Improving Kademlia Lookup Privacy through Query Obfuscation

Erik Daniel*
Technische Universität Dresden
Dresden, Saxony, Germany
erik.daniel@tu-dresden.de

Guillaume Michel*
Interplanetary Shipyard
Zug, Zug, Switzerland
guillaume@ipshipyard.com

Florian Tschorsch
Technische Universität Dresden
Dresden, Saxony, Germany
florian.tschorsch@tu-dresden.de

Abstract

This paper addresses privacy challenges inherent in Distributed Hash Tables (DHTs). While DHTs facilitate efficient content lookup, privacy concerns arise due to query mechanisms revealing user interests. In the paper, we focus on Kademlia-based DHTs and propose to obfuscate the lookup item by presenting three obfuscation methods: double hashing, Private Set Intersection, and prefix fetching. Based on our privacy improvements, we present a protocol specification for the libp2p kad-dht, a popular Kademlia implementation. The methods are analyzed in the context of measurement values derived from the public IPFS network, which uses kad-dht.

CCS Concepts

• **Networks** → **Peer-to-peer protocols**; • **Security and privacy** → *Privacy-preserving protocols.*

Keywords

P2P Overlay Network, Kademlia, Privacy

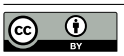
ACM Reference Format:
Erik Daniel, Guillaume Michel, and Florian Tschorsch. 2025. Improving Kademlia Lookup Privacy through Query Obfuscation. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3672608.3707703

1 Introduction

Distributed Hash Tables (DHT) provide an efficient solution for organizing Peer-to-Peer (P2P) overlay networks, facilitating content and peer discovery. Essentially, a DHT works as a distributed key-value store. In the DHT topology, a key serves as a unique identifier for both peers and content. The key is linked to the peer's address and the content provider's identifier, respectively. The key distribution varies by DHT. A prime example is Kademlia [11].

Kademlia is used in many P2P overlay networks, such as the InterPlanetary File System (IPFS) [4], BitTorrent [5], Swarm [18], and SAFE [10]. In Kademlia, entries of the DHT are distributed to the closest peer based on the XOR distance between the value and the peer identifier. By systematically traversing peers in the DHT, it becomes possible to locate the content provider for specific items. This approach enables a peer to identify a content provider by querying only a fraction of the entire network.

*These authors contributed equally to the paper.



SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707703

While a DHT allows scalable content lookup, the mechanism reveals information about a user's interest. The query reveals the interest of the initiator in the keys. The key, a pseudonymous representation of the content, indirectly reveals the content. The value of the answer reveals the content provider. This allows a DHT peer to learn the interest of all senders, requesting keys in its area of the key-space. Although only part of the network has to be queried to find the content provider, collaborating peers can learn interests of peers. For privacy, source or content of a query needs protection.

While obfuscating the sender of a query [8, 14, 19] protects the sender's identity, a sensitive item of a query might be even more critical. A malicious entity might select its observation target based on interest in a specific item. In this scenario, query privacy is necessary to mitigate target recognition in the first place.

In this paper, we focus on improving the lookup privacy in Kademlia-based DHTs by obfuscating query content. We utilize Private Set Intersection (PSI) to hide the item of interest from all queried peers. Specifically, we show three protection methods: a lightweight hashing approach, a computation heavier approach using an Elliptic Curve Diffie-Hellman PSI protocol (ECDH-PSI), and requesting a range of items. The presented methods can be combined for different privacy-utility trade-offs. For better understanding, we explain our methods, in the context of the libp2p kad-dht[1], which is used among others for routing in the IPFS P2P overlay network. Focusing on kad-dht allows us to gain insights of the overhead through sampling the IPFS network.

Our analysis is based on measurements of the public IPFS network's DHT. We show that our presented methods have different privacy and performance advantages: All methods obfuscate the content of a request from DHT nodes at the cost of a certain overhead. While the methods do not add RTTs, the response size of queries increases. A mix of hashing and PSI provides the biggest privacy improvements and the largest overhead. Combining hashing and prefix fetching makes it easier to crawl the DHT. If crawling content is deemed acceptable, the mix of hashing and prefix fetching has the best privacy-utility trade-off. Otherwise, prefix fetching in combination with PSI provides the best trade-off.

Our main contribution consists of three combinable approaches to improve lookup privacy in Kademlia. The remainder is structured as follows: In Section 2, we present related work. We explain the functionality of Kademlia and our privacy improvements in Section 3 and a protocol specification for IPFS in Section 4. Section 5 analyzes the consequences of our methods based on measurements of the public IPFS network. Section 6 concludes the paper.

[1] https://github.com/libp2p/specs/blob/master/kad-dht/README.md

2 Related Work

The goal of a privacy-enhanced DHT lookup is to prevent other peers from learning the interests of a client. This can be achieved by obfuscating the item of the request, which can be achieved via different methods. It is possible to use cryptographic methods or by requesting multiple elements. NISAN [15] tries to privately select an onion router for anonymous communication, adding privacy protection to a Chord [17] DHT. In NISAN, instead of requesting a specific item, the client selects the next hop from the fingertable of another peer. Backes et al. [2] use cryptography to hide the query from intermediate peers in a robust DHT, i.e., a Chord-based DHT with quorums. The lookup on intermediate peers uses oblivious transfer to find closer peers. The final element lookup is done in plain, revealing that the peer is interested in an element or can use again oblivious transfer. Mazmudar et al. [12] propose to improve the method from Backes et al. by using Private Information Retrieval (PIR) for the element lookup. In [13], Mazmudar et al. show another approach to use PIR for DHT requests in IPFS. A hashing approach is proposed by Katsantas et al. [9], proposing to hash the object three times to increase the privacy. Our approaches focus on query obfuscation as well. We employ similar means; however, they are combinable and distinctly centered on Kademlia.

The lookup mechanism of DHTs such as Chord and Kademlia is designed in a way that with each step a peer closer to the element can be contacted. Therefore, the nature of the lookup already reveals information about the item of interest. This can be used for range estimation attacks [20]. This observation leads us to another approach to increase the privacy: obfuscating the identity of the requestor. The simplest method to get plausible deniability for the requestor is to use a recursive lookup. In a recursive lookup each peer forwards the request, therefore, requesting peers could make a query on behalf of another peer. However, with a recursive lookup the requestor loses control of the lookup with the first hop and only learns the final result. Furthermore, a recursive lookup prevents item obfuscation, since the following peers need to be able to determine the failure or completion of the lookup. McLachlan et al. [14] propose Torsk, a system for secure and private selection of onion router. They obfuscate the association of lookup target and initiator by using so-called buddy peers. Via a random walk a buddy is selected, which performs the lookup on behalf of the peer. Additionally, each peer performs cover lookups to mitigate timing attacks. Although, it has been shown that the buddy selection process is vulnerable to a denial of service attack [20]. Next to the proxy lookup from Torsk, it is also possible to use anonymous communication systems like Tor [8] to hide the source of a request.

Wang and Borisov [19] present the Octopus DHT. The Octopus lookup consists of multiple anonymous paths, providing cover lookups, impeding range estimation.

Our proposed methods are compatible with source obfuscation.

3 Query Obfuscation

A DHT structures the P2P overlay network. In general, a DHT is a distributed key-value store. Each record consists of a key K and a corresponding value V_K, stored across the network. The distribution of records depends on the area for which a peer is responsible. Commonly, range and position in the network of every peer depends on its peer identifier (PID). The PID can be self-assigned, but must be unique from a pre-determined identifier (ID) space. The peer is responsible for keys closest to its PID. In Kademlia, "closeness" is based on the symmetric and unidirectional XOR distance.

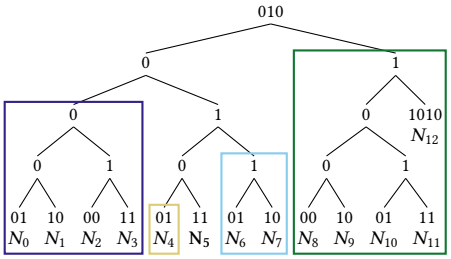


Figure 1: Kademlia tree with $n = 8$, showing buckets of N_5.

For routing in the network, every peer maintains a routing table of up to n k-buckets, where n is the length of ID space. The k is a redundancy factor and determines the maximum size of the k-buckets. Each bucket entry consists of a key-value pair, a peer's PID as the key and contact information as the value. The bucket of a peer is determined by the XOR distance of remote PID and reference PID. The bucket is chosen based on the shortest prefix length of the XOR distance, sorting the peers in a binary prefix tree structure. Fig. 1 shows an example with an $8\,b$ ID space.

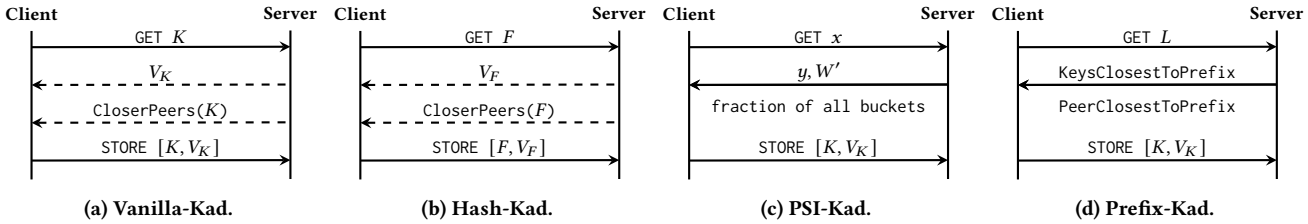
The data is identified by a content identifier (CID). The CID has the same identifier space and length as the PID. This is commonly achieved by using a cryptographic hash function, which provides a self-certifying name. A cryptographic hash function allows fast computation of a fingerprint of its input, which is unfeasible to reverse. Both PID and CID are usually identified using the digest of the same hash function, ensuring that both PID and CID identifiers lie in the same key space. Through a hashing function, the fingerprint allows to verify the integrity and correctness of the file.

Fig. 2a shows the sequence for lookup (`GET`) and store operation (`STORE`) of a key K with the value V_K. To find content providers, a peer sends a lookup request to the peers closest to the CID based on its routing table. If a queried peer stores any provider record associated with the queried CID, it returns the value V_K associated with the key, the PIDs of the content providers. Otherwise, it returns the k closest peers to the queried CID (`CloserPeers`(K)). This repeats until either a content provider is found or no new peers are discovered. To publish data in the DHT, a content provider looks up the k peers closest to the CID and then sends a `STORE` with the key-value pair (K,V_K) of CID and its own PID.

While the lookup process of Kademlia is efficient, it reveals the interest and identity of requesters to all peers involved in the lookup process. In the following, we refer to this approach as Vanilla-Kad. We further use *server* to indicate a peer, which provides DHT entries and *client* to indicate a peer requesting a DHT entry. Fig. 2 shows changes to the message sequences, due to our approaches.

3.1 Threat Model

The goal of our proposed privacy mechanisms is to protect the content of a request. For the privacy mechanisms, we assume a single or partial adversary controlling at most a small fraction of



(a) Vanilla-Kad. **(b) Hash-Kad.** **(c) PSI-Kad.** **(d) Prefix-Kad.**

Figure 2: Kademlia message sequences for lookup and store. The dashed messages indicate a choice for one message.

the nodes, evenly split across the DHT ID space. The adversary possesses information gained through protocol-compliant behavior, observing or replaying queries that the servers received, and is not able to observe all communication. This includes that the adversary may gain information by hiding records and manipulating the list of returned peers, eventually redirecting the sender to other colluding peers. Additionally, the adversary may collect information from other external sources. Finally, the adversary is computationally bounded and cannot break the underlying cryptographic primitives.

For ease of understanding, we decided to explain our privacy mechanism in Section 3.3 in the context of a semi-honest adversary. Please note, however, that it is possible to use another PSI protocol, e.g., [6], which is able to detect malicious behavior. Moreover, we mostly focus on privacy aspects and do not consider underlying Kademlia security issues. In general, our methods provide security similar to Vanilla-Kad, while remaining compatible to proposed security improvements [3].

3.2 Double Hashing (Hash-Kad)

To obfuscate the query key K, the client requests a fingerprint F of the key, instead of the key. F is obtained by using a cryptographic hashing function H:

$$F = H(K). \tag{1}$$

We call this method *Hash-Kad*. The method can also be considered as *double hashing*, since in Vanilla-Kad, CIDs are commonly the hash of the file. An overview of Hash-Kad's sent and received messages can be seen in Fig. 2b.

Hash-Kad uses the CID's fingerprint as key and encrypted PIDs as values. Consequently, a lookup request sends `GET` F. The return values are the content providers' encrypted PIDs, using the CID as encryption key, signed by the content provider, denoted as

$$V_F = SIGN_{CP}(ENC_K(V_K)). \tag{2}$$

Similar to Vanilla-Kad, in case the key is not stored, the server returns the k closest peers to the fingerprint. The store operation also sends only the fingerprint and the signed and encrypted value V_F. This method effectively conceals the PID of the content providers from peers who do not have knowledge of the CID. Using a fingerprint of the CID as identifier still allows a curious server to replay a request. However, the curious server will be unable to decrypt the PIDs, as it does not know the CID. Moreover, for fetching content from a content provider the CID is used, not the fingerprint.

The encryption of the PID with the CID serves as proof, that a content provider did not publish a random value. This prevents a malicious server from baiting clients by simply publishing a hash without knowledge of the CID. The signing of the record prevents malicious servers from serving an arbitrary forged record corresponding to the requested CID. The client can computationally verify validity of the record and that the creator knows the CID.

This approach provides some obfuscation through the obstruction of servers from associating a client's request with the specific requested content. As a result, servers cannot deduce which client is accessing which piece of content. This level of protection remains robust as long as servers remain unaware of the CID requested by clients. Consequently, the confidentiality of a request predominantly relies on the undisclosed nature of the requested CID. Curious servers can learn CIDs from other sources, i.e., website advertising the content, or crawling the Internet. The CIDs only need to be hashed to get fingerprints, which can be checked against stored entries or clients' requests, revealing interest of clients.

3.3 Private Set Intersection (PSI-Kad)

The query for a particular item can be interpreted as a set intersection problem. In this case, the client's interest in particular keys is one set and the keys of the stored records from the server are the other set. The client is interested in the intersection of the sets. A method to reveal this intersection privately without revealing information about the sets' elements is Private Set Intersection (PSI). Hash-Kad's hashing of the elements and comparing the resulting hashes can be considered as a naïve but insecure variant of PSI [16]. However, the result of the intersection would only eliminate intermediate servers, as the client only learns possession or absence of the key and not the associated value. To also obfuscate the last hop, it is possible to utilize a specific PSI approach, which also allows data transfer. For our approach, denoted as *PSI-Kad*, we adapt an ECDH-PSI [1] protocol to allow data transfer.

An overview of PSI-Kad's sent and received message can be seen in Fig. 2c. The goal of the request is to receive the value of the DHT record with the key c_i, V_{c_i}. The key is changed based on a cyclic group and a Hash function H, which maps the ID to the cyclic group. The client calculates:

$$x_i = H(c_i)^{r_C}, \tag{3}$$

with r_C being a random number chosen by the client. The client sends a `GET` x. Independent of the server's DHT, the server returns:

$$y_i = x_i^{r_S} = (H(c_i)^{r_C})^{r_S}, \tag{4}$$

with r_S being a random number chosen by the server. To complete the set intersection the client needs knowledge about the records

stored by the server. For PSI without data transfer, the server transforms each key (s_j) of the stored records using r_S with:

$$w_j = H(s_j)^{r_S} \tag{5}$$

and sends the set W to the client. To allow data transfer, the server encrypts the value of the record V_{s_j} with w_j:

$$w'_j = ENC_{w_j}(V_{s_j}), \tag{6}$$

and returns the set of all encrypted w'_j, W'. With y_i and W' the client can complete the set intersection. The transformation of the key can be reversed with r_C:

$$z_i = ((H(c_i)^{r_C})^{r_S})^{\frac{1}{r_C}} = H(c_i)^{r_S}. \tag{7}$$

If the resulting z_i can be used to decrypt a w'_j the client learns V_{c_i}.

If there exist no w'_j that can be decrypted with z_i, the client needs to find another peer. The transformed key x_i, which the client sends, contains no usable information for the server. Therefore, the server cannot suggest closer peers based on the requested key. Furthermore, the server should not know if it is even necessary to send closer peers, i.e., if the server is only an intermediate peer. For PSI-Kad, we need to provide clients with possibly closer peers. We propose two methods to get closer peers: return always parts of the routing table, return peers based on the client.

The client is guaranteed to receive a closer peer, if we provide the whole routing table. In Kademlia, keys are stored at the absolute closest peers according to the XOR distance of PID and key. Keys are searched by querying the closest peers according to a peer's routing table. Therefore, the target of the client should be from the client's perspective in the same bucket as the server. The server could give each bucket a weight based on the distance from itself; the closer the peers in a bucket, the more peers are included in the return value. As an example, we assume the routing table of peer N_5 as shown in Fig. 1. N_5 could return N_4, N_6, N_7, N_1, N_2, and N_{10}. Most of the peers are close to the requested peer and only a few peers far away. Still, peers are selected randomly from all buckets, providing any client with closer peers.

The other method uses the information provided by a client. Since the client already chooses the server for a specific reason, it should be sufficient if the server sends all buckets from its own bucket up to the bucket in which the client is included. As an example, consider again the routing table of peer N_5 as shown in Fig. 1. When N_5 sends a request to N_8, then a closer peer should be known by N_8–N_{12}. For N_8, N_0–N_7 are all in the same bucket. Since N_8 was contacted, it is unlikely that these peers are closer. Similar to the other method, the server could also provide only parts of the buckets based on the distance. While this approach has the potential to produce less overhead, it prevents the usage of proxies for queries, which could hide the identity of the client.

Both approaches reduce the redundancy of the lookup, returning only parts of the closest buckets. Reducing the redundancy makes it easier to manipulate the lookup path and increases the susceptibility to churn. The store operation of PSI-Kad is the same as Vanilla-Kad.

Since the approach is based on a semi-honest PSI protocol, it can only protect against semi-honest servers. Due to the additional transformation of the key, the server does not learn any information about the key and is even unable to replay the query. The only knowledge a server gains from the request is the number of keys the client is interested in and that its PID is closer to the key than the client's PID. A disadvantage of this approach is that the server needs to send potentially large amounts of data for the client to be able to complete the PSI. It should be noted that other PSI protocols are also possible (e.g., [6]). Another PSI protocol can increase the protection against malicious behavior, however, the PSI protocol should also enable private transfer of the value.

Algorithm 1: `ClosestToPrefix`

```
Data: prefix, k, all_keys
Result: selected_keys
selected_keys ← ∅
l ← len(prefix)
for counter = 0 to 2^l − 1 do
    if len(selected_keys) ≥ k then
        break
    leaf ← prefix ⊕ bin(counter, l) // bin(x,l) returns binary representation
      of x in l b
    matching_keys ← FindMatchingKeys(leaf, all_keys)      // returns all keys
      matching leaf
    if len(matching_keys) ≤ k − len(selected_keys) then
        selected_keys ← selected_keys ∪ matching_keys
    else
        random_selection ←
          SelectNRandom(matching_keys, k − len(selected_keys))
        selected_keys ← selected_keys ∪ random_selection
return selected_keys
```

3.4 Prefix Fetching (Prefix-Kad)

Without any cryptographic measures, PSI-Kad basically requests all records from the server and checks if it stores the key of interest. Note that requesting the entire list of all records already conceals the item of interest. The only knowledge the server gains in this scenario is the interest of the client in something. Since this approach has a potentially large overhead, we can trade off the overhead by compromising some degree of privacy. Therefore, we propose *Prefix-Kad,* which sends as request a prefix L, `GET` L. An overview of sent and received messages using Prefix-Kad can be seen in Fig. 2d. The prefix reveals some information about the key, however, the number of keys that need to be transferred can be reduced. The main goal of prefix fetching is to retain plausible deniability by concealing the request in an anonymity set. The answer of a request are multiple items, the keys closest to the prefix. As in PSI-Kad, a server should be unable to determine success of a request, making it necessary to also send closer peers.

The closer peers are determined based on L. Computing the XOR distance between binary bitstring of different lengths results in an undefined behavior. Hence, it becomes non-trivial to discern the k closest keys to a prefix within the ID space. Nevertheless, we can identify keys matching a prefix, and compute a XOR distance between a prefix and the prefix of the key of same size. To address this challenge, we introduce Algorithm 1 that efficiently identifies the k closest keys to a given key prefix. If less than k peers have a match with L, the server also considers peers with a shorter prefix length l, e.g., $l-1$, $l-2$, …, $l-l$, until at least k candidate peers are found, or all known peers are candidates. If the server has more than k candidate peers, the server chooses at random from the candidates with the shortest common prefix.

Prefix fetching can also be used in combination with Hash-Kad or PSI-Kad. The combination can mitigate some disadvantages of

the approach, e.g., enriching prefix fetching with pseudonymity. In case of Hash-Kad, the client reveals too much information and the amount of information leaked is reduced. The leaked information is reduced by requesting the prefix of the hash instead of the complete hash. For the server to identify the item, it needs the original CIDs of items to determine a range of interested items.

3.5 Privacy Discussion

In Vanilla-Kad, servers can infer a client's interest in a CID, even without providing the record by replaying requests to retrieve content. Our proposed methods increase the difficulty of linking requests to content but still reveal that a client showed interest in something. In the following, we discuss privacy aspects, i.e., prefix length, reader and provider privacy, and potential improvements.

3.5.1 Prefix length. The theoretical anonymity set of an obfuscated request is $\approx 2^n$, with n being the length of the ID space. Each revealed bit halves the theoretical anonymity set. In practice, the anonymity set is much smaller, since not all keys are used. The prefix length can be chosen, based on the known number of keys in the network, to achieve a fixed size anonymity set. The goal is that each prefix request yields an approximate count of N records, matching the prefix, where N represents a system parameter. Clients could select a specific N depending on their privacy level needs.

The prefix length l in bits can be computationally derived from N and the total number of CIDs disseminated across the DHT: $l \leftarrow log_2(\frac{\#CIDs}{N})$. Given the inherent challenges in precisely gauging the total number of CIDs published in a decentralized network, an approximation to dynamically and continuously determine l is needed: Peers need to keep track of the average number of records returned for each request made with the current prefix length. Let a symbolize the average match count for a prefix length l. At any given moment, if $a > 2N$, l is incremented, and conversely, if $a < \frac{N}{2}$, l is decremented. For continuity, the calculated value of l should be persisted beyond a peer's lifetime.

Due to Kademlia's lookup procedure of contacting the closest peers, the anonymity set is further reduced. A server can estimate the range of the interest based on the XOR distance of its own PID and the client's PID. The range can be further reduced with each contacted peer, although the intermediate peer would need to share query information. This is similar to the range estimation attack on the Chord-based NISAN lookup [15] described in [20].

3.5.2 Reader Privacy. Since a request can match multiple CIDs, prefix fetching generally increases the anonymity set of a query. The combination of Prefix-Kad and PSI-Kad, however, reveals more information compared to each separate approach. Prefix-Kad allows the client to satisfy multiple interests with a single request. In contrast, PSI-Kad requires the client to send a request for each item of interest. Compared to Prefix-Kad, the combination reveals the number of items a client is interested in, while in PSI-Kad the anonymity set is the whole range of CIDs. Yet, the combination still improves the feasibility of PSI-Kad and prevents curious peers from network crawling.

3.5.3 Provider Privacy. Our proposed methods are not designed to improve the privacy of content providers. Yet, Hash-Kad offers slight improvements through the modified publishing process. By publishing a hashed CID and encrypted provider record to the DHT, only those with knowledge about the CID can retrieve the content. This allows content providers to obscure specific CIDs from servers unaware of the corresponding CIDs. While PSI-Kad could complement Hash-Kad's publishing method, their combination offers no additional privacy benefits.

Prefix-Kad can simplify large-scale identification of content providers: Prefix fetching enables retrieval of multiple records per request, potentially exposing all providers and their content to a curious client. In contrast, Hash-Kad's publishing process limits crawling by requiring extensive knowledge of CIDs or brute-forcing CIDs. This advantage remains even with Prefix-Kad, as it does not alter publishing. PSI-Kad offers no privacy improvement for providers over Vanilla-Kad.

3.5.4 Privacy Improvements. Our proposed methods obfuscate the target of a request. The server still learns the existence and origin of a request. This poses a risk for possible attacks, e.g., range estimation attack or frequency analysis, which can be mitigated using cover traffic or proxy requests. For cover traffic, peers periodically search random keys via prefix fetching. It is not necessary to complete the lookup as long as it follows a consistent path. Alternatively, peers could send follow-up requests of completed lookups, misleading range estimations. Proxy requests involve other peers completing lookups on behalf of the clients, e.g., via an anonymous and secure channel.

PSI-Kad can mitigate range estimation attacks by using a different random value for each request, making them indistinguishable. However, when combined with Prefix-Kad, the prefix could still be used to link requests.

4 Double Hashing with Prefix Fetching in IPFS

For a better understanding, we put our protocols in the context of kad-dht. The kad-dht is used among others in the IPFS network for peer and data lookup. IPFS is an actively developed P2P overlay network, allowing content-addressed exchange of data with a large user-base, making it a prime example for adaption.

In IPFS, a CID is a multihash (MH) with additional metadata, such as the content's codec, ensuring content authenticity and system adaptability. The MH is the digest of a hashing function (typically SHA-256) and an identifier for the hashing function. The PID in IPFS is the encoded MH of a peer's public key.

IPFS utilizes a 256 b ID space for content routing. DHT content is identified using a DHT identifier (D-ID), the SHA-256 digest of the CID. Since the CID intrinsically comprises the content's hash, rehashed for its D-ID, content is effectively addressed by a secondary hash. At the time of writing, the IPFS DHT request contains the CID, not its direct D-ID. This design creates complications, especially for prefix requests as defined in Section 3.4. Because the request needs the CID, the preimage of the derived D-ID, it makes prefix requests for DHT keys impractical. Consequently, the operational method of the IPFS DHT must be modified.

In the following, we outline the protocol specification and implementation challenges for double hashing (Hash-Kad) with prefix fetching (Prefix-Kad) for IPFS. Hash-Kad combined with Prefix-Kad can be considered as the basis for all our proposed protocols. All protocols can be derived from this specification by setting specific

parameters or adjust some calculations, e.g., by setting the prefix always to the maximum, it is pure Hash-Kad. A detailed protocol specification can be found in IPIP-373.[2]

4.1 Content Publication

Content providers compute the fingerprint F with SHA-256 from the salted MH of the CID. Similarly, an `EncKey` and `ServerKey` are derived from the MH with another distinct public salt for domain separation. The content provider's PID is encrypted with `EncKey`, generating `EncPID`. The `EncPID` and record expiration time (`TS`) are signed by the content provider, creating a `Signature`.

As in Vanilla-Kad, the content provider initiates a Kademlia lookup for the closest peers to F. The closest peers receive a publish request encompassing `[F, EncPID, TS, Signature, ServerKey]`. Data is sent via a secure and authenticated connection, i.e., servers can verify the identity of content providers. The DHT record contains `[F, ServerKey, PID, EncPID, TS, Signature]`. Servers validate the `Signature`, guaranteeing authenticity of requests, discarding invalid requests. Servers remain oblivious to the content of `EncPID`, given they ignore the MH, from which `EncKey` is derived.

4.2 Content Retrieval

For content retrieval, clients derive F, `EncKey`, and `ServerKey` from MH of the CID, to lookup F. To combine Hash-Kad with Prefix-Kad, clients select and lookup a prefix of F, as described in Section 3.4.

The server's response contains closer peers and matching entries, encrypted using `ServerKey`. The record consists of `EncPID`, `TS`, `Signature`, and if available, contact address of content providers.

The client filters out entries which do not match F. For matching records, the clients undergo a three-step validation: (1) decrypt `EncPID` with `EncKey`, (2) validate `Signature` against decrypted `PID`, and (3) assesses validity of `TS`. Afterwards, the client either tries to contact the content provider or needs to discover the contact information of the decrypted PID. IPFS uses the same DHT for peer and content routing, hence it requires another DHT lookup. Content is retrieved through the CID via the Bitswap protocol [7].

5 Analysis

In the following, we analyze the expected communication, computation and storage overhead. The methods have different overheads, load distributions, and privacy benefits. Our proposed methods do not modify the original message sequence. Therefore, the methods do not require additional messages or additional RTTs, however, additional data needs to be transferred, and the messages need additional processing. To put the overhead into perspective, we conducted a measurement of the public IPFS network. We also provide a summary and overview of the methods showing the differentiated feasibility of each method.

5.1 Measurement Setup

The purpose of our network measurement is to gain insights about the DHT usage in a productive network. Specifically, we are interested in an estimate for the number of records stored by a server, the churn of the records, and the number of requests a server experiences. This allows us to analyze load and overhead of the proposed methods. The stored records also allow us to quantify the risk of range estimation attacks, prefix lengths, and possible privacy gains.

Table 1: Overview of the measurement results

Measurement Period	P1	P2
Communicated Peers	36 462	71 573
Content Provider	16 419	25 339
PUT-VALUE	1 019	3 448
GET-VALUE	47	576
ADD-PROVIDERS	2 273 855	14 414 476
GET-PROVIDERS	17 464	165 218
FIND-NODE	391 472	2 830 486
Total Messages	2 683 857	17 414 204
Unique Put keys	112	257
Unique Provider keys	955 754	2 656 265
Unique Get keys	4 290	27 080

For the measurement, we chose the IPFS network as an example network which uses a Kademlia-based DHT, due to the active and comparatively large user base. We deployed a kubo[3] DHT-server with the default configuration and go-libp2p-kad-dht[4] with additional logging. Kubo is a Go implementation of IPFS and at the time of writing the most widely deployed implementation. It should be noted that IPFS nodes behind network address translation act as DHT clients only, meaning they will not be queried to route requests. By default, all nodes with a public IP address behave as DHT servers, actively participating in the routing process.

The deployed node measured the DHT activity by logging incoming DHT messages and changes in the routing table. We conducted a passive measurement, meaning the node adds no additional traffic beyond the default behavior. The measurement node was deployed on a virtual machine (VM) in Central Europe, specifically in Finland. The VM had 2 ARM vCPU, 4 GB RAM, Ubuntu 22.04.3 LTS as the OS with Kernel version 5.15.0-89, a public IPv4, and IPv6 address. The client was compiled with go version go1.21.4 linux/arm64. We measured in two different time spans from 2023-11-21 to 2023-11-28, one spanning 24 h (P1) and the other spanning 144 h (P2). The measurements are conducted on the same node with the same PID, however, the node had a clean restart in between. A summary of the measurement results can be seen in Table 1.

5.2 Overhead

The libp2p-kad-dht has different messages: for querying a value (`GET`), for storing a record (`PUT/ADD`), finding peers (`FIND`), or for reachability checks (`PING`). In Vanilla-Kad, a client sends a `GET` request for a key lookup, servers respond to these lookup queries with k closer peers or the value(s) associated with the key.

From a client's perspective this does not change for all methods. Lookup requests stay a single `GET` request, however, with a modified key. The modifications at most increase the request by a few bytes.

For servers, the overhead is bigger. Except for Hash-Kad, servers are expected to return multiple records. Additionally, servers should return closer peers. It should be noted that although bandwidth

[2] https://github.com/ipfs/specs/pull/373

[3] https://github.com/ipfs/kubo (kubo/0.25.0-dev/)

[4] https://github.com/libp2p/go-libp2p-kad-dht (v0.25.0)

overhead increases, active processing overhead remains low. Most of the required additional processing can be done asynchronously.

To put the communication overhead into perspective, we use a back-of-the-envelope calculation derived from our measurements: The server stores around $956k$ (P1) records, which have mostly a single content provider. Given a PID size of 38 B and CID size of 32 B, we get $\approx 70\,B$ per record. Therefore, the maximum overhead (the server sends all records) would amount to $70\,B \cdot 956k \approx 63.82\,MiB$. The prefix reduces this overhead, depending on the desired privacy. Assuming, for example, at most 64 records yields $70\,B \cdot 64 \approx 4.38\,KiB$. With additional encryption (AES-256, Galois-Counter Mode, +16 B) and record signatures (Ed25519, +64 B), size of a record increases to 150 B, resulting in a message overhead of $150\,B \cdot 64 = 9.375\,KiB$.

In PSI-Kad and Prefix-Kad, servers need to always send possibly closer peers, which adds an overhead to the record overhead. This overhead consists of the k-closest peers or a fraction of the routing table. Our measurements showed an almost constant size of the routing table between 210-220 peers. Although the size remained constant, the routing table experienced quite a high churn rate. If we again assume the worst case of sending the whole routing table, it increases a single message by $215 \cdot 38\,B \approx 7.98\,KiB$. It should be noted that intermediate servers also send $k \cdot 38\,B$. Considering a $k = 20$ as used in IPFS, the normal message size is $20 \cdot 38\,B \approx 760\,B$.

In summary, query answer size increases depending on the prefix by a factor of $\approx 10^1$ to 10^5 for Prefix-Kad. PSI-Kad yields the worst case overhead of Prefix-Kad. For Hash-Kad, record size is doubled.

So far we mainly analyzed the overhead of the lookup process. In case of storage, the message size is mostly untouched; except for Hash-Kad, which changes the information a server stores and receives. In terms of computation and memory, Hash-Kad and PSI-Kad produce additional overhead for the server. Records in Hash-Kad are bigger due to the encryption and additional required data, requiring the server to store more data. This makes it more costly for content providers to announce stored content and increases the used memory of servers. In PSI-Kad, the cost for content providers stays the same. However, servers need to transform and store additional data, making a publish process more costly. Although, this transformation is a one-time cost for new values, republished values should be unaffected. The memory overhead is comparable to a DHT duplicate with other adjusted key-value pairs.

5.3 Anonymity Set

The prefix length determines the size of the possible anonymity set of a request, influencing the number of matching values. Vanilla-Kad's lookup mechanism already reduces the anonymity set of a request, e.g., the range estimation attack considers that a peer requests peers close to a key. Prefix length is also important for efficiency, reducing the number of keys returned by a server. To estimate the severity of a range estimation attack and determine possible prefix lengths, we investigate the common prefix length (CPL).

In Fig. 3, we show CPL of the measurement node's PID with values of the record keys, keys of `GET` request, and PID of all peers which were at least once in the routing table. From the peers in the routing table, we can see roughly a halving of the number of peers for each CPL until a CPL of 9. The maximum CPL with the peers in the routing table is in both measurement periods 16 and less than 50 peers have a CPL of 10 or more. Consequently, most of the record keys have a CPL between 9–16, with quite a few keys with a CPL up to 25 and only a few keys with a CPL bigger than 25. The maximum CPL of the stored keys were 30 (P1) and 32 (P2). Surprisingly, there are quite a few keys which have a CPL of 5 or less and even 35 (P1) and 349 (P2) keys with a CPL of 0. This seems unusual, due to the sufficient number of peers with a PID closer than our measurement node. This can be caused by misconfigured servers or as a result of the *Optimistic Provide*[5] peer allocation. Optimistic Provide is a mechanism to improve the speed of record publishing, it determines based on probabilistic calculations, if a closer peer exists and can therefore publish records prematurely.

The CPL of the requested `GET-PROVIDERS` of Fig. 3 can be used to estimate some practical choices of prefix lengths. Similar to the CPL of the records, we can see that the majority of requested keys have a CPL larger than 9. However, we can see more requests with lower CPL. In contrast to the unusual `ADD-PROVIDERS` which should only be sent to the absolute closest peers, `GET-PROVIDERS` messages are sent to the locally determined closest peer. While our routing table indicates that there are closer peers available, the requesting peers might have a more limited view on the network. Comparing the `GET-PROVIDERS` keys with the stored keys, we can determine a minimal prefix length required to retrieve less than $N = 64$ values. This minimal prefix length naturally varies based on the distance to our measurement node. For a CPL of 9 and above, where most of the records are expected to be, we have an average prefix length of 24. For CPLs between 5 and 8, the minimal prefix length is between 10 and 20. In case of even lower CPLs, prefix lengths between 4 and 9 reduce the candidates to under 64 records.

The CPL analysis shows that the range estimation attack based on the PID still provides a comparatively large theoretical anonymity set. Considering the neighbors, the range estimation reveals only $\approx 12\,b$. Fig. 3 also shows that a client can choose different prefix lengths based on the distance of the key to requested server. In our case, there are 35/349 entries which have a CPL of 0 with the measurement node's PID. Therefore, a prefix length of 1 b for any of these entries results already in a low number of records. In contrast, a prefix length of the first 10 b of the measurement node's PID would match more than $500k$ records. Adjustments to the chosen prefix length based on return values as proposed in Section 3.5.1, should consider the distance of a peer to the requested key, before changing the prefix length. Based on our measurements, we can conclude that a prefix length of $\approx 24\,b$ provides less than 64 records at the peers which are theoretically closest to the key. A prefix length of 24 b provides a theoretical anonymity set of $2^{232} \approx 6.9 \cdot 10^{69}$. However, the measurement node should be the closest peer, which means the size of the real anonymity set is ≤ 64.

5.4 Comparison

We presented three different approaches to obfuscate the item of a request, protecting requests with their own trade-offs.

Hash-Kad has a low overhead, adds authenticity to the DHT records and provides content providers with some protection. However, Hash-Kad provides only a low protection especially when the CID is known and can be easily hashed to check the fingerprint.

[5] https://github.com/libp2p/go-libp2p-kad-dht/pull/783

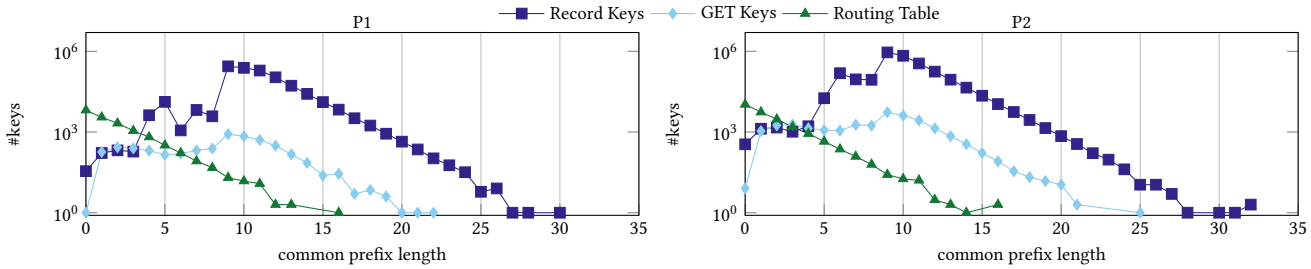


Figure 3: CPL to the PID of the measurement node.

PSI-Kad provides strong protection against single adversaries, which gain no information beyond time and number of items of a request. However, depending on the number of elements in the network, the overhead can be very large in terms of transferred data and computation. Furthermore, colluding peers can gain more information about a request based on the lookup path.

Prefix-Kad can provide similar privacy guarantees as PSI-Kad, depending on the chosen prefix length or the number of requests. However, the similar privacy guarantees incur a similar large overhead in the form of transferred data. A longer prefix reveals some more information about the requested item, reducing the anonymity set, however the overhead can be significantly reduced. A big disadvantage of prefix fetching is the ability to crawl the DHT. All entries of the DHT can be retrieved without protection.

For the best privacy-utility trade-off, we propose to combine Prefix-Kad and either Hash-Kad or PSI-Kad. The prefix fetching can reduce the overhead of PSI-Kad, while removing the problems due to crawling the DHT. This allows request obfuscation with a reasonable overhead and otherwise similar behavior to Vanilla-Kad. Prefix-Kad in combination with Hash-Kad can increase the gained privacy of Hash-Kad and could provide additional protection against crawling the DHT. In Prefix-Hash-Kad, crawling would only allow to retrieve a list of encrypted records. Furthermore, crawling the DHT for encrypted records can provide new opportunities, e.g., caching and the collection of statistics. While Prefix-Hash-Kad has lower protection compared to Prefix-PSI-Kad, we believe the advantages provide a better trade-off. Hence, our protocol specification (see Section 4) also proposes this combination.

6 Conclusion

Request obfuscation is one way to improve the privacy of Kademlia lookups. We proposed three methods to obfuscate the item of interest. The three methods can be combined with each other for additional privacy or performance benefits. Furthermore, our methods could be combined with sender obfuscation methods to further increase the privacy of users.

Acknowledgments

This work was partly supported by the German Federal Ministry of Education and Research (BMBF) and the Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center ScaDS.AI.

References

[1] Nick Angelou, Ayoub Benaissa, Bogdan Cebere, William Clark, Adam James Hall, Michael A Hoeh, Daniel Liu, Pavlos Papadopoulos, Robin Roehm, Robert Sandmann, et al. 2020. Asymmetric Private Set Intersection with Applications to Contact Tracing and Private Vertical Federated Machine Learning. *arXiv preprint arXiv:2011.09350* (2020). https://arxiv.org/pdf/2011.09350.pdf

[2] Michael Backes, Ian Goldberg, Aniket Kate, and Tomas Toft. 2012. Adding query privacy to robust DHTs. In *ASIACCS '12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. 30–31.

[3] Ingmar Baumgart and Sebastian Mies. 2007. S/kademlia: A Practicable Approach Towards Secure Key-Based Routing. In *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems*. 1–8.

[4] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *arxiv* abs/1407.3561 (2014). http://arxiv.org/abs/1407.3561

[5] Bram Cohen. 2003. Incentives build robustness in BitTorrent. In *P2PEcon '03: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*. 68–72.

[6] Emiliano De Cristofaro and Gene Tsudik. 2010. Practical private set intersection protocols with linear complexity. In *FC '10: Proceedings of the 14th International Conference on Financial Cryptography and Data Security*. 143–159.

[7] Alfonso De la Rocha, David Dias, and Yiannis Psaras. 2021. Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin. *San Francisco, CA, USA* (2021), 11.

[8] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security '04: Proceedings of the 13th USENIX Security Symposium*. 303–320.

[9] Thomas Katsantas, Yannis Thomas, Christos Karapapas, and George Xylomenos. 2024. Enhancing IPFS privacy through triple hashing. (2024), 1–6.

[10] Nick Lambert and Benjamin Bollen. 2014. The SAFE Network a New, Decentralised Internet. (2014).

[11] Petar Maymounkov and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *IPTPS '02: Proceedings of the 1st International Workshop on Peer-to-Peer Systems*. 53–65.

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

[13] Miti Mazmudar, Shannon Veitch, and Rasoul Akhavan Mahdavi. 2024. Peer2PIR: Private Queries for IPFS. *arXiv* arXiv:2405.17307 (2024).

[14] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. 2009. Scalable Onion Routing with Torsk. In *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*. 590–599.

[15] Andriy Panchenko, Stefan Richter, and Arne Rache. 2009. NISAN: Network Information Service for Anonymization Networks. In *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*. 141–150.

[16] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on OT Extension. In *USENIX Security '14: Proceedings of the 23rd USENIX Security Symposium*. 797–812.

[17] 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 *SIGCOMM'01: Proceedings of the 2001 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 149–160.

[18] Viktor Trón. 2023. The Book of Swarm. online. https://www.ethswarm.org/The-Book-of-Swarm.pdf v2.0 pre-release 1.3.

[19] Qiyan Wang and Nikita Borisov. 2012. Octopus: A secure and anonymous DHT lookup. In *ICDCS '12: Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems*. 325–334.

[20] Qiyan Wang, Prateek Mittal, and Nikita Borisov. 2010. In Search of an Anonymous and Secure lookup: Attacks on Structured Peer-to-Peer Anonymous Communication Systems. In *CCS '10: Proceedings of the 17th ACM Conference on Computer and Communications Security*. 308–318.