equiv \mathcal{D}''_{\lambda,n,i'_{t+1}}$. We conclude that $\mathcal{D}_{\lambda,n,i_{t+1}} \overset{c}{\approx} \mathcal{D}_{\lambda,n,i'_{t+1}}$, as required. □

Claim B.8 (Online query independence). *For any security parameter $\lambda \in \mathbb{N}$, database size $n = n(\lambda) \in \mathbb{N}$, and number of queries $Q = Q(n) < n$, consider a client that makes online queries $q_1, \cdots, q_t$ for a sequence of indices $i_1, \cdots, i_t \in [n]^t$, where $2 \leq t \leq Q$, using the PIR scheme of Construction B.3:*

$$\begin{aligned} (\mathsf{ck}_0, \_) &\leftarrow \mathsf{HintQuery}(1^\lambda, n) \\ (\mathsf{ck}_1, \_, q_1) &\leftarrow \mathsf{Query}(\mathsf{ck}_0, i_1) \\ (\mathsf{ck}_2, \_, q_2) &\leftarrow \mathsf{Query}(\mathsf{ck}_2, i_2) \\ &\cdots \\ (\mathsf{ck}_t, \_, q_t) &\leftarrow \mathsf{Query}(\mathsf{ck}_{t-1}, i_t). \end{aligned}$$

Then, q_t is independent of $(q_1, \cdots, q_{t-1})$.

Proof. Query q_t is the output of $\mathsf{Query}(\mathsf{ck}_{t-1}, i_t)$ and, more specifically, q_t depends only on i_t and on the primary sets in ck_{t-1}. However, by applying Claim B.6 inductively, we know that the distribution of primary sets in each ck_{t-1} is statistically identical to the distribution of primary sets in ck_0 and is independent of all prior queries $q_1, \cdots, q_{t-1}$. This implies that q_t is independent of all queries that came before it. □

We now prove that the PIR scheme of Construction B.3 is correct and secure.

Claim B.9 (Correctness). *If the underlying batch PIR scheme and fully homomorphic encryption scheme are correct, then the PIR protocol in Construction B.3 satisfies correctness for Q queries.*

Proof. Consider any security parameter $\lambda \in \mathbb{N}$, database size $n = n(\lambda) \in \mathbb{N}$, and maximum number of online queries $Q = Q(n) < n$. Let $D \in \{0,1\}^n$ be any database held by the server. We show that the PIR protocol in Construction B.3 correctly recovers Q database values from D with negligible failure probability.

We execute the PIR protocol on Q queries by first running the offline phase once and then running the online phase Q times. To execute each phase, we run λ instances of the scheme in parallel. We say that this execution *fails* if, in some online phase, none of the λ instances satisfy that:

1. at least one primary set contains the index queried, and
2. the bit r that is randomly sampled from $\mathsf{Bernoulli}\left(\frac{1}{Q} - \frac{1}{n}\right)$ takes value 0.

We first demonstrate that, if the execution does not fail, then the client successfully recovers the Q database values she queried for. We prove this statement by induction over the number of online phases: for each online phase $1 \leq t \leq Q$, we show that, if the execution has not failed, then

- before the t-th online phase, in all λ instances, the client's hint holds the correct parity p_j of the database bits indexed by each primary set S_j, and
- after the t-th online phase, in some instance, the client successfully reconstructs D_{i_t} on query $i_t \in [n]$.

After the offline phase (and thus before the first online phase), in each instance, the client by construction holds the correct database parities for each of her primary sets. As no failures occur, in the first online phase, there exists some instance in which (1) the client holds a primary set S that contains i_1, and (2) the client samples r to be 0. In this instance, the client then asks the server for the parity a of the database bits indexed by $S \setminus \{i_1\}$. In the corresponding offline phase, the client already retrieved the parity p of the database bits indexed by S. Therefore, the client correctly reconstructs D_{i_1} to be $a \oplus p$.

We have shown that the required property holds for the first online phase. Next, we assume that the same property holds for the t-th online phase and show that it also holds for the $(t+1)$-th online phase. After the t-th online phase, the client may have to refresh her distribution of primary sets, by discarding

the primary set she used and promoting the t-th backup set, S_b, to become a new primary set. Crucially, the client only does this refreshing procedure if she correctly recovered D_{i_t} in the t-th online phase (as she only does so if both $r = 0$ and she found a primary set containing i_t). In the offline phase, the client already retrieved the parity of the database bits indexed by S_b. However, if S_b does not contain i_t, the client must update S_b by deleting one of its elements, i_r, and inserting i_t. Then, the client computes the parity of the bits indexed by the *updated* S_b from the following values:

- the parity of the bits originally indexed by S_b (correctly retrieved in the offline phase, by running the batch parity retrieval circuit under fully homomorphic encryption),
- D_{i_r} (correctly retrieved in the offline phase, by the batch PIR scheme), and
- D_{i_t} (correctly retrieved in the last online phase).

Thus, at the start of the $(t+1)$-th online phase, in each of the λ instances, the client holds the correct database parities for each of her primary sets. As no failures occur, the client will successfully recover $D_{i_{t+1}}$ in some instance, by the same argument as in the base case. We conclude that, after Q online phases, the client has correctly recovered all Q database values queried, if no failures occur.

Next, we examine the probability with which failure events occur.

1. By Fact B.5, after only the offline phase, the probability that any $i \in [n]$ does not appear in any of the λQ primary sets is negligible in λ. By applying Claim B.6 inductively, after each online query made by the client, the distribution of primary sets remains identical, implying that same property still holds.

   Then, by a union bound, the probability that none of the primary sets contain the index queried, in any of the λ instances, for any of the Q online phases, is also negligible in λ.
2. Each time the bit r is sampled, r takes value 1 with probability $(1/Q - 1/n)$. For any given online phase, the probability that none of the λ instances samples r to be 0 is then negligibly small in λ.

   Finally, by a union bound, the probability that, for any of the Q queries, none of the λ instances samples r to be 0 is also negligible in λ.

We conclude that the PIR scheme fails with negligible probability, implying that Π satisfies correctness for Q queries. □

Claim B.10 (Security). *If the underlying pseudorandom permutation and batch PIR scheme are computationally secure, and the underlying fully homomorphic encryption scheme is semantically secure, then the PIR protocol in Construction B.3 satisfies security for Q queries.*

Proof. Consider any efficient adversary $\mathcal{A}$, security parameter $\lambda \in \mathbb{N}$, database size $n = n(\lambda) \in \mathbb{N}$, and maximum number of online queries $Q = Q(n) < n$. We show that any instance of the PIR protocol of Construction B.3 satisfies security for Q queries.

As in the security proof of Lemma 3.1, we design a sequence of $Q+1$ hybrid games, presented in Experiment B.3. Again, game 0 corresponds to Experiment 2.2 for $b=1$, while game Q corresponds to Experiment 2.2 for $b=0$. We define $G_{\mathcal{A},\lambda,Q,n,t}$ to be the event that game t outputs "1" when parametrized by these values, and we denote the adversary $\mathcal{A}$'s view in game t by View_t. To prove security, we again show that the adversary's views in adjacent games are computationally indistinguishable. As $\mathcal{A}$ is computationally bounded, this means that $\mathcal{A}$ has at most a negligible advantage in distinguishing adjacent games. Since the number of games is polynomially bounded, we conclude that

$$|\Pr[G_{\mathcal{A},\lambda,Q,n,0}] - \Pr[G_{\mathcal{A},\lambda,Q,n,Q}]| \leq \mathrm{negl}(\lambda).$$

Experiment B.3 (Single-server security games $t = 0, \cdots, Q$). Parameterized by an adversary $\mathcal{A}$, PIR scheme Π, security parameter $\lambda \in \mathbb{N}$, number of queries $Q \in \mathbb{N}$, and database size $n \in \mathbb{N}$.

1. Compute:

$$(\mathsf{ck}, q) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, n)$$
$$\mathsf{st} \leftarrow \mathcal{A}(1^\lambda, q)$$

2. For $l = 1, \ldots, Q$, compute:

$$(\mathsf{st}, i_0, i_1) \leftarrow \mathcal{A}(\mathsf{st})$$
$$i \leftarrow \begin{cases} i_0, & \text{if } l \leq t \\ i_1, & \text{otherwise} \end{cases}$$
$$(\mathsf{ck}, \_, q) \leftarrow \Pi.\mathsf{Query}(\mathsf{ck}, i)$$
$$\mathsf{st} \leftarrow \mathcal{A}(\mathsf{st}, q)$$

3. Output $b \leftarrow \mathcal{A}(\mathsf{st})$

For the PIR scheme of Construction B.3, View_t amounts to:

- The offline hint request. This hint request consists of λQ encrypted primary sets, Q encrypted backup sets, and a batch PIR query.
- Q online queries, $(q_1, \cdots, q_Q)$. Each online query consists of a set $S \subset [n]$, where $|S| = n/Q - 1$.

In View_t, the first t online queries are to an index i_0 chosen by the adversary, while the remaining $(Q-t)$ online queries are to an index i_1 chosen by the adversary.

With a hybrid argument, we show that the adversary's views in any two consecutive games in the sequence, View_t and View_{t+1} (for $0 \leq t < Q$), are computationally indistinguishable. The hybrid argument follows these steps:

- We begin with distribution View_t.

– We replace the encrypted primary sets and the encrypted backup sets by encryptions of fixed strings, relying on the semantic security of the encryption scheme.
– We replace the batch PIR query by a batch PIR query to a set of fixed indices, relying on the computational security of the batch PIR scheme.
– We replace query q_Q by a query to a fixed index, relying on the facts that:
  • the last query q_Q is independent of all queries that came before it (Claim B.8), and
  • q_Q is computationally indistinguishable from a query to a fixed index (Claim B.7).

  Applying the same reasoning, we one-by-one replace all queries from q_{Q-1} until q_{t+2} with queries to a fixed index.
– We replace query q_{t+1} with the $(t+1)$-th query in View_{t+1}, relying again on the fact that q_{t+1} is computationally indistinguishable from a query to any index in $[n]$ (Claim B.7) and on query independence (Claim B.8).
  Applying the same reasoning, we one-by-one replace all queries from q_{t+1} until q_Q with the corresponding query from View_{t+1}.
– We replace the batch PIR query to a fixed set of indices by the batch PIR query of View_{t+1}, relying on the security of the batch PIR scheme.
– We replace the encryptions of fixed strings by the encrypted primary and backup sets in View_{t+1}, relying on the semantic security of the encryption scheme.
  Then, the resulting distribution is exactly View_{t+1}, completing the argument.

We conclude that Π satisfies security for Q queries. □

Finally we analyze the PIR scheme's efficiency. By inspection:

– The server's amortized, per-query computation is $\widetilde{O}_\lambda(n/Q)$, assuming the server runs our quasi-linear-size circuit from Lemma B.2 under gate-by-gate fully homomorphic encryption.
– The client's amortized, per-query computation is $\widetilde{O}_\lambda(Q + n/Q)$.
– The client uses $\widetilde{O}_\lambda(Q)$ bits of storage.
– The scheme's amortized, per-query communication is $\widetilde{O}_\lambda(n/Q)$ bits (and the server and the client never communicate $O(n)$ bits in a single phase).