Do you feel a chill? Using PIR against chilling effects for censorship-resistant publishing[*]

Miti Mazmudar
University of Waterloo
Waterloo, ON, Canada
miti.mazmudar@uwaterloo.ca

Stan Gurtler
University of Waterloo
Waterloo, ON, Canada
tmgurtler@uwaterloo.ca

Ian Goldberg
University of Waterloo
Waterloo, ON, Canada
iang@uwaterloo.ca

ABSTRACT

Peer-to-peer distributed hash tables (DHTs) rely on volunteers to contribute their computational resources, such as disk space and bandwidth. In order to incentivize these node operators of privacy-preserving DHTs, it is important to prevent exposing them to the data that is stored on the DHT and/or queried for. Vasserman et al.'s CROPS aimed at providing plausible deniability to server nodes by encrypting stored content. However, node operators are still exposed to the contents of queries. We provide an architecture that uses information-theoretic private information retrieval to efficiently render a server node incapable of determining what content was retrieved in a given request by a user. We illustrate an integration of our architecture with the aforementioned system. Finally, we simulate our system and show that it has a small communication and performance overhead over other systems without this privacy guarantee, and smaller overheads with respect to the closest related work.

KEYWORDS

Censorship-resistant publishing, query privacy, private information retrieval

1 INTRODUCTION

Censorship-resistant systems allow users access to online content when direct access to such content is restricted by a nation-state adversary, namely a *censor*. For instance, Tor [9] is an anonymity network that supports users in accessing websites that are censored by nation-states. Censorship-resistant *publishing* systems allow publishers to submit their work onto multiple servers, also called *nodes*, such that a censor cannot take down or tamper with the published content. Several censorship-resistant publishing systems have been proposed to date, such as Vasserman et al.'s CROPS [32], Waldman and Mazières' Tangler [34] and Stubblefield and Wallach's Dagster [30].

Node operators within the censor's region of influence may be legally obliged to report any prohibited content that is stored on their machines, or that is requested by users in queries. Many node operators would thus rather not learn the content that they are storing or the content of the queries that they are receiving. For instance, the Signal messaging app is designed to minimize the types of users' data and metadata that are stored in the Signal servers, so that obliging with legislative requests for users' data can only cause minimal privacy harms to its users [27]. The aforementioned systems support node operators in *plausibly denying* any knowledge of the content that they store, by encrypting chunks of publishers' documents and storing key material separately from these chunks. Within Vasserman et al.'s CROPS, these chunks are indexed by hashes of keywords that describe the document. However, as the keywords' hashes are exposed in clients' queries, these node operators can no longer plausibly deny knowing what content was queried.

Censorship-resistant publishing systems have largely focused on securing the *supply* of documents for users. Frequently, the goal is to ensure that for each document, at least one server is available to provide any user a copy of that document. As long as node operators learn what document was requested, a censor may coerce them into revealing this information. Consequently, node operators may be discouraged from contributing their storage and bandwidth for such systems. Users who recognize this flaw may not allow themselves to even seek documents — a form of self-censorship known as a *chilling effect*. In these cases, censorship-resistant publishing systems have still failed to make the document available to all users, because they did not secure the *demand* of documents. Thus, exposing the contents of queries to node operators disincentivizes both node operators and users from using these types of systems.

Private information retrieval (PIR) is designed to solve the problem of retrieving a row from a database on a server, without revealing the index of that row to the server. Information-theoretic PIR (IT-PIR) in particular typically requires multiple replicas of the database. The user sends different queries to each server holding a replica, such that as long as less than a threshold fraction of the servers collude, they do not learn anything about the contents of the query, and the user can reconstruct the desired row from the servers' responses [8, 12].

To motivate our key insights on using PIR within censorship-resistant publishing systems, we examine their structure. Censorship-resistant publishing systems are commonly built atop structured peer-to-peer (P2P) networks, as they support redundancy and do not suffer from a single point of failure. (Tor onion services, in contrast, do not afford these two properties, as we will discuss in Section 4, and are orthogonal to our goals. We remark that we do not aim to provide client anonymity, but rather plausible deniability for node operators.) Even though individual nodes can store content cheaply, for the two aforementioned reasons, structured P2P networks remain relevant today for developing decentralized file storage and routing systems, such as IPFS [15]. Structured P2P networks link peer nodes' identifiers to the content that they store and use distributed hash tables (DHTs) for routing search and insertion queries. DHTs, such as Kademlia [21] and Chord [29], are analogous to conventional hash tables, with the exception that the entire key-value store is systematically split across nodes in the P2P network, such that queries can be conducted efficiently.

[*]This is an extended version of our paper that appeared in the 20th ACM Workshop on Privacy in the Electronic Society (WPES '21) [22].

P2P networks have been known to suffer from attacks by malicious users who aim to prevent a legitimate user from obtaining a correct copy of their desired content [19, 28]. Related research [3, 5, 10] shows that if only a small fraction of nodes in the P2P network are malicious, it is possible to group joining peers into *quorums* such that, with high probability, all quorums in the network will only contain a bounded fraction of malicious nodes. This result is known as the *goodness invariant*. Young et al. [36] develop protocols for efficient, robust communication across quorums that prevent spamming attacks, whereas Backes et al. [4] extend the former's protocols to support conducting DHT queries without revealing the query content to any in-path nodes.

Our key insight lies in observing that IT-PIR can be instantiated over quorums in the DHT. Carefully analyzing and setting the parameters involved, we leverage the DHT goodness invariant to satisfy the IT-PIR non-collusion assumption. We present an IT-PIR architecture that can be integrated with an existing censorship-resistant publishing system, such as CROPS, to enable clients to retrieve documents from the DHT privately. Specifically, neither the target node, nor any in-path nodes, learn which document was retrieved. In addition to our enhancement of using PIR to protect the nodes from learning which document is being retrieved, we use Young et al.'s protocols for robust communication across quorums, and Backes et al.'s protocols to hide the content of routing queries from intermediate nodes.

We begin with a description of the building blocks we use, namely PIR, DHTs, and censorship-resistant publishing systems, in Section 2. We discuss robust DHTs, including related work in query privacy over DHTs, in Section 3, and describe our threat model in Section 4. We then present the following contributions:

- We provide an architecture, similar to that of regular DHTs in structured P2P networks, such that a user can retrieve a document from the network without revealing what document was retrieved to the target node that stores the document, nor to any in-path nodes. We also illustrate an integration of this interface with that of an existing censorship-resistant publishing system (Section 5).
- We detail the security parameters of our system and show in our analysis that our system works when the DHT quorums have less than $\frac{1}{4}$ malicious nodes (Section 6).
- We include a message and communication complexity analysis of our system and find that it has a low communication complexity in comparison to the best-known scheme for privately retrieving a document from a node in a DHT [4] (Section 7).
- We simulate the communication complexity of our system and find that it is explained by our complexity analysis. We also simulate the latency and throughput overheads of our system, to demonstrate its viability for deployment (Section 8).

Finally, though we have presented our interface in the context of a censorship-resistant publishing system, we discuss general deployment of our interface in Section 9, while highlighting other use cases and addressing bootstrapping issues. We conclude in Section 10.

2 BACKGROUND AND RELATED WORK

In this section, we describe the building blocks used in our system. We begin with a brief description of private information retrieval and factors affecting our choice of a PIR scheme. Second, we provide a brief overview of DHTs, including existing literature on attacks in DHTs. Third, we delve into censorship-resistant publishing systems, with a focus on plausible deniability, and motivate our interface, which prevents exposing node operators to the content of queries.

2.1 Private Information Retrieval

Private information retrieval (PIR) allows public databases, held by one or more untrusted servers, to be queried by clients while guaranteeing that the servers cannot learn which index of the database was accessed. Computational PIR (CPIR) schemes require the computational hardness assumptions of certain cryptographic problems, and can be used even when only one server stores the entire database. On the other hand, information-theoretic PIR (IT-PIR) schemes do not require any computational hardness assumptions; however, they can only be implemented when multiple servers have copies of the database [8].

In IT-PIR schemes, the client constructs one query for each server such that using the responses to these queries, it can reconstruct the desired row of the database. A certain threshold number of these servers must not share the queries they receive with each other, as otherwise, they could reconstruct the desired row and identify it, just as the client does. This is known as the *non-collusion assumption*. Chor's IT-PIR scheme [8] requires all nodes to respond correctly to the client's query, for the latter to be able to reconstruct the desired row, and thus does not tolerate any malicious nodes. *Byzantine-robust* IT-PIR schemes, such as that of Goldberg [12] support efficient reconstruction of the correct row, while allowing a fraction of servers to be offline and out of those that are online and provide responses, a fraction can provide incorrect responses. Although Aguilar-Melchor et al.'s [1] and Angel et al.'s [2] CPIR schemes have lower computational complexity than prior schemes, they incur a high computational overhead over the aforementioned IT-PIR schemes.

Thus, if one can reasonably argue that the aforementioned non-collusion assumption holds true, then IT-PIR schemes are preferable to CPIR schemes. First, they outperform the latter in terms of computational overhead, and second, they do not require any computational hardness assumptions. We detail in Section 5 why the non-collusion assumption for IT-PIR can be reasonably made within our context, leading us to use IT-PIR. We use Goldberg's IT-PIR scheme [12] as it is robust against failing and Byzantine nodes, both of which exist in P2P networks.

2.2 DHTs

A file that is to be stored in a structured P2P network has a collision-resistant one-way hash, typically of its contents, as its identifier. A node on structured P2P networks also has a truncated one-way hash as its identifier; such a node stores all files whose identifiers, when truncated, are equal to the identifier of the node. As no single entity stores all of the content on the network, peer-to-peer networks do not suffer from a single point of failure. Each node in a structured P2P network maintains a *routing table*, which consists of node identifiers and their routing information for a small number of other nodes in the DHT. In order to search for or insert a file, nodes obtain the routing information for the relevant node identifiers by querying one of its neighbours and then querying one of that neighbour's neighbours, and so on, through iterative routing. (Recursive routing, on the other hand, has the node's neighbour itself looking up the next neighbour that is closest to the target node, and so on. We do not consider recursive routing as providing privacy guarantees over routing queries in recursive routing systems is fundamentally difficult, as Backes et al. [4] point out.) The maximum number of nodes that a given node needs to contact is known as the *path length* (ℓ). DHTs guarantee that the path length is logarithmic in the number of nodes in the network.

DHTs have been known to be vulnerable to many attacks that prevent a user from obtaining a copy of a file. Existing censorship-resistant publishing systems do not consider these attacks, and thus, do not integrate defenses that have been proposed against such attacks. A censor can easily exploit these attacks, which we motivate below, to break the availability of files stored on the overlay censorship-resistant publishing system.

If nodes are allowed to choose their own node identifiers, then a censor may simply join the system as a set of legitimate nodes, at various node addresses that would be storing prohibited documents. It can then simply drop all future requests that it obtains for storing or retrieving prohibited content. The censor can also direct its new nodes to join at addresses that are close to a target honest node, and cause this node's routing requests to be directed to its nodes, instead of other honest nodes. This is known as an eclipse attack, which restricts honest nodes' view of the network.

As a defence against these attacks, P2P networks can require existing nodes to assign a random identifier to incoming nodes. However, malicious nodes may still attempt to join and leave the network until they are placed at suitable network addresses, to influence honest nodes' routing table entries in a Byzantine join attack. Several researchers [5, 10] have proposed protocols to allocate joining nodes into quorums, such that by repeatedly rejoining, any malicious nodes do not increase their chances of being cast into a group with a majority of other such malicious nodes. We explicate Awerbuch and Schiedeler's cuckoo rule (CR) [3], as well as Sen and Freedman's improvement over this rule, namely the commensal cuckoo rule (CCR) [26], in Section 3.

To resolve routing requests across quorums, nodes in a quorum can reply back with the network addresses of the nearest quorum to the target address, akin to iterative routing in regular DHTs. However, malicious nodes in such quorums may respond back with incorrect routing table entries, or inundate honest nodes in other quorums with fake requests. Young et al. [36] present two robust communication protocols to support nodes in efficiently communicating across these quorums, while being able to either detect or prevent spamming attacks. However, these protocols only provide *integrity* guarantees over the routing information within a quorum-based DHT. They do not provide *confidentiality* of the routing query content; the node address being queried is known to nodes on the path to the target node and to the target node itself. The censor may thus compel all nodes in its region, as they may serve as *in-path* nodes to other nodes, to reveal the node addresses looked up by users in its region. It can then possibly harm these users for attempting to contact external nodes that serve prohibited content. Backes et al. [4] improve on Young et al.'s protocols by hiding the queried key from all in-path nodes, thereby providing query privacy (QP). As we discuss in the next subsection, honest node operators benefit from learning as little as possible about the key that was queried.

We describe Awerbuch and Schiedeler's CR strategy, Sen and Freedman's CCR strategy, Young et al.'s Robust Communication Protocols, and Backes et al.'s Query Privacy protocols in Section 3. Our interface uses these protocols to harden existing censorship-resistant publishing systems against these attacks.

2.3 Censorship-resistant publishing

Censorship-resistant publishing (CRP) systems, such as Waldman and Mazières' Tangler [34], Waldman et al.'s Publius [35], and Vasserman et al.'s CROPS [33], support publishing and retrieving documents in the face of a censor that may attempt to take down prohibited content. Censors may outright block any censorship-resistant publishing (CRP) systems within their administrative region. We assume that blocking the system entirely would impose an undesirably large social or economic cost to the censor. This is the collateral damage assumption, and we revisit it in Section 4.

CRP systems are built atop DHTs formed by machines that are located across different administrative regions, each of which contributes disk space and bandwidth. However, DHTs by themselves do not ensure the confidentiality, integrity, or availability of the content stored on them; CRPs include mechanisms to publish and retrieve documents while providing these three crucial properties. To increase the availability of the content stored on them, CRP systems store redundant copies of documents or of their parts across the underlying DHT. In Publius [35], secret sharing is used to produce shards or *shares* of documents, such that the document can be reconstructed using a threshold number of shares. Publishing a single document in Tangler [34] necessitates replicating multiple other existing documents within the system, again through secret sharing. To store a document within CROPS, several replicas are created, and split into chunks using erasure coding, which is similar to secret sharing in terms of the aforementioned reconstruction guarantee. The lack of a single point of failure and support for redundant storage render DHTs attractive, with respect to the availability property, for censorship-resistant publishing.

All of the aforementioned systems encrypt documents to preserve their confidentiality, before breaking them down into shards for redundancy. Encryption supports node operators in *plausibly denying* knowledge of what content they store. Consider a censor that legally obliges all node operators in its region to report any prohibited content that they store. In Vasserman et al.'s CROPS, a node that stores an erasure-coded chunk of a document does not learn anything about it, based on the hardness assumption of the cryptographic hash used to index the chunk.

Importantly, CROPS helps users easily discover content on the system through keywords, while preserving plausible deniability over the stored content and ensuring its integrity. To publish a document, the publisher specifies relevant keywords for it. After encrypting the document and performing erasure coding on the ciphertext document, the publisher creates replicas of two manifest

1CR. A joining node p is assigned a random address, which lies in some k-region of a quorum Q_T.
1CCR. Primary join: Along with step 1CR, quorum Q_T ensures that it has received at least $k-1$ secondary joins before letting node p join.
2CR. All other nodes with addresses in this k-region are relocated to other random k-regions.
2CCR. Secondary join: Instead of step 2CR, $k' = k \cdot \|Q_T\|/\bar{s}$ nodes with addresses in the quorum Q_T are relocated to other random quorums (where $\bar{s}$ is the desired average quorum size).

FIGURE 1: **Descriptions of cuckoo rule (CR) [3] and commensal cuckoo rule (CCR) [26].**

files, namely the content and key manifests. A cryptographic hash of one important keyword, along with the replica number, is used to index the manifest replicas. Each manifest file is signed by the publisher to preserve its integrity. The content and key manifests separate the material required to locate the erasure-coded chunks (cryptographic hashes of the chunk content) from that required to obtain a plaintext document (symmetric key for decryption). (The manifests also contain cryptographic hashes to verify the integrity of the ciphertext document, which is reconstructed from erasure-coded shares, as well as of the plaintext document.) Nodes that store one of a key or a content manifest replica need to obtain the other one, and sufficient erasure-coded chunks, in order to obtain the plaintext document that the replicas refer to.

As the client's query includes keyword hashes, the censor may legally oblige the node operator to reveal all queries that it received. The censor may then determine if any of the keyword hashes in a query match those of prohibited keywords [14, 18], using rainbow tables. It may also require the node operator to report the network addresses of all users, and possibly harm the users who attempted to fetch prohibited content. Thus, simply providing plausible deniability over *stored* content is insufficient; node operators should also be prevented from learning what content is being *retrieved*.

We address this problem by allowing a client to privately retrieve either erasure-coded chunks or manifests (collectively known as *files* from here on). Although we focus on CROPS as a use case for our system, we provide private file retrieval for censorship-resistant publishing systems in general. As we discuss in Section 9, our interface may also be used with other CRP systems and DHT-based applications.

3 ROBUST DHTS

In this section, we describe protocols to form quorums (Awerbuch and Schiedeler's CR [3], Sen and Freedman's CCR [26]), to route queries robustly across quorums (Young et al.'s RCP [36]), as well as to conduct these routing queries privately (Backes et al.'s QP [4]). The QP protocols can also be applied to conduct file retrieval queries privately, and so we also outline the advantages of our DHTPIR sytem over those.

Forming quorums: Awerbuch and Scheideler [3] propose a cuckoo rule joining strategy, wherein incoming nodes cause existing nodes at nearby addresses to be kicked out (or cuckooed) to other addresses, as shown in Figure 1. A virtual address space of size n is split into *quorums*, which are split further into k-regions that span a fraction $\frac{k}{n}$ of the address space.

Awerbuch and Schiedeler show that for a given *global* bound on the ratio of malicious to honest nodes (say ϵ), their strategy results in approximately equally sized quorums of $s = O(\log n)$ nodes (the *balancing* condition), where the ratio of malicious to honest nodes in each quorum is upper-bounded by a value greater than ϵ (the *correctness* condition). However, their result is *asymptotic*, and so left an open question of whether it holds for networks with realistic numbers of peers.

We implemented a simulator for the cuckoo rule in Rust, and determined that for reasonable numbers of peers in the network (say 10 million), there is a severe tradeoff between ϵ and s: to maintain the correctness of relatively small quorums ($s < 100$), the network can only withstand a very small fraction of adversary-controlled nodes ($\epsilon < \frac{1}{1000}$). Having large quorums of 100 peers or more makes the distributed protocols required within Young et al.'s schemes expensive.

Sen and Freedman [26] performed a similar simulation and their findings echo ours. Additionally, they find that the aforementioned tradeoff worsens for large network sizes; that is, small quorums in larger networks can only withstand much smaller fractions of malicious nodes. To address this issue, they propose a *commensal cuckoo joining strategy*, which modifies each of the two aforementioned steps of the cuckoo rule, as shown in Figure 1. In their joining strategy, the desired average quorum size is set beforehand (to $\bar{s}$), in contrast with other schemes that require it to increase with the network size. We use the commensal cuckoo rule strategy to form quorums in our join protocol. We discuss its security implications within our threat model in Section 6.

Robust routing across quorums: Young et al. [36] propose two efficient robust communication protocols (RCP) that prevent spamming attacks from nodes in other quorums by using a threshold signature scheme. In order to communicate with a node in a target quorum, the client node must obtain a time-stamped proof of robust communication from one of the neighbours of that quorum, in the form of a threshold signature of the neighbouring quorum. Young et al.'s deterministic RCP-I scheme *prevents* spamming attacks by imposing a high communication cost for the client; the client communicates with each of the s nodes in each of ℓ quorums on the path to the target quorum, for $s \cdot \ell$ nodes in total. Their probabilistic RCP-II scheme enables the target quorum's nodes to *detect* such an attack, and only requires the client to communicate with an expected $O(s + \ell)$ nodes. RCP schemes include distributed protocols for synchronizing the quorum's cryptographic state whenever a new node joins the quorum.

Query privacy across quorums: Backes et al. [4] present two query privacy (QP) protocols, namely QP-I and QP-II, which build on Young et al.'s RCP-I and RCP-II protocols respectively to hide the node address key being queried from all nodes in all quorums in path to the target quorum. They develop a protocol based on oblivious transfer (OT) rather than PIR, in order to restrict a client from privately obtaining routing information from any intermediate quorum about *multiple* neighbouring quorums through a single valid query.

Table 1: **Contrasting our scheme with existing ones for private and robust communication in a DHT. $\mathcal{D}$ is the total size (in bytes) of the files stored at each node in an s-node quorum. RR = robust routing; QP = In-path query privacy; PR = availability and, if so, communication complexity of private file retrieval.**

System	RR	QP	PR
Base DHT	-	-	-
Young et al.'s RCP [36]	●	-	-
Backes et al.'s QP [4] w/o last hop	●	●	-
Backes et al.'s QP [4] w/ last hop	●	●	$\mathcal{D}$
DHTPIR (our system)	●	●	$2s\sqrt{\mathcal{D}}$

However, as they observe, their OT protocol requires the queried node to send the entry-wise encrypted data store over which the OT query is to be conducted back to the client. In their use case, this data store is simply the small routing table maintained at each DHT node, which contains $O(\log n)$ routing records for a network of size n. To privately retrieve a file stored in the file store of the target quorum, Backes et al. suggest optionally extending this same OT protocol with one more hop to the target quorum. However, doing so would require sending the entire encrypted file store back to the client, resulting in a large communication complexity.

To retrieve a file from a quorum of s nodes, which holds a database of size $\mathcal{D}$ bytes, Backes et al.'s scheme incurs a communication complexity of $\mathcal{D}$, whereas our scheme only has a complexity of $2s\sqrt{\mathcal{D}}$. In Table 1 we contrast the properties provided by Young et al., Backes et al., and our schemes to DHT clients and compare the communication complexity for Backes et al.'s and our scheme in providing private file retrieval. We compute the range of values of $\mathcal{D}$ for which our system performs better in Section 7.

4 THREAT MODEL

As mentioned earlier, we build our system atop DHTs as they support redundant storage of documents and do not suffer from a single point of failure, which renders them suitable for censorship-resistant publishing. (Tor and Tor onion services are orthogonal to our work, as discussed in the non-goals below.) Our adversary is a single nation-state censor that has an administrative region of influence; other parts of the world are deemed safe. We consider the case of multiple such censors below. We assume that this nation-state cannot simply censor the entire system or host an alternative, as blocking the system entirely would impose an undesirably large social or economic cost to the censor. We recognize that this collateral-damage assumption may not hold for powerful nation-state adversaries: such an adversary could coerce its population to simply use a publishing system that is not privacy protective. However, we believe that it is a reasonable assumption that may hold for many nation-states. We require the same cryptographic hardness assumptions as Backes et al. [4], namely the GDH problem for threshold signatures and the DDH problem for the OT protocol.

Our adversary's goals are to determine which document was retrieved in a particular content retrieval request and to prevent the operator from responding correctly to a request. Our adversary has several capabilities that it can exploit towards furthering its goals. First, the censor may observe the network traffic within its region of influence, possibly using deep packet inspection techniques on any plaintext data. Second, the censor can search for sensitive content by posing as a client and can map files to virtual addresses of nodes that store them. Third, the censor's control over nodes in the network is bounded; in particular, for a network with n honest nodes, the censor may insert up to $\epsilon \cdot n$ malicious nodes, where $\epsilon \approx 0.02$. The censor may remove these nodes from the network in the future, and have them rejoin the network, which would allow the node to join a different quorum.

Multiple nation-state censors may be interested in censoring different content or content retrieval requests across nodes within their respective regions of influence. Since nodes operated by one nation-state censor may collude with those operated by another censor, we require that the ratio of nodes controlled by all these censors, to non-controlled nodes, is less than ϵ. These nodes may behave as Byzantine nodes by, for example, responding to routing requests with incorrect responses, in an attempt to misdirect a client's routing request towards other Byzantine nodes. They may also collude and share the content retrieval requests that they obtain, or forward them to the censor. These censor-controlled nodes may also simply fail, by not replying to queries. Furthermore, they may attempt to inundate honest nodes with routing or content request messages at the application layer.

Finally, the censor cannot dynamically compromise existing nodes; that is, it cannot coerce honest nodes within its region of influence to act as Byzantine or failing nodes. It may not compel honest nodes to retain (and share) the content retrieval requests they receive. (Note that honest nodes in a given quorum would be geographically distributed in arbitrary administrative regions and may be out of the censor's region of influence.) The censor may not block communication between honest nodes.

Our interface provides the following guarantees in the face of this adversary:

- Minimizing data within the content of document retrieval queries — *honest* server operator nodes and small coalitions of Byzantine nodes within a quorum do not learn which document was retrieved.
- Correct responses to document retrieval requests — a document retrieval request will be answered correctly, even if censor-controlled nodes provide incorrect or no responses.

Non-goals: We do not provide any additional protections over content *publishing* requests. Although we use RCP's spam-limiting feature to prevent DoS amplification for the DHTPIR protocol messages, protecting nodes from general DoS attacks is orthogonal to this work. We do not aim to protect the client's identity, and thus, we also do not prevent adversaries from targeting clients, such as by traffic analysis. Regarding the first goal, although nodes controlled by the adversary that lie *within* the quorum from which the file is fetched also cannot learn which file was retrieved by colluding among themselves, they may collude with nodes from other quorums. By learning the set of quorums from which files were fetched, they may determine which document was fetched, in a quorum fingerprinting attack; we do not aim to protect against this attack, as DHTPIR aims to protect server nodes, and not directly clients.

DHTPIR may be instantiated over Tor onion services to provide client anonymity. However, Tor itself does not support the

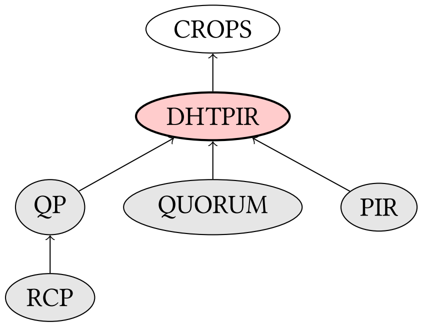


Figure 2: **API interfaces required by, and provided by, our DHTPIR interface. The bold red ellipse represents our interface, and the grey ellipses show the layers of APIs required by our interface. The white ellipse represents an existing system within which our interface can be used in order to provide document retrieval privacy, in addition to the robust routing and in-path privacy provided by the lower layers.**

redundant storage of documents that is afforded by peer-to-peer networks like DHTs. Although Tor onion services may have multiple physical servers serving the same onion service, all servers must share a private key, putting them under shared administrative control. In this sense, Tor onion services suffer from a single point of failure, unlike DHTs. As node operators of Tor onion services learn which document was retrieved through a given document retrieval request, they do not have plausible deniability over the content of such requests. Tor exit node operators have been blamed, and even charged, for retrieving content that a client had retrieved through their node [11]; privacy-enhancing technologies that prevent the operators from being exposed to the information they store or forward may decrease such risks.

5 OUR DESIGN

We first begin with an overview of our interface for integrating private file retrieval to DHT-based censorship-resistant publishing systems. We begin with a description of the building blocks that our interface requires, and then describe our implementation of this interface, while analyzing possible Byzantine behaviour. Through this analysis, we motivate the constraints that our instantiations of the building blocks must follow.

5.1 Building blocks

Our interface can be used within censorship-resistant publishing systems, such as Vasserman et al.'s CROPS [33], to insert files into quorums through the DHTPIRputFile function and to query quorums privately for files through the DHTPIRfindFile function, both provided by our DHTPIR API. Functions in the DHTPIR interface invoke existing protocols in layers, depicted in Figure 2. A list of all API functions can be found for reference in Appendix A.

We use Backes et al.'s query privacy (QP) protocol to privately fetch the desired quorum's routing information; the corresponding function constitutes the QP API layer. (This protocol internally invokes Young et al.'s RCP schemes.) We provide a QUORUM API layer to support inter-quorum communication, given the target quorum's routing information. We use Goldberg's IT-PIR algorithm to privately fetch files from the target quorum; functions to generate, perform, and process PIR queries form the PIR API layer. We rely on Sen and Freedman's commensal cuckoo-hashing technique [26] to establish quorums across the DHT; that is, to assign node addresses and quorum neighbours to new nodes joining it. We briefly describe functions at the QP, QUORUM, and PIR layers below.

QP layer: The client uses the QPgetTargetQuorum function to obtain the routing information of the target quorum while preserving the confidentiality of the file's identifier from all nodes in the DHT. This function executes Backes et al.'s RCP-QP-I or RCP-QP-II algorithms for the input file identifier id. This function obtains the target quorums' nodes' network addresses $[A_1, \cdots A_s]$ and public encryption keys $[PK_1, \cdots PK_s]$ from one of the quorum's neighbours. It also obtains a proof ϕ of robust communication from that neighbour. The proof shows that the given client node sent a legitimate request to contact the target quorum, through a signature over the node's network address (A), its public key (PK) and a timestamp ts (ϕ in our notation is equivalent to $\mathcal{S}_{\ell-1}$ for a query path length ℓ in Backes et al.'s notation). Finally, this function returns a verification key VK, obtained from the target quorum's neighbour, which allows the client to verify messages signed by a threshold number of nodes in the target quorum.

QUORUM layer: The client uses the QUORUMsendQuery function for sending an encrypted message to each of the nodes in a target quorum and obtaining a reply from them, given the nodes' addresses $([A_1, \cdots A_s])$ and public keys $[PK_1, \cdots PK_s]$ as well as a proof of robust communication ϕ. This function can also be configured to send a single message to an arbitrary quorum node, retrying with different nodes until it obtains a valid reply. The recipient nodes in the target quorum listen for messages from other nodes in the QUORUMqueryListener function. Depending on the flags in the message, a recipient node may be delegated to forward different values back to its peers and send a response from these peers back to the client node. All honest nodes regularly execute a Byzantine agreement protocol among themselves to guarantee that they agree on the state of the database (including a perfect hash function as we discuss later) stored at other honest nodes.

PIR layer: We observe that all nodes in a quorum share the same database of files; we provide an overview of the join protocol that enables this shared state at the end of this section. The client uses the PIRgetQueryVectors and PIRrecreateFile functions to generate and process IT-PIR queries while the target quorum's nodes use the PIRperformQuery function to perform the IT-PIR query itself. Perfect hash functions (PHFs) [6] are often used within PIR schemes to map a set of keywords for documents stored at server nodes to a set of indices, such that the client can simply compute the desired index if it knows the keyword. The client obtains the PHF f beforehand using the DHTPIRfindFile function, and then passes it as an input to the PIRgetQueryVectors function, along with the desired file identifier id and the number of (server) nodes s. This function computes the index at which the desired file will be stored, by computing the value of f for the key id, and obtains an index i. It then creates s query vectors such that when responses to these vectors are combined in the PIRrecreateFile function, they will result in file i of the PIR database.

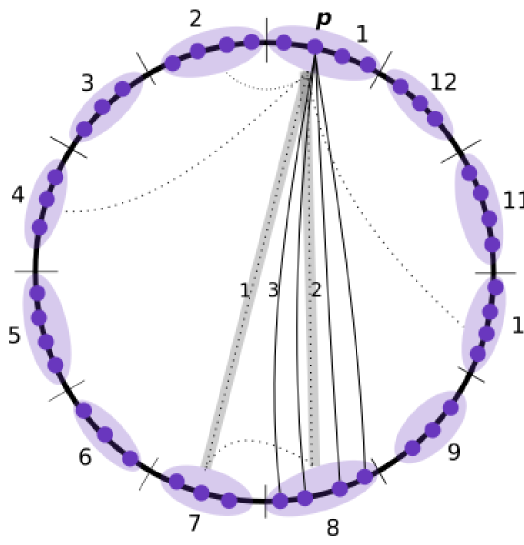


FIGURE 3: **Layers of protocols used within our work. Quorums are numbered and shown in purple ovals. Node p from quorum 1 wishes to fetch a file that is stored by quorum 8. Routing table (RT) entries maintained by quorum 1 are shown by dotted lines. Quorum 1 uses the QP protocol to query quorum 7 (grey background lines), whose RT entry it knows. Quorum 7 then provides the RT entries for quorum 8, which is queried again through QP. PHF and PIR requests and responses are shown in solid lines; node p fetches the PHF from quorum 8, and then sends a PIR request to each node in quorum 8, from which it obtains a PIR response, to reconstruct the file.**

The PIRrecreateFile function reconstructs the file from s PIR query responses using Goldberg's EASYRECOVER algorithm [12, Fig.3]. We show in Section 6 that the preconditions for using this efficient algorithm are met. The PIRperformQuery function computes a matrix multiplication of the query vector with a block-wise field representation of the database, using Goldberg's IT-PIR routine [12, Fig.2].

5.2 Our interface

We present our functions for searching for a file, namely DHTPIRfindFile, and inserting a file, namely DHTPIRputFile, shown in Algorithms 1 and 2, respectively. Both of these functions expect at least an identifier for the file to be inserted or searched. These functions use this identifier to locate the file within the DHT (as in regular DHTs), with the exception that in our case, we wish to locate all nodes in a quorum; that is, a *set* of nodes in the DHT, all of which will store the file. Specifically, both functions execute the Query Privacy API function QPgetTargetQuorum with this identifier (id) to obtain network addresses and long-term public keys of the quorum's s nodes ($[(A_1, PK_1), \cdots (A_s, PK_s)]$), a proof ϕ of robust communication from a quorum linked to the target quorum, and finally, a verification key VK to verify messages that are signed by a threshold of the target quorum nodes (lines 2–4).

We motivate rest of these functions by starting with a simple protocol for each function and then optimizing its communication

Algorithm 1 Our file search function

```
1:  function DHTPIRFINDFILE($id$)
2:      dest ← QPgetTargetQuorum($id$)
3:      $[(A_1, PK_1), \cdots (A_s, PK_s)] || \phi || VK$ ← dest
4:      nodes ← $[(A_1, PK_1), \cdots (A_s, PK_s)]$
5:      msg ← "SEARCH_PHF_PARAMS"
6:      reply || sig ← QUORUMsendQuery($\phi$, nodes, msg)
7:      PHF || ts ← reply
8:      if verifySig(reply, sig, $VK$) == False then
9:          return Error: "Could not verify PHF shares."
10:     end if
11:     if ts + Current time > $T$ then
12:         return Error: "Got Expired PHF"
13:     end if
14:     queries ← PIRgetQueryVectors($id$, PHF, $s$)
15:     msg ← "SEARCH_PIR_QUERIES" || queries
16:     blocks ← QUORUMsendQuery($\phi$, nodes, msg)
17:     file ← PIRrecreateFile(blocks)
18:     return file
19: end function
```

complexity. While designing the optimized protocols, we ensure that we meet the goals mentioned in Section 4, namely, we guarantee that an honest node, or a bounded-size coalition of Byzantine nodes in a quorum, cannot learn the content of a client's file retrieval query. We also meet the second aforementioned goal, that is, such a coalition of Byzantine nodes cannot cause a legitimate search or insertion query to be executed incorrectly or dropped. Additionally, we explore what actions a Byzantine *client* node can perform and consequently, we equip honest nodes with signature verification and rate-limiting mechanisms to detect Byzantine client nodes that attempt to overwhelm them with DHTPIR messages.

File retrieval: We illustrate the file retrieval process in Figure 3. To conduct a search query, the client needs to identify the index of its file in the target quorum's nodes' database. To do so, it obtains the perfect hash function (PHF) used in that quorum; we detail this step shortly. Through a call to the PIRgetQueryVectors function, the client computes the index of the desired file identifier id using the PHF and constructs s PIR query vectors (lines 14–15). It sends one vector to each node and obtains all PIR responses through the QUORUMsendQuery function (line 16). The client reconstructs the file using these PIR responses through the PIRrecreateFile function (line 17). We parameterize the threshold for the IT-PIR scheme as follows (references for these and subsequent constraints to the relevant inequalities are included in Section 6).

Constraints for threshold of IT-PIR scheme:

1. Byzantine nodes in a quorum *cannot* reconstruct a PIR response through collusion (inequality 6).
2. Honest nodes can *efficiently* reconstruct the desired file, when a given fraction of nodes per quorum is Byzantine (inequality 7).

Naively, the client could obtain the PHF by simply querying each node in the target quorum and majority filtering the received responses. Since all target nodes must agree on a PHF for indexing the database, ostensibly the client could query one target node for the PHF. (We describe PHF computation while discussing our file insertion function.) However, this delegate node may be Byzantine and may simply not respond. In this case, through the QUORUMsendQuery function, the client retries sending the message to a different node until it receives a reply (lines 5–6). (Again, we observe that eventually the sending node will reach an honest node as only a minority of nodes in the quorum are Byzantine.)

A Byzantine delegate node may also collude with other Byzantine nodes to send a PHF that is built after excluding certain files' indices from the input set, or after swapping some files' indices around. Thus, the client should receive a confirmation from enough honest nodes in the quorum that a valid PHF was returned. Therefore, the delegate node obtains signature shares over a hash of the PHF and sends the threshold signature back to the client. As the client knows the quorum verification key VK, it can verify this signature (lines 6–10). Yet, a Byzantine node may simply return a valid PHF and signature shares that reflect the state of the database in the past. To guarantee freshness, we require the delegate node to concatenate a timestamp with the PHF, before computing the signature shares (lines 11–13). We parameterize the threshold for Young et al.'s threshold signature scheme as follows.

Constraints for Young et al.'s threshold signature scheme (inequality 2):

3. Byzantine nodes cannot produce enough valid signature shares through collusion.
4. Byzantine nodes cannot prevent the reconstruction of a signature by simply failing to respond.

File insertion: We present our file insertion function in Algorithm 2. Intuitively, for inserting a file into the target quorum, the client needs to send its copy to each node in the target quorum. To preserve the confidentiality and integrity of the file content, consider a baseline wherein the client encrypts the file to each of the target nodes, given their public keys PK_c. Again, as an optimization, it could send the file to only one of the nodes that acts as a delegate node and forwards the file to all other nodes in the quorum. We deploy the same technique as for the PHF, to ensure that these nodes get the correct file, namely, the delegate node should return threshold signature shares over a hash of the file to the client (line 6). The client ensures that the signature is over the expected hash of the file (lines 7–9) and verifies it using the quorum's verification key VK (lines 10–12).

A Byzantine delegate node may send the file to only a minimum number of honest nodes in the quorum such that it can compose a threshold signature over the hash of the file, to reply seemingly honestly to the client. When the nodes synchronize their data store through a Byzantine agreement protocol, it can deny having obtained such a file, and thus render the honest nodes as malicious or de-synchronized [7]. We specifically parameterize the fraction of Byzantine nodes as follows.

Constraint for desynchronization attack [7]:

5. The number of honest nodes that retain this file at the end of the aforementioned attack is sufficient to run the Byzantine agreement protocol correctly (inequality 5)

Algorithm 2 Our file insertion function

```
1: function DHTPIRputFile($id, F$)
2:     dest ← QPgetTargetQuorum($id$)
3:     $[(A_1, PK_1), \cdots (A_s, PK_s)] || \phi || VK$ ← dest
4:     nodes ← $[(A_1, PK_1), \cdots (A_s, PK_s)]$
5:     msg ← "INSERT_FORWARD" || $F$
6:     reply || sig ← QUORUMsendQuery($\phi$, nodes, msg)
7:     if reply ≠ Hash(F) then
8:         return Error: "Did not get the correct hash of the file in response."
9:     end if
10:    if verifySig(reply, sig, $VK$) == False then
11:        Error: "Could not verify the signature of the remote quorum."
12:    end if
13:    return
14: end function
```

Finally, we remark that a minority of Byzantine nodes that receive any file may not store it at all; instead, they may simply compute the requisite hashes for insertion queries and drop all search queries. Within our model, we assume that all honest nodes follow all protocols correctly and therefore, that they store the files. Thus, given our goodness invariant holds, enough honest nodes will reply back with correct response blocks to future search queries, such that the impact of the lazy Byzantine nodes is nullified.

PHF updates: The target quorum's nodes can recompute a PHF after each file insertion, or in occasional batches, at the cost of temporary file retrieval misses until the insertion protocol fully completes. We remark that the PHF computation cost is very small — approximately 0.25 μs per file in the database. Therefore, if nodes receive a file retrieval request while re-computing the PHF, the impact on latency due to the PHF computation is negligible, as evidenced in our latency simulation experiments. In other words, the PHF recomputation does not necessitate a period of quiescence or low network churn. After recomputing the PHF, each node should recompute its own signature share over the PHF and broadcast it to other nodes in the quorum. Any node that is chosen as a delegate node may then respond with the updated PHF and signature shares.

Byzantine client node: We have assumed so far that the client behaves as an honest node, that is, it runs the QUORUMsendQuery function correctly. However, the client may spam a target quorum's nodes through an incorrect execution of this function. We now describe rate-limiting mitigations for spamming attacks within the recipient node's QUORUMqueryListener function.

The receiving node expects each incoming message to contain a valid proof. This check extends to file insertion messages forwarded by a delegate node, thereby preventing a Byzantine delegate node from inundating other nodes with fake insertion messages. As mentioned within the QP layer description in Section 5.1, a valid proof should consist of a signature σ of a neighbouring quorum over the new node's network address A, its public key PK, and a timestamp ts. The receiving node only accepts a proof that is obtained within a predetermined time interval T of the timestamp ts from the network address A.

Table 2: **Notation for security constraints**

n	Number of peers in the network
ϵ	Network-wide ratio of Byzantine nodes to honest nodes
s	Total number of peers in a quorum
h	Honest peers in a quorum that reply to an IT-PIR query
b	Byzantine peers in a quorum (includes failing nodes)
t	Max colluding peers for IT-PIR queries [12]
τ	Threshold for Young et al.'s threshold signature scheme [36]

With these restrictions, a Byzantine client can only spam a receiving node within a time interval of T units from the timestamp in the proof, and may only do so by sending messages with the network address that is in the proof. Furthermore, the receiving node ensures that a valid proof will only be processed once for each different flag in the accompanying message within the time interval T. This check allows for an honest node to use a valid proof to conduct one insertion query and one search query with a target quorum.

Join protocol: We provide a brief overview of the join protocol here and explicate it further, along with the leave protocol, in Appendix B. We use Sen and Freedman's commensal cuckoo rule [26] to assign addresses to primary and secondary joining nodes. We use threshold signatures within Young et al.'s robust communication protocols [36] to securely notify quorums of these joins, and to securely transfer the stored database to these nodes. We also describe the leave protocol in Appendix B. As we run Kate and Goldberg's distributed key generation (DKG) scheme [17] within the join and leave protocols to redistribute a secret throughout the quorum, we must ensure that its pre-requisite is met. We thus obtain the following constraint:

Constraint for DKG [17]:

6. The fraction of Byzantine nodes per quorum should satisfy the upper bound for the DKG protocol (inequality 3).

6 SECURITY PARAMETERS ANALYSIS

In this section, we show that as long as the fraction of Byzantine nodes per quorum is less than $\frac{1}{4}$, constraints 1–6, which were identified in the previous section, hold. We present the notation used in our analysis in Table 2. We first establish some invariants using this notation.

Invariants: Out of s peers in a quorum, a large number of h peers provide honest replies whereas the rest of the peers, say b, are Byzantine and either fail to provide a reply or provide incorrect replies. Thus, we have that $s = h + b$. This equation holds in each quorum of s nodes in the network, which may have varying sizes. However, the particular value of s is irrelevant for the following calculations, and thus we normalize our equations by dividing by s and denote the resulting fractions by appending a subscript 0 to the original symbols, yielding the following, where $b_0 < h_0$:

$$h_0 = 1 - b_0 \tag{1}$$

RCP constraints: The threshold for Young et al.'s threshold signature scheme, namely τ, is bound by constraints 3 and 4, as follows.

$$b_0 < \tau_0 \leq h_0 = 1 - b_0 \tag{2}$$

The DKG protocol used by Young et al.'s scheme imposes the following bound on b_0 (constraint 6):

$$s \geq 3b + 1 \Leftrightarrow 1 \geq 3b_0 + \tfrac{1}{s} \tag{3}$$

These threshold signature shares are used to indicate a change of the state of the quorum upon insertion of a file. As a node is delegated to forward file insertion requests to all other nodes, we must prevent the desynchronization attack that was described in Section 5 to motivate constraint 5. We detail this attack here.

A Byzantine node might only forward such a request to its fellow Byzantine nodes and the minimum number of honest nodes required to obtain a threshold signature ($\tau - b$ nodes in our case) and then collude with other Byzantine nodes to pretend that it never obtained such a file. In this case, only $\tau - b$ nodes retain a copy of the new file whereas the remaining $s - (\tau - b)$ do not have such a copy. For the Byzantine agreement protocol that synchronizes the state of the data store across different nodes to be successful, a majority of nodes must retain a copy of the newly inserted file. That is, we must have $\tau - b > s - (\tau - b)$. Equivalently, after substituting equation 1, we obtain $\tau_0 > b_0 + \frac{1}{2}$.
As this inequality tightens the lower bound in inequality 2, we obtain the following range for τ_0:

$$b_0 + \tfrac{1}{2} < \tau_0 \leq 1 - b_0 \tag{4}$$

This inequality is satisfiable as long as:

$$b_0 < \tfrac{1}{4} \tag{5}$$

Note that any admissible value of b_0 that satisfies inequality 5 also satisfies inequality 3.

IT-PIR: In our notation, at most t nodes can collude without being able to reconstruct the desired row of a database from an IT-PIR query. As all of the b Byzantine nodes in a quorum *may* collude, constraint 1 requires that t can only be as small as the number of Byzantine nodes:

$$t \geq b \tag{6}$$

Second, Goldberg shows that if the following condition holds, a client can guarantee efficient reconstruction of the desired row from the responses of the s peers, b of which may be Byzantine and thus send incorrect responses, so to satisfy constraint 2, we need:

$$h > \tfrac{s+t}{2} \Leftrightarrow h_0 > \tfrac{1+t_0}{2} \tag{7}$$

Substituting equation 1 and simplifying, we get:

$$1 > 2b_0 + t_0 \tag{8}$$

Therefore, as expected, with an increase in the fraction of Byzantine peers, the IT-PIR threshold should decrease in order to reconstruct the message, without any other changes. Inequalities 6 and 8 constrain the range of this threshold, as follows.

$$1 - 2b_0 > t_0 \geq b_0 \tag{9}$$

For a value of t_0 that satisfies inequality 9 to exist, the difference between the upper and lower bounds of that inequality should be strictly greater than $\frac{1}{s}$, as follows.

$$1 > 3b_0 + \tfrac{1}{s} \tag{10}$$

Any admissible b_0 that satisfies the strictest inequality for Young et al.'s protocols, that is, inequality 5, also satisfies the strictest inequality for Goldberg's IT-PIR scheme, that is, inequality 10. Our requirement that the fractions of Byzantine nodes in each quorum be strictly less than $\frac{1}{4}$ is therefore sufficient for the requirements of Young et al.'s robust quorums, and also Goldberg's IT-PIR scheme.

Quorum size/ϵ trade-off: For a network with at most n nodes, the desired average size of quorums formed through the commensal cuckoo rule can be set beforehand to $\bar{s}$. The adversary can only insert at most $\epsilon \cdot n$ nodes into this network and re-join these nodes, as per the join protocol in Appendix B, in an attempt to overwhelm a quorum. Increasing the target $\bar{s}$ allows us to increase ϵ at the expense of a greater intra-quorum communication cost.

We extend Sen and Freedman's [26] simulation and we estimate the maximum value of ϵ that can allow only $b_0 < \frac{1}{4}$ of each quorum to be malicious. We conduct the simulations across a range of network sizes (n) and over 100K rounds of re-join attempts. We find that small networks ($n = 6400, 12800$) can sustain ϵ up to 0.03, while maintaining a desired average quorum size of $\bar{s} = 25$ and reaching a maximum observed quorum size of $s = 74$. When larger networks ($n = 10 \cdot 2^{\lambda} \mid \lambda = 18, 19, 20$) were configured with a slightly smaller target average quorum size of $\bar{s} = 20$, the maximum observed quorum size was less than 75, and can withstand ϵ up to 0.02. Therefore, even for large network sizes, we conclude that the commensal cuckoo rule can construct reasonably sized quorums that satisfy the $b_0 < \frac{1}{4}$ constraint.

7 COMPLEXITY ANALYSIS

In this section, we analyze the performance of our system in terms of its message and communication complexity for the client and server nodes, for file insertions and retrievals. In terms of computation complexity, our PIR-based [12] file retrieval function only requires efficient Lagrange interpolation for the client. Recent improvements in optimizing Lagrange interpolation [31] can be utilized in an implementation of our scheme, to allow for fast reconstruction of files from blocks returned by the s nodes of the target quorum.

Before the client can send search or insertion queries, it needs to obtain the routing information of the target quorum through the QPgetTargetQuorum function. The message complexity incurred by the client in this call depends on whether RCP-I or RCP-II is chosen; as discussed in Section 2, RCP-I has a deterministic message complexity of $O(s \cdot \ell)$ whereas RCP-II has a randomized expected message complexity of $O(\ell + s)$.

First, we show that the message complexity of our insertion and search algorithms is $O(s)$. As ℓ will be $O(\log n)$ in expectation and quorums are of size $s = \Theta(\log n)$, $O(s) = O(\ell)$, the message complexity to run the whole protocol is determined by which of the two aforementioned protocols is chosen. Second, we compare the message complexity for the target quorum to participate in our PIR-based protocol to the case when the quorum participates in Backes et al.'s OT-based one.

When an honest node in the target quorum is delegated to insert a file or respond with a PHF, it exchanges a round of messages with other nodes in the quorum ($2 \cdot (s-1) = O(s)$ messages) to obtain threshold signatures over the file or PHF hashes. Consider the baseline discussed previously in Section 5 wherein for file insertion, the client simply sends the file to all nodes, without requiring any acknowledgements back, or, for PHF retrieval, the client expects a PHF back from all nodes. As compared to this baseline, the target quorum incurs an identical message complexity, namely $O(s)$ for the entirety of file insertion, or for PHF retrieval; the messages are sent within the quorum, instead of to the client. However, the client may send the file or PHF request messages to multiple Byzantine nodes that drop them, and thus keep resending them until it reaches an honest node. Modelling the number of times that a node needs to be sampled, without replacement, until a *single* honest node is found, as a negative hypergeometric distribution, the expected number of nodes contacted is upper bounded by $\frac{1}{h_0}$. Thus the client incurs a lower message complexity (by a factor of h_0), for file insertion and PHF retrieval, than the baseline.[1]

In the next step for file retrieval, the client sends s PIR query vectors, each of size $\sqrt{\mathcal{D}}$ [12]. The client thus sends a total of $\frac{s}{h} + s$ messages and thus the message complexity for the client for file retrieval is $O(s)$. As each PIR query vector is of size $\sqrt{\mathcal{D}}$, the total communication complexity of the messages sent by the client will be $s \cdot \sqrt{\mathcal{D}} + s = O(s \cdot \sqrt{\mathcal{D}})$.

Suppose that each node in the target quorum stores $\mathcal{F}_Q$ document chunks, each of which is L bytes long, that is, a total of $\mathcal{D} = \mathcal{F}_Q \cdot L$ bytes. Botelho et al.'s perfect hash function construction [6] for indexing these chunks allows the delegate node to send the PHF in $\kappa \cdot \mathcal{F}_Q$ bytes, where $.243 < \kappa < .3375$. Each PIR response is of size $\sqrt{\mathcal{D}}$ bytes [12]. When all Byzantine nodes send PIR responses, along with the honest nodes, the quorum incurs a worst-case communication complexity of $s \cdot \sqrt{\mathcal{D}}$ bytes. Note that the large PIR responses outweigh the small PHFs for the communication complexity of the target quorum for reasonably sized quorums and numbers of stored files per quorum.

In contrast, consider Backes et al.'s proposed extension to their OT-based QP-RCP protocols, so that OT is used to retrieve the file itself privately from the data store in the target quorum. In this case, a delegate node responds back with the entry-wise encrypted data store that is $\mathcal{F}_Q L$ bytes long as well as a total of $2 \cdot \mathcal{F}_Q + 1$ OT parameters, each of a small constant length γ bytes. For our scheme to incur a lower communication complexity for the target quorum than Backes et al.'s, we must have:

$$s \cdot \sqrt{\mathcal{D}} + \kappa \cdot \mathcal{F}_Q < \mathcal{D} + (2\mathcal{F}_Q + 1) \cdot \gamma$$

As $\kappa \ll 2 \cdot \gamma \ll L$, we drop the terms containing κ and γ, to obtain:

$$\mathcal{D} = \mathcal{F}_Q \cdot L > s^2 \tag{11}$$

The expected number of files per quorum (expected value of $\mathcal{F}_Q$) will be the total number of files in the network (say $\mathcal{F}_N$) divided by the number of quorums ($\frac{n}{s}$), yielding $\frac{s\mathcal{F}_N}{n}$. Substituting this value for $\mathcal{F}_Q$ into inequality 11, we get the following lower bound on the number of files in the network for our scheme to have a lower communication complexity for quorums, than Backes et al.'s:

$$\mathcal{F}_N > \frac{sn}{L} \tag{12}$$

[1]As the client waits to obtain a response before retrying with another node, it incurs a higher expected *latency* in our new scheme than the baseline. The client may send the file in parallel to multiple delegate nodes for insertion queries, significantly increasing its chances of reaching an honest node, and reducing the latency at the cost of a higher message complexity. Similarly, it may query several nodes in parallel for the PHF, each of which will return the signature over the PHF, for search queries.

As $s \ll L$, our scheme requires less communication complexity for quorums than Backes et al.'s private file retrieval variant, when any reasonably large numbers of files are stored in the network.

8 EVALUATION

The communication and computation costs for retrieving a file for our system should scale when the number of nodes or users that connect to the P2P network (n) increases, as well as when the number of files that are stored in the network (F_N) increases. We have implemented a simulation of our system in order to evaluate these costs for the client and target quorums' nodes in our system. We briefly describe our simulation, and then we focus on experiments that demonstrate the scalability of our system with F_N, although our simulations show that these costs for our system scale well with both n and F_N. We also remark that the communication and computation costs for publishing documents remain reasonably low. Our simulation code and output, as well as additional graphs, can be found at https://git-crysp.uwaterloo.ca/dhtpir/simulations.

Simulation setup: In our simulation, one node serves as the client and creates F_N randomly generated document chunks, each of size 1 KiB. It stores each chunk on the quorum identified by the ID of that chunk. The client then attempts to retrieve each chunk from the network. We individually simulate each of the layers used within our system: we start with a simple DHT-based P2P network — a Base DHT that implements file insertion and retrieval. For all other layers in our simulation, we instantiate $q = 100$ quorums, each of which has $s = 10$ nodes. (We simulated our interface for (q, s) ranging from $(10, 5)$ to $(2000, 16)$; we discuss results of the $(100, 10)$ case as it is representative of this range.)

The next layer is a simulation of Young et al.'s probabilistic robust communication protocol (RCP-II) to efficiently route queries across quorums. That is, our RCP simulation supports file insertion and retrieval queries by replacing the routing protocol in the Base DHT simulation with the RCP-II protocol. (We remark that none of the nodes in our simulation behaves in a Byzantine manner.) We then simulate Backes et al.'s QP extension to Young et al.'s RCP-II protocol, to conduct private routing queries, in the RCP+QP simulation. Finally, we include two protocol simulations that enable clients to privately retrieve files. We simulate our own DHTPIRfindFile and DHTPIRputFile functions within the DHTPIR simulation. We also simulate the closest related work, namely extending Backes et al.'s OT protocol for the *last hop* to the target quorum, in order to privately retrieve a chunk from it; we refer to this protocol simulation as LastHop. We delegate one node from the target quorum to conduct the OT-based protocol for LastHop, and use threshold signatures just as with RCP-QP-II.

For our performance simulation, we use the following estimates of network and computation speeds. We set the network bandwidth to 50 Mb/s, and the round-trip time (RTT) to 150 ms. We estimate the PIR computations to run at the rate of 0.25 s per GB of database, whereas encryption operations run at 1 GB/s (about 3 cycles/byte for AES-NI). For all experiments, we increased F_Q roughly exponentially, that is, $F_Q \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$. (A total of $F_N = F_Q \cdot q$ chunks are stored in the network.)

Communication complexity: We plot the total communication complexities over 25 runs for the DHTPIR, LastHop and RCP+QP

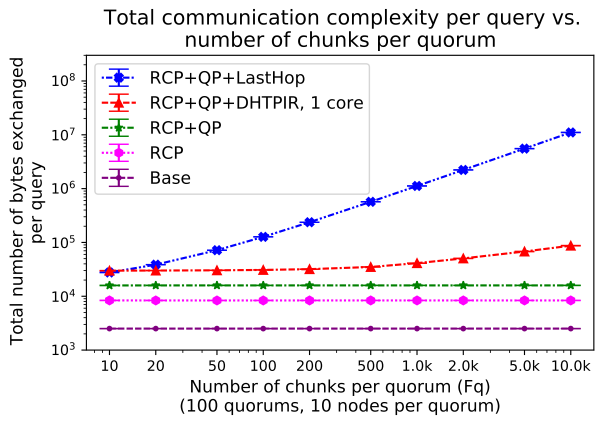


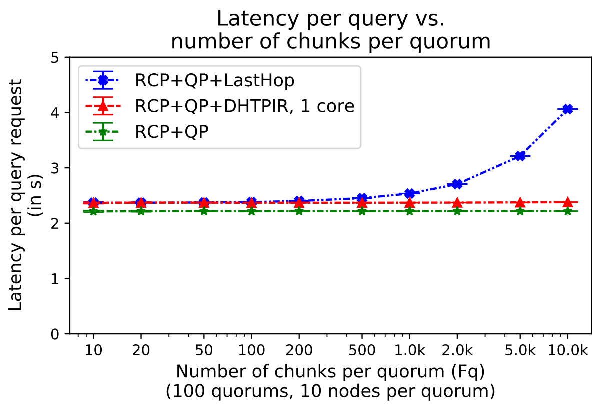
FIGURE 4: **Measurement of the total communication complexity of one query for one chunk, including the request and the response, as the number of chunks stored in each quorum increases roughly exponentially. This plot shows the average communication complexity over 25 runs of the simulation. Note the logarithmic scales. Error bars are shown, but they are too small to see.**

simulations in Figure 4. For the LastHop simulation, as explained earlier, the OT request sent by the client is constant in size with respect to increasing F_Q, whereas the OT setup and response sent by the delegate node is proportional to F_Q in size. Thus, the total communication complexity of the LastHop simulation increases linearly in F_Q in Figure 4.

In contrast, DHTPIR provides a smaller communication complexity for file retrieval *responses*, at the cost of a larger communication complexity for the corresponding *requests*. The total communication complexity overhead of DHTPIR over RCP includes the cost for the PHF request, PIR request and PIR response. This overhead increases by approximately the asymptotic amount $(s \cdot \sqrt{F_Q \cdot L})$ of bytes discussed in Section 7, as expected.

Performance simulations: We measure the latency of DHTPIR and LastHop simulations to retrieve a file. We also measure their throughput in terms of number of simultaneous clients' file retrieval requests that can be handled by the entire system. (We measure the latency and throughput of *routing* requests for RCP+QP, as routing requests are the bottleneck for that protocol, which has no privacy protection for the actual retrieval.) We plot the latencies of these three systems as F_Q increases in Figure 5a. Both LastHop and DHTPIR incur two additional round trips over RCP+QP: to obtain PHF or OT parameters and then to obtain a PIR or OT response. However, as LastHop includes the entire database (up to $\approx$ 10 MB in the figure) within the OT response, we can see in Figure 5a that as the number of chunks per quorum (F_Q) increases, LastHop incurs a linearly increasing latency overhead over RCP+QP. On the other hand, for DHTPIR, as the PIR requests and responses only grow proportional to $\sqrt{F_Q}$, we do not observe a significant increase in latency due to computational costs or communication costs, over the range of F_Q in Figure 5a.

We plot the throughput of RCP+QP, LastHop, and DHTPIR as F_Q increases in Figure 5b. For LastHop, as only a single delegate node in the target quorum may compute the OT response (and computing



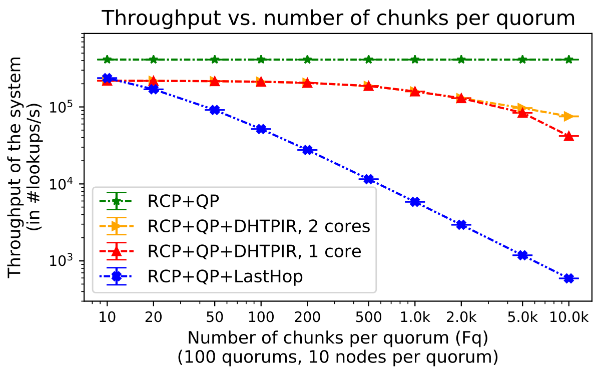


FIGURE 5: **Measurement of the (a) latency and (b) throughput, as the number of chunks stored in each quorum increases roughly exponentially. These plots show the average latency and throughput over 25 runs of the simulation. Note the logarithmic y-axis in (b). Error bars are shown, but they are too small to see.**

threshold signatures is much faster), multiple nodes can process distinct OT requests in parallel. For DHTPIR, *each* node participates in computing the PIR response for a single PIR request; adding another core allows for parallel processing of multiple *distinct* PIR requests. (Adding more cores has quickly diminishing effects, however, as the network speed soon becomes the bottleneck.)

We can see in Figure 5b that DHTPIR experiences a small constant drop in its throughput over RCP, until $F_Q \approx 2000$ — it stays network-limited in this range — beyond which, adding another core results in a significant improvement in throughput, which can be extrapolated to expand with database size. Thus, DHTPIR affords opportunities to optimize its throughput for nodes that store large databases, through multi-core processing. (We do not consider processing a single PIR request across multiple cores or threads; though efficient parallelizable matrix multiplication algorithms can be used to achieve even greater throughput for DHTPIR [20].) Even though each *node* in LastHop can process multiple distinct OT requests in parallel, it incurs a large reduction in throughput as compared to RCP, proportional to the size of the database ($F_Q \cdot L$). As LastHop is significantly network limited, adding more cores would not increase its throughput. DHTPIR thus achieves a throughput that is about two orders of magnitude larger than LastHop and this gap widens with database size.

Our performance simulations show that DHTPIR incurs almost no change in latency and only a modest drop in throughput over the baseline RCP+QP simulation (which again does not protect the privacy of the queries from the target quorum at all), across increasing database sizes. DHTPIR shows a marked improvement in latency and throughput across larger databases over LastHop, and scales significantly better in comparison. Our simulation thus provides us reasonable grounds to believe in the efficiency of a DHTPIR implementation.

9 DEPLOYMENT

Although we have focused on CROPS in order to motivate our interface, an implementation of our interface can be used to publish and privately retrieve a block of data within other publishing systems, such as in Tangler. Moreover, designers of an upcoming distributed file storage system, namely the Interplanetary File System (IPFS), have proposed establishing quorums [16] as well as privately fetching content from nodes using Young et al. and Backes et al.'s systems [13], among other privacy features [23]. As our design integrates the aforementioned systems, it would be a useful starting point to implementing such proposals to provide plausible deniability for server nodes. Our design can also support other applications that are built atop DHTs, such as distributed health and medical data repositories. A system like ours is also useful to integrate into DHTs used for commercial settings, to minimize the amount of potentially sensitive data nodes can collect.

In the CRP setting, we rely on the developers of the underlying CRP system to provide methods to bootstrap a user with the complete client-side software, which would include a DHTPIR implementation. For instance, a service like GetTor [24] can be used to deliver the software. To join the network, DHT-based systems typically require a new node to contact a node from a small list of permanently online nodes [25], which is included in the software package. These nodes are trusted to provide correct routing information; that is, to not conduct an eclipse attack. For a CRP system that uses DHTPIR, including the quorum verification keys of these nodes' current quorums with this list allows a new node to detect and prevent an eclipse attack, through the robust communication protocols. Similarly, we also rely on popular adoption of the underlying CRP system. For instance, the system can be seeded with popular, non-prohibited content, thereby inflicting a high collateral damage onto the censor if the entire system were to be banned.

10 CONCLUSION

Censorship-resistant publishing systems are built atop DHTs and enable users to store sensitive documents onto multiple server nodes in different administrative regions, such that a censor cannot easily take down or tamper with the published content. Server node administrators may be compelled by censors to reveal the documents retrieved by a given user. We propose an interface that uses information-theoretic private information retrieval (IT-PIR) to prevent node operators from being exposed to information about which document was retrieved.

We integrate existing work on hardening peer-to-peer networks so that quorums of nodes only contain at most a certain fraction of Byzantine nodes; we allow the censor to run such colluding Byzantine nodes. Our key insight lies in using quorums as a coalition of server nodes that store a set of files over which IT-PIR queries can be performed. We simulate our system and find that its total communication cost is up to two orders of magnitude smaller than the closest related work, while simultaneously achieving lower latency and higher, parallelizable throughput, all of which scale reasonably with the database size. We hope that our design spurs further research and development efforts into building robust censorship-resistant publishing systems.

ACKNOWLEDGMENTS

We thank the Royal Bank of Canada and NSERC grant CRDPJ-534381 for funding this work. This research was undertaken, in part, thanks to funding from the Canada Research Chairs program. This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo.

REFERENCES

[1] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR: Private Information Retrieval for Everyone. *Proceedings on Privacy Enhancing Technologies* 2016, 2 (2016), 155–174.

[2] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 962–979.

[3] Baruch Awerbuch and Christian Scheideler. 2009. Towards a Scalable and Robust DHT. *Theory of Computing Systems* 45, 2 (2009), 234–260. https://doi.org/10.1007/s00224-008-9099-9

[4] Michael Backes, Ian Goldberg, Aniket Kate, and Tomas Toft. 2012. Adding Query Privacy to Robust DHTs. In *7th ACM Symposium on Information, Computer and Communications Security* (Seoul, Korea) *(ASIACCS '12)*. 30–31.

[5] Ingmar Baumgart and Sebastian Mies. 2007. S/Kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*. IEEE, 1–8.

[6] Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. 2013. Practical Perfect Hashing in Nearly Optimal Space. *Information Systems* 38, 1 (2013), 108–131. https://doi.org/10.1016/j.is.2012.06.002

[7] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*.

[8] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private Information Retrieval. *Journal of the ACM* 45, 6 (1998), 965–981. https://doi.org/10.1145/293347.293350

[9] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium* (San Diego, CA). 21–21.

[10] Amos Fiat, Jared Saia, and Maxwell Young. 2005. Making Chord Robust to Byzantine Attacks. In *Proceedings of the 13th Annual European Conference on Algorithms* (Palma de Mallorca, Spain) *(ESA'05)*. Springer-Verlag, Berlin, Heidelberg, 803–814.

[11] Eva Galperin. 2017. Access Now and EFF Condemn the Arrest of Tor Node Operator Dmitry Bogatov in Russia. https://www.eff.org/deeplinks/2017/04/access-now-and-eff-condemn-arrest-tor-node-operator-dmitry-bogatov-russia.

[12] Ian Goldberg. 2007. Improving the Robustness of Private Information Retrieval. In *2007 IEEE Symposium on Security and Privacy (SP '07)*. IEEE, 131–148.

[13] gpestana. 2018. Privacy preserving DHTs. https://github.com/gpestana/notes/issues/8.

[14] Anne Henochowicz. 2015. The Human Side of Censorship: Keyword Filtering and Censorship Directives on the Chinese Internet. In *Proceedings of the 5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 2015)*. Washington D.C., USA. https://www.usenix.org/sites/default/files/conference/protected-files/foci15_slides_henochowicz.pdf

[15] Michelle Hertzfield, Jessica Schilling, and Oli Evans. 2019. How IPFS Works - IPFS Documentation. https://docs.ipfs.io/introduction/how-ipfs-works/.

[16] jbenet. 2015. IPFS Feedback. https://github.com/ipfs/notes/issues/318.

[17] Aniket Kate and Ian Goldberg. 2009. Distributed Key Generation for the Internet. In *29th IEEE International Conference on Distributed Computing Systems*. 119–128. https://doi.org/10.1109/ICDCS.2009.21

[18] Jeffrey Knockel, Masashi Crete-Nishihata, Jason Q. Ng, Adam Senft, and Jedidiah R. Crandall. 2015. Every Rose Has Its Thorn: Censorship and Surveillance on Social Video Platforms in China. In *Proceedings of the 5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 2015)*. Washington D.C., USA. https://www.usenix.org/system/files/conference/foci15/foci15-paper-knockel.pdf

[19] J. Liang, N. Naoumov, and K. Ross. 2006. The Index Poisoning Attack in P2P File Sharing Systems. In *Proceedings IEEE International Conference on Computer Communications (INFOCOMM)*. 1–12. https://doi.org/10.1109/INFOCOM.2006.232

[20] Wouter Lueks and Ian Goldberg. 2015. Sublinear Scaling for Multi-Client Private Information Retrieval. In *Financial Cryptography and Data Security*. 168–186.

[21] Petar Maymounkov and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems*, Peter Druschel, Frans Kaashoek, and Antony Rowstron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 53–65.

[22] Miti Mazmudar, Stan Gurtler, and Ian Goldberg. 2021. Do you feel a chill? Using PIR against chilling effects for censorship-resistant publishing. In *20th ACM Workshop on Privacy in the Electronic Society*.

[23] mgoelzer. 2018. [Protocol Design] How to create a fully private DHT. https://github.com/libp2p/developer-meetings/issues/6.

[24] The Tor Project. 2020. What is GetTor? https://gettor.torproject.org/.

[25] Jessica Schilling, Chris Waring, and Bertrand Falguiere. 2020. Modify the bootstrap peers list. https://docs.ipfs.io/how-to/modify-bootstrap-list/.

[26] Siddhartha Sen and Michael J. Freedman. 2012. Commensal Cuckoo: Secure Group Partitioning for Large-Scale Services. *ACM SIGOPS Operating Systems Review* 46, 1 (2012), 33–39. https://doi.org/10.1145/2146382.2146389

[27] Signal. 2016. Grand jury subpoena for Signal user data, Eastern District of Virginia. https://signal.org/bigbrother/eastern-virginia-grand-jury/.

[28] A Singh, T-W Ngan, P Druschel, and DS Wallach. 2006. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proceedings IEEE International Conference on Computer Communications (INFOCOMM)*. IEEE, 1–12.

[29] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. 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* (San Diego, California, USA) *(SIGCOMM '01)*. 149–160.

[30] Adam Stubblefield and Dan S. Wallach. 2002. *Dagster: Censorship-Resistant Publishing Without Replication.* Technical Report. Rice University.

[31] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. 2020. Towards Scalable Threshold Cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*.

[32] Eugene Y. Vasserman, Victor Heorhiadi, Nicholas Hopper, and Yongdae Kim. 2012. One-Way Indexing for Plausible Deniability in Censorship Resistant Storage. In *Proceedings of the 2nd USENIX Workshop on Free and Open Communications on the Internet*. USENIX, Bellevue, WA.

[33] Eugene Y. Vasserman, Victor Heorhiadi, Yongdae Kim, and Nicholas J. Hopper. 2011. *Censorship resistant overlay publishing.* Technical Report 11-027. University of Minnesota.

[34] Marc Waldman and David Mazières. 2001. Tangler: A Censorship-Resistant Publishing System Based On Document Entanglements. In *Proceedings of the ACM Conference on Computer and Communications Security*. ACM, 126–135.

[35] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. 2000. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of the 9th USENIX Security Symposium*. USENIX, 59–72.

[36] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. 2013. Towards Practical Communication in Byzantine-resistant DHTs. *IEEE/ACM Transactions on Networking* 21, 1 (2013), 190–203.

A API FUNCTIONS USED AND PROVIDED

As described in Section 5, our interface can be used within censorship-resistant publishing systems, such as Vasserman et al.'s CROPS [33], to insert files into quorums through the DHTPIRputFile function and to query quorums privately for files through the DHTPIRfindFile function, both provided by our DHTPIR API. Functions in the DHTPIR interface in turn invoke existing QP (which itself invokes RCP), QUORUM, and PIR protocols in a layered manner, as shown in Figure 2. A list of all API functions can be found for reference in Table 3.

TABLE 3: **Overview of the functions in each layer of the interfaces presented in Section 5. The layer name is included as the first word of the function name and is capitalized.**

Function	Arguments	Output	Description
DHTPIRputFile	File identifier id File F	None	*Privately* inserts a file F with the ID id. Replaces the native putFile function.
DHTPIRfindFile	File identifier id	File F	*Privately* finds a file with the ID id. Replaces the native findFile function.
QPgetTargetQuorum	File identifier id	Proof of robust communication: ϕ Target nodes' network addresses: $[A_1 \cdots A_s]$ and public keys: $[PK_1 \cdots PK_s]$ Quorum verification key: VK	Obtain the output information for the closest quorum that stores the file with the ID id, without any intermediate node learning the ID.
QUORUMsendQuery	Proof: ϕ Target nodes' network addresses: $[A_1 \cdots A_s]$ and public keys: $[PK_1 \cdots PK_s]$ Messages: $[M_1 \cdots M_s]$	Received messages: $[R_1 \cdots R_s]$	Used for securely sending a message to a target quorum and receiving replies from it.
QUORUMqueryListener	None	None (replies internally to received messages)	Used for securely receiving messages from nodes in other quorums and sending replies to them, possibly after coordinating with other nodes in own quorum.
PIRgetQueryVectors	File identifier id Perfect hash function f Number of nodes s	Query vectors $[q_1 \cdots q_s]$	Computes s query vectors, one for each server that holds a database indexed by the output of a perfect hash function f, in order to obtain a file at index id.
PIRrecreateFile	File blocks $[f_1 \cdots f_s]$	D	Returns a file F that is reconstructed from the s input blocks.
PIRperformQuery	Query vector q	Response block d	Queries the node's file store for the input query q, and returns a response block d.

B JOIN PROTOCOL

A new node p interacts with multiple quorums as it joins the network; we illustrate the join protocol through the sequence diagram in Figure 6. When a new node joins the network, it first contacts a bootstrapping peer q, which is a part of quorum Q_1 for instance, and sends its public key PK_p. Peer q then sends network addresses and public keys of its peers, signed by quorum Q_1, thereby allowing node p to send this bootstrapping message to node q's peers. Nodes in the quorum Q_1 collectively generate a random address using their shares of the quorum's shared secret s. They generate a cryptographic hash of the new node's public key PK_p, and include a timestamp for freshness. Treating this hash as a group element, they raise it to the power of their share, and send it to the new node. Node p combines these shares to obtain a random address id_p.

To contact the target quorum Q_T that spans address id_p, node p obtains a valid proof of communication by running the QPgetTargetQuorum function. This function may be run with either RCP-I or RCP-II as the underlying communication protocol, though there are two small changes when it is run for primary or secondary joins. First, the lookup message m_p also includes its securely assigned virtual address id_p and a string to demarcate the message as a primary (or secondary) join, on top of the node's network address A_p, public key PK_p and a timestamp ts. We need to distinguish primary joins from a secondary join, as a new Byzantine node could masquerade as a secondary join to avoid causing the eviction of other Byzantine nodes in its target quorum. Second, for a primary join, as node p does not belong to a quorum, the bootstrapping peer's quorum Q_1 produces a threshold signature over the message m_p, which is denoted by $[m_p]_1$ within our diagram. Through this function, node p then proceeds to contact multiple quorums, to ultimately obtain the usual response; that is, a threshold signature over m_p by the nearest neighbour of the target quorum, namely quorum Q_{T-1}, as well as network addresses and public keys of nodes in the target quorum $((A_i, PK_i)_{i=1}^{s})$, and the verification key of the target quorum VK_T. Node p then forwards the message m_p and the threshold signature $[m_p]_{T-1}$ to the target quorum.

The target quorum verifies the threshold signature $[m_p]_{T-1}$, using the verification key of its neighbour quorum $VK_{Q_{T-1}}$, as usual. When the signed message contains the primary join flag, a node in Q_T must first vet the join, as proposed within the commensal cuckoo rule (step 1CCR in Figure 1). That is, each node first ensures that the quorum has had at least $k-1$ secondary joins since the last primary join into the quorum. Here, k is set such that an average of k nodes are cuckoo-ed out in a k-region that spans a fraction $\frac{k}{n}$ of the address space. (In case this check fails, the target quorum replies back with a signed failure message that is bound to the input message m_p. Node p repeats the aforementioned process with the bootstrapping peer, until it reaches a new target quorum that allows the primary join.) Nodes in Q_T then compute the number k' of its nodes that must be evicted from it and rejoin another quorum, following Sen and Freedman's commensal cuckoo rule. They then recompute the size of its quorum based on k'. Finally, each node in Q_T generates a message m_{st} that consists of the database hash, the new quorum size, and a timestamp. Each node computes its signature share over this message, and forwards it to one delegate node, say node j. This delegate node derives a shared secret with node p, using the latter's public encryption key and its own private key SK_j and uses this shared secret to encrypt the database to node p. Node j merges the threshold signature shares to a threshold signature ρ_T and forwards it along with the message m_{st} and the encrypted database, to the new node p.

The new node verifies the threshold signature ρ_T using the quorum verification key VK_T that it obtained earlier. It ensures freshness of the message m_{st} by checking its timestamp. It uses the peer node j's public encryption key PK_j that it obtained earlier to establish the same shared secret sk, and decrypts the encrypted database in the message. It computes a hash of this database and checks it with the hash contained within the message, to detect if peer j maliciously sent an incorrect copy of the database. (In this case, it sends $(m_p, [m_p]_{T-1})$ to another node of the target quorum.) The new node runs Kate and Goldberg's distributed key generation (DKG) scheme [17], in conjunction with the *remaining* nodes of the target quorum, to derive a signing key share for itself and verification key shares for these nodes. This scheme requires that the fraction of Byzantine nodes per quorum should be less than a third, as explicted in constraint 6. The threshold for this threshold signature scheme should be maintained to satisfy constraints 3 and 4. Thus, after a certain number of new node insertions, a new signing/verification key may need to be generated and the neighbouring quorums would need to be informed of the verification key. The target quorum also updates its routing table entries to include the network address A_p and public key PK_p of the new node, and excludes addresses and keys of the k' old nodes. The RCP-II protocol involves informing the quorum's neighbours of these updates in the routing table entries; we defer the reader to Section IV-C in Young et al.'s paper for details [36].

The nodes to be evicted from the target quorum do not participate in the DKG. After computing these threshold signatures, each node in the target quorum proceeds to evict k' of its nodes as follows. They generate a random node address, in a similar manner as before, compute a cryptographic hash of it, and expand this hash to construct k' new random addresses for the nodes to be evicted. They then identify which nodes in the quorum should be evicted, based on their current addresses and the new set of addresses. For each evicted node, the target quorum generates a message m_i and threshold signature $[m_i]_T$ over it; m_i is identical to the message m_p in structure for all fields, but differs in that its string flag indicates a secondary join. Each evicted node follows a similar process as the new node did with the bootstrapping quorum, in order to communicate with its new target quorum. (Such nodes may now safely delete their databases.) It runs the QPgetTargetQuorum function as node p did earlier in the protocol, and forwards m_p as well as a threshold signature by the neighbour of its new target quorum, $[m_i]_{S-1}$ to the new target quorum Q_S. The quorum Q_S accepts all incoming secondary joins and secondary joins do not result in further cuckoos. So, with the exception of the corresponding check for primary joins and the generation of IDs, and signed join messages (id_i, m_i, ρ_i), this quorum and the secondary joining node repeat the process of sharing the database, and generating $(\|Q_S\|+1)$ new key shares.

Leave protocol: If a node decides to leave a quorum (outside of a primary join), then it can send a time-stamped message which is signed by its own private signing key. The remaining nodes may then need to re-run the DKG protocol, or regenerate the quorum's signing/verification key, depending on the current value of the threshold for the threshold signature scheme.

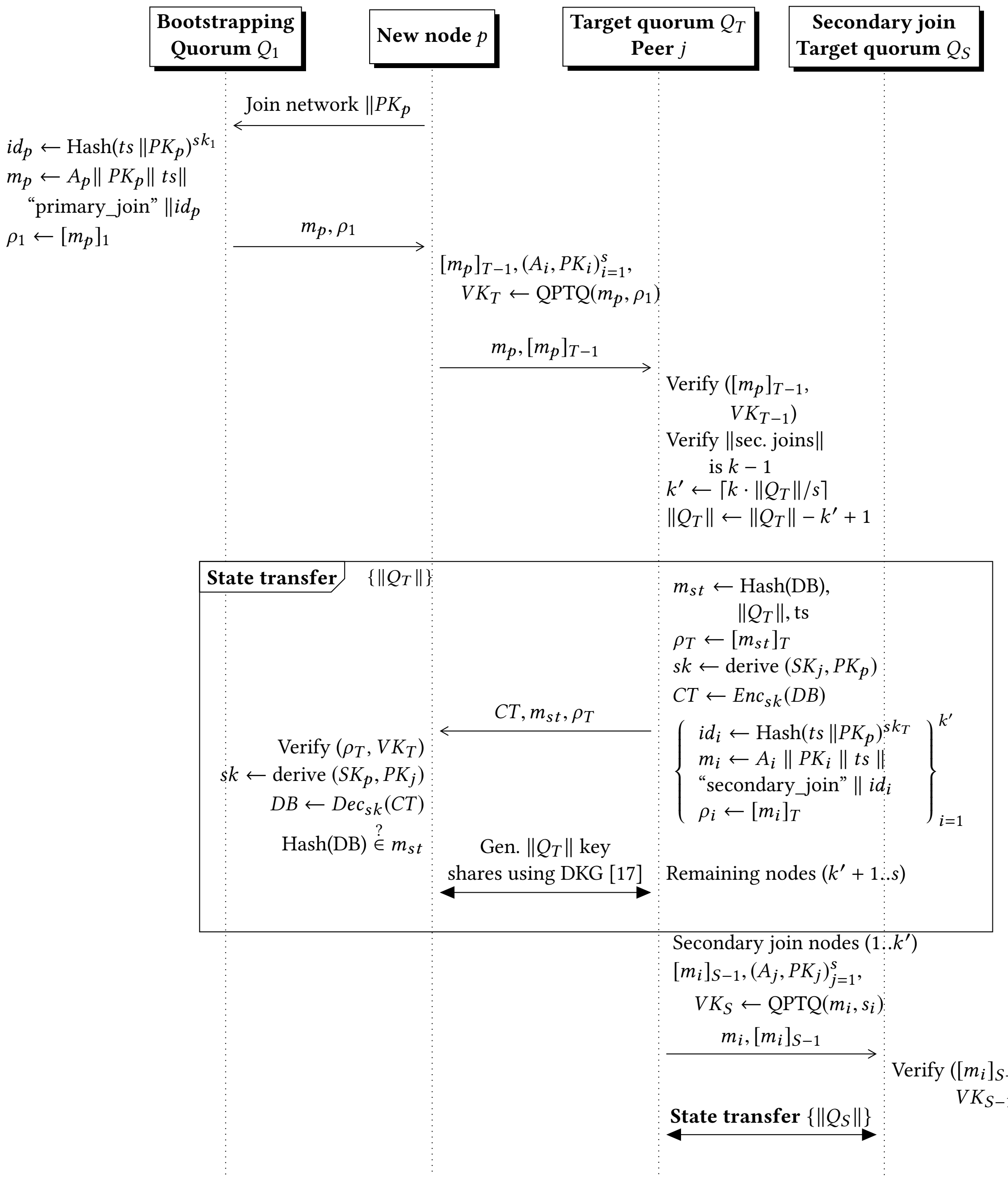

Bootstrapping Quorum Q_1
New node p
Target quorum Q_T Peer j
Secondary join Target quorum Q_S
Join network $\|PK_p$
$id_p \leftarrow \text{Hash}(ts\ \|PK_p)^{sk_1}$
$m_p \leftarrow A_p\|\ PK_p\|\ ts\|$ "primary_join" $\|id_p$
$\rho_1 \leftarrow [m_p]_1$
m_p, ρ_1
$[m_p]_{T-1}, (A_i, PK_i)_{i=1}^{s}, VK_T \leftarrow \text{QPTQ}(m_p, \rho_1)$
$m_p, [m_p]_{T-1}$
Verify $([m_p]_{T-1}, VK_{T-1})$
Verify $\|$sec. joins$\|$ is $k-1$
$k' \leftarrow \lceil k \cdot \|Q_T\|/s \rceil$
$\|Q_T\| \leftarrow \|Q_T\| - k' + 1$
State transfer $\{\|Q_T\|\}$
$m_{st} \leftarrow \text{Hash(DB)}, \|Q_T\|, \text{ts}$
$\rho_T \leftarrow [m_{st}]_T$
$sk \leftarrow \text{derive}\ (SK_j, PK_p)$
$CT \leftarrow Enc_{sk}(DB)$
CT, m_{st}, ρ_T
Verify (ρ_T, VK_T)
$sk \leftarrow \text{derive}\ (SK_p, PK_j)$
$DB \leftarrow Dec_{sk}(CT)$
$\text{Hash(DB)} \overset{?}{\in} m_{st}$
$\left\{ \begin{array}{l} id_i \leftarrow \text{Hash}(ts\ \|PK_p)^{sk_T} \\ m_i \leftarrow A_i \parallel PK_i \parallel ts \parallel \\ \text{"secondary\_join"} \parallel id_i \\ \rho_i \leftarrow [m_i]_T \end{array} \right\}_{i=1}^{k'}$
Gen. $\|Q_T\|$ key shares using DKG [17]
Remaining nodes $(k'+1..s)$
Secondary join nodes $(1..k')$
$[m_i]_{S-1}, (A_j, PK_j)_{j=1}^{s}, VK_S \leftarrow \text{QPTQ}(m_i, s_i)$
$m_i, [m_i]_{S-1}$
Verify $([m_i]_{S-1}, VK_{S-1})$
State transfer $\{\|Q_S\|\}$

FIGURE 6: **A new node p informs nodes in the bootstrapping quorum Q_1 of its public key PK_p in a bootstrapping message. This quorum then generates a random identifier id_p using its shared secret sk_1, creates a primary join routing lookup message m_p which includes id_p, and quorum 1's threshold signature over m_p, denoted by $[m_p]_1$. Q_1 then forwards $(m_p, [m_p]_1)$ to node p. Node p then contacts the target quorum Q_T that spans id_p, by passing its message and signature to the QPgetTargetQuorum (QPTQ) function. Q_T verifies this signature as usual, and only accepts this primary join if it has had sufficient secondary joins since the last primary join. If so, each node in Q_T computes the number of nodes that must be kicked out from it (k'), and updates the quorum size. All nodes then compute threshold signatures over a timestamped message that includes a hash of the database and the new quorum size. A delegate peer j forwards this message and shares to node p, along with a copy of the database that is encrypted to PK_p. In parallel with node p processing this message, all nodes in Q_T construct random IDs for the nodes to be evicted ($id_1..id'_k$) just as quorum Q_1 did for node p, and identify which nodes should be evicted based on these IDs (nodes 1–k'). They also generate signature shares over secondary join lookup messages, as nodes in quorum Q_1 did for the primary join node p. In the meantime, node p verifies the signature and the integrity of the database. Along with the remaining nodes, node p runs the DKG protocol to generate a new signature share and Q_T verification key shares. The evicted nodes (nodes 1–k') then contact their respective target quorums (quorum Q_S here) through the QPTQ function as before, using $(m_i, [m_i]_{S-1})$. After signature verification, all nodes in the target quorum engage in an identical state transfer process with node i as before. Similarly, they also run the DKG protocol with node i to establish new key shares.**