Malicious Security Comes (almost) for Free in Private Information Retrieval

Brett Falk, Pratyush Mishra, Matan Shtepel

University of Pennsylvania

June 17, 2024

Abstract

Private Information Retrieval (PIR) enables a client to retrieve a database element from a semi-honest server while hiding the element being queried from the server. Maliciously-secure PIR (mPIR) [Colombo et al., USENIX '23] strengthens the guarantees of plain (i.e., semi-honest) PIR by ensuring that even a misbehaving server (a) cannot compromise client privacy via selective-failure attacks, and (b) must answer every query *consistently* (i.e., with respect to the same database). These additional security properties are crucial for many real-world applications.

In this work we present a generic compiler that transforms any PIR scheme into an mPIR scheme in a black-box manner, minimal overhead, and without requiring additional cryptographic assumptions. Since mPIR trivially implies PIR, our compiler establishes the equivalence of mPIR and PIR. By instantiating our compiler with existing PIR schemes, we immediately obtain mPIR schemes with $O(N^\epsilon)$ communication cost. In fact, by applying our compiler to a recent doubly-efficient PIR [Lin et al., STOC '23], we are able to construct a *doubly-efficient* mPIR scheme that requires only $\mathrm{polylog}(N)$ communication and server and client computation. In comparison, all prior work incur a $\Omega(\sqrt{N})$ cost in these metrics.

Our compiler makes use of a smooth locally-decodable codes (LDCs) that have a robust decoding procedure. We term these codes "subcode"-LDCs, because they are LDCs where the query responses are from an error-correcting code. This property is shared by Reed-Muller codes (whose query responses are Reed-Solomon codewords) and more generally lifted codes.

Applying our compiler requires us to consider decoding in the face of *non-signaling adversaries*, for reasons analogous to the need for non-signaling PCPs in the succinct-argument literature. We show how to construct such decoders for Reed–Muller codes, and more generally for smooth locally-decodable codes that have a robust decoding procedure.

Keywords: PIR, LDC, Malicious security

Contents

1 Introduction

Private information retrieval (PIR) is a cryptographic primitive that allows a client to retrieve a record from a database without revealing to the database server(s) which record was retrieved. Since the seminal works of [CGKS98; KO97], there has been tremendous progress in new constructions [CCKM00; GPS04; GR05; DG15; LMW23; HHGMV23], applications [MOTBG11; Rya14; KC21; HDCZ23] and several deployments [blyss; spiral].

While almost all work on PIR schemes has focused on the setting of *semi-honest* servers, the question of how to handle *malicious* servers was noticed even in the earliest works on PIR. Indeed, Kushilevitz and Ostrovsky [KO97] observed already in their seminal work that a malicious server could learn whether a client is querying for a specific record by returning a malformed response. The question of how to obtain *maliciously-secure* PIR (mPIR) schemes has received renewed interest recently, with several works proposing formal definitions and new constructions of mPIR [CNCWF23; WZLY23; DT24; CL24]. Below we briefly describe the problems caused by malicious servers and survey existing work on mPIR.

Why malicious security? Unlike a semi-honest PIR server, a malicious server can return incorrect results to a client, violating not only correctness guarantees, but importantly also *privacy* guarantees. Examples of such violations include:

- **Selective-failure attacks:** Consider a malicious PIR server, $\mathcal{A}$, who wishes to learn whether a client is querying for the i-th record of a database db. To test this hypothesis, $\mathcal{A}$ executes the client's query on a corrupted database db' whose i-th record has been replaced with garbage data. Even if the client can somehow detect that the server's response is malformed, its hands are tied: it cannot "complain" about the corruption, as this would leak to $\mathcal{A}$ that it indeed queried for the i-th record. In general, if the client's future actions depend (in an observable manner) on the server's response, then $\mathcal{A}$ can selectively corrupt the database and learn information about the client's query.

- **Inconsistent views:** Many applications of PIR require the ability for different clients to privately query. If the server can return different results to different clients, then the clients' views of the database can become inconsistent, possibly violating the application's security guarantees.

Recent work. As noted above, while the possibility of such attacks was observed early on [KO97], only recently have formal definitions been proposed to model malicious server behavior [CNCWF23; DT24]. These definitions state that any "maliciously-secure PIR (mPIR)" protocol must satisfy not only the standard *correctness* guarantee, but must also satisfy strong *privacy* and *coherence* guarantees. Constructions matching these definitions have been proposed by [CNCWF23; DT24; WZLY23; CL24]. Informally, privacy captures selective-failure attacks by requiring that the view of any efficient adversarial server must be efficiently simulatable *even when* the adversary sees whether the client aborted, while coherence captures inconsistent view attacks by requiring that all clients querying the same index receive the same response.[1] Unfortunately, all these constructions are ad-hoc: they apply construction-specific tweaks to particular PIR schemes to achieve malicious security. Further, they achieve at best $\Omega(\sqrt{n})$ communication, server and client computation, (and 'digest' size). This leaves open the following foundational question:

Is there a general transformation that constructs more efficient mPIR from PIR in a black-box manner?

[1] Achieving this notion in the single-server setting requires all clients to hold a (succinct) commitment to the database, and so single-server mPIR protocols necessarily require a preprocessing phase [BIM04].

1.1 Our results

In this work, we answer the foregoing question by providing a generic transformation that compiles any PIR scheme to an mPIR scheme (Theorem 1). We demonstrate the flexibility of our compiler by using it settle several questions regarding the efficiency of, and assumptions required for mPIR. We detail our results below.

(1) A generic compiler from PIR to mPIR. To obtain our transformation, we carefully combine PIR schemes with vector commitments and a special class of locally-testable and decodable codes, obtaining the following theorem:

Theorem 1 (PIR $\Rightarrow$ mPIR)**.** *Consider the following ingredients:*
– *A suitable locally-decodable code (Definition 4.12) with codeword size n and query complexity* q.
– *A PIR scheme with communication complexity $c(\cdot)$.*
– *A vector commitment scheme with proof size $p(\cdot)$.*
Then, there exists a maliciously-secure PIR (mPIR) scheme with the following efficiency properties on a database of size $O(n)$:
- *communication cost is $c(n) \cdot p(n) \cdot \mathsf{q}(n)$;*
- *server computation is the time to respond to* q *(semi-honest) PIR queries on a database of size n; and*
- *client computation is the time to verify* q *opening proofs and decode the corresponding responses using the LDC decoder.*

We highlight two key challenges we encountered in proving the foregoing theorem.
- *Privacy:* One key technical challenge that arises when proving the foregoing theorem is in demonstrating privacy. In more detail, in our construction, a client's request for the i-th "logical" database record gets mapped to multiple PIR queries that are correlated with i. Although the security of the semi-honest PIR protocol *hides* the indices being queried, it does *not* prevent the adversary from potentially introducing correlated error patterns. This problem has been noted when using PIR to construct succinct arguments [DLNNR04; KRR14], where the issue was resolved by showing that the underlying argument system was secure against *non-signaling adversaries*. We resolve this problem in a similar way, by carefully designing the underlying protocol so that it remains secure against this type of attacker (see Section 5).
- *Server efficiency:* Notice that whenever $\mathsf{q} = \omega(1)$ and the base PIR has $\Theta(k)$ online server computation (which is the usual case), a straightforward use of PIR to answer each query separately leads to $\omega(k)$ server computation. Instead, we show how to obtain the optimal server cost of $O(k)$ by relying on a novel batching technique that is compatible with our stringent privacy requirements. Surprisingly, naïvely applying batch codes [IKOS04], the standard tool for batching PIR queries, is *insecure* in the malicious setting, and we have to develop new batching techniques (see Section 6).

(2) Minimal assumptions for mPIR. Because single-server PIR schemes imply collision-resistant hashes [IKO05], instantiating our compiler with a Merkle-tree-based vector commitment [Mer89] allows us to obtain mPIR schemes that rely on the same assumptions as the underlying PIR. Because mPIR trivially implies PIR, we obtain the following corollary:

Corollary 1. *The existence of single-server PIR is equivalent to the existence of single-server mPIR.*

(3) Doubly-efficient mPIR. Recently, [LMW23] proposed a "doubly-efficient" PIR scheme that achieves $\mathrm{polylog}(n)$ communication and server computation. A natural question is whether their techniques can be used to obtain doubly-efficient *mPIR*. We answer this question affirmatively by instantiating our compiler with the Reed–Muller LDC of sub-constant rate and polylogarithmic locality and the doubly-efficient PIR of [LMW23], obtaining the following corollary:

Corollary 2 (doubly-efficient mPIR)**.** *There exists an mPIR scheme where the communication complexity, client computation and server computation are* $\mathrm{polylog}(n, \lambda)$.

We illustrate the efficiency of our constructions in Table 1, which shows that the latter require lower communication and smaller digests than all prior work.

Scheme	Communication	Computation	Digest	Assumptions	Methodology
[CNCWF23] (single server)	$O(N^{1/2})$	$O(N)$	$O(N^{1/2})$	LWE, DDH	Ad-hoc
[WZLY23] (single server, amortized)	$O(N^{1/2})$	$O(N^{1/2})$	$O(N^{1/2})$	OWF	Ad-hoc
[DT24]	$O(N^{1/2})$	$O(N)$	$O(N^{1/2})$	DDH	Ad-hoc
[CL24]	$O(N^{1/2})$	$O(N)$	$O(N^{1/2})$	ROM & LWE	Ad-hoc
Ours (Theorem 2)	$O(N^{\varepsilon})$	$O(N)$	$O(1)$	PIR	Compiler
Ours + dePIR (Corollary A.1)	$O(\text{polylog } N)$	$O(\text{polylog } N)$	$O(1)$	RingLWE	Compiler

Table 1: Comparison of mPIR constructions. Above N is the database size. All schemes incure a λ overhead in all metrics; we omit this for brevity. In the "Ours (Theorem 2)" row, the figures are obtained by applying our generic compiler to any PIR scheme with $O(N^{\varepsilon})$ communication and $O(N)$ computation.

2 Technical overview

Our goal is generically transform *any* semi-honest PIR scheme to a maliciously-secure PIR (mPIR) while incurring *minimal* overhead scheme and without introducing additional cryptographic assumptions. We begin by reviewing (m)PIR.

Background: defining (m)PIR. A PIR protocol is a protocol between a client and a server consisting of the following algorithms:

- Query takes as input a query index, $i^\star$, and outputs a query, query, to be sent to the server.
- Respond takes as input a query, query, executes it on the database, db, and outputs a response resp for the client.
- Decode takes as input a response, $\mathsf{resp}_{i^\star}$, decodes it, and outputs the $i^\star$-th record $\mathsf{db}_{i^\star}$ in db.

A PIR protocol satisfies **correctness** if for every database db and every query index $i^\star$, if Query is run on $i^\star$, and Respond, Decode are run honestly then Decode outputs the correct value, $\mathsf{db}_{i^\star}$. A PIR protocol satisfies **privacy** if for every database db and every query index $i^\star$, the query, $\mathsf{query}_{i^\star}$, leaks no information about $i^\star$ to the server. (This is formalized via a standard simulation-based definition.)

An **mPIR protocol** is a PIR protocol where the server may deviate from the protocol. To obtain reasonable security in the face of a malicious server, an mPIR protocol must satisfy two additional properties.

- ***Malicious* privacy:** a malicious server should not be able to use *selective-failure* attacks to learn information about the client's queries. For example, a malicious server should not be able to corrupt a portion of the database and then use the client's behavior post-response (e.g., whether it has aborted) to deduce information about the queried location.

- **Coherence:** a malicious server cannot respond to two different queries for the i-th database location with inconsistent values: either both responses should be the same value, or at least one response should lead to rejection.

We provide a formal definition of these properties in Section 4.1.1, and now describe how to construct mPIR.

Starting point: mPIR from PIR + Merkle trees. We use as a starting point the following idea of Kushilevitz and Ostrovsky [KO97] that augments a PIR scheme to provide improved integrity properties:

1. In a preprocessing phase, the server computes a Merkle tree over the database, db, and publishes the resulting Merkle Root, cm.
2. To perform a query for the $i^\star$-th record, the client invokes the semi-honest PIR query algorithm as usual.
3. The server answers this query not with respect to db, but with respect to an *augmented* database $\mathsf{db}' := ((\mathsf{db}_1, \pi_1), \ldots, (\mathsf{db}_n, \pi_n))$, where π_i is the Merkle tree path for db_i.
4. The client decodes the response and verifies that the Merkle path is valid with respect to cm.

This compiler achieves various attractive properties: it is modular, efficient, does not require additional assumptions,[2] and makes black-box use of the underlying semi-honest PIR protocol. Unfortunately, while this scheme does satisfy coherence, it does not achieve privacy in the face of selective-failure attacks: a malicious server can execute the client's query with respect to a *corrupted* database that is identical to db' everywhere *except* at the $j^\star$-th location; at the latter location, the server replaces the proof $\pi_{j^\star}$ with $\perp$. For most PIR schemes, the client will abort if and only if $j^\star = i^\star$, inducing a selective failure.[3] Given this flaw, [KO97] noted that this construction is secure against a *covert* adversary, but left it as an open question how construct a PIR protocol secure against a malicious adversary.

[2] The existence of Collision-Resistant Hash Functions and hence Merkle Trees is implied by PIR [IKO05].

[3] We note that that Colombo et al. [CNCWF23] show that this approach can be made to work for a specific class of multi-server PIR schemes; see Section 3 for a detailed discussion.

2.1 Attempt 1: robustness from locally-decodable codes

The problem with the foregoing approach is that the server can introduce a few localized corruptions to the database and trigger a selective failure attack. A natural idea to prevent such localized corruptions would be to encode the database with an error-correcting code, as intuitively this would "spread" any local errors across the entire codeword. Making this idea communication-efficient requires using a *locally decodable code (LDC)*.

Below we elaborate on this idea, focusing for simplicity of exposition on the Reed–Muller (RM) locally-decodable code. (We note that our final construction works with *any* suitable locally-decodable codes)

Background: local decoding for Reed–Muller codes. For a finite field $\mathbb{F}_p$, a number of variables m, and a *total* degree d, a codeword in the Reed–Muller code is the list of evaluations over $\mathbb{F}_p^m$ of an m-variate polynomial of *total* degree d. That is, for an $\binom{m+d}{d}$-symbol message $y_1, \ldots, y_{\binom{m+d}{m}} \in \mathbb{F}_p$, the corresponding Reed–Muller codeword is the list of evaluations of the m-variate, d-total degree, unique polynomial f satisfying $f(i) = y_i$ for all $i \in [\binom{m+d}{m}]$. Reed–Muller codes enjoy several attractive properties, including quasilinear time encoding [HS13] and efficient local decoding, which we describe next.

The standard local decoder for Reed–Muller works by exploiting the fact that *the restriction of a multivariate polynomial to a line is a univariate polynomial*. In more detail, given access to a (possibly corrupted) codeword $\tilde{f}$, the client can decode the $i^\star$-th message symbol $f(i^\star)$ by sampling a random line $\ell(X) = mX + i^\star$ that passes through $i^\star$, and querying for all evaluations of $\tilde{f}(\ell(X))$. If $\tilde{f}$ is close to a Reed–Muller codeword f, then with high probability the evaluations of $\tilde{f}(\ell(X))$ will be close to a *Reed–Solomon* codeword encoding $f(i^\star)$, allowing us to recover $f(i^\star)$ by invoking any standard (non-local) decoder for Reed–Solomon codes.

We are now ready to describe our proposed fix to the Merkle-tree-based mPIR scheme.

Proposal: mPIR from Merkle trees + LDC. Offline, the server publishes a commitment cm to E, the Reed–Muller encoding of the database, db. Online, to retrieve the $i^\star$-th record in db, the client samples a random line ℓ that passes through $i^\star$ and performs *semi-honest* PIR queries to retrieve the codeword symbols (i.e., evaluations) that lie on ℓ. The PIR server answers these queries with respect to E', the augmented analogue of E where the codeword symbols are accompanied by Merkle proofs. Upon receiving the responses, the client invokes the Reed–Solomon decoder to reconstruct the $i^\star$-th record, treating every invalid Merkle proof as an "erasure". If the number of erasures on the line ℓ is less than $p - d$, then the local decoder will succeed and the client will output the $i^\star$-th element of the database. Else, the client outputs $\perp$.

It is straightforward to see that this scheme satisfies correctness and coherence (i.e., any non-$\perp$ output is correct). Unfortunately, it does not satisfy privacy.

The problem is subtle: although for any fixed corruptions set of corrupted points C, the *expected* number of corrupted responses received is when the client queries $i^\star$ is the same as when it queries $j^\star$, for carefully chosen C, the *variance* can dramatically vary. In particular, consider an adversary which chooses a particular point, $j^\star$, picks an ϵ-fraction of the lines which go through $j^\star$, and corrupts them entirely. Now, if $i^\star = j^\star$, then with probability ϵ, the decoder would choose one of the corrupted lines and output $\perp$. On the other hand, if $\epsilon < \frac{p-d}{p}$ it's not too hard to show that the probability a line through $i^\star \neq j^\star$ has more than $p - d$ corruptions can be made much smaller (close to) 0.

More generally, the issue is that while Reed–Muller codes (and LDCs in general) guarantee that probability of decoding success is high for all indices (assuming few corruptions), we require a stronger guarantee: the probability of decoding success should be close for every pair of distinct indices. Indeed, such a guarantee would ensure that no matter the corruption pattern, decoding for $i^\star$ would be as successful as decoding for $j^\star$, and would hence allow us to establish privacy.

A natural idea to obtain such a guarantee would be to try and increase the probability of decoding success by repetition, i.e. making more queries, attempting to decode all of them, and outputting any valid decoding result. Unfortunately, the number of repetitions required to drive the gap between decoding success probabilities to negligible is large, leading to a significant overhead.

Instead, we take the opposite approach: instead of trying to overcome corruptions, we try to *detect* them in a manner that is independent of the query index $i^\star$. We elaborate on this idea in the next section.

2.2 Attempt 2: detecting corruptions with test queries

To detect corruptions, we introduce "test queries" into our client's decoding algorithm. Specifically, we supplement the "decoding" set $\mathcal{D} \subset \mathbb{F}_p^m$ (containing queries corresponding to l random lines through $i^\star$) with a "test" set $\mathcal{T} \subset \mathbb{F}_p^m$ consisting of $O(\lambda)$ uniform and independently chosen test queries, where λ is a security parameter. After sampling these sets, the client prepares a PIR query for every index $i \in \mathcal{T} \cup \mathcal{D}$, shuffles the queries randomly, and sends them to the server. Upon receiving responses, the client checks the Merkle proofs of all the test queries, and outputs $\perp$ if any of them are invalid, *even if it could otherwise decode*. If no test queries are corrupted, the client decodes the lines as usual.

A reasonable approach to proving that the above approach achieves privacy would be as follows. First, since the test set $\mathcal{T}$ is sampled independently of $i^\star$, the probability that the client finds a corrupted test query is *independent* of $i^\star$. Thus, if the client finds a corrupted test query, it can safely abort without "leaking" its query index, $i^\star$, to the adversarial server. Second, since $\mathcal{T}$ is of size $O(\lambda)$, is sampled uniformly from $\mathbb{F}_p^m$, and is shuffled with the decoding queries, if the client receives *no* corrupted test queries, then most (i.e. $1 - o(1)$-fraction) of the adversary's responses are uncorrupted. Thus, the client needs to only query a modest number of lines to successfully decode with high probability.

The problem with the preceding argument is that it implicitly assumes that the privacy property of the (semi-honest) PIR protocol ensures that the adversary's corruption pattern is independent of the underlying query indices. Unfortunately, the privacy of the underlying PIR protocol only gives us the weaker guarantee that the adversary's responses must follow a *non-signaling* distribution. This obstacle has been encountered in the succinct argument literature [DLNNR04; KRR14] when attempting to use PIR to make multiple correlated queries into a PCP.

To avoid this problem, we show how to modify this construction so that the "test" queries are information-theoretically hidden.

2.3 Our construction

We now describe how to fix the foregoing issues and prove Theorem 1.

At a high level, our construction leverages smoothness of Reed–Muller codes to better hide the test queries within the decoding queries. The resulting decoding algorithm, described below, is deceptively simple:

1. Sample the decoding set $\mathcal{D}$ by sampling $O(\lambda)$ many independent lines through $i^\star$.
2. Sample the test set $\mathcal{T}$ by sampling a uniformly random point on each line sampled in Step 1.

We will now argue that this decoder achieves "query-independent" abort even against *unrestricted* (i.e. not necessarily non-signaling) adversaries. By smoothness of Reed–Muller, and because a uniformly random point on a uniformly random line through $i^\star$ is uniformly random in $\mathbb{F}_p^m$, $\mathcal{T}$ is uniformly random and hence *independent* of $i^\star$. This means that *any* corruption pattern that the adversary chooses will be detected with all-but-negligible probability, and the client will abort without revealing $i^\star$.

We show how to expand this intuition into a formal proof in Section 5. Along the way, we generalize our construction to work with a special class of LDCs that we call "subcode"-LDCs.

Let us now analyze the efficiency of our construction. It is straightforward to see that communication complexity and client computation complexity remain sublinear, since they incur an overhead of $O(\mathsf{q}_{\mathsf{LDC}} \cdot \lambda)$, where $\mathsf{q}_{\mathsf{LDC}}$ is the query complexity of the LDC. (When instantiated with the Reed–Muller code, $\mathsf{q}_{\mathsf{LDC}} \approx O(N^{\epsilon})$.) However, as noted in Section 1.1, a straightforward usage of PIR to independently retrieve each query in $\mathcal{T} \cup \mathcal{D}$ results in server computation that scales as $O(N \cdot \mathsf{q}_{\mathsf{LDC}} \cdot \lambda)$. When instantiated with the Reed–Muller code, $\mathsf{q}_{\mathsf{LDC}} \approx O(N^{\epsilon})$, and so the server's incurs a superlinear cost of $O(N^{1+\epsilon})$. We show how to reduce this cost to $O(N)$ next in Section 2.3.1.

2.3.1 Shaving the query-complexity overhead with lossy batching

The standard approach for reducing the computational overhead of preforming a batch of q PIR queries is *batch codes* [IKOS04]. Roughly speaking, a q-batch encoding of a database $\mathsf{db} = \mathsf{db}_1, \ldots, \mathsf{db}_n$ is a list of buckets $B_1, \ldots, B_m$ where each bucket B_i is a subset of $\{\mathsf{db}_1, \ldots, \mathsf{db}_n\}$, and each database element is stored in several buckets.[4] Given any subset of indices $Q \subset [n]$ of size at most q, an efficient batch decoder can recover db_Q by querying exactly *one* element per bucket. A PIR client can retrieve a batch of records db_Q from a server holding a batch–encoded db by sending a single PIR query to each bucket as prescribed by the batch decoding algorithm. If the encoding has constant rate (i.e. $\sum_{i \in [m]} |B_i| = O(n)$) then the batch of q PIR queries is processed with only $O(1)$ overhead. Indeed, even for $\mathsf{q} = O(n^{\epsilon})$, we know batch codes with constant rate [IKOS04].

Unfortunately we cannot use batch codes as-is, because for a given index $j \in Q$, the bucket B from which db_j should be retrieved can depend on *all* of Q. As we explain next, this dependency can be exploited by a malicious server. Consider batch encoding the Reed–Muller encoding of the database, as would naturally be done for our scheme. Let $i, j, k \in \mathbb{F}_p^m$ be entries noticeably more likely to be queried if $i^\star$ is the query index (e.g. three points on a line through $i^\star$). With noticeable probability, i, j, k are stored in the same bucket B. Since only one of them could be queried in B, we have that i is *less* likely to be queried in bucket B if $i^\star$ is the query index then otherwise. Thus, an adversary corrupting i *only* in bucket B could get noticeable advantage at guessing $i^\star$ by observing the discrepancy in i's corruption probability. That is because even though our scheme ensures the probability that $i \in \mathcal{T}$ does not depend on $i^\star$, the probability $i \in \mathcal{T}$ *and* queried in bucket B *does* depend on $i^\star$.[5]

Fortunately we can circumvent this issue by leveraging a neat property of our setting: with high probability, the Reed–Muller decoder will succeed even when a small constant fraction of queries are "dropped." We leverage this property by modifying our decoder as follows: given a query set $Q \leftarrow D_{\mathsf{d}}(i^\star)$ as input, for each $x \in Q$, our batch code query algorithm picks a random i such that $x \in B_i$.[6] If the algorithm has already picked B_i for some previous x', it will "drop" x (i.e. not query x at all). Else, it will query x at B_i. By setting the parameters of the batch codes appropriately, we are able to gurentee that except from with negligable probability, at most a small (constant) fraction of elements are dropped.

Correctness still holds because Reed–Muller decoding can handle a small constant fraction of drops by treating them as erasures. Privacy holds because every x that isn't dropped is queried at a bucket that is chosen *independently* of Q. At a very high level, our analysis leverages this to show that even an adversary that (partially) knew which elements were dropped couldn't use this information to violate a client's privacy. We leave the full proofs and further discussion to Section 6.

[4] We note that in some batch codes, it is not the case that buckets are subsets of $\{\mathsf{db}_1, \ldots, \mathsf{db}_n\}$. However, all sufficiently efficient batch codes known to us are vulnerable to the attack described below.

[5] Note we could query each bucket with an *m*PIR, but our computational complexity would not improve.

[6] A subtlety in the analysis requires that we first "try" to query elements of the test set $\mathcal{T}$, then elements of $\mathcal{D}$, but we omit this detail here.

2.3.2 From semi-malicious to malicious security

So far, we have considered the case where the commitment contains a valid codeword. However, in general an adversarial server might publish an arbitrary string cm which has no relation to any codeword. To obtain security in such a strongly adversarial setting, we rely on *proximity tests* (Definition 4.14). A proximity test for a code $\mathcal{C}$ is a (randomized) local algorithm which queries an arbitrary string w at few locations and determines with high probability whether w is close to some codeword in $\mathcal{C}$. Thus, to certify that cm is a commitment to a string w that is δ-close to a Reed–Muller codeword, the client makes queries to the server in the clear according to an appropriate proximity tester. The client aborts (independently of $i^\star$) if the proximity test rejects. Given that the proximity test accepted, the client can be sure that w is at most δ-far from $\mathcal{C}$. For a sufficiently small δ, the proofs previously outlined need only slight tweaks.

3 Related Work

The potential for selective failure attacks was observed in the first single-server PIR work [KO97], but the first rigorous treatment of the problem was not given until [CNCWF23].

Multi-server setting. As discussed in Section 2, [CNCWF23] show how to compile any linear-reconstruction k-server PIR scheme into a malicious secure PIR scheme, assuming vector commitments scheme and that at least one server is honest. Following up, Wang et al. [WZLY23] adapt ideas from sublinear computation multi-server PIRs [CK20; SACM21; KC21; LP23] to build multi-server PIR protocols with amortized $O(\sqrt{n})$ server computation. Unfortunately, their construction offers rather weak security: a particular designated server must be *semi-honest* or the scheme breaks.

Single-server setting. In order for a client to verify a PIR response, the client needs a commitment, cm, to the "true" database. In the multi-server setting, each server can include this commitment in their PIR response, and the client will check if all these commitments match.

In the single-server setting, things are a bit more complicated. In [CNCWF23], it is assumed the commitment, cm, is produced by an honest "data-holder" and provided to the client before the PIR protocol begins. In this "semi-malicious" setting, [CNCWF23] provide two novel single-server constructions from LWE or DDH. At a high level, the schemes share a similar structure: offline, the honest data-holder publishes a $O(n)$-sized (homomorphic) commitment cm to $\mathsf{db} \in \{0,1\}^n$ (it is crucial db_i is in a small subdomain). Online, clients query the database by sending an $O(n)$-length additively homomorphic encryption of a 1-hot vector, e. The server responds by taking the dot-product of e and db, and the client decrypt the response *using the commitment* cm. It can then be shown that, if the server computes the inner-product of e with some corrupted database $\mathsf{db}' \neq \mathsf{db}$, corruptions would propagate in a "random linear combination" fashion, leading to the client decoding some $v \notin \{0,1\}$ and aborting regardless of the query index. Through a "standard square-root rebalancing trick" (i.e., execute $\sqrt{n}$-many PIR protocols on $\sqrt{n}$-sized databases on the *same* client query), [CNCWF23] obtain a semi-malicious single-server PIR scheme with $O(n)$ computation, $O(\sqrt{n})$ communication.

In addition to their 2-server mPIR scheme, [WZLY23] give a semi-malicious single-server PIR protocol with *amortized* $O(\sqrt{n})$ computation, by delegating the role of one of the servers to the client. Unfortunately, in their scheme, the *client* must re-process the *entire* database (in a streaming fashion) every $O(\sqrt{n})$ queries.

[DT24] study the "fully-malicious" setting where cm may be adversarially generated (potentially independently of db). Since in this setting it is impossible define the "true" databases, [DT24] relax security to only require the server's responses are consistent with *some* database. However, cm is still required to ensure consistent views between clients. In addition to these definitions, [DT24] show that by supplementing each query with λ "random" (rather than 1-hot) queries, the DDH-scheme of [CNCWF23] can be made fully-malicious. Continuing in the fully malicious setting, [CL24] build upon the LWE-based scheme of [CNCWF23], and show that (in the random oracle model) the server can provide an (extractable) proof that cm is an well-formed commitment. They leverage this to achieve online throughput close to that of the most practical semi-honest schemes [HHGMV23] (and much better than that of the LWE-based scheme of [CNCWF23]).

Summary. To summarize, the asymptotic performance [CNCWF23; DT24; CL24]'s schemes seems to be bound $O(n)$ computation and $\Theta(\sqrt{n})$ communication, commitment size, and client computation, by the square-root rebalancing trick. All of the existing single-server schemes rely on concrete cryptographic hardness assumptions (like LWE or DDH). This comparison is summarized in Table 1.

Terminology. [CNCWF23] use the term "authenticated PIR" to refer to the setting in which the server is malicious and the data holder is honest. "fully-malicious authenticated PIR" was used by [DT24] to refer to

the setting in which both the server and data holder are malicious. [CL24] use the term "verifiable PIR" to refer to the malicious-commitment setting, while [WZLY23] use it to refer to the honest-commitment settings and related works [ZWH21; BKP22] use it to give completely different guarantees (see Section 3.1)

Despite these names, there is no authentication happening in these protocols, whereas they are capturing security in the presence of a malicious server, much the same as malicious security in a generic MPC protocol. Thus, we propose to call the honest-commitment, malicious-PIR server setting "semi-maliciously secure PIR" (shortened: smPIR) and the malicious-commitment, malicious-PIR server "maliciously-secure PIR" (shortened: mPIR).[7]

3.1 Other related work

In this section we will present a summary and comparison to related works which do not operate in our precise setting, but have some similarities with our work.

Related maliciousness compilers. Eriguchi et al. [EKN24] consider the setting where there is some client holding an input, x, and a large number of servers holds a function f (from some restricted class which includes the PIR function $f(i) = \mathsf{db}_i$). The client wants to minimally interact with the servers to learn $f(x)$ while keeping x hidden from the servers. In this setting, they show generic compiler which increase the number of servers or the number of rounds, and sometimes introduce computation assumptions to upgrade protocols secure against semi-honest servers to malicious server. While this is interesting, we view their work as mostly tangential to ours, as few servers is the setting of interest in PIR. Ben-David et al. [BKP22] define (and construct) a notion of *verifiable PIR* where the server can prove to the client some limited yet expressive range of database properties. Their construction depends on succinct batch arguments, which means that it requires specific computational assumptions, and only achieves a $\mathrm{poly}(\lambda, N)$ server time. They can also prove that the server executed the PIR honestly, and indeed this is the trivial way to construct PIR from mPIR.

Previous connections between PIR and LDCs. The connection between *multi-server* PIR and smooth LDC runs deep: given a smooth LDC with query complexity q, one can easily construct a q-server PIR by having the client request a different symbol from each server [Yek10, Lemma 7.2]. [CHR17] introduce another connection between LDC and PIR when they tried to use Reed–Muller codes to build doubly-efficient PIR. However, their construction is only heuristic and has since been shown insecure [BHMW21]. We introduce a different connection between PIR and LDCs: we use LDCs to get *single-server* mPIR, rather than multi-server PIR.

Tangentially related multiserver works. A variety of works attempt to construct multi-server "verifiable" PIR schemes [ZS14; WZ18; Cao+23; KDKWZ23; KZ23], but do not consider selective failure attacks. Despite this, some of their schemes have sufficiently many servers such that they can necessarily give correctness. Zhao et al. [ZWH21] construct a single server "verifiable" PIR scheme, but their construction does not seem to have clear definitions and security proofs.

[7] We note Angel et al. [ACLS18] used "mPIR" to describe their "multi-query PIR." However, "batch PIR" [IKOS04] is the standard name for this gadget.

4 Preliminaries

Notation. Throughout this work we use λ to denote the security parameter. We use $[n]$ to denote the set $\{1, \ldots, n\}$. For a vector, $v = (v_1, \ldots, v_n)$, we slightly abuse notation and let $x \in v$ denote $\exists i \in [n]$ such that $v_i = x$, let $v \cup \{y\}$ denote the new vector $v' = (v_1, \ldots, v_n, y)$, and for a set/vector $\{s_1, \ldots, s_k\} = S \subset [n]$ we let $v_S = (v_{s_1}, \ldots, v_{s_k})$. For a random variable $\mathcal{S}$ over (a subset of) $2^{[n]}$ we let $v_{\mathcal{S}}$ denote sampling $S \leftarrow \mathcal{S}$ then outputting v_S For a vector $x = (x_1, \ldots, x_n)$ and a function f we define $f(x) = (f(x_1), \ldots, f(x_n))$. For two functions f, g, we denote by $\delta(f, g)$ the fraction of inputs on which f and g differ.

All adversaries considered in this paper will be non-uniform. When describing the client's behavior, we use the language "output $\perp$" and "abort" synonymously.

4.1 Private information retrieval

A private information retrieval (PIR) scheme is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Preprocess}, \mathsf{Query}, \mathsf{Respond}, \mathsf{Decode})$ with the following syntax:

- **Setup:** on input 1^λ, Setup outputs public parameters $\mathsf{pp} = (\mathsf{cpp}, \mathsf{spp})$ for the PIR server and client, respectively.
- **Preprocessing:** on input public parameters pp, and a database $\mathsf{db} \in \Sigma^k$, $\mathsf{Preprocess}$ outputs a public state pst and a server state sst.[8]
- **Client query:** on input public parameters cpp, the public state pst, and a query index $i^\star \in [k]$, Query outputs a message, query, that will be sent to the server and a private client state, cst.
- **Server response:** on input the server state sst and a client query query, $\mathsf{Respond}$ outputs a response resp.
- **Client decoding:** on input the client state cst and a server response resp, Decode outputs an answer $\mathsf{ans} \in \mathsf{db}$.

We require a PIR scheme to satisfy the following properties:

- **Correctness:** For every database $\mathsf{db} \in \Sigma^k$, and query index $i^\star \in [k]$, the following holds:

$$\Pr\left[\mathsf{ans} \neq \mathsf{db}_{i^\star} \;\middle|\; \begin{array}{r} (\mathsf{cpp}, \mathsf{spp}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pst}, \mathsf{sst}) \leftarrow \mathsf{Preprocess}(\mathsf{spp}, \mathsf{db}) \\ \hline (\mathsf{query}, \mathsf{cst}) \leftarrow \mathsf{Query}(\mathsf{pst}, \mathsf{cpp}, i^\star) \\ \mathsf{resp} \leftarrow \mathsf{Respond}(\mathsf{sst}, \mathsf{query}) \\ \mathsf{ans} \leftarrow \mathsf{Decode}(\mathsf{cst}, \mathsf{resp}) \end{array}\right] = \mathrm{negl}(\lambda) \quad .$$

- **Privacy:** For every (efficient) adversarial server $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, large enough security parameter $\lambda \in \mathbb{N}$, database $\mathsf{db} \in \Sigma^k$, and query location $i^\star \in [k]$, there exists an efficient simulator $\mathcal{S}$ such that the following distributions are (computationally) indistinguishable:

$$\left\{ x \;\middle|\; \begin{array}{r} (\mathsf{cpp}, \mathsf{spp}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pst}, \mathsf{sst}) \leftarrow \mathcal{A}_0(\mathsf{cpp}, \mathsf{spp}, \mathsf{db}) \\ \hline (\mathsf{query}, \mathsf{cst}) \leftarrow \mathsf{Query}(\mathsf{cpp}, \mathsf{pst}, i^\star) \\ x \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}) \end{array}\right\} \quad \text{and} \quad \left\{ x \;\middle|\; \begin{array}{r} (\mathsf{cpp}, \mathsf{spp}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda) \\ (\mathsf{pst}, \mathsf{sst}) \leftarrow \mathcal{A}_0(\mathsf{cpp}, \mathsf{spp}, \mathsf{db}) \\ \hline (\mathsf{query}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}.\mathsf{Query}(\mathsf{cpp}, \mathsf{pst}) \\ x \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}) \end{array}\right\}$$

[8] We note not all PIR schemes exhibit preprocessing, but it has been shown preprocessing can reduce the overhead of PIR [BIM04]. Our scheme requires preprocessing.

We refer to this variant as "plain" or "semi-honest" PIR.
Several variants of the previous definition have been studied in the literature. We recap some relevant ones below.

Multi-server PIR. A natural extension would be to consider a setting with *multiple* servers of which only some are corrupted. Formally, a (k, t)-PIR scheme is one with k servers $S_1, \ldots, S_k$, of which at most t are corrupt [CGKS98]. Below we outline how to adapt the previous definition to this setting.

In the multi-server setting, Query is changed to output k queries $(\mathsf{query}_1, \ldots, \mathsf{query}_k)$, one for each of server, $S_1, \ldots, S_k$. The i-th server runs Respond on query_i and outputs a response resp_i. The client invokes Decode on the responses $(\mathsf{resp}_1, \ldots, \mathsf{resp}_k)$ and outputs an answer ans.

The privacy game is changed as follows. Let $A \subset [k]$ be the set of corrupted servers such that $|A| \leq t$. In both worlds, the client receives honest responses $\{\mathsf{Respond}(\mathsf{query}_h)\}_{h \in [k] \setminus A}$ from the honest parties, and adversarial responses $\{\mathsf{resp}_a\}_{a \in A} \leftarrow \mathcal{A}(\mathsf{query}_A)$. Privacy is now required to hold for every possible subset A.

Multi-query privacy. In our security proofs, we will rely on a notion of privacy that considers multiple queries made via independent PIR instances. The standard definition of (semi-honest) privacy implies this one via a standard hybrid argument.

Definition 4.1. *A PIR scheme is said to achieve* **multi-query privacy** *if for every (efficient) adversarial server $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, large enough security parameter $\lambda \in \mathbb{N}$, every database $\mathsf{db} \in \Sigma^k$, and all query vectors $q, q' \in [k]^{\mathsf{q}}$, the following distributions are (computationally) indistinguishable:*

$$\left\{ x \;\middle|\; \begin{array}{l} \textit{for each } i \in [\mathsf{q}], \\ \quad (\mathsf{cpp}_i, \mathsf{spp}_i) \leftarrow \mathsf{Setup}(1^\lambda) \\ [(\mathsf{pst}_i, \mathsf{sst}_i)]_{i \in [\mathsf{q}]} \leftarrow \mathcal{A}_0([\mathsf{spp}_i]_{i \in [\mathsf{q}]}, \mathsf{db}) \\ \hline \textit{for each } i \in [\mathsf{q}], \\ \quad (\mathsf{query}_i, \mathsf{st}_i) \leftarrow \mathsf{Query}(\mathsf{cpp}_i, \mathsf{pst}_i, q_i) \\ \hline x \leftarrow \mathcal{A}_1(\mathsf{sst}, [\mathsf{query}_i]_{i=1}^{\mathsf{q}}) \end{array} \right\} \quad \textit{and} \quad \left\{ x \;\middle|\; \begin{array}{l} \textit{for each } i \in [\mathsf{q}], \\ \quad (\mathsf{cpp}_i, \mathsf{spp}_i) \leftarrow \mathsf{Setup}(1^\lambda) \\ [(\mathsf{pst}_i, \mathsf{sst}_i)]_{i \in [\mathsf{q}]} \leftarrow \mathcal{A}_0([\mathsf{spp}_i]_{i \in [\mathsf{q}]}, \mathsf{db}) \\ \hline \textit{for each } i \in [\mathsf{q}], \\ \quad (\mathsf{query}_i, \mathsf{st}_i) \leftarrow \mathsf{Query}(\mathsf{cpp}_i, \mathsf{pst}_i, q'_i) \\ \hline x \leftarrow \mathcal{A}_1(\mathsf{sst}, [\mathsf{query}_i]_{i=1}^{\mathsf{q}}) \end{array} \right\}.$$

4.1.1 Maliciously-secure PIR

A *maliciously-secure* private information retrieval (mPIR) scheme is a PIR scheme that satisfies a stronger notion of privacy and an additional *coherence* property. We formalize these notions below, highlighting any difference from standard PIR properties in blue.

- **Malicious Privacy:** For every (efficient) adversarial server $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, large enough security parameter $\lambda \in \mathbb{N}$, database $\mathsf{db} \in \Sigma^k$, and query location $i^\star \in [k]$, there exists an efficient simulator $\mathcal{S}$ such that the following distributions are (computationally) indistinguishable:

$$\left\{ x \;\middle|\; \begin{array}{r} (\mathsf{cpp}, \mathsf{spp}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pst}, \mathsf{sst}) \leftarrow \mathcal{A}_0(\mathsf{cpp}, \mathsf{spp}, \mathsf{db}) \\ \hline (\mathsf{query}, \mathsf{cst}) \leftarrow \mathsf{Query}(\mathsf{cpp}, \mathsf{pst}, i^\star) \\ (\mathsf{resp}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}) \\ \mathsf{ans} \leftarrow \mathsf{Decode}(\mathsf{cst}, \mathsf{resp}) \\ \hline x \leftarrow \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{ans} \stackrel{?}{=} \perp) \end{array} \right\} \quad \text{and} \quad \left\{ x \;\middle|\; \begin{array}{r} (\mathsf{cpp}, \mathsf{spp}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda) \\ (\mathsf{pst}, \mathsf{sst}) \leftarrow \mathcal{A}_0(\mathsf{cpp}, \mathsf{spp}, \mathsf{db}) \\ \hline (\mathsf{query}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}.\mathsf{Query}(\mathsf{cpp}, \mathsf{pst}) \\ (\mathsf{resp}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}) \\ \mathsf{ans} \leftarrow \mathcal{S}.\mathsf{Decode}(\mathsf{st}_{\mathcal{S}}, \mathsf{resp}) \\ \hline x \leftarrow \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{ans} \stackrel{?}{=} \perp) \end{array} \right\}$$

- **Coherence:** For every efficient adversarial server $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, and every query index $i^\star \in [k]$, the following holds:

$$\Pr\left[\begin{array}{l|r} \mathsf{ans}' \notin \{\mathsf{ans}, \perp\} & (\mathsf{spp}, \mathsf{cpp}) \leftarrow \mathsf{Setup}(1^\lambda) \\ & (\mathsf{pst}, \mathsf{sst}) \leftarrow \mathcal{A}_0(\mathsf{spp}) \\ \hline & (\mathsf{query}, \mathsf{cst}) \leftarrow \mathsf{Query}(\mathsf{cpp}, \mathsf{pst}, i^\star) \\ & (\mathsf{query}', \mathsf{cst}') \leftarrow \mathsf{Query}(\mathsf{cpp}, \mathsf{pst}, i^\star) \\ \hline & (\mathsf{resp}, \mathsf{resp}') \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}, \mathsf{query}') \\ \hline & \mathsf{ans} \leftarrow \mathsf{Decode}(\mathsf{cst}, \mathsf{resp}) \\ & \mathsf{ans}' \leftarrow \mathsf{Decode}(\mathsf{cst}', \mathsf{resp}') \end{array}\right] = \mathrm{negl}(\lambda)$$

In this work, we consider an alternate definition of coherence that is more convenient to work. Below we state this equivalent definition.

Definition 4.2 (Alternate characterization of Coherence)**.** *For every efficient adversarial server $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, with all but negligible probability over honestly-generated public parameters $(\mathsf{spp}, \mathsf{cpp}) \leftarrow \mathsf{Setup}(1^\lambda)$, and public and server states $(\mathsf{pst}, \mathsf{sst}) \leftarrow \mathcal{A}_0(\mathsf{spp})$, there exists a database db such that for every $i^\star$,*

$$\Pr\left[\begin{array}{l|r} \mathsf{ans} \notin \{\mathsf{db}_{i^\star}, \perp\} & (\mathsf{query}, \mathsf{cst}) \leftarrow \mathsf{Query}(\mathsf{cpp}, \mathsf{pst}, i^\star) \\ & \mathsf{resp} \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}) \\ & \mathsf{ans} \leftarrow \mathsf{Decode}(\mathsf{cst}, \mathsf{resp}) \end{array}\right] = \mathrm{negl}(\lambda)$$

Lemma 4.3. *The alternate characterization of coherence (Definition 4.2) is equivalent to the notion of coherence.*

Proof of Lemma 4.3. Suppose there is an adversary $\mathcal{A}$ that can violate the coherence property. We define a new adversary $\mathcal{A}' = (\mathcal{A}'_0, \mathcal{A}'_1)$, for the security game in Definition 4.2, with $\mathcal{A}'_0 = \mathcal{A}_0$.

Now, we run *two* instances of the security game in Definition 4.2, and define $\mathcal{A}'_1$ as follows. In the first step of the security game of Definition 4.2, since we are running two instances of the game, we obtain

$$(\mathsf{query}, \mathsf{cst}) \leftarrow \mathsf{Query}(\mathsf{cst}, \mathsf{pst}, i^\star)$$
$$(\mathsf{query}', \mathsf{cst}) \leftarrow \mathsf{Query}(\mathsf{cst}', \mathsf{pst}, i^\star)$$

Now, $\mathcal{A}'_1$ passes these responses to $\mathcal{A}_1$ to obtain $(\mathsf{resp}, \mathsf{resp}') \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}, \mathsf{query}')$. Then in the first instance of the game, $\mathcal{A}'_1$ responds with resp and in the second instance of the game $\mathcal{A}'_1$ responds with resp'. If we let

$$\mathsf{ans} \leftarrow \mathsf{Decode}(\mathsf{cst}, \mathsf{resp})$$
$$\mathsf{ans}' \leftarrow \mathsf{Decode}(\mathsf{cst}', \mathsf{resp}')$$

By assumption on $\mathcal{A}$, with non-negligible probability ans and ans' are distinct, and not equal to $\perp$. When this happens $\mathcal{A}'_1$ violates the security of Definition 4.2. Thus we have that Definition 4.2 implies coherence.

For the opposite direction, suppose there is an adversary $\mathcal{A}$ that can violate the security property in Definition 4.2. We define an adversary, $\mathcal{A}'$, for the coherence game as follows. We let $\mathcal{A}'_0 = \mathcal{A}_0$. When $\mathcal{A}'_1$ receives $\mathsf{sst}, \mathsf{query}, \mathsf{query}'$ in the coherence game, $\mathcal{A}'_1$ runs two copies of $\mathcal{A}_1$

$$\mathsf{resp} \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query})$$

$$\mathsf{resp}' \leftarrow \mathcal{A}_1(\mathsf{sst}, \mathsf{query}')$$

then $\mathcal{A}'_1$ returns $\mathsf{resp}, \mathsf{resp}'$. By assumption on $\mathcal{A}$, with non-negligible probability ans and ans' are distinct (not both equal to $\mathsf{db}_{i^\star}$), and not equal to $\perp$. Thus we have that coherence implies Definition 4.2. □

Remark 4.4 (semi-malicious PIR)**.** The coherence requirement in malicious PIR is rather strong: it requires coherence to hold even when the preprocessed public state pst was generated maliciously. Prior work [CNCWF23; WZLY23] considered a notion called *integrity* that requires that pst is generated honestly via the $\mathsf{Preprocess}$ algorithm. In this case, there is some *true* database, db, so we can require that $\mathsf{db}_{i^\star}$ in Definition 4.2 is from this *true* database. In this work, we focus on the stronger notion *fully*-malicious PIR, which does not assume that pst is honestly generated. Thus Definition 4.2 does not $\mathsf{db}_{i^\star}$ comes from the "true" database, since a malicious adversary could produce pst that is not consistent with *any* underlying database.

Doubly-efficient mPIR. A doubly-efficient maliciously-secure PIR scheme [CHR17] is one where $\mathsf{Preprocess}$ runs in time $O(k \cdot \mathrm{polylog}(k, \lambda))$, $\mathsf{Query}, \mathsf{Respond}$, and Decode run in time $O(\mathrm{polylog}(k, \lambda))$, and dig, query, and resp are of size $O(\mathrm{polylog}(k, \lambda))$, where $k = |\mathsf{db}|$.

4.2 Vector commitments

A vector commitment scheme is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- **Setup:** on input 1^λ, Setup outputs a committer key, ck, and a verifier key, vk.
- **Commit:** on input the committer key, ck, and a vector $v \in \Sigma^n$, Commit outputs a commitment, cm, and decommitment material $\mathfrak{d}$.
- **Prove:** on input the committer key, ck, decommitment material, $\mathfrak{d}$, and an index to open, $i^\star \in [n]$, Prove outputs an opening proof, π.
- **Verify:** on input the verifier key, vk, a commitment, cm, an index, $i \in [n]$, claimed opening, v_i, and opening proof π_i, Verify outputs a bit indicating whether the i-th element of the vector v committed in cm equals v_i.

A vector commitment scheme satisfies the following properties:

- **Correctness:** For every vector $v \in \Sigma^n$ and index $i \in [n]$, the following holds:

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, \mathsf{cm}, i, v_i, \pi_i) = 1 \;\middle|\; \begin{array}{r}(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{cm}, \mathfrak{d}) \leftarrow \mathsf{Commit}(\mathsf{ck}, v) \\ \pi_i \leftarrow \mathsf{Prove}(\mathsf{ck}, \mathfrak{d}, i)\end{array}\right] = 1 - \mathrm{negl}(\lambda) \quad .$$

- **Binding**: For every efficient adversary $\mathcal{A}$, the following holds:

$$\Pr\left[\begin{array}{c}\mathsf{Verify}(\mathsf{vk}, \mathsf{cm}, i, v_i, \pi_i) = 1 \\ \wedge \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{cm}, i, v'_i, \pi'_i) = 1 \\ \wedge \\ v_i \neq v'_i\end{array} \;\middle|\; \begin{array}{r}(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{cm}, i, v_i, v'_i, \pi_i, \pi'_i) \leftarrow \mathcal{A}(\mathsf{ck})\end{array}\right] = \mathrm{negl}(\lambda) \quad .$$

Theorem 4.5 (CRH from non-trivial computational PIR [IKO05])**.** *Suppose there exists a (semi-honest) single-server PIR protocol with query complexity $q(n)$ and computational complexity $c(n)$, Then there exists a Collision-Resistant Hash function which takes n-bit strings to $\omega(\lambda \cdot q(n))$-bit digests, and requires $\omega(\lambda \cdot c(n))$ computation.*

Combining this with the Merkle Tree construction, we have

Theorem 4.6 (Merkle Trees [Mer89])**.** *Assume the existence of a secure single-server PIR protocol, then there exists a vector commitment scheme.*

4.3 Algorithms with oracle access

Standard descriptions of local computation algorithms are often given in terms of algorithms which have query access to an oracle. The local decoding problems we consider in this paper involve the decoder interacting with a special kind of oracle that is allowed to respond with $\perp$, effectively erasing the response to the query. Below we specify this behavior formally.

Definition 4.7. *A* q*-query oracle* $\mathcal{O}_F$ *for a fixed* $F \in \Sigma^n$ *is defined by a computationally non-signaling strategy* $\{\mathcal{P}_q\}_{q\in[n]^{\mathsf{q}}}$ *with range* $\{\not\perp, \perp\}$*, and operates as follows when given as input* $q = (q_1, \ldots, q_{\mathsf{q}})$*:*
1. *Sample* $b \leftarrow \mathcal{P}_q$*.*
2. *For each* $i \in [\mathsf{q}]$*, if* $b_i = \perp$*, set* $r_i := \perp$*; else set* $r_i := F_{q_i}$*.*
3. *Output* $r = (r_1, \ldots, r_{\mathsf{q}})$*.*

We say that the oracle is *honest*, and denote it by $\mathcal{H}_F$, if the strategy used by the oracle never returns $\perp$.

We assume that every algorithm $A^{\mathcal{O}_F}(x)$ interacting with an oracle $\mathcal{O}_F$ can be split into two parts, a *query* algorithm that takes as input x and outputs a query vector q and state st_q, and a *reconstruction* algorithm that takes as input st_q and the oracle response $\mathcal{O}_F(q)$ and outputs some result. The query complexity, q, of an algorithm $A^{\mathcal{O}_F}(x)$ is the size of the query vector q.

4.4 Background on coding theory

Definition 4.8. *An* **error-correcting code** *of length* n *over an alphabet* Σ *is a subset* $\mathcal{C} \subseteq \Sigma^n$*. A code* $\mathcal{C}$ *is said to be* **linear** *if* Σ *is a field* $\mathbb{F}$*, and* $\mathcal{C}$ *is a linear subspace of* Σ^n*.*

The **rate** of a code $\mathcal{C}$ is $R = \frac{\log|\mathcal{C}|}{n}$, and its **relative distance** is $\delta = (\min_{c_1\neq c_2} \Delta(c_1, c_2))/n$. We denote the encoding algorithm of a code $\mathcal{C}$ by E.

In our construction, we make use of *locally-decodable codes* (LDCs), which allow the decoder to recover a single symbol of the message by reading only a small number of symbols from the (corrupted) codeword.

Definition 4.9. *A code* $\mathcal{C} \subseteq \Sigma^n$ *is called a* $(\mathsf{q}, \delta, \epsilon)$**-locally-decodable code** *if for all* $x \in \Sigma^k$*, all* $i^\star \in [k]$*, and all* $Y \in \Sigma^n$ *such that* $\Delta(Y, \mathsf{E}(x)) < \delta$*, there exists an efficient decoder* $(D_{\mathsf{q}}, D_{\mathsf{d}})$ *such that*

$$\Pr\left[D_{\mathsf{d}}(\mathsf{st}_q, q, r) \neq x_{i^\star} \;\middle|\; \begin{array}{l}(\mathsf{st}_q, q) \leftarrow D_{\mathsf{q}}(i^\star)\\ r := Y_q\end{array}\right] \leq \epsilon \quad .$$

The query complexity q *is the size of the query vector* q *.*

Definition 4.10 (Smooth LDC [KT00])**.** *An LDC* $\mathcal{C}$ *is said to be* c**-smooth** *if for all* $i^\star \in [k]$ *and* $j \in [n]$*,*

$$\Pr\left[j \in q \;\middle|\; q \leftarrow D_{\mathsf{q}}(i^\star)\right] \leq \frac{c}{n} \quad .$$

If the smoothness parameter c equals the query complexity q, we just say that the LDC is smooth.

Theorem 4.11 ([KT00, Theorem 1])**.** *Every* $(\mathsf{q}, \delta, \epsilon)$*-locally-decodable code is* q/δ*-smooth.*

Locally-decodable codes can successfully decode even if an adversary is allowed to corrupt a δ-fraction of the underlying codeword. In our setting, we need a slightly stronger object, which guarantees correct decoding even when an adversary can adaptively corrupt a δ-fraction *of each response*. This motivates Definition 4.12, which allows the local decoding algorithm to tolerate *adaptive* corruptions. At a high level, a code $\mathcal{C}$ is a subcode LDC if there exists another code $\mathcal{C}_{\mathsf{sub}} \in \Sigma^{\mathsf{q}}$ such that the local reconstruction algorithm D_{d} for $\mathcal{C}$ is simply the decoder for $\mathcal{C}_{\mathsf{sub}}$. We formalize this notion next.

Definition 4.12 (subcode LDCs)**.** *Let $\mathcal{C} \subseteq \Sigma^n$ be a $(\mathsf{q}, \delta, \epsilon)$-LDC with message length k. Then $\mathcal{C}$ is a γ-**subcode LDC** if there exists a code $\mathcal{C}_{\mathsf{sub}} \subseteq \Sigma^{\mathsf{q}}$ with minimum distance γ, such that for all $i^\star \in [n]$, for all $F \in \mathcal{C}$, if $q \leftarrow D_{\mathsf{q}}(i^\star)$, then $F_q \in \mathcal{C}_{\mathsf{sub}}$.*

The canonical example of a subcode LDC is the Reed–Muller code: it is smooth, and the subcode is the Reed–Solomon code. (Other examples include multiplicity codes [KSY14] and lifted codes [Guo16])

Definition 4.13. *The Reed–Muller code $\mathrm{RM}[\mathbb{F}_p, d, m]$ over a prime field $\mathbb{F}_p$ is the set of evaluations of m-variate polynomials of* **total** *degree at most d over $\mathbb{F}_p^m$:*

$$\mathrm{RM}[\mathbb{F}_p, d, m] = \left\{ [f(x)]_{x \in \mathbb{F}_p^m} \;\middle|\; f \in \mathbb{F}_p^{\leq d}[X_1, \ldots, X_m] \right\} \quad .$$

4.4.1 Proximity testing

Let $\mathcal{O}$ be an oracle representing a function from $\mathcal{D} \to \mathcal{R}$. Then a proximity tester with respect to a certain set of functions $\mathcal{F}$ is an algorithm that makes few queries to $\mathcal{O}$ and determines whether $\mathcal{O}$ is close to $\mathcal{F}$. Formally:

Definition 4.14. *A (δ, ϵ)-**proximity tester** for a set of functions $\mathcal{F}$ is a pair of algorithms $\mathsf{PT} = (\mathsf{PT}.Q, \mathsf{PT}.D)$ satisfying the following syntax and properties.*

Syntax. *The randomized query algorithm $\mathsf{PT}.Q$ samples a query set $q \subset \mathcal{D}$ such that $|q| = t$. Let $\mathcal{O}$ be the oracle being tested, and $\mathcal{O}(q)$ the responses of $\mathcal{O}$ to the queries in q. The decision algorithm $\mathsf{PT}.D$ takes as input the queries q and the oracle responses $\mathcal{O}(q)$, and then accepts or rejects.*

- **Completeness***: if $\mathcal{O} \in \mathcal{F}$ then*

$$\Pr\left[\mathsf{PT}.D(q, \mathcal{O}(q)) = 1 \;\middle|\; q \leftarrow \mathsf{PT}.Q(1^\lambda) \right] = 1.$$

- **Soundness***: if $\mathcal{O}$ is at least δ-far from $\mathcal{F}$, i.e., for all $f \in \mathcal{F}$, $\mathcal{O}(x) \neq f(x)$ for at least a δ-fraction of the domain $\mathcal{D}$, then*

$$\Pr\left[\mathsf{PT}.D(q, \mathcal{O}(q)) = 0 \;\middle|\; q \leftarrow \mathsf{PT}.Q(1^\lambda) \right] > \epsilon.$$

We say that the query complexity of PT is $\mathsf{q}(\mathsf{PT})$.

Theorem 4.15 (proximity test for RM codes [JPRZ09])**.** *For some prime p let $\mathcal{F}$ be the set of polynomials of* **total** *degree d from $\mathbb{F}_p^m$ to $\mathbb{F}_p$ (i.e. Reed–Muller codewords). Then, there exists a proximity tester PT for $\mathcal{F}$ with query complexity*

$$O\left(p^{\left\lceil 2\frac{d+1}{p-1} \right\rceil} \right) \quad .$$

It is straightforward to see that we can sequentially repeat the test above $O(\lambda)$ times to obtain a tester that rejects any $\mathcal{O}$ of distance greater than $\mu = \Omega(1)$ with probability $1 - \mathrm{negl}(\lambda)$, for any $\mu > 0$.

Corollary 4.16. *For all $\delta = \Omega(1)$ there exists $(\delta, \mathrm{negl}(\lambda))$-low-degree tester with query complexity*

$$O\left(\lambda \cdot p^{\left\lceil 2\frac{d+1}{p-1} \right\rceil} \right) \quad .$$

5 Constructing mPIR from subcode LDCs

We now describe a methodology for compiling PIR to mPIR using LDCs.

Construction overview. At a high level, our compiler follows the outline described in Section 2, but replaces the Reed–Muller code with an arbitrary "smooth subcode LDC" $\mathcal{C}$ (Definitions 4.10 and 4.12). We briefly recall the outline here.

In a preprocessing phase, the server encodes the database with $\mathcal{C}$ and publishes a vector commitment to the encoding. Then, in the online phase, given query index $i^\star$, the client first invokes a special "consistent" local decoder for $\mathcal{C}$ (described below) to generate a query set, and makes a batch of parallel PIR queries for each query in the set. In addition to the queries, the client checks that the server committed to a string close to an LDC codeword by performing a proximity test. After receiving the server's responses to these queries, the client first checks if the proximity test accepts, and if it fails, the client aborts. If the proximity test accepts, the client decodes the server's responses according to the consistent decoding procedure.

We now expand on this outline by describing the transformation in detail in Fig. 1.

We devote the rest of this section to proving the following theorem.

Theorem 5.1. *Given the following ingredients:*
- *Single-server PIR scheme* PIR.
- *Vector commitment* VC.
- *Smooth subcode LDC* $\mathcal{C} \subseteq \Sigma^n$ *with encoder* E *and query complexity* q.
- *Proximity test* PT *for* $\mathcal{C}$ *with distance* $\delta_{\mathrm{PT}} < \min\{\mathrm{dist}(\mathcal{C}), \mathrm{dist}(\mathcal{C}_{\mathsf{sub}})\}/3$ *and error* $\mathrm{negl}(\lambda)$.

Then the construction in Fig. 1 is a maliciously-secure PIR (mPIR) scheme, with query complexity $\lambda \cdot \mathsf{q} \cdot c\left(\frac{k}{R}\right) + p_{\mathrm{PT}}(n)$*, where* R *is the rate of* $\mathcal{C}$*,* $c(n)$ *is the query complexity of the underlying (semi-honest) PIR protocol,* PIR*, and* $p_{\mathrm{PT}}(n)$ *is the communication complexity of the proximity test.*

Remark 5.2 (Efficiency)**.** The mPIR protocol guaranteed by Theorem 5.1 makes $O(\lambda \cdot \mathsf{q})$ semi-honest PIR queries. Although this overhead is rather large, in the following section (Section 6) we will show how to reduce this overhead with a novel "lossy" batching technique.

To prove Theorem 5.1 we must show that the mPIR scheme satisfies correctness, privacy, and coherence. Because correctness follows in a straightforward manner from the correctness of the ingredients, we focus on proving privacy (Lemma 5.3) and coherence (Lemma 5.5).

Lemma 5.3 (Privacy)**.** *The mPIR protocol outlined in Fig. 1 satisfies privacy.*

Proof. To show this, we describe a simple simulator $\mathcal{S}$.

The simulator simply chooses a random index $i^\star_{\mathcal{S}} \in [k]$ and runs the honest protocol with respect to this index $i^\star_{\mathcal{S}}$. So $\mathcal{S}.\mathsf{Setup} = \mathsf{Setup}$, $\mathcal{S}.\mathsf{Query}(\mathsf{cpp}, \mathsf{pst}) = \mathsf{Query}(\mathsf{cpp}, \mathsf{pst}, i^\star_{\mathcal{S}})$ and $\mathcal{S}.\mathsf{Decode}(\mathsf{st}_{\mathcal{S}}, \mathsf{resp}) = \mathsf{Decode}(\mathsf{st}_{\mathcal{S}}, \mathsf{resp})$.

We are now left to show that the distribution of responses created by interacting with this simulator is computationally indistinguishable from the distribution of responses created by the adversary's interaction with the real protocol (for an unknown index $i^\star$). Note that (in the unlikely event) that the simulator's guess $i^\star_{\mathcal{S}} = i^\star$, the simulation would be perfect. So in effect, we will show that the simulation will be good even when $i^\star_{\mathcal{S}} \neq i^\star$.

First, consider a modified simulator, $\mathcal{S}'$, that maintains the same setup ($\mathcal{S}'.\mathsf{Setup} = \mathcal{S}.\mathsf{Setup} = \mathsf{Setup}$), and query procedure ($\mathcal{S}'.\mathsf{Query} = \mathcal{S}.\mathsf{Query}$) but changes the decoding procedure to ignore failures outside of the λ test queries. More specifically, $\mathcal{S}'.\mathsf{Decode}$ will ignore failures in Step 4 of D_{d}, and only look at the λ

- Let PIR be a single-server PIR scheme.
- Let VC be a vector commitment scheme.
- Let $(D'_{\mathsf{q}}, D'_{\mathsf{d}})$ be the decoder for a smooth subcode LDC $\mathcal{C} \subseteq \Sigma^n$ with encoder E and query complexity q'.
- Let $\mathsf{PT} = (\mathsf{PT}.Q, \mathsf{PT}.D)$ be a proximity test for $\mathcal{C}$ with distance $\delta_{\mathrm{PT}} < \min\{\mathrm{dist}(\mathcal{C}), \mathrm{dist}(\mathcal{C}_{\mathsf{sub}})\}/3$ and error $\mathrm{negl}(\lambda)$.

Below we construct a "consistent" decoder $(D_{\mathsf{q}}, D_{\mathsf{d}})$ for $\mathcal{C}$ with query complexity $\mathsf{q} := \mathsf{q}' \cdot \lambda$. Using this decoder, we construct our mPIR protocol as follows.

- $\mathsf{mPIR.Setup}(1^\lambda)$:
  1. Compute vector commitment keys: $(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{VC.Setup}(1^\lambda)$.
  2. For each $i \in [\mathsf{q}]$, compute PIR client and server public parameters: $(\mathsf{cpp}_i, \mathsf{spp}_i) \leftarrow \mathsf{PIR.Setup}(1^\lambda)$
  3. Output mPIR client and server public parameters: $\mathsf{cpp} := (\mathsf{vk}, [\mathsf{cpp}_i]_{i\in[\mathsf{q}]})$ and $\mathsf{spp} := (\mathsf{ck}, [\mathsf{spp}_i]_{i\in[\mathsf{q}]})$.
- $\mathsf{mPIR.Preprocess}(\mathsf{spp}, \mathsf{db} = (\mathsf{db}_1, \ldots, \mathsf{db}_k) \in \Sigma^k)$:
  1. Parse $\mathsf{spp} = (\mathsf{ck}, [\mathsf{spp}_i]_{i\in[\mathsf{q}]})$.
  2. Encode the database using $\mathcal{C}$: $E := \mathsf{E}(\mathsf{db})$.
  3. Compute commitment to E: $(\mathsf{cm}, \mathfrak{d}) \leftarrow \mathsf{VC.Commit}(\mathsf{spp}, E)$.
  4. For each $i \in [n]$, compute opening proof for E_i: $\pi_i \leftarrow \mathsf{VC.Prove}(\mathsf{spp}, i, \mathsf{cm}, \mathfrak{d}, E_i)$
  5. Construct E' such that for each $i \in [n]$, $E'_i = E_i \;||\; \pi_i$
  6. For each $i \in [\mathsf{q}]$, compute preprocessing material for PIR: $(\mathsf{pst}_i, \mathsf{sst}_i) \leftarrow \mathsf{PIR.Preprocess}(\mathsf{spp}_i, E')$.
  7. Output $\mathsf{pst} = (\mathsf{cm}, [\mathsf{pst}_i]_{i\in[\mathsf{q}]})$ and $\mathsf{sst} = (E', [\mathsf{sst}_i]_{i\in[\mathsf{q}]})$.
- $\mathsf{mPIR.Query}(\mathsf{cpp}, \mathsf{pst}, i^\star)$:
  1. Parse $\mathsf{cpp} = (\mathsf{vk}, [\mathsf{cpp}_i]_{i\in[\mathsf{q}]})$ and $\mathsf{pst} = (\mathsf{cm}, [\mathsf{pst}_i]_{i\in[\mathsf{q}]})$.
  2. Compute the consistent decoding query vector: $(\mathsf{st}_q, q) \leftarrow D_{\mathsf{q}}(i^\star)$. (see bottom of figure for definition.)
  3. For each $i \in [\mathsf{q}]$, compute a PIR query for q_i: $(\mathsf{query}_i, \mathsf{cst}_i) \leftarrow \mathsf{PIR.Query}(\mathsf{cpp}_i, \mathsf{pst}_i, q_i)$.
  4. Invoke the proximity test to obtain proximity test query locations: $h \subset [n] \leftarrow \mathsf{PT.Query}(1^\lambda)$.
  5. Output $\mathsf{query} := ([\mathsf{query}_i]_{i\in[\mathsf{q}]}, h)$ and $\mathsf{cst} := (\mathsf{vk}, \mathsf{cm}, q, h, \mathsf{st}_q, [\mathsf{cst}_i]_{i\in[\mathsf{q}]})$.
- $\mathsf{mPIR.Respond}(\mathsf{sst}, \mathsf{query})$:
  1. Parse $\mathsf{sst} = (E', [\mathsf{sst}_i]_{i\in[\mathsf{q}]})$, and $\mathsf{query} = ([\mathsf{query}_i]_{i\in[\mathsf{q}]}, h)$.
  2. For each $i \in [\mathsf{q}]$, compute PIR response: $\mathsf{resp}_i \leftarrow \mathsf{PIR.Respond}(\mathsf{sst}_i, \mathsf{query}_i)$.
  3. Output mPIR response $\mathsf{resp} := ([\mathsf{resp}_i]_{i\in[\mathsf{q}]}, E'_h)$.
- $\mathsf{mPIR.Decode}(\mathsf{cst}, \mathsf{resp})$:
  1. Parse $\mathsf{cst} = (\mathsf{vk}, \mathsf{cm}, q, h, \mathsf{st}_q, \{\mathsf{cst}_{,i}\}_{i\in[\mathsf{q}]})$ and $\mathsf{resp} = ([\mathsf{resp}_i]_{i\in[\mathsf{q}]}, E'_h)$.
  2. Check that proximity test responses are correct: for each $i \in [|h|]$, $\mathsf{VC.Verify}(\mathsf{vk}, \mathsf{cm}, h_i, v_i, \pi_i) = 1$.
  3. Check that the proximity test passes: $\mathsf{PT}.D(H, \{v_i\}_{i\in H}) = 1$.
  4. For each $i \in [\mathsf{q}]$:
     (a) Compute PIR response: $(v_i \;||\; \pi_i) \leftarrow \mathsf{PIR.Decode}(\mathsf{cst}_i, \mathsf{resp}_i)$.
     (b) If $\mathsf{VC.Verify}(\mathsf{vk}, \mathsf{cm}, q_i, v_i, \pi_i) = 1$, set $r_i := v_i$; else set $r_i := \bot$.
  5. If any of the foregoing checks fail, output $\bot$.
  6. Output $\mathsf{ans} \leftarrow D_{\mathsf{d}}(\mathsf{st}_q, (r_1, \ldots, r_{\mathsf{q}}))$. (see bottom of figure for definition.)

Query procedure $D_{\mathsf{q}}(i^\star)$**:**
1. For each $j \in [\lambda]$, sample queries for $\mathcal{C}$: $(s_j, \mathsf{st}_j) \leftarrow D'_{\mathsf{q}}(i^\star)$.
2. Set query vector $q := \cup_{j\in[\lambda]} s_j$.
3. Output q and state $\mathsf{st}_q := (\mathsf{st}_1, \ldots, \mathsf{st}_\lambda)$.

Decoding procedure $D_{\mathsf{d}}(\mathsf{st}_q, z)$**:**
1. Parse $\mathsf{st}_q = (\mathsf{st}_1, \ldots, \mathsf{st}_\lambda)$ and $z = (z_1, \ldots, z_\lambda)$.
2. For each $j \in [\lambda]$, sample a random $t_j \leftarrow z_j$.
3. If any $t_j = \bot$, output $\bot$.
4. Else, output $\mathrm{majority}_{j\in[\lambda]}(D'_{\mathsf{d}}(z_j, \mathsf{st}_j))$.

Figure 1: mPIR from a subcode LDC and black box PIR.

responses corresponding to the queries t_j (defined in Step 2 of D_{d}), and abort if any of these responses fail their vector commitment proof. Note that $\mathcal{S}'$ will still abort as usual if any of the responses fail their proximity test, but $\mathcal{S}'$ will *not* output $\perp$ if the majority of D'_{d} fail (Step 4)

Now, we show that the distributions induced by $\mathcal{S}$ and $\mathcal{S}'$ are *statistically* indistinguishable.

Remark 5.4. The claim that $\mathcal{S}$ and $\mathcal{S}'$ produce statistically indistinguishable transcripts is an information theoretic-claim that will rely on (1) The smoothness of the LDC, and (2) the subcode error-correction ability of the LDC. This also resolves the issue of "non-signaling" adversaries that appears in [DLNNR04; KRR14]. We will show that even an adversary who gets to see all the queries *in the clear* cannot distinguish the distributions induced by $\mathcal{S}$ and $\mathcal{S}'$, so this step of the proof does not rely on the privacy of the PIR protocol *at all*. PIR privacy will show up in the next step (showing that the output of $\mathcal{S}'$ is indistinguishable from that of a second simulator $\mathcal{S}''$).

Note that distributions received by $\mathcal{A}_0$ and $\mathcal{A}_1$ are the same when interacting with $\mathcal{S}$ and $\mathcal{S}'$, so the only difference is when $\mathcal{A}_2$ receives $\mathsf{ans} = \perp$. By the definition of $\mathcal{S}$ and $\mathcal{S}'$, if $\mathcal{S}'.\mathsf{Decode}$ outputs $\perp$, then $\mathcal{S}.\mathsf{Decode}$ will output $\perp$ as well. On the other hand, there are cases where $\mathcal{S}.\mathsf{Decode}$ outputs $\perp$, but $\mathcal{S}'.\mathsf{Decode}$ does *not*, and so we must show that these happen with only negligible probability. To see this, note that this will only occur when the majority of the λ LDC queries fail to decode. By assumption, the code $\mathcal{C}$ is a γ-subcode LDC, so the adversary can only cause this type of decoding failure if it corrupts at least $\gamma \cdot \mathsf{q}$ of the responses for at least $\frac{\lambda}{2}$ of the LDC queries. So, suppose the adversary corrupts at least this many responses. Since $\mathcal{S}'$ will sample one response uniformly at random to check, the probability that the adversary can corrupt this many responses but $\mathcal{S}'$ does *not* abort, is at most

$$(1-\gamma)^{\frac{\lambda}{2}} = \mathrm{negl}(\lambda) \tag{1}$$

Note that this even holds for an information-theoretic adversary, because the decoder's "test" queries are not even defined until after the adversary has made its corruptions. This shows that the distributions induced on the adversary's output x are statistically close when the adversary is interacting with $\mathcal{S}$ or $\mathcal{S}'$.

Next, we further modify the simulator into a new simulator $\mathcal{S}''$ that maintains the same setup and decoding procedures as $\mathcal{S}'$ (i.e., $\mathcal{S}''.\mathsf{Setup} = \mathcal{S}'.\mathsf{Setup} = \mathcal{S}.\mathsf{Setup} = \mathsf{Setup}$ and $\mathcal{S}''.\mathsf{Decode} = \mathcal{S}'.\mathsf{Decode}$), but changes the query procedure as follows.

- $\mathcal{S}''$ will generate λ queries for $i^\star_{\mathcal{S}}$ using the subcode LDC decoder D'_{q}.
- For each LDC query set query'_i, $\mathcal{S}''$ will choose a random element $t_i \in \mathsf{query}'_i \subset [n]$. Let $D_i := \mathsf{query}'_i \setminus \{t_i\}$.
- $\mathcal{S}''$ will generate PIR queries for each index $\{t_i\}_{i=1}^{\lambda}$.
- Instead of making PIR queries for the remaining locations in $\bigcup_i D_i$, $\mathcal{S}''$ will make PIR queries for $\lambda \cdot (\mathsf{q}-1)$ *random* locations.
- $\mathcal{S}''$ will define query' to be the (shuffled) set of $\lambda \cdot \mathsf{q}$ queries

Now, we need to show that the distribution of the adversary's response is (computationally) indistinguishable when interacting with $\mathcal{S}'$ or $\mathcal{S}''$. To do this, we will rely on multi-query privacy (Definition 4.1) of the underlying semi-honest PIR, as well as the fact that *each* PIR query is generated with respect to an independently sampled setup.

Suppose the mPIR adversary's response, x, is distinguishable when interacting with $\mathcal{S}'$ compared to $\mathcal{S}''$. Then we could ask the (semi-honest) multi-query PIR privacy challenger to provide $\lambda \cdot (\mathsf{q}-1)$ PIR queries that are either from the distribution induced by $\mathcal{S}'$ (i.e., consistent with the query pattern of λ LDC queries) or uniformly distributed within $[n]$. In the first case, the distribution will be consistent with $\mathcal{S}'$, and the second case, the distribution will be consistent with $\mathcal{S}''$. Now, we have to be careful here, because the adversary's response, x, depends not just on $\mathcal{S}''.\mathsf{Query}$, but also on $\mathcal{S}''.\mathsf{Decode}$, and if we obtain PIR queries from the

multi-query PIR challenger, we can no longer decode the responses to those queries, which might change the behavior of the decoder (and hence also the behavior of $\mathcal{A}_2$). This is not a problem, however, because $\mathcal{S}''.\mathsf{Decode}$ (which is the same as $\mathcal{S}'.\mathsf{Decode}$) ignores the responses for all but the λ test queries, and so its response will be the same whether or not it has the decoding keys for the remaining queries.

At this point, we have shown that the adversary's output is (computationally) indistinguishable whether it is interacting with $\mathcal{S}$ or $\mathcal{S}''$. The final step here is to notice that since the LDC is smooth, when $\mathcal{S}''$ chooses exactly one "test" point from each query query'_i, the collection of these test points is uniform, and independent of the query index, $i^\star_{\mathcal{S}}$ chosen by the simulator. So the distribution of the adversary's output, when interacting with $\mathcal{S}''$, is the same no matter what $i^\star_{\mathcal{S}}$ was chosen.

Finally, we observe that when the simulator's guess is correct ($i^\star_{\mathcal{S}} = i^\star$), then the simulator perfectly mimics the real execution of the protocol. Since have shown that the adversary's output is (computationally) independent of the simulator's guess $i^\star_{\mathcal{S}}$, we conclude that the adversary's output is computationally indistinguishable from the from the output of the real execution of the protocol for every guess $i^\star_{\mathcal{S}}$. □

Lemma 5.5 (Coherence)**.** *The mPIR protocol outlined in Fig. 1 satisfies coherence.*

Proof. By Lemma 4.3, it is sufficient to prove that the protocol satisfies Definition 4.2.

Note that $\mathsf{mPIR.Setup}$ outputs the vector commitment keys $(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{VC.Setup}(1^\lambda)$. Since the vector commitment $(\mathsf{cm}, \mathfrak{d}) \leftarrow \mathsf{VC.Commit}(\mathsf{spp}, E)$ is created during preprocessing, so cm is part of the public state pst that is created by $\mathcal{A}_0(\mathsf{spp})$ in Definition 4.2.

Intuitively, this commitment, cm, together with the proofs of proximity should bind the adversary to some database db, and we show this formally below.

First, note that the coherence property does not impose any restrictions on when the mPIR decoder outputs $\perp$, so it is sufficient to only consider the case when the mPIR decoder does *not* output $\perp$. Thus, we can consider the case when *all* the $\lambda \cdot \mathsf{q}$ proximity tests pass.

This means that (with all-but-negligible probability), we may assume that the commitment cm in pst commits to a vector $F \in \Sigma^n$ that is at most δ_{PT}-far from $\mathcal{C}$. Together with the binding property of the vector commitment scheme, we may assume that (with all-but-negligible probability) every adversarial response that is accepted by the decoder is consistent with this vector F.

Now, we can define db to be the unique message such that $\Delta(F, E(\mathsf{db})) < \delta_{\mathrm{PT}}$. To prove coherence, we will show that on query $i^\star$, if the mPIR decoder does not output $\perp$, then (with all-but-negligible probability) the mPIR decoder must output $\mathsf{db}_{i^\star}$.

By assumption $\delta_{\mathrm{PT}} < \mathrm{dist}(\mathcal{C})$, so *if the adversary answered all queries with respect to* F, the mPIR decoder would recover $\mathsf{db}_{i^\star}$ with probability at least $(1 - \epsilon)$ for each of the λ queries. A Chernoff bound would then show that the probability that a majority of the λ queries decoded to $\mathsf{db}_{i^\star}$ would be negligible in λ. We have to be slightly more careful, however, since the adversary can also selectively erase any number of responses—a power that is beyond a traditional LDC adversary.

So, we first derive a bound on the number of erasures the adversary can introduce before the decoder outputs $\perp$. Suppose the adversary modifies an η-fraction of the q responses for a given LDC query (i.e., makes modifications so that the decommitments fail). Then, since the decoder has chosen one uniformly random "test" point, and will reject if this decommitment fails, the decoder will output $\perp$ with probability at least η.

So if the adversary corrupts an η-fraction of at least $\lambda/10$ of the λ LDC queries, the decoder will abort with probability at least $1 - (1 - \eta)^{\lambda/10}$, which is negligibly close to 1. Thus we may assume that (if the decoder doesn't abort), then $9/10$ths of the λ LDC queries have at most $\eta \cdot \mathsf{q}$ corruptions (relative to F). We will call these "good" queries.

Since F was committed to during pre-processing (when the adversary published cm as part of pst), and since the LDC is assumed to be smooth, the chance that any given index queried by the decoder hits an index i, where $F_i \neq E(\mathsf{db})_i$, is δ_{PT} (since $\mathrm{dist}(F, E(\mathsf{db})) < \delta_{\mathrm{PT}}$).

Consider one of the "good" LDC queries (i.e., the collection of q responses), where the adversary introduces at most $\eta \cdot \mathsf{q}$ erasures. Then the total number of corruptions in this query (relative to the "true" codeword $E(\mathsf{db})$) is $\eta \cdot \mathsf{q} + X$, where X is the number of points where $F_i \neq E(\mathsf{db})_i$ hit by the LDC query. Since the LDC is smooth, the probability that any corrupted point is hit is δ_{PT}, and these are negatively dependent, so we can apply a Chernoff Bound [DP09, Theorem 4.3]. A basic Chernoff bound [DP09, Theorem 1.1] shows that

$$\Pr\left[\eta \cdot \mathsf{q} + X > \mathrm{dist}(\mathcal{C}_{\mathsf{sub}}) \cdot \mathsf{q}\right] < e^{-2(\mathrm{dist}(\mathcal{C}_{\mathsf{sub}}) - \eta - \delta_{\mathrm{PT}})^2 \cdot \mathsf{q}} \tag{2}$$

So the probability that a given "good" query fails to decode $\mathsf{db}_{i^\star}$ is given by Eq. (2). For concreteness, if we choose $\eta < \mathrm{dist}(\mathcal{C}_{\mathsf{sub}})/3$, then $\mathrm{dist}(\mathcal{C}_{\mathsf{sub}}) - \eta - \delta_{\mathrm{PT}} > \mathrm{dist}(\mathcal{C}_{\mathsf{sub}})/3 > 0$, and the right-hand side of Eq. (2) will go to zero exponentially quickly in q. By assumption $\mathsf{q} > \lambda$, so the right-hand side of Eq. (2) will go to zero exponentially quickly in λ.

The mPIR decoder will succeed when a majority of the λ queries decode $\mathsf{db}_{i^\star}$, since (with all-but-negligible probability) at least $9/10 \cdot \lambda$ of the LDC queries are "good," the LDC decoder will only fail if $4/9$ of the "good" queries fail to decode (note $4/9$ of the "good" queries plus $1/10$ of the total queries is $1/2$ of the total queries). Applying another Chernoff bound, we have

$$\Pr\left[\text{ } 4/9 \text{ of the "good" queries fail }\right] < \Pr\left[Y > E[Y] + t\right] < \exp\left(-\frac{2t^2}{\frac{9}{10}\lambda}\right) \tag{3}$$

In this case $E[Y] < e^{-2(\mathrm{dist}(\mathcal{C}_{\mathsf{sub}}) - \eta - \delta_{\mathrm{PT}})^2 \cdot \mathsf{q}} \cdot \frac{9}{10}\lambda$, so $t = \left(\frac{4}{10} - e^{-2(\mathrm{dist}(\mathcal{C}_{\mathsf{sub}}) - \eta - \delta_{\mathrm{PT}})^2 \cdot \mathsf{q}}\right) \cdot \frac{9}{10}\lambda$ Thus

$$\Pr\left[\text{ } 4/9 \text{ of the "good" queries fail }\right] < \exp\left(-2\left(\frac{4}{10} - e^{-2(\mathrm{dist}(\mathcal{C}_{\mathsf{sub}}) - \eta - \delta_{\mathrm{PT}})^2 \cdot \mathsf{q}}\right)^2 \cdot \frac{9}{10}\lambda\right) \tag{4}$$

which is negligible in λ. □

Remark 5.6 (Multi-server mPIR)**.** For brevity, this section is written in the language of compiling a single-server PIR scheme to a single-server mPIR. Our compiler also works to compile a multi-server PIR to multi-server mPIR. In this case, we can remove the proximity test, and instead have each server execute $\mathsf{Preprocess}$ with every query and send the resulting commitment cm. The client will abort if it receives different commitments from the servers. Since at least one commitment is honest, this ensures that if the client doesn't abort, it is holding an honestly generated commitment.

6 Boosting the mPIR with lossy batching

In the previous section, we constructed a PIR to mPIR compiler with $\Theta(\mathsf{q})$ server computation overhead, where q is the query complexity of the "consistent" decoding procedure of Fig. 1. The most efficient instantiation of this procedure has $\mathsf{q} = O(n^\epsilon)$, and thus our compiler has $O(n^\epsilon)$ overhead, yielding an mPIR with superlinear $O(n^{1+\epsilon})$ computation. In this section we show how to lower the computational cost to $O(n)$ using a "lossy batching" trick.

The key idea, similar to batch codes [IKOS04], is to distribute the database into m buckets, and then replace q queries into the codeword of length n by m queries into these (small) buckets. The problem with using batch codes directly, is that in our proof of security (Lemma 5.3), we crucially use the fact that the test queries are distributed uniformly and independently of the query $i^\star$. With a batch code, the *entire* set of $\lambda \cdot \mathsf{q}$ queries are mapped to m buckets, and even if the λ test queries are distributed uniformly in $[n]$ (independent of the other indices being queried), in a batch code, the locations of the test queries in each bucket *may become dependent on the other indices being queried.*

To overcome this, we outline a simple, "lossy" batching procedure that preserves the independence of the test queries. We give our main construction in Figs. 2 and 3, and prove that it is an mPIR with the required efficiency guarantees result in Theorem 2.

Theorem 2 (PIR $\Rightarrow$ mPIR)**.** *Given*
– *A smooth "subcode LDC" (see Definition 4.12) with rate $R := \frac{k}{n}$ and query complexity $q(k)$ and $(O(1), \mathrm{negl}(\lambda))$ proximity test with query complexity $p_{\mathrm{PT}}(n)$.*
– *A PIR scheme with communication complexity $c(n)$ and server computation $s(n)$.*
– *A vector commitment scheme with size $s(n)$ and proof size $p_{\mathrm{VC}}(n)$.*
Then, there exists a maliciously-secure PIR (mPIR) scheme with
- *digest size is $s(n) = s\left(\frac{k}{R}\right)$;*
- *communication cost is $c\left(\frac{k}{R}\right) \cdot q(k) + p_{\mathrm{PT}}(n)$, i.e. the communication cost to respond to $q(k)$ PIR queries into a codeword of size $\frac{k}{R}$, with alphabet size $|\Sigma| + p_{\mathrm{VC}}(n)$ (since each entry of size $|\Sigma|$ is augmented with a proof of size $p_{\mathrm{VC}}(n)$.*
- *server preprocessing time is the sum of the times to encode the database, to commit to the resulting codeword of size n, to produce proofs for each of the n elements of the codeword;*
- *server online computation is the time to respond to $q(k)$ PIR queries on a database of size n;*
- *client online computation is the time to verify $q(k)$ opening proofs and decode the response.*

Instantiating the claim above, we have that

Corollary 6.1. *Assuming there exists a PIR with $O(n^\epsilon)$ communication and $O(n)$ computation, there exists an mPIR with the same complexity.*

The "PIR recursion" technique [KO97] provides a construction of (semi-honest) PIR from any additively homomorphic encryption scheme. Applying Corollary 6.1 to one of these constructions, we obtain the following result.

Corollary 6.2. *There exists an $O(n^\epsilon)$ communication, $O(n)$ computation mPIR from LWE and the Decisional Composite Residuosity (DCR).*

Construction overview. Our "batched" mPIR compiler is outlined in detail in Fig. 1 (with modifications highlighted in blue). The batched mPIR compiler works as follows. In preprocessing, after computing the encoded database $E' = E'_1, \ldots, E'_n$, the server maps each codeword symbol to a (pseudo)random bucket as

We describe an improvement to the mPIR protocol of Fig. 1, lowering the server's computational complexity by $O(\mathsf{q}(\mathcal{C}))$. Changes are in blue.

- Let PIR be a single-server keyword PIR scheme.
- Let VC be a vector commitment scheme.
- Let $(D'_{\mathsf{q}}, D'_{\mathsf{d}})$ be the decoder for a smooth subcode LDC $\mathcal{C} \subseteq \Sigma^n$ with encoder E and query complexity q.
- Let $\mathsf{PT} = (\mathsf{PT}.Q, \mathsf{PT}.D)$ be a proximity test for $\mathcal{C}$ with distance $\delta_{\mathrm{PT}} < \min\{\mathrm{dist}(\mathcal{C}), \mathrm{dist}(\mathcal{C}_{\mathsf{sub}})\}/3$ and error $\mathrm{negl}(\lambda)$.
- Let $\{\mathsf{PRF}_s\}$ be a pseudorandom function (mapping from $[n]$ to $[m]$) family indexed by seed $s \in \{0,1\}^\lambda$.

This construction relies on the modified "consistent" decoder $(D^{\mathsf{b}}_{\mathsf{q}}, D^{\mathsf{b}}_{\mathsf{d}})$ described in Fig. 3.

- $\mathsf{mPIR.Setup}(1^\lambda)$:
  1. Sample PRF seed $\mathsf{s} \leftarrow \mathsf{PRF.Keygen}(1^\lambda)$
  2. Compute vector commitment keys: $(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{VC.Setup}(1^\lambda)$.
  3. For each $i \in [\mathsf{q}]$, compute PIR client and server public parameters: $(\mathsf{cpp}_i, \mathsf{spp}_i) \leftarrow \mathsf{PIR.Setup}(1^\lambda)$
  4. Output mPIR client and server public parameters: $\mathsf{cpp} := (\mathsf{vk}, [\mathsf{cpp}_i]_{i\in[m]})$ and $\mathsf{spp} := (\mathsf{ck}, [\mathsf{spp}_i]_{i\in[m]})$.
- $\mathsf{mPIR.Preprocess}(\mathsf{spp}, \mathsf{db} = (\mathsf{db}_1, \ldots, \mathsf{db}_k) \in \Sigma^k)$:
  1. Parse $\mathsf{spp} = (\mathsf{ck}, [\mathsf{spp}_i]_{i\in[\mathsf{q}]})$.
  2. Encode the database using $\mathcal{C}$: $E := \mathsf{E}(\mathsf{db})$.
  3. Compute commitment to E: $(\mathsf{cm}, \mathfrak{d}) \leftarrow \mathsf{VC.Commit}(\mathsf{spp}, E)$.
  4. For each $i \in [n]$, compute opening proof for E_i: $\pi_i \leftarrow \mathsf{VC.Prove}(\mathsf{spp}, i, \mathsf{cm}, \mathfrak{d}, E_i)$
  5. Construct E' such that for each $i \in [n]$, $E'_i = E_i \;||\; \pi_i$
  6. For each $j \in [m]$, construct the j-th bucket: $B_j := \{E''_i \mid \mathsf{PRF}_s(i) = j\}$
  7. For each $j \in [m]$, compute preprocessing material for PIR on the j-th bucket: $(\mathsf{pst}_j, \mathsf{sst}_j) \leftarrow \mathsf{PIR.Preprocess}(\mathsf{spp}_j, B_j)$.
  8. Output $\mathsf{pst} = (\mathsf{cm}, [\mathsf{pst}_i]_{i\in[m]})$ and $\mathsf{sst} = (E', [\mathsf{sst}_i]_{i\in[m]})$.
- $\mathsf{mPIR.Query}(\mathsf{cpp}, \mathsf{pst}, i^\star)$:
  1. Parse $\mathsf{cpp} = (\mathsf{vk}, [\mathsf{cpp}_i]_{i\in[m]})$ and $\mathsf{pst} = (\mathsf{cm}, [\mathsf{pst}_i]_{i\in[m]})$.
  2. Compute the consistent bucketed query vector: $(\mathsf{st}_q, q) \leftarrow D^{\mathsf{b}}_{\mathsf{q}}(i^\star)$ where $q \in [n]^m$. (see Fig. 3 for definition.)
  3. For each $i \in [m]$, compute a PIR query for q_i: $(\mathsf{query}_i, \mathsf{cst}_i) \leftarrow \mathsf{PIR.Query}(\mathsf{cpp}_i, \mathsf{pst}_i, q_i)$.
  4. Invoke the proximity test to obtain proximity test query locations: $h \subset [n] \leftarrow \mathsf{PT.Query}(1^\lambda)$.
  5. Output $\mathsf{query} := ([\mathsf{query}_i]_{i\in[m]}, h)$ and $\mathsf{cst} := (\mathsf{vk}, \mathsf{cm}, q, h, \mathsf{st}_q, [\mathsf{cst}_i]_{i\in[m]})$.
- $\mathsf{mPIR.Respond}(\mathsf{sst}, \mathsf{query})$:
  1. Parse $\mathsf{sst} = (E', [\mathsf{sst}_i]_{i\in[m]})$, and $\mathsf{query} = ([\mathsf{query}_i]_{i\in[m]}, h)$.
  2. For each $i \in [m]$, compute PIR response: $\mathsf{resp}_i \leftarrow \mathsf{PIR.Respond}(\mathsf{sst}_i, \mathsf{query}_i)$. Note: sst_i is an encoding of bucket B_i, not the entire database.
  3. Output mPIR response $\mathsf{resp} := ([\mathsf{resp}_i]_{i\in[m]}, E'_h)$.
- $\mathsf{mPIR.Decode}(\mathsf{cst}, \mathsf{resp})$:
  1. Parse $\mathsf{cst} = (\mathsf{vk}, \mathsf{cm}, q, h, \mathsf{st}_q, \{\mathsf{cst}_i\}_{i\in[\mathsf{q}]})$ and $\mathsf{resp} = ([\mathsf{resp}_i]_{i\in[\mathsf{q}]}, E'_h)$.
  2. Check that proximity test responses are correct: for each $i \in [|h|]$, $\mathsf{VC.Verify}(\mathsf{vk}, \mathsf{cm}, h_i, v_i, \pi_i) = 1$.
  3. Check that the proximity test passes: $\mathsf{PT}.D(H, \{v_i\}_{i\in H}) = 1$.
  4. For each $i \in [m]$:
     (a) Compute PIR response: $(v_i \;||\; \pi_i) \leftarrow \mathsf{PIR.Decode}(\mathsf{cst}_i, \mathsf{resp}_i)$.
     (b) If $\mathsf{VC.Verify}(\mathsf{vk}, \mathsf{cm}, q_i, v_i, \pi_i) = 1$, set $r_i := v_i$; else set $r_i := \perp$.
  5. If any of the foregoing checks fail, output $\perp$.
  6. Output $\mathsf{ans} \leftarrow D^{\mathsf{b}}_{\mathsf{d}}(\mathsf{st}_q, (r_1, \ldots, r_m))$.

Figure 2: An $O(\mathsf{q})$ improvement to the computational complexity of the mPIR compiler from Fig. 1 using lossy batching of the queries to the smooth subcode LDC.

Query procedure $D^{\mathsf{b}}_{\mathsf{q}}(i^\star)$:

1. For each $j \in [\lambda]$, sample queries for $\mathcal{C}$: $(s_j, \mathsf{st}_j) \leftarrow D'_{\mathsf{q}}(i^\star)$.
2. For each $j \in [\lambda]$, pick a test element: $t_j \leftarrow s_j$.
3. Set query vector $q := \cup_{j\in[\lambda]} s_j$.
4. Initialize an array, $\mathrm{Bucket} = (\perp)^m$.
5. For each $j \in [\lambda]$, try to "placing" the test element into its bucket at a random order: if $\mathrm{Bucket}_{F_{\mathsf{s}}(t_j)} = \perp$, set $\mathrm{Bucket}_{F_{\mathsf{s}}(t_j)} := t_j$.
6. Try bucketing each of the remaining elements, in a random order: let $\pi \leftarrow S_{\lambda\cdot\mathsf{q}(D'_{\mathsf{q}})}$. For $i \in [\lambda \cdot \mathsf{q}(D'_{\mathsf{q}})]$, if $\mathrm{Bucket}_{F_{\mathsf{s}}(q_{\pi_i})} = \perp$, set $\mathrm{Bucket}_{F_{\mathsf{s}}(q_{\pi_i})} := q_{\pi_i}$.
7. Output q and state $\mathsf{st}_q := ((\mathsf{st}_1, s_1, t_1) \ldots, (\mathsf{st}_\lambda, s_\lambda, t_\lambda), \mathrm{Bucket})$

Decoding procedure $D^{\mathsf{b}}_{\mathsf{d}}(\mathsf{st}_q, r')$:

1. Parse the state st_q as $((\mathsf{st}_1, s_1, t_1) \ldots, (\mathsf{st}_\lambda, s_\lambda, t_\lambda)), \mathrm{Bucket}$.
2. Initialize $(z_i)_j = \perp^*$ for $i \in [\lambda], j \in [\mathsf{q}]$. For $\ell \in [m]$, if $(s_i)_j \stackrel{?}{=} \mathrm{Bucket}_\ell$, $(z_i)_j := r'_\ell$.
3. Concatenate $z := z_1, \ldots z_\lambda$ and output $D_{\mathsf{d}}(\mathsf{st}_q, z)$ (where D_{d} is the decoding procedure in Fig. 1, with $t_1, \ldots, t_\lambda$ obtained from st_q used as test queries).

Figure 3: Bucket algorithm for LDC consistent Query .

follows. Let $F : [n] \to [m]$ be the PRF sampled in part of Setup. Then for all $i \in [m]$, the server lets bucket $B_i := \left\{ E'_j \;\middle|\; F(j) = i \right\}$. At this point, the buckets may be of different sizes, but $\sum_{i\in[m]} |B_i| = n$. The mPIR server treats each bucket, B_i as a separate "database," and executes PIR.Preprocess with respect to B_i as the "database," where PIR is a semi-honest *keyword* PIR.[9]

Given a query index $i^\star$, the client invokes a modified query algorithm $D^{\mathsf{b}}_{\mathsf{q}}(i^\star)$ (described in Fig. 3) and obtains a single query q_i for each bucket $i \in [m]$. For all $i \in [m]$, the client PIR queries the i-th bucket with q_i.The client decodes the server's responses and passes it to (slightly) tweaked decoding algorithm $D^{\mathsf{b}}_{\mathsf{d}}$, outputting its output.

Proof overview. The key feature of our bucketing procedure is that the distribution of test points in buckets is independent of the rest of the query points. We achieve this by placing the test points (randomly) into buckets first, then placing the remaining elements (Fig. 3). This process ensures that the locations of the test points are uniformly distributed in the buckets, and do not depend on the other query locations (Claim 6.6), but it does introduce extra errors, when a later query is "dropped" because it has been assigned to a bucket that already has a query assigned to it. In Lemma 6.4, we show that the batched scheme still satisfies correctness, i.e., the number of queries dropped in this way does *not* prevent decoding (when the server is honest). Note that this makes critical use of the *subcode* property of the LDC. A traditional LDC might fail if we dropped a small number of responses from each query.

Once we can show that test points are uniformly distributed and independent of the query index, $i^\star$, the privacy follows from an argument that is very similar to that of Lemma 5.3.

Efficiency. In the batched scheme, the encoded database (of length n) is redistributed to m, so $\sum_{i\in[m]} |B_i| = n$. The mPIR protocol makes one (semi-honest, keyword) PIR query to each bucket. So, assuming the server complexity of the underlying PIR scheme is (sub-)linear, computational complexity of executing a query in the mPIR protocol will be (sub-)linear as well.

Lemma 6.3 (Errors induced by lossy bucketing)**.** *With all but negligible probability, no more than a* $\frac{2\cdot\lambda\cdot\mathsf{q}}{m}$*-fraction of any given LDC query is dropped by the bucketing procedure (Fig. 3).*

[9]The need for keyword PIR is roughly a technicality: while the client can evaluate F_{s} to learn that the i-th database symbol is stored in B_j, it does not know what is the "index" of i in B_j. To get around this problem, we use keyword PIR which allows the client to query the bucket based on the original label, rather than the location in the bucket. Using keyword PIR instead of standard, semi-honest PIR only introduces a small, constant overhead [PSY23].

Proof. The probability that a given query index is dropped is bounded by $\frac{\lambda \cdot \mathsf{q}}{m}$, because at the time a given query index is placed, no more than $\lambda \cdot \mathsf{q}$ buckets could be occupied by other queries. Since $\mathsf{q} = \omega(\lambda)$, a basic Chernoff bound shows that with all but negligible probability no more than a $\frac{2 \cdot \lambda \cdot \mathsf{q}}{m}$-fraction of any given query will be corrupted. Taking a union bound over all λ queries, and all k possible indices, $i^\star$, we have that, for all queries, no more than a $\frac{2 \cdot \lambda \cdot \mathsf{q}}{m}$ fraction of the query is lost due to the bucketing procedure. □

Lemma 6.4. *When $m > \frac{2 \cdot \lambda \cdot \mathsf{q}}{\gamma}$, the mPIR protocol in Fig. 2 satisfies correctness.*

Proof. To show that decoding succeeds, we have to show that (1) No test queries will be lost in the bucketing procedure and (2) the number of decoding queries lost is small enough that the correct response can be decoded.

Since the test queries are placed in buckets first (see Fig. 3), a test query will be lost only if two test queries are mapped to the same bucket. Since there are λ test queries, and m buckets, by the birthday bound, this probability will be negligible when $m \gg \lambda^2$.

Lemma 6.3 shows that with all-but-negligible probability, all of the λ queries will have less than a $\frac{2 \cdot \lambda \cdot \mathsf{q}}{m}$-fraction corruptions. Since (by assumption), the code, $\mathcal{C}$, is a γ-subcode LDC, decoding will succeed because $\gamma > \frac{2 \cdot \lambda \cdot \mathsf{q}}{m}$.

Note that since $\mathsf{q} = \omega(\lambda)$, the assumption that $\frac{2 \cdot \lambda \cdot \mathsf{q}}{m} < \gamma = O(1)$, ensures that $m \gg \lambda^2$ (as required to ensure that no test queries are dropped). □

Lemma 6.5. *The mPIR protocol in Fig. 2 satisfies privacy.*

Proof. This proof follows exactly the same structure as the proof of Lemma 5.3. We construct a simulator, $\mathcal{S}$ that chooses a random $i^\star_{\mathcal{S}} \in [k]$, and runs the honest protocol (Fig. 2) with respect to this $i^\star_{\mathcal{S}}$. To show privacy, we just need to show that the distribution produced by $\mathcal{S}$ is (computationally) independent of $i^\star_{\mathcal{S}}$, since the simulator is perfect when $i^\star_{\mathcal{S}} = i^\star$. To do this, we consider the same two "hybrid" simulators $\mathcal{S}'$, which ignores decoding failures and only outputs $\perp$ when a test query or proximity test fails. Then, we consider $\mathcal{S}''$ which modifies $\mathcal{S}'$ to replace all the non-test queries with queries to random indices.

To show that the output of $\mathcal{S}$ and $\mathcal{S}'$ are indistinguishable, we just need to show that the probability that decoding fails when all test queries are uncorrupted is negligible. Now, Claim 6.6 shows that, as before, the test queries are still information-theoretically hidden among the decoding queries, so an adversary who corrupts a constant fraction, ϵ, of decoding queries will (with all but negligible probability) corrupt at least one test query.

Now, Lemma 6.3 shows that (with all but negligible probability) the bucketing procedure introduces at most $\frac{2 \cdot \lambda \cdot \mathsf{q}}{m}$-fraction of corruptions into the decoding of any given query, so as long as $\gamma > \epsilon + \frac{2 \cdot \lambda \cdot \mathsf{q}}{m}$, decoding will succeed with all-but-negligible probability whenever the test queries pass. In other words, the probability $\mathcal{S}$ outputs $\perp$ but $\mathcal{S}'$ does not is negligible.

To show that the output of $\mathcal{S}'$ and $\mathcal{S}''$ are indistinguishable, the argument follows exactly as in Lemma 5.3. This follows from the multi-query privacy of the underlying keyword-PIR protocol.

Now, by the smoothness of the LDC, the test queries are independent of $i^\star_{\mathcal{S}}$, and all the rest of the queries in $\mathcal{S}''$ are independent of $i^\star_{\mathcal{S}}$, so the distribution induced by $\mathcal{S}''$ is independent from $i^\star_{\mathcal{S}}$. □

Claim 6.6. *For each $i \in [\lambda]$, denote by $T_i \in [n] \times [m]$ the random variable that is assigned a value the value of (t_i, x_i) where x_i is the bucket assigned to t_i, or $\perp^*$ if no such bucket has been assigned (Step 2 of $D^{\mathsf{b}}_{\mathsf{q}}$) Then T_i is independent of $i^\star$.*

Proof. We pick a random test point $t_i \leftarrow s_i$ for all $i \in [\lambda]$. Since the underlying LDC is smooth, the points $t_1, \ldots, t_\lambda$ are uniformly random points in $[n]$. Then, we "bucket" the test points according to F_{s} (See Fig. 3). This clearly does not make the distribution depend on $i^\star$. Note, however, that the $\{x_i\}$ are not independent from each other (in fact, they are negatively dependent, since two test queries cannot occupy the same bucket). □

Lemma 6.7. *The mPIR protocol in Fig. 2 satisfies coherence.*

Proof. This follows in exactly the same way as in Lemma 5.5, except now there is an extra $\frac{2 \cdot \lambda \cdot \mathsf{q}}{m}$-fraction of each query that can fail due to failed placement in a bucket (Lemma 6.4), but this extra loss is easily absorbed by the error-correction ability of the subcode LDC. □

References

[ACLS18] S. Angel, H. Chen, K. Laine, and S. Setty. "PIR with Compressed Queries and Amortized Query Processing". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 962–979.

[BHMW21] E. Boyle, J. Holmgren, F. Ma, and M. Weiss. *On the Security of Doubly Efficient PIR*. Cryptology ePrint Archive, Report 2021/1113. 2021. URL: https://eprint.iacr.org/2021/1113.

[BIM04] A. Beimel, Y. Ishai, and T. Malkin. "Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing". In: *Journal of Cryptology* 17.2 (2004), pp. 125–151.

[BKP22] S. Ben-David, Y. T. Kalai, and O. Paneth. "Verifiable Private Information Retrieval". In: *Proceedings of the 20th Theory of Cryptography Conference*. TCC '22. 2022, pp. 3–32.

[blyss] *blyssForBTC*. https://btc.usespiral.com/. 2023.

[Cao+23] Q. Cao et al. "Committed Private Information Retrieval". In: *Proceedings of the 28th European Symposium on Research in Computer Security*. ESORICS '23. 2023, pp. 393–413.

[CCKM00] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. "One-Round Secure Computation and Secure Autonomous Mobile Agents". In: *Proceedings of the 37th International Colloquium on Automata, Languages and Programming*. ICALP '10. 2000, pp. 512–523.

[CGKS98] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. "Private Information Retrieval". In: *J. ACM* 45.6 (1998), pp. 965–981.

[CHR17] R. Canetti, J. Holmgren, and S. Richelson. "Towards Doubly Efficient Private Information Retrieval". In: *Proceedings of the 15th Theory of Cryptography Conference*. TCC '17. 2017, pp. 694–726.

[CK20] H. Corrigan-Gibbs and D. Kogan. "Private Information Retrieval with Sublinear Online Time". In: *Proceedings of the 39th Annual InternationalConference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020, pp. 44–75.

[CL24] L. de Castro and K. Lee. "VeriSimplePIR: Verifiability in SimplePIR at No Online Cost for Honest Servers". In: *Proceedings of the 33rd USENIX Security Symposium*. USENIX Security '24. 2024.

[CNCWF23] S. Colombo, K. Nikitin, H. Corrigan-Gibbs, D. J. Wu, and B. Ford. "Authenticated Private Information Retrieval". In: *Proceedings of the 32nd USENIX Security Symposium*. USENIX '23. 2023, pp. 3835–3851.

[DG15] Z. Dvir and S. Gopi. "2-Server PIR with Sub-Polynomial Communication". In: *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*. STOC '15. 2015, pp. 577–584.

[DLNNR04] C. Dwork, M. Langberg, M. Naor, K. Nissim, and O. Reingold. *Succinct NP Proofs and Spooky Interactions*. Available at www.openu.ac.il/home/mikel/papers/spooky.ps. 2004.

[DP09] D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[DT24] M. Dietz and S. Tessaro. "Fully Malicious Authenticated PIR". In: *Proceedings of the 44th Annual International Cryptology Conference*. CRYPTO '24. 2024.

[EKN24] R. Eriguchi, K. Kurosawa, and K. Nuida. "Efficient and Generic Methods to Achieve Active Security in Private Information Retrieval and More Advanced Database Search". In: *Proceedings of the 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '24. 2024.

[GPS04] R. Granger, D. Page, and M. Stam. *On Small Characteristic Algebraic Tori in Pairing-Based Cryptography*. Cryptology ePrint Archive, Report 2004/132. 2004.

[GR05] C. Gentry and Z. Ramzan. "Single-Database Private Information Retrieval with Constant Communication Rate". In: *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*. ICALP '05. 2005, pp. 803–815.

[Guo16] A. Guo. “High-Rate Locally Correctable Codes via Lifting”. In: *IEEE Trans. Inf. Theory* 62.12 (2016), pp. 6672–6682.

[HDCZ23] A. Henzinger, E. Dauterman, H. Corrigan-Gibbs, and N. Zeldovich. “Private Web Search with Tiptoe”. In: *Proceedings of the 29th Symposium on Operating Systems Principles*. SOSP ’23. 2023, pp. 396–416.

[HHGMV23] A. Henzinger, M. M. Hong, H. C. Gibbs, S. Meiklejohn, and V. Vaikuntanathan. “One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval”. In: *Proceedings of the 32nd USENIX Security Symposium*. USENIX Security ’23. USENIX Association, 2023, pp. 3889–3905.

[HS13] J. van der Hoeven and É. Schost. “Multi-point evaluation in higher dimensions”. In: *Appl. Algebra Eng. Commun. Comput.* 24.1 (2013), pp. 37–52.

[IKO05] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. “Sufficient Conditions for Collision-Resistant Hashing”. In: *Proceedings of the 2nd Theory of Cryptography Conference*. TCC ’05. 2005, pp. 445–456.

[IKOS04] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. “Batch Codes and Their Applications”. In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. STOC ’04. 2004, pp. 262–271.

[JPRZ09] C. S. Jutla, A. C. Patthak, A. Rudra, and D. Zuckerman. “Testing low-degree polynomials over prime fields”. In: *Random Struct. Algorithms* 35.2 (2009), pp. 163–193.

[KC21] D. Kogan and H. Corrigan-Gibbs. “Private Blocklist Lookups with Checklist”. In: *Proceedings of the 30th USENIX Security Symposium*. USENIX Security ’21. 2021, pp. 875–892.

[KDKWZ23] S. Kruglik, S. H. Dau, H. M. Kiah, H. Wang, and L. F. Zhang. *Querying Twice to Achieve Information-Theoretic Verifiability in Private Information Retrieval*. techrxiv Report 24290563. 2023.

[KO97] E. Kushilevitz and R. Ostrovsky. “Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval”. In: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. FOCS ’97. 1997, pp. 364–373.

[KRR14] Y. T. Kalai, R. Raz, and R. D. Rothblum. “How to delegate computations: the power of no-signaling proofs”. In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. STOC ’14. 2014, pp. 485–494.

[KSY14] S. Kopparty, S. Saraf, and S. Yekhanin. “High-rate codes with sublinear-time decoding”. In: *Journal of the ACM* 61.5 (2014). Preliminary version appeared in STOC ’11., 28:1–28:20.

[KT00] J. Katz and L. Trevisan. “On the efficiency of local decoding procedures for error-correcting codes”. In: *Proceedings of the 32nd Annual Symposium on Theory of Computing*. STOC ’00. 2000, pp. 80–86.

[KZ23] P. Ke and L. F. Zhang. “Private Information Retrieval with Result Verification for More Servers”. In: *Proceedings of the 21st International Conference on Applied Cryptography and Network Security*. ACNS ’23. 2023, pp. 197–216.

[LMW23] W. Lin, E. Mook, and D. Wichs. “Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring LWE”. In: *Proceedings of the 55th Annual Symposium on Theory of Computing*. STOC ’23. 2023, pp. 595–608.

[LP23] A. Lazzaretti and C. Papamanthou. “TreePIR: Sublinear-Time and Polylog-Bandwidth Private Information Retrieval from DDH”. In: *Proceedings of the 43rd Annual International Cryptology Conference*. CRYPTO ’23. 2023, pp. 284–314.

[Mer89] R. C. Merkle. “A certified digital signature”. In: *Proceedings of the 9th Annual International Cryptology Conference*. CRYPTO ’89. 1989, pp. 218–238.

[MOTBG11] P. Mittal, F. G. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg. “PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval”. In: *Proceedings of the 20th USENIX Security Symposium*. USENIX Security ’11. 2011.

[PSY23] S. Patel, J. Y. Seo, and K. Yeo. "Don't be Dense: Efficient Keyword PIR for Sparse Databases". In: *Proceedings of the 32nd USENIX Security Symposium*. USENIX Security '23. 2023, pp. 3853–3870.

[Rya14] M. D. Ryan. "Enhanced Certificate Transparency and End-to-End Encrypted Mail". In: *Proceedings of the 21st Annual Network and Distributed System Security Symposium*. NDSS '14. 2014.

[SACM21] E. Shi, W. Aqeel, B. Chandrasekaran, and B. M. Maggs. "Puncturable Pseudorandom Sets and Private Information Retrieval with Near-Optimal Online Bandwidth and Time". In: *Proceedings of the 41st Annual International Cryptology Conference, CRYPTO 2021*. CRYPTO '21. 2021, pp. 641–669.

[spiral] *spiralwiki*. https://spiralwiki.com. 2023.

[WZ18] X. Wang and L. Zhao. "Verifiable Single-Server Private Information Retrieval". In: *Proceedings of the International Conference on Information and Communications Security*. ICICS. 2018, pp. 478–493.

[WZLY23] Y. Wang, J. Zhang, J. Liu, and X. Yang. *Crust: Verifiable And Efficient Private Information Retrieval with Sublinear Online Time*. Cryptology ePrint Archive, Report 2023/1607. 2023.

[Yek10] S. Yekhanin. "Private information retrieval". In: *Commun. ACM* 53.4 (2010), pp. 68–73.

[ZS14] L. F. Zhang and R. Safavi-Naini. "Verifiable Multi-server Private Information Retrieval". In: *Proceedings of the 12th International Conference on Applied Cryptography and Network Security*. ACNS '14. 2014, pp. 62–79.

[ZWH21] L. Zhao, X. Wang, and X. Huang. "Verifiable single-server private information retrieval from LWE with binary errors". In: *Information Sciences* 546 (2021), pp. 897–923.

A Doubly-efficient mPIR

We demonstrate the power of our compiler Theorem 2 by applying it to the doubly-efficient PIR (dePIR) of [LMW23]. To instantiate our compiler, we must specify a vector commitment scheme and a smooth subcode LDC. We use Merkle Trees as the vector commitments. For some constant c, the smooth subcode LDCs we use are Reed–Muller codes with parameters:

- message size $|\mathsf{db}| = k$,
- *total* degree (and thus query complexity) $d = \mathsf{q}_{\mathsf{LDC}} = O(\log^c(k))$,
- $m = O(\log(k)/c \log\log(k))$ variables,
- field size $p = \Theta(d)$,

These codes can be shown to have:

- block length $O(k^{1+\epsilon})$ (where ϵ depends on c)
- constant distance
- $(\mu, \mathrm{negl}(\lambda))$ proximity tests with $\mathrm{polylog}(\lambda, k)$ query complexity for all $\mu > 0$ (Theorem 4.15)
- subcode structure
- perfect smoothness
- encoding algorithms running in time $O(k^{1+\epsilon})$ [HS13][Section 6.3].

As desired. As input to dePIR's preprocessing, we now input an encoded database of length $O(k^{1+\epsilon})$ with elements size $\mathrm{polylog}(\lambda, k)$. The complexity of executing dePIR on this slightly larger database is asymptotically unchanged because $O((n^{1+\epsilon})^{1+\epsilon'}) = O(n^{1+\epsilon''})$ and $\mathrm{polylog}(k^{1+\epsilon}) = O(\mathrm{polylog}(k))$. Thus, as a corollary of Theorem 2,[10]

Corollary A.1. *Assuming the security of RingLWE ([LMW23, Section 2.2]) There exists a doubly-efficient mPIR protocol.*

[10] For this particular compilation, Theorem 5.1 would have sufficed because dePIR already had polylogarithmic server computation.