Peer2PIR: Private Queries for IPFS

Miti Mazmudar
University of Waterloo

Shannon Veitch
ETH Zurich

Rasoul Akhavan Mahdavi
University of Waterloo

***Abstract*—The InterPlanetary File System (IPFS) is a peer-to-peer network for storing data in a distributed file system, hosting over 190,000 peers spanning 152 countries. Despite its prominence, the privacy properties that IPFS offers to peers are severely limited. Any query within the network leaks the queried content to other peers. We address IPFS' privacy leakage across three functionalities (peer routing, provider advertisements, and content retrieval), ultimately empowering peers to privately navigate and retrieve content in the network. Our work highlights and addresses novel challenges inherent to integrating PIR into distributed systems. We present our new, private protocols and demonstrate that they incur reasonably low communication and computation overheads. We also provide a systematic comparison of state-of-art PIR protocols in the context of distributed systems.**

2025 IEEE Symposium on Security and Privacy (SP) | 979-8-3315-2236-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/SP61157.2025.00231

1. Introduction

Peer-to-peer (P2P) applications and networks have been widely used for decades and new networks, such as the Inter-Planetary File System (IPFS), are being rapidly developed. IPFS is a P2P network hosting over 190,000 peers spanning 152 countries [58], making it one of the most widely used distributed file systems. The IPFS ecosystem [50] includes many projects that facilitate building decentralized web applications. For example, Fleek [21] serves as a decentralized content distribution network backed by IPFS. Space Daemon [22] enables building distributed applications that process users' data privately on their own IPFS nodes.

Although IPFS has proven itself to be an effective platform for a decentralized web, the current model provides limited privacy to users and lags far behind recent advancements in protecting Internet users. There has been significant adoption on the Internet of DNS-over-Encryption protocols, such as DNS-over-HTTPS and DNS-over-TLS [29], [30], [35]. These alternatives to unencrypted DNS prevent malicious parties, ISPs, and others from observing the domains that are being accessed by users. Meanwhile in IPFS, data sent between two parties is encrypted, akin to Internet traffic being encrypted with TLS; however, any query within the IPFS network leaks to other peers the *content for which a peer is querying*, akin to a DNS query. Encrypting messages between peers prevents passive observers from viewing information, but by revealing it to intermediate peers, a client's privacy is severely undermined. Although distributed systems are lauded for their privacy guarantees, in reality, the current state-of-the-art falls behind the privacy offered in the centralized setting. Protecting the content of queries from intermediate peers is the natural next step to improve the privacy of IPFS, and provide users with end-to-end protection of their query content.

IPFS provides three high-level functionalities to its peers. First, it arranges peers in an overlay topology such that each peer can contact and communicate with any other peer efficiently. IPFS implements this functionality of *peer routing* using distributed hash tables (DHTs). Second, peers can advertise to other peers that they provide a file. The *provider advertising* functionality enables any peer on the network to discover which other peers provide a desired file. Third, a peer can directly contact another peer that provides a file and retrieve the file from them. The two peers engage in a *content retrieval* protocol to share that file. Each of these three functionalities is performed through protocols involving queries between different pairs of peers.

Consider a peer acting as a client and another as a server in IPFS, where the server peer is storing content and is assumed to be a passive adversary. In each of the three aforementioned functionalities, the client reveals what it is querying for to the server. Peer routing reveals which peer the client is attempting to contact. In provider advertisement and content retrieval queries, a server learns which file the client wishes to retrieve.

Prior work addressed only one of these problems in isolation, e.g., by enabling confidentiality for the target file in content retrieval [15], [16] or obfuscating the target peer in peer routing [44]. However, replacing one aspect of the system with a private version is insufficient if the remainder of the system leaks the queried content. For example, a server peer can learn a target file through a provider routing query, even if it is hidden in a content retrieval query. Additionally, privacy of provider advertisements has not previously been addressed.

This work proposes an end-to-end solution for IPFS that hides the *content a client queries for and retrieves* from a server. Ours is the first work to address this problem holistically across IPFS protocols for peer routing, provider advertising, and content retrieval. Not only is the content of clients' queries then protected from intermediate peers, but also from the final peer from whom they collect the content. Ultimately, no one within the network learns the content which a peer is retrieving. In developing our solution, we handle challenges inherent to DHTs, such as a dynamic network with churn, and varying amounts of content stored at each peer. Therefore, our resulting techniques are applicable to a wider range of distributed systems. Towards developing this solution, we present the following building blocks:

- a novel routing algorithm that enables an end-to-end procedure for privately contacting a peer (Section 5);
- generic binning and hashing techniques to adapt datastores to be amenable to PIR, applied to provider advertisements and content retrieval (Sections 6 and 7);
- a scalable and private adaptation of an interactive content retrieval protocol (Section 7); and,
- a systematic comparison of state-of-the-art PIR protocols in the context of distributed systems (Section 8).

Furthermore, for our use cases in DHTs, existing PIR protocols do not provide a reasonable communication-computation tradeoff. To address this gap, we present new PIR protocols, using techniques from cryptographic folklore and PIR literature (Section 8.2), which are optimized for small databases. We prove the security of our PIR protocols and demonstrate that they outperform the state-of-the-art.

Our performance evaluation demonstrates the feasibility of our system in terms of communication and computation costs. Our private peer routing and provider advertisement queries can be answered in less than 100ms and 1.5s, respectively. We believe that these are reasonable latencies for the IPFS system, given the benefit of added privacy. Furthermore, our private protocols do not incur additional rounds and our private peer routing protocol does not significantly increase the number of hops over the original routing algorithm. Our system is modular in that it enables substituting PIR protocols for each of the three functionalities, and thus Peer2PIR directly benefits as PIR protocols become more efficient. We integrate our protocols into the IPFS codebase, highlighting and addressing privacy engineering challenges faced along the way.

2. Background

2.1. Distributed Hash Tables & IPFS

Distributed hash tables (DHTs), such as Kademlia [38] and Chord [55], are analogous to conventional hash tables, with the exception that the entire key-value store is systematically split across peers in the network so that queries can be conducted efficiently. Importantly, both the content stored for the application and the routing information for the P2P network are distributed among peers. P2P networks grow dynamically: the rate at which peers leave and rejoin the network is the *node churn* rate. Though each peer can answer and conduct queries, throughout this paper, we designate peers as *clients* when they are conducting queries and as *servers* when they are responding to queries. IPFS is a distributed file system built on the Kademlia DHT [32], [58]. Next, we describe the three main functionalities provided by IPFS: contacting another peer in the DHT, discovering which peers provide desired content, and retrieving content from a peer. Figure 1 visualizes these functionalities, alongside our private alternatives.

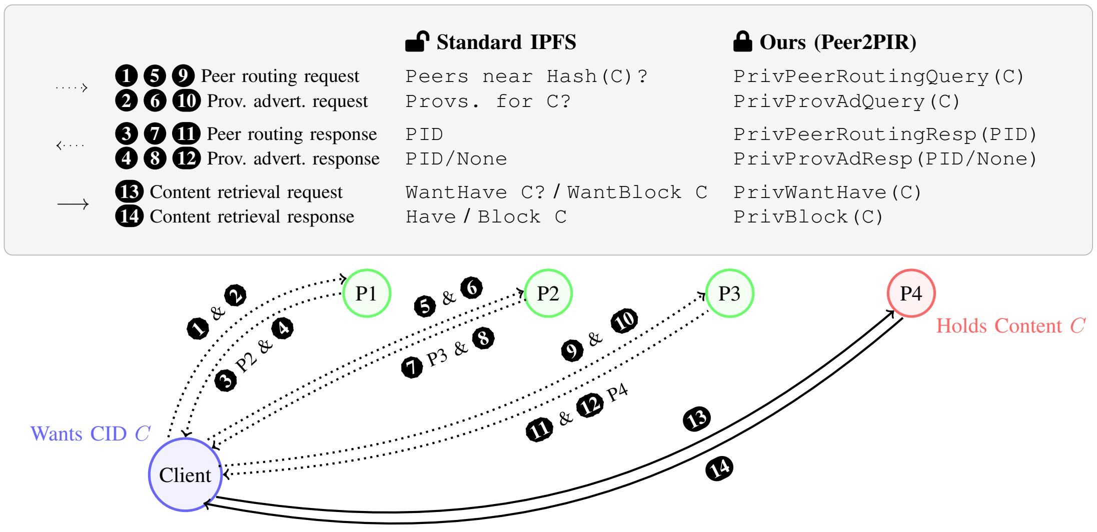
Contacting a Peer. Each peer in a DHT has a peer identifier (peer ID) that is the output of a cryptographic hash function. For example, in IPFS, peer IDs are given by a hash of the peer's public key. Each peer in the DHT maintains a *routing table* consisting of peer IDs and their routing information, for a small number of other peers in the DHT. The routing information consists of *multiaddresses* which encode IP addresses and ports, as well as peer IDs. To reach a target peer, a client obtains relevant routing information by querying one of its neighbours for the target peer ID, and then querying one of that neighbour's neighbours, and so on, through an iterative routing process. DHTs guarantee that each peer can reach any other peer in a network of size n, without a trusted central party, through approximately $\log(n)$ routing queries. Importantly, each such routing query will reveal the target peer ID to the in-path peers. The IPFS routing algorithm in the context of the Kademlia DHT is discussed in Section 5.1.

Discovering Peers Providing Content. IPFS decomposes each file into multiple content blocks, using either fixed or adaptively sized chunks [26], [49]. Each content block in IPFS is addressed by a *content identifier* (CID), which includes a collision-resistant one-way hash of the block. CIDs remove the need for a central party to address content and are also used to verify the integrity of a content block. A peer who provides a content block in IPFS, namely a provider peer, also advertises this fact to the DHT, through provider advertisements. A provider advertisement maps a given CID to its providers' peer IDs. Each peer maintains a provider store with provider advertisements. So a client who wishes to discover the peers who provide a given CID, should query the server peers' provider store for the target CID. (This query occurs simultaneously with the aforementioned routing query, as we describe later.) Evidently, through this query, the server peer learns the target CID that the client wishes to access. We describe IPFS provider advertisements in Section 6.1.

Retrieving a Content Block. Once a client determines the provider peer for a target CID, it proceeds to retrieve the content block. Content retrieval in IPFS is accomplished using Bitswap [18], a content exchange protocol for retrieving a content block with a given CID from a connected peer. We detail the Bitswap protocol in Section 7.1 and note that the server learns the target CID that the client wishes to fetch as well as the content block that it returns.

2.2. Private Information Retrieval

Private information retrieval (PIR) protocols allow a client to retrieve an item from a database, held by one or more servers, such that the servers do not learn which item is retrieved. Information-theoretic PIR (IT-PIR) schemes provide perfect secrecy in the presence of computationally unbounded adversaries. Many efficient IT-PIR schemes exist in the multi-server setting [4], [9], [14], [19]; however, they rely on assumptions that are difficult to achieve in practice, as discussed in Section 9. In contrast to IT-PIR, computational PIR (CPIR) schemes rely on hardness assumptions and assume a computationally bounded adversary. We mainly focus on CPIR schemes for our protocols.



		Standard IPFS	Ours (Peer2PIR)
⋯→	1 5 9 Peer routing request	`Peers near Hash(C)?`	`PrivPeerRoutingQuery(C)`
	2 6 10 Prov. advert. request	`Provs. for C?`	`PrivProvAdQuery(C)`
←⋯	3 7 11 Peer routing response	`PID`	`PrivPeerRoutingResp(PID)`
	4 8 12 Prov. advert. response	`PID/None`	`PrivProvAdResp(PID/None)`
⟶	13 Content retrieval request	`WantHave C?` / `WantBlock C`	`PrivWantHave(C)`
	14 Content retrieval response	`Have` / `Block C`	`PrivBlock(C)`

Figure 1: Overview of the IPFS peer routing, provider routing, and content retrieval protocols, and our private versions.

We defer the formal definition of PIR to Section B.1. While PIR schemes ensure that the server does not learn which record is retrieved, symmetric PIR (SPIR) schemes also guarantee that the client learns only the record for which they query, similar to oblivious transfer (OT).

IndexPIR vs. KeywordPIR. PIR schemes often assume the client knows the index of the desired row in a table; these are known as IndexPIR schemes. KeywordPIR schemes enable a client to retrieve a value from a key-value store, given that they know the key. KeywordPIR schemes privately match keywords, and thus incur communication overhead over IndexPIR schemes [1], [37]. PIR over key-value stores can be performed using IndexPIR, with an extra round of communication and a small probability of error. The server constructs a hash table, mapping each key to an index in the table. It places the associated value into the indexed row of the hash table and sends the chosen hash function to the client. The client uses the hash function and its desired keyword to calculate the index for its desired row in the hash table and uses IndexPIR to retrieve that row. As modern DHTs involve key-value stores in their design, we use this transform to efficiently conduct IndexPIR. We exploit the fact that CIDs themselves include hashes, to enable the client to compute the index for a CID (within the server's hash table), without an extra round.

PIR in Distributed Settings. The DHT setting presents challenges in immediately applying PIR. First, the key-value stores have varying numbers of records and record sizes. While the routing table is upper-bounded in size, the number of provider advertisements and content blocks on each peer follows a long-tail distribution, as we discuss in Section 4. Since PIR schemes are typically optimized for one database size, one scheme does not fit all three of our use cases. Second, we seek to minimize the communication overhead of PIR queries, to prevent congesting the distributed network, while maintaining reasonable computational overheads for individual peers. Furthermore, existing schemes commonly assume that clients repeatedly query the same server, amortizing communication cost over many queries. However, across all types of queries that we consider, a peer acting as a client routinely contacts different peers. Thus, a direct application of state-of-the-art PIR protocols optimized for communication overhead is not possible for our use cases.

3. Related Work

Private Routing and Provider Advertisements. The recently proposed Double Hashing [43], [44] aims to improve client privacy when querying for content in IPFS. The approach requires that a client query for a *prefix* of the CID as opposed to the entire CID, thus providing the client with plausible deniability about which content it is querying for. A server returns provider advertisements corresponding to any CID matching this prefix. The client thus obtains k-anonymity where k is the number of CIDs with a matching prefix. We provide stronger guarantees to both the client and the server. The client hides the entire CID, instead of just the suffix, from the server. The server is guaranteed that the client does not learn other CIDs stored by the server.

Private Content Retrieval. Daniel et al. [15] provide plausible deniability to peers by integrating a gossip protocol into Bitswap, in which peers forward messages on each other's behalf. While their work obscures the source of a query, we protect the content of a query. Daniel and Tschorsch [16] present two protocols to hide the target CID from the server during the content *discovery* step of Bitswap. However, the first protocol leaks the entire list of CIDs to the client, whereas the second protocol relies on computationally expensive Private Set Intersection (PSI) operations. Both protocols reveal the target CID to the server during the content *retrieval* step. Mazmudar et al. [39] incorporate IT-PIR into DHTs to hide the content that clients retrieve within the network. This approach relies on grouping peers into *quorums* which contain a bounded fraction of malicious nodes. Under this assumption, their construction can take advantage of existing protocols [6], [60] for robust DHTs, enabling integrity and confidentiality of routing queries within the DHT. We discuss challenges with incorporating similar strategies into IPFS in Section 9. Keyword search schemes for IPFS enable a client to retrieve a file, given a representative keyword rather than its CID [12]. They also reveal the keyword to server peers.

4. Threat Model

We consider a peer acting as a client and an adversarial peer acting as a server. We aim to hide the *content the client queries for and retrieves* from the server, across three queries:

- Peer routing queries: the client fetches routing information for a *target peer ID* from a server.
- Provider advertisement queries: the client fetches which peers provide a *target CID*, as well as the routing information to reach these provider peers.
- Content retrieval queries: the client fetches a block from a server peer, given its *target CID*.

We hide the target peer ID from each intermediate server peer during peer routing queries. We hide the CIDs during provider advertisement and content retrieval queries. Additionally, the server peer cannot determine these identifiers based on its response to a client's query. This strictly improves over IPFS, as it currently only encrypts communication between peers at the transport layer [51].

We assume that all peers within the network are honest but curious, i.e., they correctly follow prescribed protocols but may passively observe their execution to infer information. We briefly outline non-goals before contextualizing our guarantees for provider advertisements and content retrieval.

Non-Goals. We do not aim to protect the client's network identity. Instantiating a distributed file system such as IPFS over Tor onion services to provide client anonymity is out of our scope. We do not protect content *publishing* requests, i.e., peers know that a server peer is storing a given CID (via provider advertisements or by querying the server for the content block). Since CPIR schemes add significantly more computational overhead for the server than the client, a malicious client can exploit our CPIR-based schemes to amplify a denial of service attack against any peer. Standard rate-limiting based on the client's peer ID and network identity can deter such DoS attacks.

Peer Routing. Servers may cause an honest client to experience a restricted view of the network through *eclipse attacks*. A server peer may respond incorrectly to a client's routing request, directing it to other malicious server peers. Prünster et al. [52] demonstrated an eclipse attack against IPFS, which has since been fixed. While we do not prevent eclipse attacks, we do not make IPFS peers more vulnerable to these attacks, e.g. by releasing a client's routing table.

Provider Advertisements and Content Retrieval. We provide *server privacy* for these two use-cases. We ensure that a client can only retrieve a provider advertisement or a content block if it knows the corresponding CID. In other words, the client should not be able to retrieve information stored by a server peer corresponding to CIDs that are unknown to the client. Separately, we observe that the privacy guarantees afforded to a client depend on the amount of content stored by server peers. The amount of provider advertisements stored by each peer in IPFS follows a long tail distribution [56]. So it is highly likely that a client will query a heavyweight content provider for both provider advertisements and content blocks, and its target CID will be indistinguishable from millions of others. Moreover, assuming that a peer stores files of different sizes, the number of content retrieval queries made by the client may leak which file is being retrieved. We cannot prevent this attack without changing how content blocks are duplicated in IPFS.

Colluding servers. Server peers do not learn any information about a client's query by sharing and examining the client's PIR queries, as these rely only on computational assumptions, and not on non-collusion assumptions. Nevertheless, server peers could share that they received a peer routing query (or provider routing or content retrieval query) from the client's network address or peer ID. Based on the volume and sequence of (encrypted) traffic sent by the client to these server peers, they can then narrow down the search space of possible target peer IDs (or target CIDs) that the client is looking for. Wang et al. [59] evaluate the feasibility of such traffic correlation attacks against the Kademlia DHT with non-private peer and provider routing queries; such attacks could be extrapolated to our system. Existing information-theoretic systems for private queries over DHTs, such as Mazmudar et al.'s DHTPIR (see Section 9), are also vulnerable to such traffic correlation attacks, since similar to our work, they do not provide client anonymity.

5. Private Peer Routing

In this section, we focus on how a server answers a client's query for a target peer ID without revealing the peer ID to the server. We first outline the Kademlia DHT routing process, highlighting how a server uses its routing table. We then propose an algorithm to adapt the routing table to enable hiding target peer IDs from the server and demonstrate its correctness. Finally, we integrate a PIR scheme to privately retrieve a bucket from our modified routing table, leading to a private peer routing algorithm.

5.1. Kademlia and the IPFS Routing Table

Kademlia DHT. DHTs map both the content identifiers and peer identifiers to a common virtual address space, through a collision-resistant one-way hash. IPFS uses SHA-256 to form the virtual address space for its underlying Kademlia DHT. Kademlia measures *distance* between two addresses by computing an XOR between them and interpreting the result as an integer. For example, the addresses `0b00011` and `0b00101`, in an example 5-bit address space, are `0b00110=6` units apart. They share the first 2 bits and are said to have a *common prefix length* (CPL) of 2. Addresses with higher CPLs are said to be closer to one another.

Routing Table Structure. The routing table (RT) is indexed by CPLs. When a peer is added to an RT, the peer first computes the CPL between itself and the new peer. Each row of the RT contains multiaddresses for at most k peers whose CPL is given by the row index. In other words, higher indices of the RT contain closer peers. In IPFS, each row (also called *bucket*) can have up to $k = 20$ peers' multiaddresses. When a peer first joins the network, it contacts a set of bootstrapping peers to populate its RT. Whenever a peer comes into contact with another peer in the DHT, it may create a new bucket for this peer or fill up an existing bucket, depending on the CPL between the two peers. Buckets are created dynamically, so a bucket with index i is created only when the peer has met at least k additional peers whose CPL is greater than or equal to i. If an existing bucket is full, then the new peer is not stored in the RT, leading to Kademlia preferring longer-lived peers. In general, the RT can have buckets of unequal sizes.

Kademlia Lookup. We focus on how a server uses its RT to answer a client's query for the target peer ID. If the bucket in the RT given by the CPL between the target and the server's peer IDs is full, then the server returns that bucket. Otherwise, it selects peers from other buckets to return a set of k closest peers. It does so by computing the distance between the target peer ID and the remaining peer IDs in the RT to select the k closest peers. As long as the RT has at least k peers in total, for any lookup, the server returns the multiaddresses for k peers. The client then uses the multiaddresses to contact these peers, which are closer to the target peer than the server peer. The client iteratively queries different server peers until it reaches the target peer.

Challenges in Hiding the Target Peer ID. Our goal is to enable the server to respond to the client's query, while hiding the target peer ID. Suppose that the server only learns the bucket index, namely the CPL between its own peer ID and the target peer ID. (We ultimately hide this index as well, via PIR, as we discuss later.) If the server's bucket for this CPL is full, then it simply returns this bucket. However, if it is not full, then only knowing the CPL is insufficient to complete the original Kademlia lookup. In particular, the suffix of the target peer ID is used to compute distances to peers in other buckets, and select the closest ones. A naive approach might return the bucket corresponding to the CPL, even if it is full; however, non-full buckets would be insufficient for the client to reach the target peer, and the bucket size could leak its index. To solve the problem of non-full buckets, the server can compute the k-closest peers for every target peer ID. This approach blows up the RT, which originally has at most 256 rows, to 2^{256} rows.

We propose *normalizing* the RT, i.e., ensuring that each bucket has exactly k peers closest to any target peer ID whose CPL is given by the bucket index. We present an algorithm for normalizing the RT buckets. We introduce a small amount of randomization when selecting closest peers while knowing only the target peer's CPL. We demonstrate that our algorithm introduces only a constant overhead to the number of hops that the client must perform, for any target peer ID, in comparison to the original Kademlia lookup. In practice, as we show next, our algorithm introduces negligible to no overhead.

5.2. Routing Table with Normalized Buckets

We present our normalization algorithm (Algorithm 1) and describe it next. The client wishes to retrieve a target bucket indexed by t, where t is the CPL between the target and the server's peer IDs. As in the original lookup algorithm, if the RT has at most k peers *in total*, all peers are returned (Line 4). Since buckets are created dynamically, the RT may have $r < t$ buckets, in which case the server returns the (normalized) last bucket, containing the closest k peers that it knows (Line 6). If the target bucket is full, it is returned directly (Line 9).

Algorithm 1 Routing Table Normalization Algorithm

```
 1: $i \in [0, r = \text{len(RT)}]$, bucket $B_i$ corresponds to CPL $i$.
    Input: Target index $t$
 2: $R \leftarrow \cup_{i=0}^{r} B_i$
 3: if $|R| \leq k$ then                          ▷ RT has less than k peers
 4:     return $R$
 5: if $r < t$ then                               ▷ If $r < t$, return last bucket
 6:     $t \leftarrow r$
 7: $R \leftarrow B_t$
 8: if $|R| == k$ then
 9:     return $R$
10: else                                          ▷ $R$ is not full ($|R| < k$)
11:     if $t < r$ then                           ▷ Closer buckets can be used
12:         $C \leftarrow \cup_{i=t+1}^{r} B_i$
13:         if $|C| \leq k - |R|$ then
14:             $R \leftarrow R \cup C$
15:         else append $k - |R|$ peers from $C$ to $R$ at random
16:     if $|R| < k$ then                         ▷ Must go through farther buckets
17:         $\ell \leftarrow t - 1$
18:         while $|R| + |B_\ell| \leq k$ do
19:             $R \leftarrow R \cup B_\ell$
20:             $\ell \leftarrow \ell - 1$
21:         $B_\ell \leftarrow \{B_\ell^j\}$ ▷ Sort peers in $B_\ell$ by distance to server
            (smaller $j$ = closer to server)
22:         $m \leftarrow 1$
23:         while $|R| + |B_\ell^m| \leq k$ do
24:             $R \leftarrow R \cup B_\ell^m$
25:             $m \leftarrow m + 1$
26:         append $k - |R|$ peers from $B_\ell^m$ to $R$, at random.
27: return $R$
```

We observe that a peer in a bucket t would share the common prefix of length t not only with the server peer, but also with peers in closer buckets ($t+1 \leq i \leq r$). Whereas its common prefix with peers in farther buckets ($0 \leq i \leq t-1$) is given by the index of the farther bucket. In particular, a peer in a bucket t is closer to peers in buckets that are closer to the server and farther from peers in buckets that are farther from the server. Thus the server prefers filling a bucket with peers from closer buckets over farther ones (Line 11). Importantly, without knowing the suffix of the target peer ID, the server cannot order peers in these closer buckets in terms of their distance to the target. So, the server picks the remaining number of peers at random from all closer buckets, taking all closer peers if there are less than k such peers (Lines 13–15).

If bucket t remains non-full, the server then adds farther buckets one at a time (Lines 18–20). If the server cannot add a bucket completely, then it partitions the bucket into equivalent subbuckets, based on its distance to them (Line 21), as illustrated in Example 5.1. The server then adds each subbucket one at a time (Line 23–25), selecting peers at random if it cannot include the entire subbucket (Line 26).

Example 5.1. *A server with peer ID* `0b0011` *has 2 buckets:*

- B_2 *with the common prefix* `0b00XX`*. It can contain peers with IDs* `0b001X`*. It currently has* `0b0010`*.*
- B_1 *with the common prefix* `0b0XXX`*. It can contain peers with IDs* `0b01XX`*. It currently has* `0b0101`*,* `0b0111`*,* `0b0110`*.*

Suppose we wish to normalize B_2 *to* $k = 3$ *peers. Peers in* B_1 *that are closest to* B_2 *would share the later bit(s) of the common prefix, i.e. they would be of the form* `0b011X`*, so that* `0b001X` $\oplus$ `0b011X` *is minimized to* $[4, 6)$*. We can partition* B_1 *into:* $B_1^1 = \texttt{0b011X} = \{\texttt{0b0111}, \texttt{0b0110}\}$ *and* $B_1^2 = \texttt{0b010X} = \{\texttt{0b0101}\}$*. Then, a normalized* B_2 *would contain* $\{\texttt{0b0010}, \texttt{0b0111}, \texttt{0b0110}\}$*.*

Convergence. We prove and experimentally verify that our normalized algorithm does not significantly increase the number of hops required to reach a target peer. At a high level, the argument follows from the fact that the only cases in which the normalized algorithm diverges from the original still provide the *same guarantee* of closeness to the target peer. In the lemma, KADEMLIA refers to the original routing algorithm and PRIV-KADEMLIA refers to our normalized routing algorithm. We defer the proof to Section A.

Lemma 5.1 (Convergence of Private Routing). *If* KADEMLIA *converges in* $\lceil \log n \rceil + c_1$ *hops, then* PRIV-KADEMLIA *converges in* $\lceil \log n \rceil + c_2$ *hops for some constants* c_1, c_2*.*

We experimentally evaluated our normalized routing algorithm against the Trie-based routing algorithm currently deployed in IPFS to determine the difference in the number of hops required to reach a target peer. We simulated a network that mirrors the topology of IPFS and queried for different target peer IDs, tracking the number of hops required by each algorithm. Our experiments verify Lemma 5.1 and show that in practice, normalizing the routing table introduces negligible to no difference in the number of hops.

To capture a meaningful network topology, we crawled the active IPFS network using the Nebula crawler [57] and collected an adjacency list of all dialable peers and their neighbours. Our crawl collected 13692 peer IDs from the network; we note that no network addresses were gathered. For each peer, we set up a Trie-based RT using the current Kademlia implementation and a normalized RT using our new algorithm. We instantiated our experiment with the same parameters as IPFS: for each step of the routing process, the client concurrently queries three closest peers it knows to the target peer ID and retrieves $k = 20$ nearest peers from them. The process is repeated iteratively until the target peer is found. We tested 5000 target peer IDs at random, from a constant client peer.

All queries took an average of two hops to reach the target, with a maximum of six hops. We did not find a difference in the number of hops using the Trie-based and the normalized routing algorithms, that is, both algorithms required an identical number of hops in every iteration of our test. Thus, our normalized routing algorithm imposes negligible overhead over the original routing algorithm.

5.3. Private Algorithm Integration

Integrating our normalization algorithm into IPFS generates privacy engineering challenges. For instance, the contents of the RT in practice differ from the Kademlia model. We discuss why PIR is optimal for privately retrieving buckets from the RT, and then integrate our algorithm with PIR to develop a private peer routing solution.

Dynamic Networks. IPFS is a dynamic network with high churn, meaning that nodes often join and leave the network. Consequently, clients' RTs are regularly updated in order to reflect new or failing peers. We require normalizing the RT every time a new peer is added to it. Since routing tables are relatively small and the normalization algorithm is not computationally intensive, this does not impose any prohibitive overhead on the clients.

Joining Peer IDs and Addresses. We have thus far assumed that the RT stores multiaddresses to peers. In practice, the RT in IPFS stores only the *peer IDs*, and an additional *address book* maps peer IDs to their multiaddresses. This allows multiaddresses to be stored only once and used for peer and provider routing. A non-private lookup uses the value fetched from the RT as a key to lookup the address book; this is evidently not possible when privately retrieving buckets from the RT. Therefore, in our implementation, the server joins the two key-value stores, namely the RT storing peer IDs and address book storing multiaddresses, before responding to a private query. This enables PIR to be performed over one table rather than two. The join must be recomputed when either store is updated, which is a function of the node churn rate.

Selecting a PIR scheme. Following normalization and join steps, we must hide the target CPL from the server. The RT is limited in both its number of rows (≤ 256) and the size of each row (multiaddresses for $k = 20$ peers).

Using trivial PIR, the server would provide the entire RT to the client, foregoing normalization and joining; however, trivial PIR makes the server susceptible to eclipse attacks [52]. Alternatively, OT would restrict the client to retrieving only one row and prevent leaking other rows' contents. However, efficient OT algorithms typically involve encrypting each row of a table with a different key and then obliviously transferring a cryptographic key to a client, making the total communication cost greater than the size of the table. This introduces a trade-off between information leaked to clients about other rows in the table and communication cost. Given that IPFS does not restrict a client from repeatedly querying a server for different target peer IDs, i.e., rows in the routing table, we determine that the minimal leakage guarantee provided by OT does not justify the communication overhead. Rate-limiting on the server peer can reduce the susceptibility of server peers to eclipse attacks. We thus opt for PIR with rate-limiting.

We design an IndexPIR scheme based on RLWE, namely RLWEPIR, catered to our private routing use case. This scheme and design rationale are detailed in Section 8.

PIR Integration. A client first computes the CPL between the target peer ID and the server peer ID, which it uses to construct a PIR query. The server uses the PIR query and its joined, normalized routing table, to compute a PIR response, which is returned to the client. The client processes the response to obtain multiaddresses of k peer IDs close to the target peer ID. PIR hides the target CPL from the server.

Security. Interactions between the client and the server now consist only of PIR queries and responses, so security follows directly from that of RLWEPIR, as in Section 8.

6. Private Provider Advertisements

We describe provider advertisements in IPFS, namely, how clients determine which peers provide the desired content block, and introduce our private alternative.

Notation. The following notation is useful for the next two sections. Each peer stores m content blocks, each having a unique CID. We refer generically to a CID as c. A peer holds an ordered list L of CIDs of the content blocks they store. We refer to the i-th CID by $L[i]$. We refer to the first ℓ bits of the CID by the notation $[:\ell]$ and so $L[i][:\ell]$ refers to an ℓ-bit prefix of the i-th CID in L. We form a table D consisting of all content blocks, ordered by their CIDs, such that $D[i]$ is the content block for the i-th CID in L. Each peer also maintains a provider store P, which maps each CID for which the peer knows a provider, to the provider peers' IDs. The address book M maps a peer ID to its multiaddress. $\mathsf{SE.Enc}(k, x)$ refers to the (robust) symmetric encryption of x with the key k. For example, this can be instantiated with AES-AEZ [28]. We refer to the homomorphic encryption (HE) of x as $\mathsf{HE.Enc}(x)$. Let KDF denote a *key derivation function*, which takes as input some key material and outputs an appropriate cryptographic key. Implicitly, the KDF outputs a key of the desired length (in our case, the length of key required for the scheme SE).

6.1. IPFS Provider Advertisements

A provider advertisement maps a CID c to the peer ID p of a peer that provides c, namely a provider peer. Each peer maintains a key-value store of provider advertisements, known as a provider store. When a provider peer stores a content block addressed by c, it advertises the block to the IPFS network. The provider peer contacts its k-closest neighbours to c, following the aforementioned iterative routing process. (A routing query for a CID operates similarly to that for a peer ID, in that both identifiers get hashed to the SHA-256 virtual address space of the Kademlia DHT, as observed in Section 5.1.) The provider peer then requests each neighbour to add a provider advertisement for this CID to their provider store.

Consider a client who wishes to discover the provider peers for c. The client first runs a routing query for that CID. This routing query will ultimately lead the client to one of the k-closest peers to c. We note that our private peer routing algorithm from Section 5.3 can be used to determine k-closest peers to the desired CID, without revealing the CID to the server peer. As described above, these peers will store the provider advertisement, which would map CID c to the provider peer p. So, along with the routing query for c, the client must also query these peers' provider store for c. We refer to this as a provider advertisement query.

A server peer processes a provider advertisement query non-privately as follows. The provider advertisement datastore is keyed by the CID concatenated with the provider peer's ID, since a block with a given CID can be provided by multiple peers. Each value in the provider store is the expiry time of the advertisement. Upon receiving a provider advertisement query, a server peer looks up the provider store for keys prefixed by the given CID. It checks if each such provider advertisement has not expired, based on the value, and if so, extracts the provider peer's ID from the key suffix. Recall that the server peer also maintains an address book, which maps peer IDs to their multiaddresses. The server peer then consults its address book for the provider peers' IDs and fetches their multiaddresses. We observe that the client's desired CID continues to leak to the server in the provider advertisement query, since the server looks up its provider advertisement store based on this CID.

6.2. Private Algorithm Integration

We present our algorithm for private provider advertisement queries in Algorithm 2. This enables a client to send a PIR query hiding the CID that it is querying for. The server returns a PIR response with the peer IDs of corresponding providers, if they exist in the server's provider store.

Joining provider store to address book. Similar to the non-private peer routing query, the non-private provider advertisement query uses the provider peers' IDs from the provider store as a key to the address book which contains multiaddresses. We join the two key-value stores such that for a given key (CID), the value contains the provider peer's multiaddresses (Line 8–Line 11). We recompute the join whenever a provider or peer record is inserted. Provider advertisements have a short default expiry period (of 24 hours), and are typically reinserted after expiration.

Binning for PIR. Given that we now have a joined key-value store, we could directly apply a KeywordPIR scheme, wherein the desired CID acts as the keyword that the server can privately lookup. Instead, to avoid costs of KeywordPIR schemes, we transform our joined key-value store to one which enables an IndexPIR scheme, which is more efficient. This follows the approach described in Section 2.2, with optimizations. We exploit the fact that the CIDs can be used to directly map each key-value pair from the joined key value store to a hash table. CIDs in IPFS are computed using SHA-256, so we can treat them as indices of a hash table. All key-value pairs whose CIDs have the same ℓ-bit prefix are grouped into a single row of the table.

We refer to this process as *binning*, and it enables multiple pairs to be binned into the same row of the table. The bins would fill up non-uniformly as more provider advertisements are added. So the server pads incomplete bins to the size of the largest bin. This provides a hash table with B consistently-sized bins over which we can perform PIR. While the server could vary the number of bins with the number of provider advertisements that it stores, this would require the client to know the number of bins in advance to formulate its query. Instead, we fix B beforehand, based on the parameterization of the chosen PIR scheme, as we describe later. To retrieve a provider advertisement for a given CID, c, the client uses its $\log_2 B$-bit prefix as their desired index in the hash table (Line 2). It then constructs its PIR query based on this index (Line 3). The performance of the PIR protocol is relative to the size of the bins.

Encrypted Entries. Immediately applying PIR to the binned table may enable a client to learn more than it asked for, since any row in the table contains multiple provider advertisements. To prevent this leakage, all advertisements are encrypted under a key derived from the corresponding CID. Effectively, the CID acts as a *pre-shared secret* between a client who desires the provider peers for that CID and a server who stores them. The server uses a Key Derivation Function (KDF) to compute a key for each CID c in the joined store (Line 13). The server encrypts the multiaddresses for each CID under this key, using a *robust* symmetric encryption scheme (Line 14). None of the preceding steps require the client's PIR query, and thus, the table T of binned, encrypted provider advertisements can be precomputed in advance. The server then computes a PIR response over this table (Line 15).

After a client receives and decrypts the homomorphically encrypted PIR response, they obtain a set of ciphertext provider advertisements, which are symmetrically encrypted under keys derived from CIDs. The client can derive the key from their desired CID, $k = \mathsf{KDF}(c)$, and use it to decrypt each ciphertext. Under a robust encryption scheme, only the ciphertext encrypted with the key derived from c will decrypt to a valid plaintext. This provides security of a *symmetric PIR* scheme. In a *trivial* symmetric PIR scheme, the server would send back T to the client (Line 14). We discuss optimal PIR schemes for this use case in Section 8.

Security. The security of the chosen PIR protocol ensures that server does not learn the queried CIDs. Meanwhile, the client only receives a set of encrypted provider advertisements. The *symmetric* property of our protocol requires that the client only learns provider advertisements for the CID that they queried for, and learns nothing about other provider advertisements stored by the server. In our protocol, this reduces to the security of the robust symmetric encryption scheme and the KDF. We defer the proof to Section C.

Algorithm 2 Private Provider Advertisements using PIR with B bins over a list of provider records P

```
1: procedure PRIVATEPROVADQUERY(CID $c$)
2:     $\mathtt{qu} \leftarrow c[:\log_2 B]$                    ▷ $q \in \{0, 1, \cdots, B-1\}$
3:     $(\mathtt{sk}, (\mathtt{pk}, \mathtt{ct})) \leftarrow$ PIR.QUERY($\mathtt{qu}$)
4:     Send $(\mathtt{pk}, \mathtt{ct})$ to the server.
5: procedure PRIVATEPROVADRESP($(\mathtt{pk}, \mathtt{ct})$, $P$)
6:     Initialize tables: $J$ with $\|P\|$ rows, $T$ with $B$ bins.
7:     for $i \in \{1, 2, \cdots, |P|\}$ do
8:         $(c, \{p_1, p_2 \cdots p_j\}) \leftarrow P[i]$
9:         $m \leftarrow c$
10:        for $k \in \{1, 2, \cdots, j\}$ do
11:            $m \leftarrow m \parallel M[p_k]$    ▷ Addresses for providers of c.
12:        $b \leftarrow c[:\log_2 B]$                          ▷ Bin index.
13:        $k \leftarrow \mathsf{KDF}(c)$              ▷ Derive a key using the CID.
14:        $T[b] \leftarrow T[b] \cup \mathsf{SE.Enc}(k, m)$       ▷ Encrypt addresses.
15:    $\mathtt{ans} \leftarrow$ PIR.RESPONSE($(\mathtt{pk}, \mathtt{ct}), T$)
```

7. Private Content Retrieval

IPFS employs the *Bitswap* [18] content retrieval protocol, which, as in the provider routing protocol, leaks the CID retrieved to the server. We describe two steps of the Bitswap protocol (WANTHAVE and BLOCK) and introduce private equivalents.

7.1. Bitswap Protocol

The WANTHAVE Step. The client concurrently sends a message to multiple candidate peers, asking if they possess a block with a given CID. If the peer has the requested CID in its table, it responds with a *Have* message. Otherwise, it may respond with a *DontHave* message.Importantly, IPFS clients can use the WANTHAVE step to first query long-lived peers in its RT, thereby assessing whether they have a content block, *before* searching for provider advertisements for that block. So, the set of candidate peers corresponds to either a set of peers in the clients' RT that are queried opportunistically, or a set of peers that are output from provider advertisement queries.

The BLOCK Step. In this step, the client asks a peer, who has responded with *Have* in the previous step, to transfer the desired block. There may not be a clear separation between these two steps, depending on the block size used in the chunking algorithm [49]. For example, a server peer can immediately send a small block along with the *Have* response, thereby avoiding an extra round of interaction.

7.2. The Private Bitswap Protocol

While developing a private version of Bitswap, we could develop a single-round protocol; however, we observe that the client would incur multiplicative communication overhead when it attempts to opportunistically query multiple peers who may have the desired content block. So, we choose to maintain the structure of a two-round protocol, with a lightweight PRIVATEWANTHAVE step and a relatively heavyweight PRIVATEBLOCK step. The PRIVATEWANTHAVE step can be run opportunistically with multiple candidate peers to determine whether a peer has a content block, while hiding the CID. We provide additional information in this step to reduce the overhead of the PRIVATEBLOCK step. While Private Set Intersection (PSI) schemes can be used in the first step, as in [16], they tend to be relatively communication intensive. In the PRIVATEBLOCK step, the client privately retrieves the block from the server peer using PIR. For the remainder of this section, recall the notation introduced in Section 6.

PRIVATEWANTHAVE. In this step, the client should learn whether the server has a block with a given CID, and if so, at which index of the server's table does this block lie. A trivial approach would have the server send the entire list of CIDs to the client, allowing the client to search the list for the index of the desired CID. This would reveal to the client other CIDs that are stored by the server peer. This leakage can be resolved by the server computing the hash of every entry in the list using a public hash function. Then, when the client receives the list of all (hashed) CIDs, they can compute the hash of the CID they desire to determine its index. Assuming the hash function is one-way, the client does not learn which other CIDs are stored by the server.

Alternatively, we can reuse the process for provider advertisements to transmit information to the client. In particular, for the PRIVATEWANTHAVE step, the client and server can engage in Algorithm 2, using the list of CIDs L in place of the joined provider store J. Instead of encrypting the provider peers' multiaddresses, we encrypt the index of the CID, and thus Line 14 changes to $T[b] \leftarrow T[b] \cup \mathsf{SE.Enc}(k, \mathbf{i})$.

In the PRIVATEWANTHAVE step, the server also returns the optimal PIR protocol to use in the PRIVATEBLOCK step, depending on the number of blocks they hold, enabling an adaptive approach to efficiently retrieve a block. Optimal choices of PIR schemes for the PRIVATEWANTHAVE step are the same as those for the private provider advertisement functionality (Section 6.2), and are discussed in Section 8. The security of PRIVATEWANTHAVE is then equivalent to the security of the private provider advertisement algorithm.

PRIVATEBLOCK. Following the PRIVATEWANTHAVE step, the client knows the index of the desired block in the table held by the server. Thus, it can issue an IndexPIR query to retrieve the block. The optimal choice of PIR protocol for this step depends on the number of blocks possessed by the peer. The optimal PIR protocol is sent to the client as part of the PRIVATEWANTHAVE response, enabling the client to follow that protocol to construct a PIR query for the PRIVATEBLOCK step.

Directly applying PIR to the table of content blocks is undesirable as it leaves the server peer vulnerable to an index enumeration attack. A client could query for an arbitrary index and retrieve a block of content without knowing its corresponding CID. Thus, we require some cryptographic pre-processing of the data. We reapply a technique that we have previously identified: using the CID as a pre-shared secret between clients and servers. Each block in the table is encrypted under a key derived from the corresponding CID, so that a client receiving a PIR response can only retrieve a content block if they know its CID. We summarize how the server handles a client's PIR query in Algorithm 3.

The server only obtains a PIR query from the client, and thus it can only break the security of PRIVATEBLOCK if it can break the underlying PIR scheme. On the part of the client, the security argument is equivalent to that for the private provider advertisements algorithm.

Algorithm 3 PRIVATEBLOCK using PIR

```
1: procedure PRIVATEBLOCKRESPONSE(qu, $D$)
2:     for $j \in \{1, 2, \cdots, m\}$ do
3:         $k \leftarrow \mathsf{KDF}(L[j])$                ▷ $L[j]$ is the CID.
4:         $D[j] \leftarrow \mathsf{SE.Enc}(k, D[j])$
5:     return PIR.RESPONSE(qu, $D$)
```

8. Analysis of PIR Protocols

In this section, we analyze options for PIR protocols to be used in our algorithms from Sections 5–7. Our goal is to select schemes with the best communication–computation cost tradeoff for each of the following use cases:

1) *Few rows with small payloads:* PIR over a table of 256 rows with roughly 1.5KB entries for private routing.
2) *Bins with varying size:* PIR over bins for provider advertisements and PRIVATEWANTHAVE steps.
3) *Many rows with large payloads:* PIR over 256KB payloads, representing content blocks in PRIVATEBLOCK.

We require that the protocol maintain the same number of rounds as the non-private protocols, to adhere to the same structure. Hence, the protocol chosen for each case should only require a single round. Additionally, the PIR protocol should be efficient even if a client issues only one PIR query to a specific server, which is the case for our applications. Thus, we cannot amortize costs over many queries.

We examine existing PIR protocols in Section 8.1. While these protocols are reasonable for our PRIVATEBLOCK step, applying any of them to the prior two use cases leads to infeasible costs. To sufficiently address our use cases, we propose two PIR protocols: PAILLIERPIR and RLWEPIR, in Section 8.2. Both are tailored for the IPFS setting but may be of independent interest in other systems with similar architectures. We evaluate the costs of our proposed and existing PIR schemes for each use case in Section 8.3.

8.1. Existing PIR Protocols

Modern PIR protocols typically consist of offline and online phases. In the offline phase, content that is independent of the query is exchanged (e.g., cryptographic keys and database-dependent hints) to accelerate the online phase. In the online phase, the server computes a response to a query. To adhere to having only one round of communication, we require that any offline phase which requires interaction with the client is conducted concurrently with the online phase.

TABLE 1: Lower bounds on the size of the key material, query, and response of various protocols. The second column denotes the size of client-specific cryptographic keys used in each approach. *In HintlessPIR and YPIR, the bound depends on the database size (1 GB here).

Protocol	Key Material	Query (Encrypted)	Response (Plaintext)	Response (Encrypted)
SealPIR [5]	1.6 MB	90 KB	10 KB	181 KB
FastPIR [2]	0.67 MB	64 KB	10 KB	65 KB
OnionPIR [45]	5.4 MB	64 KB	30 KB	128 KB
Spiral [41]	13 MB	28 KB	7.5 KB	20 KB
HintlessPIR* [33]	-	453 KB	32 KB	3080 KB
YPIR* [42]	462 KB	384 KB	1 B	12 KB
PAILLIERPIR	1.14 KB	0.38 KB	0.38 KB	0.76 KB
RLWEPIR	750 KB	64 KB	7.5 KB	65 KB
RLWEPIR3	192 KB	64 KB	7.5 KB	65 KB
RLWEPIR2	128 KB	64 KB	7.5 KB	65 KB

This precludes many PIR protocols for our use cases, as discussed in the full version [40].

We observe that PIR protocols that use client-specific cryptographic keys are the only suitable candidates (we elaborate on this in the full version). This leaves SealPIR [5], FastPIR [2], OnionPIR [45], Spiral [41], HintlessPIR [33], and YPIR [42]. Table 1 summarizes the high level costs of each of these protocols. Spiral provides small queries and responses, but requires the client to send a large, 13MB cryptographic key. In contrast, SealPIR, FastPIR, and OnionPIR involve smaller cryptographic keys but require larger queries and/or responses. HintlessPIR requires no client-specific keys, but has very large response sizes. YPIR requires very small keys, but is optimized for payloads smaller than our cases. Importantly, we deemed all existing schemes to be insufficient for our first and second use cases, due to incompatible payload sizes. Our PAILLIERPIR scheme supports smaller payloads, which is relevant for peer routing. Variants of RLWEPIR produce the smallest client-specific cryptographic keys, in comparison to the aforementioned schemes, which is useful for the private provider advertisements and PRIVATEWANTHAVE steps. We consider existing PIR schemes that support 256 KB payloads, for the PRIVATEBLOCK step and evaluate them in Section 8.3.

8.2. Tailored Protocols: PAILLIERPIR, RLWEPIR

We present protocols based on Paillier or RLWE encryption in Algorithm 4; these protocols use existing techniques from cryptographic folklore. Both protocols compute an inner product between an encrypted indicator vector, supplied by the client, and a table. The protocols are straightforward in nature, which both allows for ease of integration into large systems and demonstrates the lack of attention paid to such use cases in the PIR literature.

PAILLIERPIR. This protocol is based on the Paillier cryptosystem and uses homomorphic additions (PAILLIER.Add) and scalar multiplications (PAILLIER.ScalMult). The client only sends a small public key (1 KB) to the server along with the PIR query; this key includes the Paillier composite modulus and the chosen generator. For the PIR query, the client encrypts an indicator vector corresponding to the desired row, which produces one ciphertext for each row in the table. To reduce the query size, we use a technique from Beck [8]. Instead of directly sending the indicator ciphertext vector, the client samples a random ciphertext vector (Line 3). It decrypts this ciphertext vector to obtain a randomized plaintext vector (Line 6), uses this plaintext vector to mask each element of the indicator vector (Line 7), and sends the masked plaintext vector to the server. Instead of sending the randomized ciphertext vector (Line 3), the client sends a seed used to generate this vector. The communication cost for one query is approximately halved, as given by the sum of sizes of the seed and a vector of *plaintexts*.

Algorithm 4 PAILLIERPIR and RLWEPIR. The composite modulus of Paillier is denoted as M. The plaintext and ciphertext space of RLWE are denoted as R_p and $\mathcal{C}$, respectively. db represents a database with n rows and ℓ columns. We assume the size of each cell in the database is the size of one plaintext in the corresponding scheme, i.e., in PAILLIERPIR, $\mathtt{db} \in \mathbb{Z}_M^{n\times\ell}$ and in RLWEPIR, $\mathtt{db} \in R_p^{n\times\ell}$.

```
1: procedure PAILLIERPIR.Query($i$)                          ▷ $i \in \{1,\cdots,n\}$
2:     Sample Paillier secret key sk
3:     $\mathtt{ct} \xleftarrow{\$} \mathbb{Z}_{M^2}^n$
4:     for $j \in \{1,\ldots,n\}$ do
5:         $b_j \leftarrow \mathbb{I}[i=j]$
6:         $r_j \leftarrow$ PAILLIER.Dec(sk, ct[$j$])                ▷ $r_j \in \mathbb{Z}_M$
7:         p[$j$] $= b_j - r_j \mod M$                            ▷ $p \in \mathbb{Z}_M^n$
8:     return (sk, (ct, p))
9: procedure PAILLIERPIR.Response((ct, p), db)
10:    for $j \in \{1,\ldots,n\}$ do
11:        ct′[$j$] ← PAILLIER.Add(ct[$j$], p[$j$])
12:    ans ← [0] $*\ \ell$
13:    for $k \in \{1,\ldots,\ell\}$ do
14:        ans[$k$] ← PAILLIER.ScalMult(ct′[1], db[1][$k$])
15:        for $j \in \{2,\ldots,n\}$ do
16:            $t \leftarrow$ PAILLIER.ScalMult(ct′[$j$], db[$j$][$k$])
17:            ans[$k$] ← PAILLIER.Add(ans[$k$], $t$)
18:    return ans
19: procedure RLWEPIR.Query($i$)                             ▷ $i \in \{1,\cdots,n\}$
20:    sk, pk ← GenerateKeys()
21:    for $j \in \{1, 2..\lceil n/N \rceil\}$ do
22:        if $j * N \le i < (j+1) * N$ then
23:            ct[$j$] ← RLWE.Enc(sk, $X^{i \mod N}$)
24:        else
25:            ct[$j$] ← RLWE.Enc(sk, 0)
26:    return (sk, (pk, ct))
27: procedure RLWEPIR.Response((pk, ct), db)
28:    $C \leftarrow []$
29:    for $j \in \{1, 2..\lceil n/N \rceil\}$ do
30:        $t \leftarrow$ ObliviousExpand(pk, ct[$j$])               ▷ $t \in \mathcal{C}^N$
31:        Append ciphertexts in $t$ to $C$
32:    ans ← [0] $*\ \ell$
33:    for $k \in \{1,\cdots,\ell\}$ do
34:        ans[$k$] ← RLWE.ScalMult($C$[1], db[1][$k$])
35:        for $j \in \{2,\ldots,n\}$ do
36:            $t \leftarrow$ RLWE.ScalMult($C$[$j$], db[$j$][$k$])
37:            ans[$k$] ← RLWE.Add(ans[$k$], $t$)
38:    return ans
```

The server provides a PIR response to the client's query (Line 9). In doing so, it runs a scalar multiplication for each element in the database (Lines 14, 16). The computation overhead is thus proportional to the size of the database, making it impractical for large databases. Nonetheless, this protocol is useful for databases with few, small rows. Proofs of correctness and security for PAILLIERPIR are provided in Section B. In our evaluation, we choose a 3072-bit composite modulus M for 128-bit security [7], [24].

RLWEPIR. Our second construction is based on RLWE [36] and related additive homomorphic schemes [10], [11], [20]. Plaintexts are polynomials $p(x) \in R_p = \mathbb{Z}_p/(X^N+1)$. We make use of homomorphic addition (RLWE.Add) and scalar multiplication (RLWE.ScalMult). Our construction combines common techniques in the literature for RLWE-based PIR protocols [2], [5], [17], tuned to our setting. Compared to PAILLIERPIR, RLWEPIR has lower computation but higher communication costs. A client requesting row i generates the plaintext polynomial $p(X) = X^i$, and sends the encryption of this polynomial to the server as the PIR query. The server then derives an encrypted indicator vector from the query using an oblivious expansion technique [5], [13], [17].

Oblivious expansion occurs as follows: on an input ciphertext c which encrypts $p(X) = a_0 + a_1X + \cdots + a_{N-1}X^{N-1}$, it outputs N ciphertexts, $\{c_i\}_{i=0,\cdots,N-1}$ such that c_i encrypts the scalar a_i. An automorphism key, for a given $k \in \mathbb{Z}_{2N}^*$ allows the server to compute the encryption of $p(X^k)$ from the encryption of $p(X)$. Angel et al. [5] first proposed an oblivious expansion scheme requiring a client to send $\log_2 N$ automorphism keys to the server [5], [13]. More recently, de Castro et al. [17] propose a change that requires only a constant number of automorphism keys. While this reduces the communication cost, the server must perform more automorphisms, thereby increasing the computation cost. Server-side oblivious expansion is denoted by ObliviousExpand in Line 30 and the client-side procedure to generate the necessary secret and auxiliary keys is GenerateKeys in Line 20.

In addition to RLWEPIR, we develop two variants of oblivious expansion following de Castro et al. [17]. RLWEPIR2 and RLWEPIR3 involve two and three automorphism keys, respectively. RLWEPIR sends more keys; we detail these variants and their application to the first and second use cases in Section D. In all variants of RLWEPIR, we use a polynomial modulus degree of $N = 4096$ and a plaintext modulus of 40961. The ciphertext modulus is the composite of 54-bit and 55-bit primes. These parameters ensure correctness and provide 128-bit security. The security of our protocol follows directly from prior work [5], [17]. Generating fewer automorphism keys increases the number of operations that the server performs, but does not affect the security of the protocol.

8.3. Evaluation

Experimental Setup. We evaluate the total communication cost and the server-side runtime for our three use cases, while increasing the size of the table for each use case. We perform our peer and provider routing experiments on an AMD Ryzen 9 7900X with 12 cores and 32 GB RAM, whereas our content retrieval experiment is run on an AMD EPYC 7302 with 16 cores and 6 TB RAM. Thus, for all three use cases, the entire table is stored in memory. We plot averages and standard deviations for both overheads over $N = 10$ runs, in Figure 2.

The *Runtime* metric denotes the server-side runtime to process a single private query in each use case. We selectively parallelize our bottlenecks for peer and provider routing to minimize runtimes. Client-side runtime is typically small in comparison to the server-side runtime. The *Communication* metric denotes the total round-trip network cost of one private query. For peer routing, we evaluate these overheads only for a single "hop" in the iterative routing process; in a complete lookup, the client would query server peers in multiple hops to reach a target peer. We then analyze the end-to-end latency for a complete lookup.

Private Peer Routing. We recall this case from Section 5.3, wherein we query a routing table of up to 256 rows with 1.5 KB entries. The number of rows in the routing table is bound by the bit length of peer addresses in IPFS. First, in order of increasing communication cost, we find PAILLIERPIR followed by RLWEPIR2, RLWEPIR3, and RLWEPIR. Increasing the number of automorphism keys sent increases the communication cost across the latter three schemes, as expected. Second, each of our new PIR algorithms has a constant communication overhead, even as the number of rows increases. This is because the client issues its query assuming the worst case scenario, namely that the server's RT has 256 buckets. Third, we also plot the communication overhead when the server sends the entire normalized RT, in the TRIVIALPIR plot, and observe that PAILLIERPIR outperforms TRIVIALPIR as long as the normalized RT has more than approximately 64 rows. However, TRIVIALPIR may increase the risk of eclipse attacks, and is thus unsuitable for our threat model. Finally, we find that RLWEPIR has the smallest server runtime, followed by RLWEPIR3, RLWEPIR2, and PAILLIERPIR. Our RLWEPIR3 and RLWEPIR2 schemes are bottlenecked by the oblivious expansion step; *decreasing* the number of automorphism keys requires the server to perform additional automorphisms, which increases the server runtime. Our PAILLIERPIR scheme scales poorly with increasing number of rows, as expected. We recommend RLWEPIR3 for this use case, as it provides the best communication–computation tradeoff for our goal.

Private Provider Advertisement. We evaluate PIR protocols for private provider advertisements (Section 6); our results can be extended to the PRIVATEWANTHAVE step (Section 7). We group multiple provider advertisements into bins and perform PIR over bins. We use 4096 bins for our RLWE-based protocols, since the client can encode a query as large as the polynomial modulus degree $N = 4096$ within one RLWE ciphertext with no extra cost. We use 256 bins for PAILLIERPIR. These bins store increasing numbers of provider advertisements, varying from 8 k–192 k. We conser-

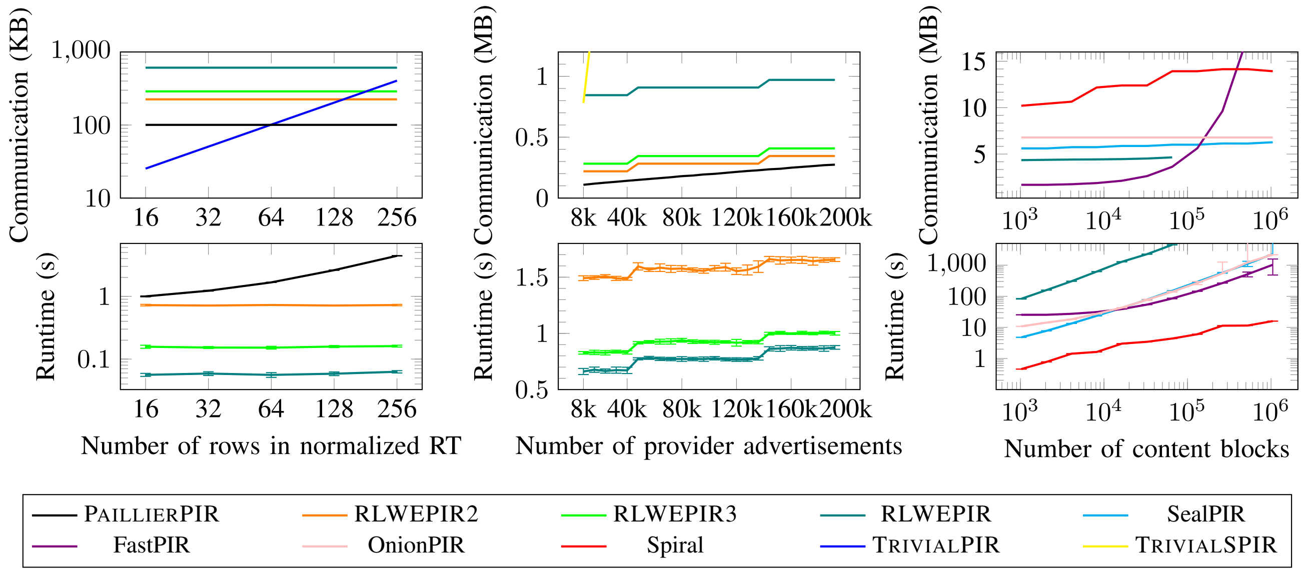


Figure 2: Communication costs (top graphs) and computation costs (bottom graphs) for private peer routing (left graphs), private provider advertisements (middle graphs), and private content retrieval (right graphs).

vatively overestimate the size of each provider advertisement to 100 B. Thus, our provider advertisement store is between 800 kB–18.75 GB.

First, as expected, our RLWEPIR2, RLWEPIR3 and RLWEPIR schemes have relatively increasing communication costs and decreasing server runtimes, similar to the peer routing use case. Second, the communication costs and runtimes of our RLWE schemes are in the form of a step function over the number of provider advertisements; these overheads increase sharply at approximately 40k and 140k advertisements. As bins fill with more provider advertisements, the server needs to compute additional ciphertexts to encode a bin in a PIR response, resulting in the sharp increases in both overheads. Third, the communication cost of PAILLIERPIR is lower than that of RLWEPIR, as expected. The *number* of rows (bins) for PAILLIERPIR, and thus the size of the PIR query, is identical to the peer routing case. As we increase the *size* of each row, the size of the PIR query remains constant, whereas the size of the PIR response increases linearly, and thus the total communication cost increases linearly. PAILLIERPIR is an order of magnitude slower than RLWEPIR, taking between 0.5–6 minutes, which is consistent with existing results [3], [34]. Thus, we do not plot its runtime in Figure 2, and do not recommend it for this use case. Fourth, we consider a trivial symmetric PIR approach, namely TRIVIALSPIR (defined in Section 6.2); this protocol quickly exceeds the communication costs of our RLWEPIR and PAILLIERPIR protocols. In conclusion, comparing the variants of RLWEPIR, we again recommend using RLWEPIR3.

Private Content Retrieval. Finally, we consider overheads for content blocks sent in the PRIVATEBLOCK step. We vary the number of content blocks stored by the server peer from 10^3 to 10^6, resulting in a content store size of 256 MB–256 GB. In the PRIVATEWANTHAVE step, the server will output the PIR protocol to use for the PRIVATEBLOCK step. So, we recommend multiple protocols for this use case, depending on the size of the content store. PAILLIERPIR and RLWEPIR have been designed for the two aforementioned use cases and provide a poor communication–computation tradeoff for this case. Thus, excluding them, we compare the remaining options from Table 1.

We observe the following from Figure 2. Spiral enables reasonably low runtimes (between 0.4–20 s) for the ranges of database sizes that we consider, while incurring a high communication overhead. SealPIR has almost half as much communication overhead as Spiral, and is at least an order of magnitude slower. At the other extreme of the communication–computation tradeoff, FastPIR has the lowest communication overhead until 100,000 blocks, and this overhead then scales linearly for larger databases. FastPIR is noticeably slower than other protocols until 10,000 blocks.

We thus recommend Spiral for the PRIVATEBLOCK step in Bitswap, due to its significantly lower computation overhead. Alternately, to reduce communication costs, lightweight peers that hold less than 10,000 blocks may opt for FastPIR, whereas heavyweight peers can opt for SealPIR.

While all tables were stored in memory for our experiments, machines with smaller RAM sizes may frequently read the disk for large content stores and provider advertisement stores. We can vertically partition the table, cache each partition or column, and use each column only once to compute the PIR response (Line 33–Line 37, Algorithm 4).

End-to-end latency analysis. A client retrieves a content block with a CID c in three steps. First, the client opportunistically searches for the content block, by running the PRIVATEWANTHAVE step *in parallel* among its neighbours, and thus the total computational latency for this step would be the same as that in Figure 2. The client does not usually find the block in the first step; they then learn the block's provider peer's ID p, by running a peer routing query for peers closest to Hash(c) and simultaneously, running a provider routing query for providers for c. Third, the client finds the provider's multiaddress, by running another peer routing query for peers closest to p.

For peer routing, each *hop*, which takes the client one bit closer to the target peer ID, is sequential. Thus, the total server-side computational latency for private peer routing would be the path length multiplied by the latency for one hop ($\approx$ 60 ms from Figure 2). The client's overheads to compute the PIR query and process the PIR response for each hop would also similarly scale with the path length. Our path length does not increase over standard IPFS in practice (Section 5.2). For the IPFS network of size $n = 200,000$ [58], the average path length in Kademlia is $\log_2(n) = 17.6$ [38], and so, the total server-side computational latency of a private peer routing query would be $\approx$ 1 s. The client would experience this latency when running a peer routing query to learn the multiaddress of a provider peer.

From Figure 2, provider routing takes longer ($t_2 \approx$ 600–900 ms) than peer routing ($t_1 \approx$ 60 ms). Thus, in the aforementioned second step, when the client runs both peer and provider routing queries in parallel, they will *potentially* know a provider peer ID p at the end of a single hop of a peer routing query (t_1). However, they will *not* know that peer p is in fact a provider, until they get the provider routing response (t_2). Instead of waiting ($\approx$ 540–840 ms) for a provider routing response for each hop, the client can proceed with the next hop of the private peer and provider routing queries. This design decision avoids bottle-necking the computational latency for each hop to the latency of the slower query, namely the private provider routing query, at the cost of redundantly querying other peers in case the client would have found a provider in a previous hop. The end-to-end latency includes both the total computational latency as well as the network round-trip time (RTT) for each hop, which we represent by τ_i. The worst case end-to-end latency $O(T)$ to find a provider peer ID is:

$$O(T) = \underbrace{\log_2(n) \cdot t_1}_{\text{Peer routing}} + \underbrace{\Sigma_{i=1}^{\log_2(n)} \tau_i}_{\text{Network RTT}} + \underbrace{\max(\tau_i) + t_2}_{\text{Provider routing}}$$

For content retrieval, each file is decomposed into a Merkle DAG of content blocks (Section 2.1, [26], [49]). The height of this DAG is logarithmic in the file size, and it bounds the number of sequential retrievals for which our private content retrieval algorithm's latency may cascade. All blocks within the same level of the tree can be fetched in parallel from the same or different provider servers. We consider two types of churn in the network: node churn (when a peer enters or leaves the network) and content churn (when a content block is added or removed by its provider). In the full version, we outline how these types of churn impact each of our algorithms.

Use Case	Recommendation	Communication	Runtime
Peer Routing	RLWEPIR3	295 KB	60 ms
Provider Advertisements	RLWEPIR3	430 KB	<1 s
Content Retrieval	Spiral FastPIR SealPIR	10–15 MB 1.7–2.1 MB 6.1–6.4 MB	0.5–15s 25–38s 200–2250s

TABLE 2: Recommended PIR Protocols. Measurements for FastPIR and SealPIR are for 10^3–10^4 and 10^5–10^6 blocks, respectively.

9. Discussion & Takeaways

We summarize our recommended PIR protocols for each use case in Table 2. While we do incur an overhead, our private algorithms for each use case are agnostic of the ingredient protocol. Hence, improved PIR protocols in future work can be plugged into our algorithms.

Based on our design of PIR protocols for the distributed setting, we present the following takeaways.

- Symmetric PIR: Robust symmetric encryption can be used to instantiate a symmetric PIR scheme, by interpreting the index of the record to fetch, which is the CID in our case, as a pre-shared secret between the client and server.
- IndexPIR: Given that CIDs are themselves hashes, we construct an IndexPIR scheme, where the client can compute the bin index by taking a prefix of the CID. Consequently, an extra round to privately fetch the index is unnecessary.
- RLWE: Reducing the number of automorphism keys sent by the client, by extending existing oblivious expansion techniques, allows us to gain runtimes comparable to RLWEPIR with much lower communication overhead.

Non-collusion Assumptions in DHTs. Distributed settings are often seen as ideal applications of non-collusion assumptions, which can enable more efficient algorithms. For example, Mazmudar et al. [39] leverage IT-PIR for content retrieval in DHTPIR. They observed that existing work in robust DHTs [60] relied on the ability to partition peers into quorums: groups of peers such that each group has a bounded proportion of malicious peers. They exploit quorums to satisfy non-collusion assumptions for IT-PIR.

However, we found that instantiating these assumptions in IPFS was challenging since robust DHTs relied on theoretical quorum formation protocols. Mazmudar et al. [39] suggest forming quorums using the Commensal Cuckoo Rule (CCR) [54]. CCR requires peers to repeatedly leave and join the network until they join a sufficiently big quorum, and thereby, drastically increases the node churn. Even if one were able to efficiently instantiate quorums, applying IT-PIR still appears infeasible, since it requires all peers within a quorum to share files. In IPFS, peers decide which files to store, whereas IT-PIR requires them to increase storage by a factor of the size of a quorum. This introduces a prohibitive overhead of copying files every time a peer changes quorums. Thus, despite appearing to be a natural setting for threshold cryptography, we conclude that the assumptions required in prior work do not hold for IPFS.

Privacy Engineering Takeaways. We outline key takeaways on integrating PETs into an existing non-private codebase for a distributed system.

- We use existing integration tests, along with observability tools such as distributed tracing [46], [48], to easily track how private information is used within existing functions in the codebase, and to determine which functions need to be reimplemented privately.
- Backwards compatibility enables gradual deployment of private implementations as more peers update their code. Instead of altering existing functionality, we add private versions through careful integration with Protobufs [25]. We could then test both private and non-private versions simultaneously for correctness. We highlight that PETs for DHTs should not only be backwards compatible at a protocol level, but should also handle data that is already published in the DHT.
- We design our private interfaces to be sufficiently generic so that we can instantiate different PET implementations, such as PAILLIERPIR and RLWEPIR, under them.
- Data structures in the codebase often differ from the model used in PETs. We join multiple tables in order to perform PIR. This is distinct from any preprocessing operations that the PET itself requires.
- Dealing with randomization: For peer and provider routing, the client sends the same message to each server in its path to the target. However, our private versions involve the client sending a different ciphertext message, namely a PIR query, to each server peer. This discrepancy may arise in both centralized (client-server) or distributed (P2P) architectures, and for other PETs such as differential privacy, and addressing it may require significant redesign to avoid code smell.

Acknowledgements

This research has been funded by the *RFP-014: Private retrieval of data* grant from Protocol Labs (PL) [31], [53]. We are grateful to Will Scott from PL for his support throughout this project. We thank Dennis Trautwein and Guillaume Michel from PL for their assistance in integrating Peer2PIR algorithms within the LibP2P library. We also thank our anonymous shepherd for their helpful feedback. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo. Any findings or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of PL.

References

[1] Ishtiyaque Ahmad, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Pantheon: Private retrieval from public key-value store. *Proc. VLDB Endow.*, 16(4):643–656, 2022.

[2] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 313–329. USENIX Association, July 2021.

[3] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication–Computation Trade-offs in PIR. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1811–1828, 2021.

[4] Andris Ambainis. Upper Bound on the Communication Complexity of Private Information Retrieval. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, ICALP 1997, pages 401–407. Springer, 1997.

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 962–979, 2018.

[6] Michael Backes, Ian Goldberg, Aniket Kate, and Tomas Toft. Adding query privacy to robust DHTs. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12. Association for Computing Machinery, 2012.

[7] Elaine Barker. Recommendation for key management: Part 1 – general. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, 2020.

[8] Martin Beck. Randomized decryption (RD) mode of operation for homomorphic cryptography - increasing encryption, communication and storage efficiency. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 220–226, 2015.

[9] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $o(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 261–270, 2002.

[10] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886, Berlin, Heidelberg, 2012. Springer.

[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[12] Ling Cao and Yue Li. IPFS keyword search based on double-layer index. In Zhiyuan Zhu and Fengxin Cen, editors, *International Conference on Electronic Information Engineering and Computer Communication (EIECC 2021)*, volume 12172, page 1217209. International Society for Optics and Photonics, SPIE, 2022.

[13] Hao Chen, Ilaria Chillotti, and Ling Ren. Onion ring ORAM: Efficient constant bandwidth oblivious RAM from (leveled) TFHE. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 345–360, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[15] Erik Daniel, Marcel Ebert, and Florian Tschorsch. Improving Bitswap privacy with forwarding and source obfuscation. In *2023 IEEE 48th Conference on Local Computer Networks (LCN)*, pages 1–4. IEEE, 2023.

[16] Erik Daniel and Florian Tschorsch. Privacy-enhanced content discovery for Bitswap. In *2023 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2023.

[17] Leo de Castro, Kevin Lewi, and Edward Suh. WhisPIR: Stateless private information retrieval with low communication. Cryptology ePrint Archive, 2024/266, 2024. https://eprint.iacr.org/2024/266.

[18] Alfonso De la Rocha, David Dias, and Yiannis Psaras. Accelerating content routing with Bitswap: A multi-path file transfer protocol in IPFS and Filecoin. Technical report, Protocol Labs, 2021.

[19] Daniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: Practical Multi-Server PIR. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, CCSW '14, pages 45–56. Association for Computing Machinery, 2014.

[20] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, 2012/144, 2012. https://eprint.iacr.org/2012/144.

[21] Fleek. Deployments on Fleek. https://docs.fleek.co/hosting/site-deployment/. Accessed on 2025-04-07.

[22] Fleek. Space daemon. https://github.com/FleekHQ/space-daemon. Accessed on 2025-04-07.

[23] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully Homomorphic Encryption with Polylog Overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.

[24] Damien Giry. Cryptographic key length recommendation. https://www.keylength.com/en/4/. Accessed on 2025-04-07.

[25] Google. What problems do protocol buffers solve? https://protobuf.dev/overview/#solve.

[26] Marcel Gregoriadis. Analysis and Comparison of Deduplication Strategies in IPFS. Master's thesis, Humboldt-Universität zu Berlin, 2023.

[27] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3889–3905, Anaheim, CA, 2023. USENIX Association.

[28] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust Authenticated-Encryption AEZ and the Problem That It Solves. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 15–44, Berlin, Heidelberg, 2015. Springer.

[29] P. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484 (Proposed Standard), October 2018.

[30] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858 (Proposed Standard), May 2016. Updated by RFC 8310.

[31] Karola Kirsanow and Will Scott. Announcing rfp-014: The one with private retrieval. https://research.protocol.ai/blog/2022/announcing-rfp-014-the-one-with-private-retrieval/, 2022. Accessed on 2025-04-07.

[32] Protocol Labs. How IPFS works. https://docs.ipfs.tech/concepts/how-ipfs-works/, 2023. Accessed on 2024-02-04.

[33] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. Hintless Single-Server Private Information Retrieval. In *Advances in Cryptology – CRYPTO 2024*, pages 183–217. Springer, 2024.

[34] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Proceedings of the 8th International Conference on Information Security*, ISC'05, page 314–328, Berlin, Heidelberg, 2005. Springer-Verlag.

[35] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Haixin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu. An end-to-end, large-scale measurement of DNS-over-Encryption: How far have we come? In *Proceedings of the 2019 Internet Measurement Conference (IMC'19)*, pages 22–35, 2019.

[36] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, Berlin, Heidelberg, 2010. Springer.

[37] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1723–1740, Boston, MA, 2022. USENIX Association.

[38] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, Berlin, Heidelberg, 2002. Springer-Verlag.

[39] Miti Mazmudar, Stan Gurtler, and Ian Goldberg. Do You Feel a Chill? Using PIR against Chilling Effects for Censorship-resistant Publishing. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, WPES '21, page 53–57, New York, NY, USA, 2021. Association for Computing Machinery.

[40] Miti Mazmudar, Shannon Veitch, and Rasoul Akhavan Mahdavi. Peer2PIR: Private Queries for IPFS. *arXiv:2405.17307 [cs.CR]*, 2024. https://arxiv.org/abs/2405.17307.

[41] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 930–947, 2022.

[42] Samir Jordan Menon and David J. Wu. YPIR: High-Throughput Single-Server PIR with Silent Preprocessing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5985–6002, 2024.

[43] Guillaume Michel. Double-hashing as a way to increase reader privacy. https://www.youtube.com/watch?v=VBlx-VvIZqU. Accessed on 2025-04-07.

[44] Guillaume Michel. IPIP-373: Double Hash DHT Spec. https://github.com/ipfs/specs/pull/373. Accessed on 2025-04-07.

[45] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2292–2306, New York, NY, USA, 2021. Association for Computing Machinery.

[46] The OpenTelemetry Authors. What is opentelemetry? https://opentelemetry.io/docs/concepts/what-is-opentelemetry/. Accessed on 2025-04-07.

[47] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, Berlin, Heidelberg, 1999. Springer.

[48] Austin Parker, Daniel Spoonhower, Jonathan Mace, and Rebecca Isaacs. *Distributed Tracing in Practice*. O'Reilly Media, Inc., 2020.

[49] Protocol Labs. Chunking. https://docs.ipfs.tech/concepts/file-systems/#chunking. Accessed on 2025-04-07.

[50] Protocol Labs. Ecosystem directory. https://ecosystem.ipfs.tech/. Accessed on 2025-04-07.

[51] Protocol Labs. Privacy and Encryption. https://docs.ipfs.tech/concepts/privacy-and-encryption/. Accessed on 2025-04-07.

[52] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer. Total Eclipse of the Heart – Disrupting the InterPlanetary File System. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3735–3752, Boston, MA, August 2022. USENIX Association.

[53] Will Scott. Private retrieval grant 2023 roundup. https://research.protocol.ai/blog/2023/private-retrieval-grant-2023-roundup/, 2023. Accessed on 2025-04-07.

[54] Siddhartha Sen and Michael J. Freedman. Commensal cuckoo: secure group partitioning for large-scale services. *SIGOPS Oper. Syst. Rev.*, 46(1):33–39, 2012.

[55] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, 2001.

[56] Dennis Trautwein. 2022-09-20 Hydras Analysis. https://github.com/probe-lab/network-measurements/blob/master/results/rfm21-hydras-performance-contribution.md#provider-distribution. Accessed on 2025-04-07.

[57] Dennis Trautwein. Nebula. https://github.com/dennis-tra/nebula. Accessed on 2025-04-07.

[58] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. Design and evaluation of IPFS: a storage layer for the decentralized web. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, pages 739–752, New York, NY, USA, 2022. Association for Computing Machinery.

[59] Qiyan Wang, Prateek Mittal, and Nikita Borisov. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10. Association for Computing Machinery, 2010.

[60] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Practical robust communication in DHTs tolerating a byzantine adversary. In *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 263–272. IEEE, 2010.

Appendix A. Convergence of Private Routing

Lemma A.1 (Convergence of Private Routing)**.** *If* KADEMLIA *converges in $\lceil \log n \rceil + c_1$ hops, then* PRIV-KADEMLIA *converges in $\lceil \log n \rceil + c_2$ hops for some constants c_1, c_2.*

Proof. Consider only the number of hops it takes to reach the target peer given the *closest* peer in each iteration of the protocol. The argument applies equally to the next $k-1$ closest peers.

If the index of the target bucket is greater than the number of rows in the original routing table, then PRIV-KADEMLIA returns the k closest nodes to the server. This is equivalent to the output of KADEMLIA, and as such, does not increase the total number of hops. Therefore, we are only concerned with the case where the target index is less than the total number of rows in the original RT and the corresponding bucket is empty.

For each hop, if the target peer ID is in a non-empty bucket, then PRIV-KADEMLIA returns the same value as KADEMLIA. If this holds for each hop, then by assumption, PRIV-KADEMLIA converges in $\lceil \log n \rceil + c_1$ hops. Otherwise, there are some hops for which the target peer ID is in an *empty* bucket. The bound on KADEMLIA is assumed to follow by the fact that if the target ID is in a non-empty bucket, then the lookup procedure will return a peer which is at least half as close (i.e., whose distance is at least one bit shorter) than the current peer. If this is always the case, then the procedure terminates in $\log n$ steps. Therefore, for the algorithm to terminate in $\lceil \log n \rceil + c_1$, the number of times that the lookup procedure does not return a peer which is at least half as close than the current peer must be bounded by the constant c_1. That is, the number of times the target ID is in an empty bucket must be bounded by c_1. Suppose then, that there is some constant, c_1, number of hops for which the target peer ID is in an *empty* bucket and PRIV-KADEMLIA returns a different peer than KADEMLIA.

Thus, the problem reduces to the case where the following conditions hold: there are c_1 hops wherein the target index corresponds to an empty bucket which is less than the total number of rows in the original routing table. In this case, PRIV-KADEMLIA returns first the closer nodes to itself (between the target index and the longest CPL in the RT) and then resorts to returning nodes farther from itself (between the target index and 0).

If PRIV-KADEMLIA returns a node closer to itself, then this must necessarily be no farther from the target node than the server node itself. By assumption, this occurs at most c_1 times, and so PRIV-KADEMLIA converges in at most $\lceil \log n \rceil + c_1^2$ hops. Otherwise, PRIV-KADEMLIA returns a peer which is farther from itself; however, the returned peer must have the same CPL with the target node as the peer which would have been returned by KADEMLIA. Therefore, this case additionally does not increase the number of hops by more than a constant amount. □

We fix a seed in our implementation, thus selecting all peers deterministically. Since all peers within a range are equally likely to be a given distance from the target, the proof of convergence is identical. As the seed is independent of the target peer ID, the privacy of the client is preserved.

Appendix B. Correctness and Security of PaillierPIR

B.1. Preliminaries

We define our PIR protocols below, drawing upon [27], while omitting the Setup algorithm.

Definition B.1 (PIR protocol)**.** *A* PIR protocol *is a triplet of algorithms* $(\mathsf{Query}, \mathsf{Response}, \mathsf{Extract})$ *over record space $\mathcal{D}$ and database size n, satisfying the following:*

- $\mathsf{Query}(i) \to (\mathtt{st}, qu)$*: takes as input an index $i \in [n]$ and outputs a client state* $\mathtt{st}$ *and a query* qu*.*
- $\mathsf{Response}(db, qu) \to \mathtt{ans}$*: given the database* db *and query* qu*, outputs an answer* $\mathtt{ans}$*.*
- $\mathsf{Extract}(\mathtt{st}, \mathtt{ans}) \to d$*: given client state* $\mathtt{st}$ *and an answer* $\mathtt{ans}$*, outputs a record $d \in \mathcal{D}$.*

We say that a PIR protocol is correct *if, for any database $db = \{d_i\} \in \mathcal{D}^n$ and any $i \in [n]$,*

$$\Pr\left[d_i = d'_i \;\middle|\; \begin{array}{c} (\mathtt{st}, qu) \leftarrow \mathsf{Query}(i), \\ \mathtt{ans} \leftarrow \mathsf{Response}(db, qu), \\ d'_i \leftarrow \mathsf{Extract}(\mathtt{st}, \mathtt{ans}) \end{array} \right] \geq 1 - \delta,$$

for some negligible δ. Moreover, we say that the PIR protocol is ϵ-secure, *if for all polynomial-time adversaries $\mathcal{A}$ and for all $i, j \in [n]$,*

$$\begin{aligned} &|\Pr[\mathcal{A}(1^n, qu) = 1 : (\mathtt{st}, qu) \leftarrow \mathsf{Query}(i)] \\ &- \Pr[\mathcal{A}(1^n, qu) = 1 : (\mathtt{st}, qu) \leftarrow \mathsf{Query}(j)]| \leq \epsilon, \end{aligned}$$

for negligible ϵ.

Definition B.2 (Semantic security)**.** *A public-key encryption scheme* $\mathsf{PKE} = (\mathrm{KGen}, \mathrm{Enc}, \mathrm{Dec})$ *is ϵ-**IND-CPA** secure, if for any polynomial-time adversary $\mathcal{A}$, we have that*

$$\mathsf{Adv}^{\mathsf{indcpa}}_{\mathsf{PKE}}(\mathcal{A}) := \left| \Pr[\mathsf{G}^{\mathsf{indcpa}}_{\mathsf{PKE}}(\mathcal{A}) = 1] - 1/2 \right| \leq \epsilon,$$

where $\mathsf{G}^{\mathsf{indcpa}}_{\mathsf{PKE}}$ is defined as in Figure 3.

$\underline{\mathsf{G}^{\mathsf{indcpa}}_{\mathsf{PKE}}(\mathcal{A})}$:

```
1 $(pk, sk) \xleftarrow{\$} \mathrm{KGen}()$
2 $m_0, m_1 \xleftarrow{\$} \mathcal{A}(pk)$
3 $b \xleftarrow{\$} \{0,1\}$
4 $c \xleftarrow{\$} \mathrm{Enc}(pk, m_b)$
5 $b' \leftarrow \mathcal{A}(pk, c)$
6 return $[\![b = b']\!]$
```

$\underline{\mathsf{G}^{\mathsf{pirsec}}_{\mathsf{PIR}}(\mathcal{A})}$:

```
1 $i_0, i_1 \xleftarrow{\$} [n]$
2 $b \xleftarrow{\$} \{0,1\}$
3 $\mathtt{st}, \mathtt{qu} \xleftarrow{\$} \mathsf{Query}(i_b)$
4 $b' \leftarrow \mathcal{A}(\mathtt{qu})$
5 return $[\![b = b']\!]$
```

Figure 3: Security games for PIR and semantic security

We rely on the semantic security of the Paillier cryptosystem, due to [47]. We can rewrite the definition of PIR security in a game-based notion as follows.

Definition B.3 (PIR security)**.** *A PIR scheme* $\mathsf{PIR} = (\mathsf{Query}, \mathsf{Response}, \mathsf{Extract})$ *is ϵ-secure, if for any polynomial-time adversary $\mathcal{A}$, we have that*

$$\mathsf{Adv}^{\mathsf{pirsec}}_{\mathsf{PIR}}(\mathcal{A}) := \left|\Pr[\mathsf{G}^{\mathsf{pirsec}}_{\mathsf{PIR}}(\mathcal{A}) = 1] - 1/2\right| \leq \epsilon,$$

where $\mathsf{G}^{\mathsf{pirsec}}_{\mathsf{PIR}}$ is defined as in Figure 3.

B.2. Proof of Correctness

Theorem B.1. *Let M denote the composite modulus of Paillier. For any $n, \ell \in \mathbb{N}$, any database $db \in \mathbb{Z}_M^{n\times\ell}$ any $i \in \{1, 2, \cdots, n\}$, if*

$$(\mathit{sk}, (\mathit{ct}, p)) \leftarrow \textsc{Paillier}\mathsf{PIR}.\mathsf{Query}(i) \quad (1)$$

$$\mathit{ans} \leftarrow \textsc{Paillier}\mathsf{PIR}.\mathsf{Response}(db, (\mathit{ct}, p)) \quad (2)$$

$$d' \leftarrow \textsc{Paillier}\mathsf{PIR}.\mathsf{Extract}(\mathit{sk}, \mathit{ans}) \quad (3)$$

then $d' = db[i]$.

Proof. Given that $d', \mathrm{db}[i] \in \mathbb{Z}_M^{\ell}$, we will prove the theorem by proving that $d'[k] = \mathrm{db}[i][k]$ for every $k \in [\ell]$. First, using the additive properties of Paillier, we see that for every $j \in [n]$,

$$\begin{aligned} & \textsc{Paillier}.\mathsf{Dec}(\mathtt{sk}, \mathtt{ct}'[j]) && (4)\\ &= \textsc{Paillier}.\mathsf{Dec}(\mathtt{sk}, \textsc{Paillier}.\mathsf{Add}(\mathtt{ct}[j], \mathtt{p}[j])) && (5)\\ &= \textsc{Paillier}.\mathsf{Dec}(\mathtt{sk}, \mathtt{ct}[j]) + \mathtt{p}[j] && (6)\\ &= r_j + (b_j - r_j) = b_j && (7)\end{aligned}$$

where operations are modulo M in the above equations. Moreover, for every $k \in [\ell]$,

$$\begin{aligned} d'[k] &= \textsc{Paillier}.\mathsf{Dec}(\mathtt{sk}, \mathtt{ans}[k]) && (8)\\ &= \sum_{j=1}^{n} \mathrm{db}[j][k] \cdot \textsc{Paillier}.\mathsf{Dec}(\mathtt{sk}, \mathtt{ct}'[j]) && (9)\\ &= \sum_{j=1}^{n} \mathrm{db}[j][k] \cdot b_j = \mathrm{db}[i][k] && (10)\end{aligned}$$

where the last equation is due to the fact that $b_j = \mathbb{I}[j = i]$. Combining the result for all values of $k \in [\ell]$, we observe that $d' = \mathrm{db}[i]$. □

B.3. Proof of Security

Theorem B.2. *Let* Query *be as defined in* $\textsc{Paillier}\mathsf{PIR}.\mathsf{Query}$ *of Algorithm 4. For any adversary $\mathcal{A}$, we construct an algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{pirsec}}_{\textsc{Paillier}\mathrm{PIR}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{indcpa}}_{\textsc{Paillier}}(\mathcal{B}).$$

Proof. We reduce the security of the PIR scheme to the $\mathsf{IND\text{-}CPA}$ security of the Paillier cryptosystem. In particular, for any adversary $\mathcal{A}$ attacking the PIR security of $\textsc{Paillier}\mathsf{PIR}$, we construct a new adversary $\mathcal{B}$ attacking the $\mathsf{IND\text{-}CPA}$ security of Paillier. Let $d \xleftarrow{\$} \{0,1\}$ and $b \xleftarrow{\$} \{0,1\}$ denote the challenge bits in the pirsec and indcpa games respectively.

The adversary $\mathcal{B}$ selects two random indicator vectors ℓ_0, ℓ_1 and sends these as messages to the Enc challenge which returns $c \leftarrow \mathrm{Enc}(pk, \ell_b)$, where b is the challenge bit for $\mathcal{B}$. Now, $\mathcal{B}$ selects a random $r \xleftarrow{\$} \mathbb{Z}_M^n$ from the plaintext space, and computes two ciphertexts: $\widehat{r} = \textsc{Paillier}.\mathrm{Enc}(pk, r)$, and $ct_1 = \textsc{Paillier}.\mathsf{Add}(c, \widehat{r})$. $\mathcal{B}$ then calls $\mathcal{A}$ on the following input: $\mathtt{qu} \leftarrow (ct_1, -r)$. Upon receiving $\mathcal{A}$'s decision bit, say d', $\mathcal{B}$ outputs $b' = d'$.

Now the advantage of $\mathcal{B}$ is:

$$\mathsf{Adv}^{\mathsf{indcpa}}_{\textsc{Paillier}}(\mathcal{B}) = |\Pr[d' = 1 \mid b = 0] - \Pr[d' = 1 \mid b = 1]|.$$

The adversary $\mathcal{B}$ perfectly simulates the pirsec game to adversary $\mathcal{A}$ as it provides a query consisting of a ciphertext, $ct_1 = \textsc{Paillier}.\mathsf{Add}(c, \widehat{r})$, and a random plaintext $pt_1 = -r$, such that:

- ct_1 is indistinguishable from a random ciphertext. This is because ct_1 is simply the sum of the encryption of the ciphertext vector c and the random plaintext r:

$$\begin{aligned} ct_1 &= \textsc{Paillier}.\mathsf{Add}(c, \widehat{r})\\ &= \textsc{Paillier}.\mathsf{Add}(c, \textsc{Paillier}.\mathsf{Enc}(pk, r))\\ &= \textsc{Paillier}.\mathsf{Enc}(pk, \textsc{Paillier}.\mathsf{Add}(c, r))\end{aligned}$$

  Since c is indistinguishable from a ciphertext chosen at random, and r is a plaintext chosen at random, their sum is indistinguishable from a ciphertext chosen at random.
- $ct' = \textsc{Paillier}.\mathsf{Add}(ct_1, pt_1)$ (Line 11) is an encryption of the indicator vector ℓ_b, as follows:

$$\begin{aligned} ct' &= \textsc{Paillier}.\mathsf{Add}(\textsc{Paillier}.\mathsf{Add}(c, \widehat{r}), -r)\\ &= \textsc{Paillier}.\mathsf{Add}(\textsc{Paillier}.\mathsf{Enc}(pk, \textsc{Paillier}.\mathsf{Add}(c, r)), -r)\\ &= \textsc{Paillier}.\mathsf{Enc}(pk, \textsc{Paillier}.\mathsf{Add}(c, r - r))\\ &= \textsc{Paillier}.\mathsf{Enc}(pk, c)\end{aligned}$$

It follows that

$$\begin{aligned} \mathsf{Adv}^{\mathsf{indcpa}}_{\textsc{Paillier}}(\mathcal{B}) &= |\Pr[d' = 1 \mid d = 0] - \Pr[d' = 1 \mid d = 1]|\\ &= \mathsf{Adv}^{\mathsf{pirsec}}_{\textsc{Paillier}\mathrm{PIR}}(\mathcal{A})\end{aligned}$$

By the semantic security of the Paillier cryptosystem [47], the result follows.

□

Appendix C. Security of Private Provider Advertisements

We formalize the security requirements and proof for the private provider advertisements protocol (Algorithm 2). The results here also apply to the protocol used for PrivateWantHave, since the algorithm is the same. The standard PIR security of the scheme (i.e., pirsec in Figure 3) follows directly from the security of the underlying PIR scheme, since the server only receives a PIR query for a given row in a table. This scheme should also satisfy a *symmetric* property, such that the client does not learn more than the information for which they query. We capture this by an indistinguishability notion, where the client cannot distinguish between a response generated from the true database and one generated from a random database (aside from the record for which they queried). This notion is given in the security game in Figure 4, and is equivalent to notions of security for OT protocols.

```
$\mathsf{G}^{\mathsf{spirsec}}_{\mathsf{PIR},\mathcal{D}}(\mathcal{A})$:
1 $i \xleftarrow{\$} [n], b \xleftarrow{\$} \{0,1\}$
2 $v \leftarrow \mathcal{D}[i], \mathcal{D}_1 \leftarrow \mathcal{D}$
3 $\mathcal{D}_0 \xleftarrow{\$} \{0,1\}^{|\mathcal{D}|}$    $\triangleright$ random database
4 $\mathcal{D}_0[i] \leftarrow v$
5 $\mathtt{st}, \mathtt{qu} \leftarrow \mathsf{Query}(i)$
6 $\mathtt{ans} \leftarrow \mathsf{Response}(\mathcal{D}_b, \mathtt{qu})$
7 $b' \leftarrow \mathcal{A}(\mathtt{ans})$
8 return $[\![ b = b' ]\!]$
```

Figure 4: Security game for symmetric PIR security

Theorem C.1. *Let* PrivateProviderRecord *(*PPR*) be defined as in Algorithm 2. For any adversary $\mathcal{A}$ against* spirsec*, and any database $\mathcal{D}$, we give algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ such that:*

$$\mathsf{Adv}^{\mathsf{spirsec}}_{\mathsf{PPR}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{KDF}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{indcpa}}_{\mathsf{SE}}(\mathcal{B}_2).$$

Proof. We proceed via a series of game hops. Let G_0 denote the original spirsec game for the PPR protocol for an arbitrary database $\mathcal{D}$, i.e., $\mathsf{G}^{\mathsf{spirsec}}_{\mathsf{PPR},\mathcal{D}}$.

In G_1, we modify G_0 to use a truly random function in place of the KDF, unless it is called on the value v (the CID being queried). This in particular replaces the key k with a random value k^*, unless it is derived from the CID that is being queried. We can bound the difference by a reduction $\mathcal{B}_1$ to the PRF security of KDF:

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1] \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{KDF}}(\mathcal{B}_1).$$

Next, in G_2, we replace the input of SE.Enc with a random value of the same length – again, only on evaluations where the key is already random (and not on evaluations where the key is derived from the queried CID). Since the input key k^* is random, we can bound this hop by a reduction $\mathcal{B}_2$ to the IND-CPA security of SE:

$$\Pr[\mathsf{G}_1] - \Pr[\mathsf{G}_2] \leq \mathsf{Adv}^{\mathsf{indcpa}}_{\mathsf{SE}}(\mathcal{B}_2).$$

Now, we can observe that regardless of the challenge bit b, the response $\mathtt{ans}$ given to the adversary is the same. Hence, $\mathcal{A}$ has no greater advantage than by guessing the challenge bit b and $\mathsf{Adv}^{\mathsf{G}_2}_{\mathsf{PPR}}(\mathcal{A}) = 0$. This concludes the proof. □

Appendix D. Variants of Oblivious Expansion

The oblivious expansion procedure is described in Algorithm 5. This procedure uses a substitution operation over ciphertexts in RLWE-based schemes [23]. Recall that plaintexts in RLWE-based cryptosystems are polynomials $p(x) \in R_p$. The substitution operation, $\mathtt{Sub}(k, ct, pk)$, computes the encryption of a plaintext $ct' = \mathsf{Enc}(pt(X^k))$ given the encryption of plaintext $ct = \mathsf{Enc}(pt(X))$, for a given $k \in \mathbb{Z}^*_{2N}$ using an automorphism public key pk.

Angel et al. [5] generate the automorphism keys for all substitutions of form $k = 1 + N/2^i$ for $i \in [\log_2 N]$, requiring $\log_2 N$ keys in total. They also prove correctness of oblivious expansion, given the substitution operation [5, Appendix A.2]. Since the routing table only has 256 rows, only $\log_2 256 = 8$ automorphism keys are required for RLWEPIR for peer routing. However, since provider routing involves more rows, more automorphism keys are required.

de Castro et al. [17] observe that to reduce the number of automorphism keys, we can compute the substitution of an arbitrary k' by successively applying the substitution k, i.e., $p(X) \to p(X^k) \to p(X^{k^2}) \to \cdots$. This approach only requires the automorphism key for k. Since no generator can generate the set $\{1 + N/2^i\}_{i \in [\log_2 N]} \subset \mathbb{Z}^*_{2N}$, they choose two elements in $\mathbb{Z}^*_{2N}$ that can generate the entire set. For $N = 4096$, they choose 3 and 1173 as two generators.

While their approach reduces the number of required keys, it requires more base substitutions, which impacts performance. To find a middle ground, we can use three keys instead of just two, and compute all the necessary substitutions with fewer base substitutions. We conduct an efficient grid search of the space, for $N = 4096$, while minimizing the objective function stated by de Castro et al., that is, the total number of base substitutions required by the oblivious expansion algorithm. We find that 3, 5, and 1167 generate the necessary substitutions that we need with the fewest base substitutions.

Algorithm 5 Oblivious Expansion with substitution

```
1: procedure ObliviousExpand($\mathtt{pk}, \mathtt{ct}$)
2:     $cts \leftarrow [\mathtt{ct}]$
3:     for $i \in [\log_2 N]$ do
4:         for $j \in [2^i]$ do
5:             $c \leftarrow cts[j]$
6:             $t \leftarrow \mathtt{Sub}(N/2^i + 1, c, pk)$
7:             $cts[j] \leftarrow c_0 + t$
8:             $cts[j + 2^i] \leftarrow X^{-2^i} \cdot (c_1 - t)$
9:     return $\{c_i\}_{i \in [N]}$
```

Appendix E. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

This paper shows how to integrate PIR into IPFS, allowing clients to access data without revealing to the nodes which data they seek. It shows how to optimize each aspect of IPFS for privacy and introduces two new PIR algorithms that are suitable for this use case. Notes are shared on lessons learned from integrating this privacy-preserving functionality into an existing, non-private codebase.

E.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

E.3. Reasons for Acceptance

- The paper establishes protocols for privately retrieiving data in a peer-to-peer setting, an important direction with limited previous attention.
- The paper's PIR schemes may be useful in other contexts because of the tradeoffs they explore.
- In addition to the core technique, this paper discusses practical deployment considerations and provides takeaways, including integration challenges, node churn effects, and potential attack vectors. These make the solution more practical.
- The authors present a detailed comparison with existing PIR protocols in the evaluation.

E.4. Noteworthy Concerns

1) The formal threat model is not clear.
2) The paper only considers a semihonest security model.
3) The evaluation focuses on area-under-curve for communications overhead, which might miss the impact induced on network connectivity.
4) The evaluation does not focus on I/O overhead.