Iris: Dynamic Privacy Preserving Search in Authenticated Chord Peer-to-Peer Networks

Angeliki Aktypi
University of Oxford
angeliki.aktypi@cs.ox.ac.uk

Kasper Rasmussen
University of Oxford
kasper.rasmussen@cs.ox.ac.uk

***Abstract*—In structured peer-to-peer networks, like Chord, users find data by asking a number of intermediate nodes in the network. Each node provides the identity of the closet known node to the address of the data, until eventually the node responsible for the data is reached. This structure means that the intermediate nodes learn the address of the sought after data. Revealing this information to other nodes makes Chord unsuitable for applications that require query privacy so in this paper we present a scheme IRIS to provide query privacy while maintaining compatibility with the existing Chord protocol. This means that anyone using it will be able to execute a privacy preserving query but it does not require other nodes in the network to use it (or even know about it).**

In order to better capture the privacy achieved by the iterative nature of the search we propose a new privacy notion, inspired by k-anonymity. This new notion called (α, δ)*-privacy*, allows us to formulate privacy guarantees against adversaries that collude and take advantage of the total amount of information leaked in all iterations of the search.

We present a security analysis of the proposed algorithm based on the privacy notion we introduce. We also develop a prototype of the algorithm in Matlab and evaluate its performance. Our analysis proves IRIS to be (α, δ)*-private* while introducing a modest performance overhead. Importantly the overhead is tunable and proportional to the required level of privacy, so no privacy means no overhead.

I. Introduction

Structured Peer-to-Peer (P2P) networks, provide a scalable and robust lookup service allowing a requester to identify the provider of sought-after information. The Chord [45] lookup service is one of the first structured P2P networks to be widely deployed, and has been used in systems such as the Cooperative File System (CFS) [11], UsenetDHT [43], OverCite [46] and ConChord [1]. It has also been proposed in the literature as a resource service discovery mechanism in grid computing [34] and WSN [20] and an alternative to the traditional centralised design for DNS [10] and telephony [41] systems. More recently, CFS has been proposed by the Tor project [12] as an efficient key-value lookup system with authenticated updates to allow the retrieval of the introductory points for onion services. Chord also underpins the NKN (New Kind of Network) [18] blockchain network infrastructure focused on decentralising network resources used as the base for many decentralised applications (dapps) including nMobile, D-chat, nShell and nConnect.

In Chord, the participants only have a partial view of the network, and there is no central entity to assist with searches. When a requester needs to resolve a query, it collaborates with other nodes from the network, asking them if they have the target information. This poses a privacy challenge, as all the nodes that participate in the routing will know what information is being searched for. Malicious nodes can exploit this information to punish requesters based on their activity or disrupt their communication. For example, in Tor, allowing directory nodes to know the query's target can allow them compile statistics about which onion services are being accessed [13].

Although privacy was not initially a design objective in Chord, achieving it is clearly desirable in many (most?) situations [44], [48], [42]. In fact, a number of authors already studied the privacy of structured P2P networks. However, most existing works propose anonymity schemes that conceal the requester's identity [15], [27]. This is a good option if the P2P network does not need to provide authentication, but for authenticated connections, and networks with long term identities, they break the authentication of the communicating parties.

In this work, we assume a network with authenticated nodes, and we wish to maintain the authentication while still providing privacy. Authenticated nodes enable a number of benefits and there are several existing works proposing different schemes, e.g., [7], [4], [36], [2]. Existing P2P applications such as CFS [11], NKN [18], the Inter Planetary File System IPFS [5] and the Storj distributed storage platform [19], all assume authentication is in place, with CFS and NKN being built on Chord. Our proposed scheme IRIS, achieves search privacy while maintaining authentication by hiding the content of the requester's queries, rather than their identity. This allows nodes to have long-term identities that can be used to initiate queries in the network, without revealing the content of their search queries.

Since the content of a search query forms the basis of the existing routing procedure in Chord, hiding that information comes with a number of challenges. First of all we have to guarantee correctness (i.e., convergence to the target object)

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240392
www.ndss-symposium.org

for the new routing algorithm, while concealing the target from the intermediate nodes. The intermediate nodes need to know what address the requester is targeting, to determine how to identify the next hop. We replace the target with a different one that gets us closer to the real target without revealing too much information. Getting this right is the crux of IRIS and the details are described in Section VI.

The search algorithm follows an iterative process in which the requester queries a new node at every step, gradually converging on the target. Because of this iterative process, the adoption of privacy notions such as k-anonymity to quantify the information leakage, would result in a different privacy guarantee for each iteration. We need a new privacy guarantee that can provide an overall value with which we can compare strategies and quantify requirements. It needs sufficient granularity to express the level of information leakage achieved at every step of the iterative retrieval process. For this we propose (α, δ)*-privacy*. This notion of privacy will provide the foundation to argue about the privacy level achieved within a search.

Because Chord is already being used by deployed applications, and the peer-to-peer nature of the deployments means that there is no central authority that can mandate a software upgrade, it is critical that our solution can co-exist with regular Chord nodes, i.e., nodes that do not run our IRIS protocol. The literature contains several proposed schemes that necessitate significant modification either to the organization of the nodes [29], [22] or to the data structure [14], and we believe that that lack of compatibility with existing systems is partly to blame for the lack of adoption. We make sure the design of IRIS is backwards-compatible with the existing search algorithm (and node behavior) for Chord networks. IRIS makes use of the low-level Chord algorithms as building blocks to achieve this. The requester has the freedom to decide the level of privacy he wants to achieve, without demanding any deviation from the vanilla Chord algorithms from the queried nodes. In this way, IRIS can be used directly in already deployed applications.

The act of concealing the real target of a search introduces a modest overhead in terms of the number of requests needed to eventually reach the real target. While some overhead is acceptable as the price of privacy, we want to make sure IRIS is usable in practice. We show thorough analysis and simulation that the overhead introduced by IRIS is logarithmic in the number of hops, which makes it acceptable. More importantly, the overhead is proportional to the level of privacy the requester wants to achieve. That level is tunable by the requester and zero privacy means zero overhead, so a performance critical application can pick and choose which searches need (which level of) privacy.

Our contributions can be summarized as follows:

- We propose a new privacy metric, which we call (α, δ)-*privacy*, that allows us to quantify the information leakage in structured P2P networks.
- We design IRIS, a new algorithm that leverages the `lookup` operation inherently built in Chord overlay to allow for query privacy based on the requester's requirements.
- We prove the security of our algorithm with respect to the new privacy metric we introduce.
- We further confirm empirically through simulations of the communication overhead that IRIS introduces and the privacy it achieves for different populations of colluding adversarial nodes.

Paper Structure: We start with a brief background and notation on Chord structured P2P lookup service in Section II. We provide a detailed description of the designing goals and challenges our proposal addresses in Section III, followed by a formal definition of its system and adversary model in Section IV. We then define (α, δ)*-privacy*, the metric we introduce to quantify privacy in networks that follow the Chord lookup service. We continue by describing IRIS, a privacy-preserving algorithm for Chord. We analyze the privacy guarantees that IRIS offers in Section VII and evaluate its performance through simulations in Section VIII. We compare our approach with prior related works that provide privacy guarantees in P2P architectures in Section IX, before closing the paper in Section XI.

II. BACKGROUND

In this section, we provide background on Chord, the communication scheme on which IRIS builds on enhancing the privacy guarantees it provides.

Chord is a structured P2P network that offers a decentralized and scalable search service. It defines how K key-value pairs are stored across N peers and allows the retrieval of the value associated with a given key by locating the peer to which this key is assigned. Both the peers that participate in the network and the keys that are stored get an m-bit identifier I from an address space. The address space contains 2^m discrete identifiers from the set $\{0, 1, ..., 2^m-1\}$ and is often visualized as a ring. The peers and the keys are depicted as anchor points on the ring, sorted in increasing order in a clockwise direction. A cryptographic hash function $h(\cdot)$ is used to calculate and to uniformly distribute the identifiers on the address space. Often—without that being a functional requirement—a peer's identifier is calculated by applying h on its public key or its IP, while a key identifier is produced by hashing the key (the data) or its name descriptor. To distinguish between the identifiers of peers and the identifiers of keys, we refer to them as *nodes* and *objects*, respectively.

Each node stores the value of every object from a range on the address space in a table referred to as the *object table*. Each object is assigned to (stored at) the node that is equal to or follows the object in the address space. We refer to the node that stores the value of an object as the *responsible* node for this object. Due to their uniform distribution the average distance between the network nodes and objects is equal to $\nu = (2^m - 1)/N$ and $\kappa = (2^m - 1)/K$, respectively. Thus, on average every node is responsible for ν/κ objects.

Nodes have a partial view of the network, knowing the communication information of only selected nodes. Every node saves the details of their *predecessor*, namely the node that comes before them along the address space. They also save the details of m nodes that succeed them in the address space in a table referred to as their *routing table*. The j^{th} entry of the routing table of a node N_i has the information of the responsible node for $N_i + 2^{j-1}$, where $1 \leq j \leq m$. The node stored in the routing table's first entry is called the node's *successor*. This structure ensures that nodes can get the communication information of every other node in the network, asking no more than $log_2(N)$ other nodes.

A. Modeling Chord

We model how Chord works, aiming to provide the backbone on which IRIS builds upon and enhances. We describe only protocols that need to be executed to allow specific data to be stored in the network and retrieved by a requester. Thus, we exclude from our model protocols used in network maintenance, *e.g.*, `leave` and `update`. We identify four low-level protocols as described below:

1) `bootstrap`$(N_n) \rightarrow (N_r, RT_r)$: Protocol executed between the requester and an existing member of the network N_n that returns the address of the requester N_r and the requester's initial routing table RT_r.
2) `lookup`$(N_n, O_k) \rightarrow (N_n{}')$: Protocol executed between the requester and N_n. The protocol takes the address of the communication partner node N_n, and the address of the data object O_k, and returns a new node address that is closer to the data object. If $N_n = N_n{}'$ the responsible node for O_k has been found.
3) `fetch`$(N_n, O_k) \rightarrow$ data OR nil: Protocol to retrieve data with address O_k from node N_n.
4) `push`$(N_n, O_k, Data) \rightarrow$ Ack OR Nack: Protocol to upload data with object address O_k to node N_n.

When a node wants to store or to retrieve data from the network, it invokes a number of the aforementioned low-level protocols. We abstract the steps that nodes perform in each case as two high-level algorithms indicated below:

1) `store`$(RT_h, O_k, Data) \rightarrow$ Ack OR Nack: Algorithm to store data in the network. As depicted in Algorithm 1, it takes three arguments, the routing table of the node that holds the data RT_h, the object O_k of the inserted data and the $Data$ itself. It returns a binary status value that indicates success or failure.
2) `retrieve`$(RT_r, O_k) \rightarrow Data$ OR nil: Algorithm to retrieve data from the network. As depicted in Algorithm 2, it takes the routing table of the requester RT_r, and the object of the requested data. The algorithm returns the $Data$ or 'nil' to indicate that no data can be found at this address.

In both algorithms the executing node needs to identify the responsible node for the object that it wants to store or retrieve. Since in Chord there is no centralized entity that can assist with the search, and nodes do not have a global view of the network, the node has to ask other nodes, if they are responsible for the queried object. The node asks first the node from its routing table that most closely precedes the target object. Every queried node checks if the requested object belongs in the address range between its own and its successor identifier. If this is not true, the queried node, similar to how the initiator picked the first node, identifies the next queried node. If the target object is between the queried node and its successor—there is no node that is closer to the target than the queried node—the queried node returns its successor and the recursive execution of the `lookup` protocol is terminated. The initiator then executes with the identified responsible node the `push` or `fetch` protocol and the algorithm terminates.

Algorithm 1 Chord's Store Algorithm

```
function STORE($RT_h, O_k, Data$)
    $N_n \leftarrow$SELECTCLOSESTNODE($RT_h, O_k$)
    repeat
        $N_n{}' = N_n$
        $N_n \leftarrow$LOOKUP($N_n{}', O_k$)
    until $N_n == N_n{}'$
    return PUSH($N_n, O_k, Data$)
end function
```

Algorithm 2 Chord's Retrieve Algorithm

```
function RETRIEVE($RT_r, O_k$)
    $N_n \leftarrow$SELECTCLOSESTNODE($RT_r, O_k$)
    repeat
        $N_n{}' = N_n$
        $N_n \leftarrow$LOOKUP($N_n{}', O_k$)
    until $N_n == N_n{}'$
    return FETCH($N_n, O_k$)
end function
```

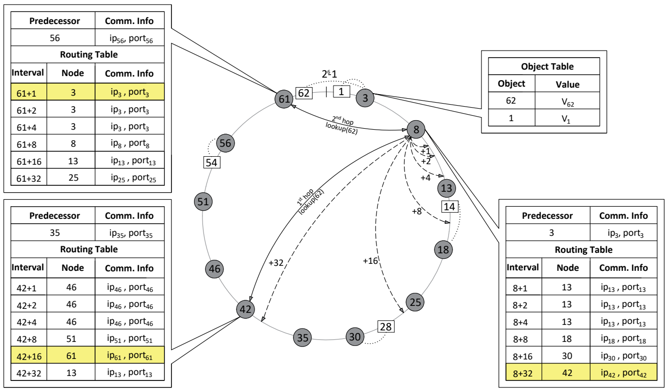
In Figure 1, we can see an example execution of the `retrieve` algorithm in a Chord network. The requester, node 8 searches for the responsible node of object 62. Initially, the requester checks its routing table to identify the node that most closely precedes object 62 and selects node 42 with which it executes the `lookup` protocol first. As object 62 is not between node 42 and its successor, *i.e.*, node 46, node 42 relays the requester to node 61. Next, node 8 executes `lookup` with node 61. For node 61, the queried object 62 is between its own and its successor identifier, *i.e.*, node 3; thus, node 3 is the responsible node for object 62. Node 61 responds to node 8 by specifying node 3. Node 8 executes with node 3 the `fetch` protocol and the `retrieve` algorithm is terminated.

III. PROBLEM STATEMENT AND DESIGN GOALS

This section explains the challenges of developing a private query mechanism in Chord, followed by an outline of the design goals such a mechanism must achieve.

A. Problem Statement

The adoption of Chord P2P networks in real-world applications such as CFS and NKN, underscores the critical need to provide robust privacy guarantees in these networks.



Predecessor		Comm. Info
56		ip_{56}, $port_{56}$
Routing Table		
Interval	Node	Comm. Info
61+1	3	ip_3 , $port_3$
61+2	3	ip_3 , $port_3$
61+4	3	ip_3 , $port_3$
61+8	8	ip_8 , $port_8$
61+16	13	ip_{13} , $port_{13}$
61+32	25	ip_{25} , $port_{25}$

Predecessor		Comm. Info
35		ip_{35}, $port_{35}$
Routing Table		
Interval	Node	Comm. Info
42+1	46	ip_{46} , $port_{46}$
42+2	46	ip_{46} , $port_{46}$
42+4	46	ip_{46} , $port_{46}$
42+8	51	ip_{51} , $port_{51}$
42+16	61	ip_{61} , $port_{61}$
42+32	13	ip_{13} , $port_{13}$

Object Table	
Object	Value
62	V_{62}
1	V_1

Predecessor		Comm. Info
3		ip_3, $port_3$
Routing Table		
Interval	Node	Comm. Info
8+1	13	ip_{13} , $port_{13}$
8+2	13	ip_{13} , $port_{13}$
8+4	13	ip_{13} , $port_{13}$
8+8	18	ip_{18} , $port_{18}$
8+16	30	ip_{30} , $port_{30}$
8+32	42	ip_{42} , $port_{42}$

Fig. 1. *An example of the Chord's* `retrieve` *algorithm.* Node 8 executes `retrieve` to fetch the data associated with object 62. The participating nodes in the network are depicted as grey circles and the registered objects as white squares.

When a node is asked for the location of a target, as part of the search process, the target is an address that does not by itself reveal much information. It is essentially a hash of the content. However, the node is free to request that same target himself, and obtain the corresponding data. This allows any node in the network to monitor and track the data that is being searched for by others. Because each node in an authenticated Chord network has a long-term identity, it is possible to build a profile of each identity by tracking network activity, and reveal additional information over time. This information, when exploited by actors such as surveillance agencies, advertisement companies, or states that apply censorship, can have severe implications.

As described in Section II, the Chord `lookup` protocol requires that the requester reveals the target object to every queried node. The nodes decide where to forward a message based on the target address. In every hop, the distance to this address gets smaller, guaranteeing convergence. A trivial solution that replaces the target with a random identifier cannot guarantee convergence (in a reasonable amount of time) and thus, cannot be applied. Picking a fixed address calculated based on a predefined offset may not reveal the target directly, but it still allows for easy discovery if the offset is known or can be guessed.

Even assuming we can hide the target address in the request, some information can be obtained by looking at the relative position (address) of the requester. The Chord search algorithm dictates that requester selects the node from its routing table that most closely precedes the target. Knowing the address of the requester, and the structure of the nodes in a normal routing table, a node can narrow down where in the address space the target object is likely to be.

B. Design Goals

In the design of a privacy preserving lookup algorithm for IRIS, we consider the following objectives:

1) *Correctness*: convergence to the node responsible for the target must be guaranteed in a reasonable amount of time (hops). Ideally the trade-off between the level of privacy and the convergence speed should be determined by the requester, on a per-object basis.
2) *Query privacy*: given an authenticated requester, the inference that any malicious queried node makes regarding the target of the query should not violate the level of (α, δ)*-privacy* chosen by the requester for that query.
3) *Interoperability*: hiding the target object should leverage already deployed infrastructure without any change in the network organization or in the communication of the queried nodes—IRIS should work, and be secure, even if the requester is the only node using it.

Finally there is the question of whether to conceal the target address from the final node in the search process. The one that is responsible for the target. We have chosen not to incorporate that into IRIS, if such a property is required one can use a number of existing options to accomplish that, e.g., Private Information Retrieval, or a more naive solution where the nodes sends all the objects it controls. We consider this to be an orthogonal problem to the one of providing privacy from the nodes along the search path, and we will not address that further in this paper.

IV. SYSTEM AND ADVERSARY MODEL

We assume a set of N nodes and K named data objects, organized in an authenticated Chord peer-to-peer network as described in Section II. Nodes can communicate directly as long as they have each other's communication details (IP address, etc.). Communication is done on top of an authenticated channel, e.g., [31], [7], [2], and as a consequence nodes cannot lie about their long term identity or network address. The requester's goal is to be able to search for arbitrary data objects without revealing the nature of the data to any intermediary nodes as part of the search process.

We consider an internal attacker who participates in the routing process by controlling a fraction f of the nodes of the network. The attacker can act through all the nodes under his control by initiating requests or responding to incoming connections, but cannot identify or listen to connections among the remaining honest nodes. The attacker is active, deviating from the Chord algorithm at will, *e.g.*, redirecting the requester to another malicious node, claiming responsibility for an object or initiating requests to enumerate the registered nodes and objects. However, the attacker cannot break the underlying authentication scheme used in the routing algorithm; thus cannot lie about the address they control. The adversary knows IRIS and to make the adversary as powerful as possible we give him full knowledge of the α and δ parameters the requester chooses. This would not normally be known to an attacker but we choose to provide them to the attacker in our model, to account for the possibility that these parameters could be guessable in a practical scenario. They are chosen by the user after all, so maybe some common choices (or software defaults) emerges.

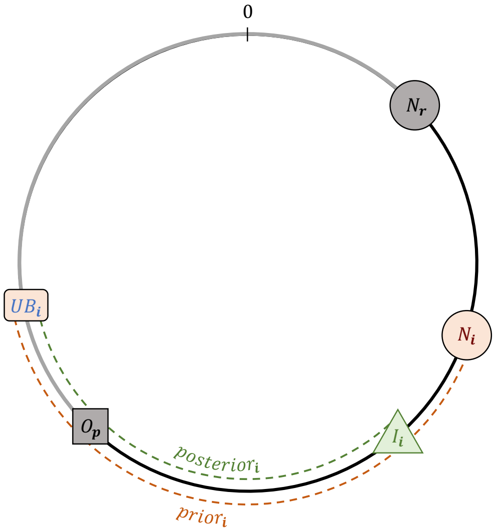


Fig. 2. *The privacy metric.* The orange dashed line indicates the $prior_i$ range of N_i against N_r. The green dashed line shows the $posterior_i$ range of N_i after knowing I_i. Both ranges are computed based on UB_i, *i.e.*, the upper bound of node N_i's estimate regarding the range in which belongs the actual target of node N_r.

The attacker's goal is to discover the target object of a query. Specifically the attacker must know the target object with a probability higher than that allowed by the (α, δ)-*privacy* notion described in Section V, for parameters α and δ chosen by the requester.

V. Alpha-Delta Privacy

The development of a privacy preserving algorithm necessitates the need for a way to measure the privacy guarantees it provides. Due to the iterative process of the Chord `retrieve` algorithm, during which different nodes with different distances to the target are queried every time, privacy notions such as k-anonymity are not suitable as they only allow us to argue about the privacy achieved at every step of the retrieval process and not the retrieval algorithm as a whole. To overcome that, before designing our privacy preserving retrieval algorithm, we introduce a new *privacy notion* that offers the granularity to argue about the privacy achieved at the level of a completed query, *i.e.*, that simultaneously applies to every step that a search incorporates.

To solve this problem we define (α, δ)-*privacy* to measure the privacy level of an Iris-request. This notion is parameterized by two values α and δ which are chosen freely by the requester. They can be different for each new search the requester performs, and could in theory even be changed between iterations of a single run of Iris. Despite this, our adversary model requires that the attacker knows both of these values. All our results are presented with this in mind and can thus be considered the worst-case privacy for the requester.

TABLE I
List of symbols and notation used in this paper

2^m	size of the address space
N	number of participating nodes in the network
K	number of registered objects in the network
N_r	identifier of the requester
RT_r	routing table of the requester
O_p	identifier of the target object
N_t	identifier of the node responsible for O_p
α, δ	privacy parameters explained in Section V
S	starting address of the search $S = O_p - \delta$
N_i	identifier of the node being queried
N_{i+1}	identifier of the next node to be queried
R_i	reference point selected against node N_i: $R_i \in_R [N_i, O_p)$
I_i	identifier for which N_i is queried: $I_i = R_i + (N_i - R_i) \cdot \alpha$
d_i	distance between N_i and O_p
UB_i	upper bound of the target range that node N_i can estimate
$prior_i$	target range N_i can estimate by knowing δ
$posterior_i$	target range N_i can estimate by knowing δ and I_i
f	fraction of colluding adversaries in the network

The parameter δ serves a similar role to k in k-anonymity as it describes the size of the initial anonymity-set in which the real target object must reside. α describes how quickly we progress towards the target object, and thus the decay in privacy per iteration. In order to understand how this works it is necessary to introduce a few details about the search process.

In each iteration of Iris a potential attacker is queried for the address of a target node. We explain this process in detail in Section VI but for now it is sufficient to know that this query reduces the size of the address range where the actual target can be by a certain amount. This gives rise to two address ranges, a $prior_i$ range that an attacker could deduce from knowledge of δ and previous search iterations performed with colluding attackers, and a $posterior_i$ range that incorporates the knowledge gained from the ongoing request. These two ranges are visualized in Figure 2, note that $posterior_i$ is always smaller than $prior_i$. With this we can describe α as the minimum allowable ratio between the posterior and prior knowledge of a requester in any iteration.

$$\alpha \leq \min_i \left(\frac{posterior_i}{prior_i} \right) \forall i$$

An equivalent way to think about α is as (one minus) the maximum allowable gain in knowledge by any intermediate node. We can now state the definition of (α, δ)-*privacy*.

Definition 1 ((α, δ)-*privacy*)**.** *A search algorithm is* (α, δ)-*private if the following two conditions hold: (1)* $prior_0 \geq \delta$ *for the first queried node* N_0*; and (2)* $posterior_i/prior_i \geq \alpha$ *for every iteration* $i > 0$

Choices for δ are values in the interval $[0, 2^m - 1]$ where 2^m is the size of the address space. Similarly choosing $\alpha \in [0, 1)$ allows a requester to tune the trade off between privacy and performance. The closer α is to 1 the less additional information N_i gains about the intended target.

VI. Iris

In this section, we describe Iris, the mechanism we develop to allow for (α, δ)-*private* queries in Chord. Table I defines the symbols and notation we use. We start by outlining the core idea behind its design. We then provide a detailed description with an execution example.

A. *Overview*

The ordered address space that is leveraged by the nodes in Chord provides a numerical basis to position nodes and calculate the distance between them. In every hop, the requester, by asking for the target object, finds nodes that have a smaller distance than the requester to the target. Yet, finding nodes that satisfy this condition can also be achieved if instead of the target object the requester queries for another identifier that is between the requester and the target object. Due to the ordered address space, getting closer to this intermediate identifier allows simultaneously the requester to get closer to the target object.

We built on this observation to develop Iris. In Iris, the requester, rather than asking the queried node for the target object, asks for an intermediate identifier. In this way, the requester gets closer to the target without however revealing the target. The requester iterates this process asking every time for another intermediate identifier so as to find the responsible node for the target object. In the section below we define how this process is done. We make Iris such as achieving (α, δ)-*privacy* when using Chord.

B. *Mechanism Description*

Iris replaces the regular `retrieve` algorithm from Chord. It takes two additional parameters, α and δ that determine the level of privacy to use for the request: $\texttt{iris}(RT_r, O_p, \alpha, \delta) \rightarrow \mathit{Data}$ OR nil.

The δ parameter allows the requester to control the size of the address range to which the target belongs, which the queried node can estimate. In Chord's `retrieve` algorithm, the requester asks first the node in its routing table that is closest and does not succeed the target address. This greedy heuristic gives to the queried nodes an estimate regarding the target of the request—the target does not succeed the requester's successor, which is deterministically defined, that comes after the queried node. Because nodes have a more dense view of the address space closer to them, this estimate becomes more accurate the closer the queried node is to the requester. To overcome this leakage Iris modifies Chord's `retrieve` protocol node selection by changing how the requester picks the first node to query. The δ parameter is used by the requester to calculate an address S that precedes the target object O_p by δ. The requester selects the first node to query to be its successor that most closely succeeds S but precedes O_p. If none of its successors belongs in this interval the first node to be queried is the requester's successor that most closely precedes S. With this selection process every queried node N_i, regardless of how close to the requester is, assuming that the node knows δ, can only deduce that the target of the request is one out of δ addresses from the address space that succeed N_i.

Algorithm 3 Iris's Retrieve Algorithm

```
function Iris($RT_r, O_p, \alpha, \delta$)
    $N_i \leftarrow$SelectStartNode($RT_r, O_p, \delta$)
    repeat
        ${N_i}' = N_i$
        $R_i \leftarrow$RandomAddressBetween($N_i, O_p$)
        $I_i \leftarrow$lerp($R_i, N_i, \alpha$)
        $N_i \leftarrow$Lookup(${N_i}', I_i$)
    until $N_i == {N_i}'$
    return Fetch($N_i, O_p$)
end function
```

The α parameter controls how fast the request converges to the target. The requester hides the target of the query from the intermediate nodes by substituting O_p with another address I_i, which succeeds the queried node but precedes the target. To pick I_i the requester firstly selects a reference point R_i by selecting uniformly at random an address in the interval $[N_i, O_p)$. By bounding the range selection of the randomly picked points to the actual target of the query, Iris guarantees correctness with the least possible number of steps, at the cost of allowing colluding attackers that get closer to the target to have a better guess regarding the target, *i.e.*, being able to calculate a *prior* range of smaller size. The requester then calculates I_i as the linear interpolation between the address of the queried node and the reference point, based on the α parameter, *i.e.*, $I_i = R_i + (N_i - R_i) \cdot \alpha$. The reference point provides privacy to the requester against a colluding adversary. Calculating I_i as the linear interpolation between the address of the queried node and the target, based on the α parameter, *i.e.*, $I_i = O_p + (N_i - O_p) \cdot \alpha$, would provide no privacy against our strong adversary model. By querying N_i for I_i the requester in every hop converges by a rate α to the reference point. As R_i comes after the queried node and precedes the target, the requester by converging to R_i converges simultaneously to O_p.

Algorithm 3 depicts Iris's pseudo-code. After calculating I_i, the requester asks N_i for I_i leveraging the Chord `lookup`. The queried node following the Chord `lookup` algorithm replies to the requester by indicating either another node closer to I_i or its responsible node. If the node to which the requester is relayed still precedes the target identifier, the requester repeats the process. The requester is free to renew the value of α between iterations.

The requester stops querying nodes when being relayed to a node that succeeds O_p. In this case, due to how the ownership of objects is defined in Chord, the responsible node of the target is simultaneously responsible for the queried identifier. The requester then precedes by executing the `fetch` protocol with the node responsible for O_p, getting back either the Data or nil if this object does not exist, and the `iris` algorithm terminates.

Application Example: An example execution of Iris is depicted in Figure 3. Let's assume a requester node 44 wants to find the node that stores the values of a service by the name

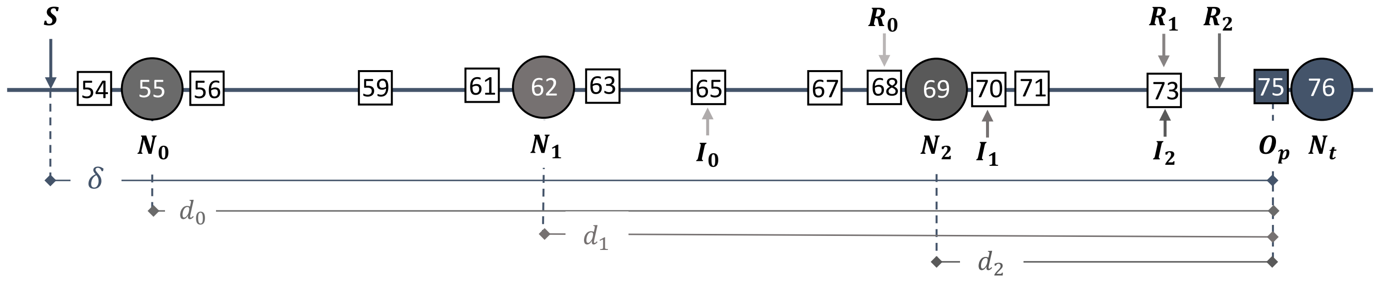


Fig. 3. Iris*'s application example.* The requester targeting object $O_p = 75$ selects $\delta = 22$ and $\alpha = 0.25$. queries back to back nodes for identifiers chosen in the interval $[53, 75)$. In every iteration the interval degrades, converging at the end to node $N_t = 76$.

$'secret'$. To achieve that, 44 needs to identify the responsible node of the targeted object $O_p = h('secret') = 75$. To avoid revealing the targeted object to the network, node 44 executes Iris. First, 44 tunes the δ parameter to be equal to 22 and calculates $S = 53$ by abstracting 22 from 75. Node 44 after consulting its routing table selects $N_0 = 55$, as this is the first of its successors belonging in the interval $[53, 75]$. Node 44 then selects parameter $\alpha = 0.25$ to control the rate of convergence of the query. After picking randomly $R_0 = 68$ in the interval $[55, 75)$, node 44 calculates I_0 based on $|I_0 - 68| = 0.25 \cdot |55 - 68|$, thus, queries node 55 for the identifier 65. Node 55 relays the requester to node 62. The requester then checks if node 62 comes after the targeted object 75. As this is not the case, it continues by picking a new random identifier $R_1 = 73$ from the range $[62, 75)$ and calculating a new queried identifier $I_1 = 70$. Node 44 queries node 62 for 70 having, as a result, to be relayed to node 69. Finally, the node 69 when queried for 73—calculated based on the reference point 74—relays node 44 to node 76 that comes after the target object 75. Node 44 executes the `fetch` protocol with node 76 asking for object 75 to retrieve the stored mapped data.

VII. Security Analysis

In this section we analyze Iris's *correctness* and we prove that Iris is an (α, δ)*-private* algorithm. We further analyze theoretically the advantage that Iris gives to a powerful adversary.

A. Correctness

We start the security analysis of Iris by formally proving its correctness, *i.e.*, guaranteeing that the requester upon executing the algorithm succeeds in identifying the node that stores the searched value.

Let N_t be the first successor of the requester's target object O_p, *i.e.*, N_t is the responsible node of O_p. Consider the i-th iteration of the algorithm where the requester executes the Chord `lookup` with node N_i specifying the address I_i, getting back N_{i+1} that is the next node to query. Recall from Section VI-B that the requester queries node N_{i+1} for another address I_{i+1}; thus, N_{i+1} is not necessarily the predecessor of I_i, yet, due to how the `lookup` progresses N_{i+1} is closer than N_i to I_i.

Let d_i be the distance between N_i and O_p, and R_i be a uniformly randomly picked address in $[N_i, O_p)$. On average, R_i is picked in the middle of the interval, thus, $|R_i - N_i| = |O_p - R_i| = d_i/2$. The queried address I_i is calculated based on R_i and the parameter α that the requester selects. More precisely, I_i is selected such as $|R_i - I_i| = \alpha \cdot |R_i - N_i|$; thus, $|R_i - I_i| = \alpha \cdot d_i/2$. Assuming that the distance between N_{i+1} and I_i is zero we have that $|R_i - N_{i+1}| = \alpha \cdot d_i/2$. From Figure 3, we observe that the distance the requester has to O_p at the $(i + 1)$-th iteration is the sum of the distance the queried node has to R_i plus the distance of R_i to O_p. Considering the above calculations, this distance is equal to $d_{i+1} = \alpha \cdot d_i/2 + d_i/2$. By referencing every step to the initial distance the requester has to the target object d_0, we have that in the n-th iteration the distance of the requester to O_p is given by Equation (1). Because $\alpha \in [0, 1)$, we have that $\lim_{n\to\infty} d_n = 0$, thus, the algorithm converges on the target.

$$d_n = d_0 \cdot \left(\frac{\alpha + 1}{2}\right)^n \tag{1}$$

Nodes are responsible for the identifiers that fall between their predecessor and their own node identifier; thus, the responsible node for O_p, follows O_p on the address space and there is no other node placed between them. Iris has the selection interval of the selected queried identifiers not to exceed the targeted object. Querying identifiers that only precede the targeted object guarantees that if a randomly picked object has its responsible node succeeding O_p, this node is also responsible for O_p. The `iris` algorithm is terminated to the predecessor of the node that is responsible for the queried identifier. Thus, Iris terminates when the distance the requester has to the target object becomes equal to the average distance the nodes have on the address space ν. Setting $d_n = \nu$, the number of iterations the requester needs on average to identify N_t while executing Iris is:

$$\begin{aligned} d_n = \nu \Rightarrow (\alpha + 1)^n \cdot \frac{d_0}{2^n} = \nu \\ \Rightarrow n = \frac{log_\alpha d_0 - log_\alpha \nu}{log_\alpha 2 - log_\alpha(\alpha + 1)} \end{aligned} \tag{2}$$

1) Secure Routing: Based on the `lookup` protocol, the requester is relayed to every other but the first node by the previous queried node. This can be leveraged by a colluding attacker who can relay the requester to a malicious node that—given that it succeeds the target—will be accepted by the requester as the responsible node, thus, learn the target.

Let's assume that the requester is relayed to N_{i+1} from N_i. The requester to conclude if N_{i+1} is the responsible node for O_p, can use bound checking [8]. Assuming N active nodes, the distances between consecutive active nodes can be modeled as approximately independent exponential random variables with mean equal to $\nu = (2^m - 1)/N$. Given $f \cdot N$ colluding nodes, the distances between consecutive colluding nodes can also be modeled as approximately independent exponential random variables with mean equal to $d_a = (2^m - 1)/(f \cdot N)$. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the distributions of active and colluding nodes, respectively. The requester can identify if N_{i+1} is the responsible node for O_p by determining if d_x, that is the distance between N_{i+1} and N_i, is drawn from distribution $\mathcal{F}_1$ and not $\mathcal{F}_2$. The requester does not know N but based on its routing table and its distance to its predecessor, can make an estimation d_r concerning the address range for which each node is responsible. For N_{i+1} to be N_t we need to have $d_x \geq T$ where $T = \gamma \cdot d_r$ and $\gamma \in (1, 1/f)$. Based on our threat model the attacker does not control the responsible node, thus $d_x > T$. According to [8], to obtain the minimum false positives and false negatives, γ must be equal to $\gamma = 1/f$.

B. Query Privacy

Here we prove that IRIS is an (α, δ)-*private* algorithm following the definition introduced in Section V. We start by analyzing the query privacy guarantees that IRIS provides against an adversary that has no more than one node under control that can, however, be any of the queried nodes. We then continue by considering a more powerful adversary that controls multiple nodes in the network. Based on our system and adversary model, in our analysis we assume that nodes are authenticated, i.e., they cannot lie about the address they control.

1) Lone Adversary: Let us now consider a lone adversary that controls only node N_i. Recall from Section VI-B the way the requester selects the queried nodes, *i.e.*, N_0 is the requester's successor immediate after or before address S that precedes the target by δ. Based on this selection process, N_i can deduce the following about the requester's target. If one of the requester's successors belongs in $[N_i, N_i + \delta]$ then the requester searches something that belongs in $[N_i, UB_i]$ where $UB_i = N_i + \delta$. If there are no successors of the requester in $[N_i, N_i + \delta]$ then the requester searches something that belongs in $[N_i, UB_i]$ where $UB_i = N_i + \delta + x$, denoting as x the larger than δ distance a queried node can have from the actual target. Based on our adversary model we assume that the δ parameter is known to the attacker. Thus, the worst scenario is the prior knowledge of N_i to be equal to $|UB_i - N_i| = \delta$.

Node N_i is queried by the requester for the identifier I_i; thus, the posterior knowledge of N_i is $|UB_i - I_i|$. I_i is picked based on the reference point R_i that succeeds I_i but precedes O_p thus UB_i on the address space. Thus, the following holds $|UB_i - R_i| + |R_i - I_i| = |UB_i - I_i|$ and $|UB_i - R_i| + |R_i - N_i| = |UB_i - N_i|$. By definition $|R_i - I_i| = \alpha \cdot |R_i - N_i|$, thus, we have:

$$\begin{aligned} \frac{posterior_i}{prior_i} &= \frac{|UB_i - I_i|}{|UB_i - N_i|} = \frac{|UB_i - R_i| + |R_i - I_i|}{|UB_i - R_i| + |R_i - N_i|} \\ &= \frac{|UB_i - R_i| + \alpha \cdot |R_i - N_i|}{|UB_i - R_i| + |R_i - N_i|} \\ &= \alpha \cdot \frac{\frac{|UB_i - R_i|}{\alpha} + |R_i - N_i|}{|UB_i - R_i| + |R_i - N_i|} \\ &= \alpha \cdot c_a \end{aligned} \tag{3}$$

In Equation (3), as c_a has in its numerator the parameter $\alpha \in [0, 1)$ as denominator we can conclude that $c_a > 1$, thus, the ratio between the posterior and prior knowledge of the adversary is greater than α. Hence, we can conclude that IRIS is an (α, δ)-*private* algorithm against a lone adversary.

2) Colluding Adversary: Let us now consider the case of a colluding adversary that controls a fraction f of the nodes in the network with N_j and N_i being two consecutive adversarial nodes, both queried by the requester when searching for O_p. The worst case scenario is for the attacker correctly to assume that the two different queries serve the same search. Due to the random reference point in the calculation of the queried address at step 6 in Algorithm 3, the attacker cannot calculate O_p. However, from Figure 4, we observe that node N_i can use as an upper bound UB_j instead of UB_i. Thus, the prior knowledge of N_i is equal to $|UB_j - N_i|$. Now considering that $UB_j = N_j + \delta$, we have:

$$prior_i = \delta - |N_i - N_j| \tag{4}$$

A colluding attacker with average distance between colluding nodes bigger than δ is only queried once, hence, this case is equal to a lone attacker from a security perspective. From Equation (4), we observe that for a colluding attacker for whom the nodes under control have average distance d_a, the minimum distance the first adversarial node has to the estimate upper bound is equal to δ. However, assuming that t colluding nodes are queried throughout the `iris` execution, at the end the minimum distance the adversarial node has to the estimate upper bound is equal to $\delta' = \delta - t \cdot d_a$.

Regarding the prior and posterior knowledge ratio of N_i, if in Equation (3) we replace UB_i with UB_j we have $posterior_i/prior_i = \alpha \cdot c_b$. As UB_i succeeds UB_j, we have that $|UB_i - R_i| > |UB_j - R_i|$, thus, $c_a > c_b$. Hence, IRIS achieves a lower ratio between the posterior and prior knowledge against a colluding adversary compared to a lone adversary, yet still lower bounded by α. From the above we can conclude that IRIS is an (α, δ)-*private* algorithm against a colluding adversary.

C. Attacker Advantage

Let us now consider the advantage that IRIS gives to the attacker, what the attacker can deduce regarding the target of the requester based on the information available to the attacker. We consider the worst case scenario, assuming the attacker knows the α and the δ parameters the requester has chosen.

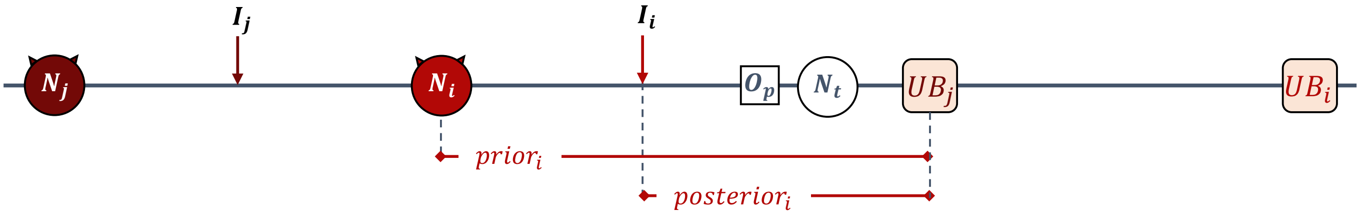


Fig. 4. *A colluding adversary.* Assuming that N_j is the first asked colluding adversary, every other colluding node that the requester queries can use UB_j instead of UB_i in their calculation to infer the target.

Following IRIS, the requester chooses I_i based on α and the randomly picked address R_i. Assuming the attacker knows α, when queried for an identifier I_i, the attacker can calculate the R_i the requester picked. Based on this deduction, in our analysis we examine the probability IRIS gives for the target address to have a specific value o given a randomly picked value x by the requester.

We have assumed that the attacker knows the δ parameter. This prior knowledge, as in previous section explained, can be used by the attacker to calculate an upper bound for the address of the target, *i.e.*, the target will be an address that is no further away than δ addresses from the attacker's address. However, this is only true for the attackers that succeed S address, where $S = O_p - \delta$. Any attacker preceding S, even with the knowledge of δ, cannot calculate a correct upper bound, *i.e.*, the target will succeed the attacker's address by more than δ addresses. The attacker has no way to conclude where on the address space is placed in reference to the address S, thus, the best thing the attacker can do is to guess that succeeds S even if this might not be the case.

Without loss of generality we can position the attacker at the start of the address space $N_i = 0$. As described in Section VI-B, R_i is selected uniformly at random in the interval that extends from the address of the attacker to one address before the address of the target $R_i \in [N_i, O_p - 1]$, thus, $x \in [0, o-1]$. The UB_i is δ addresses away from the address of the attacker. Assuming $N_i = 0$ we will have $UB_i = \delta$. Given a specified value of the random point x, from the attacker's perspective it is equally likely the target address to be any of the addresses that succeeds x and precedes δ (with δ included).

In Figure 5 we illustrate by O the possible addresses that the target can have given that the randomly picked address is equal to or equal and less than a specific value x. Given $R_i = x$, in Figure 5(a) and in Figure 5(c), we examine what are the possible case(s) for the target to be equal to value o whereas, in Figure 5(b) and in Figure 5(d) we consider the cases for the target to be equal or less than o. Counting down the number all the examined events o and all possible addresses O (that incorporate the number of o) for every $x \in [0, o-1]$, we have that:

$$P(O_p = o | R_i = x) = \frac{1}{\delta - x} \tag{5}$$

$$P(O_p <= o | R_i = x) = \frac{o - x}{\delta - x} \tag{6}$$

$$P(O_p = o | R_i <= x) = \frac{x}{x(\delta - x) + \frac{x(x-1)}{2}} = \frac{2}{2\delta - x - 1} \tag{7}$$

$$P(O_p <= o | R_i <= x) = \frac{x(o - x) + \frac{x(x-1)}{2}}{x(\delta - x) + \frac{x(x-1)}{2}} = \frac{2o - x - 1}{2\delta - x - 1} \tag{8}$$

From Equations (5) to (8) we observe that given the information the attacker has it is equally likely that the target is any of the possible addresses. Thus, while considering a powerful adversary that knows both the α and the δ parameters still IRIS succeeds in hiding the target from the attacker.

VIII. Evaluation

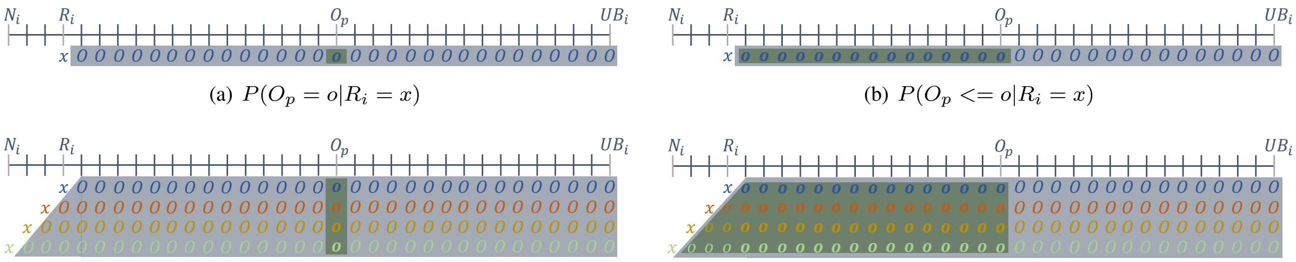
We have created a Chord network simulation where we can vary any of the network parameters independently, and run experiments with any combination of parameters. We use this to run a large number of simulations with networks that differ in size, number of nodes, fraction of adversaries, number of data objects, etc.

We simulate IRIS in this environment with a range of choices for the α and δ parameters, and analyze how they affect performance and correctness. We simulate different fractions of adversaries in the network and analyze the privacy degree we achieve, accounting for colluding adversaries and perfect ability to guess the requester's parameters, in accordance with our threat model. We confirm the probabilistic advantage an attacker has, and find that no attacker advantage exceeds α, thus confirming the (α, δ)-*privacy* of IRIS.

We start by describing the setup of our simulation before moving on to the presentation of the results of our experiments.

A. Simulation Setup

We simulate IRIS using the Matlab programming language. We model an address space of 2^{23} addresses, on which we position 1000 nodes uniformly at random, selecting a fraction f of them to be colluding adversaries. We implement Chord `lookup` and run the network until a steady state has been reached. Each node keeps a routing table with $m = 23$ other nodes as specified by the Chord protocol. Given such a network, our implementation selects a requester and a target object at random from the set of non-attackers, and executes IRIS as defined in Algorithm 3. The requester is free to choose the α and δ parameters.



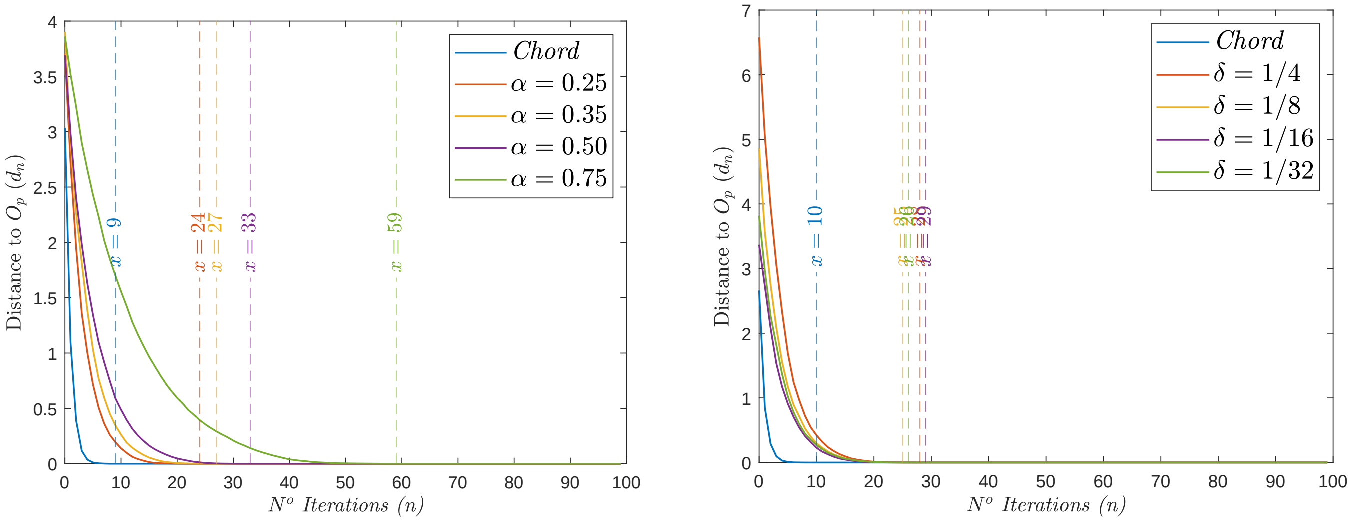
(a) $P(O_p = o | R_i = x)$

(b) $P(O_p <= o | R_i = x)$

(c) $P(O_p = o | R_i <= x)$

(d) $P(O_p <= o | R_i <= x)$

Fig. 5. *Probability Calculation:* We mark with O the addresses that O_p can obtain and with o and x the explicit value(s) of O_p and R_i, we examine.



(a) IRIS's convergence for $\delta = 1/16$.

(b) IRIS's convergence for $\alpha = 0.35$.

Fig. 6. *Comparing* IRIS*'s performance for different values of α and δ parameters.* The x-axis indicates the number of iterations needed to converge to the target, whereas the y-axis indicates the distance the queried nodes have to the target normalised by 1/16 of the address space.

We have open-sourced our simulation code, the evaluation scripts, and the presented benchmarks as artifacts, and the code is available at https://github.com/angakt/iris.

Each experiment is run k times, with an entirely new network each time. This way, our results are independent of the requester and the target positions in the network, as well as any particular distribution of attackers. When examining the communication overhead, *i.e.*, the number of iterations IRIS needs to terminate, we execute every experiment $k = 100$ times and we report the average distance in every iteration across all the executions. When examining the privacy guarantees IRIS provides, we execute every experiment $k = 500$ times and we calculate the minimum ratio of posterior to prior knowledge across every execution.

B. Simulation Results

Performance: To understand the performance overheads introduced by IRIS we vary the α and δ parameters and compare the performance with the vanilla Chord algorithm. To avoid an impact from differences in attacker strategy, we run the performance experiments in a network with no adversaries, *i.e.*, $f = 0$.

In Figure 6(a) we report the average distance to the target on every hop, in a network of 1000 participants. We evaluate IRIS with α equal to 0.25, 0.35, 0.50, and 0.75 keeping δ constant and equal to $2^m/16$. The performance of IRIS is compared to vanilla Chord and averaged over $k = 100$ experiments. The dashed vertical lines indicate the *maximum* (worst case) number of hops the requester needed to identify the responsible node for the target object.

We observe that the α parameter has a dominant effect on IRIS's convergence time: the smaller the value of α the faster the convergence. This is due to the fact that bigger α values result in more conservative steps towards the target. This gives away less information to intermediate nodes, but it also comes with a performance penalty. This result highlights an important quality of IRIS, namely the fact that the trade-off between performance and privacy can be tuned to only pay the performance penalty for the amount of privacy needed.

To asses the performance impact of δ we keep α constant ($\alpha = 0.35$) and vary the δ parameter. We again run the experiment 100 times with $f = 0$, and the results can be seen in Figure 6(b). We observe only minor difference in

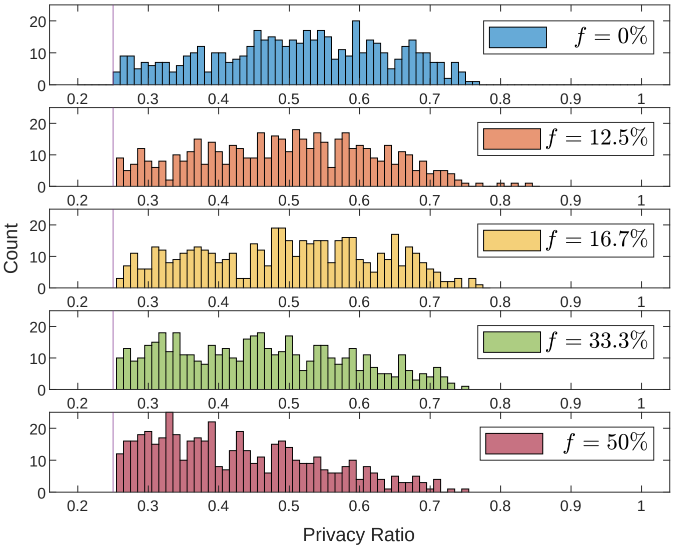


Fig. 7. The posterior and prior knowledge rate for different fractions of colluding adversaries.

the average number of hops, which matches the prediction produced by Equation (2) where it is clear that the dominant factor determining IRIS's performance is α. Changes in the δ parameter affect the fraction's numerator and have a negligible influence on the final calculated value. The intuitive explanation for this is while a larger δ causes us to start the search further from the target, we also take larger steps when we are far from the target, so the overall effect on performance is minor.

Query Privacy: To validate the query privacy guarantees that IRIS achieves with respect to the privacy notion introduced in Section V, we execute IRIS by varying the fraction of colluding adversaries. Following the analysis in Section VII-B, we examine the posterior-to-prior knowledge ratio IRIS achieves for each node along the routing path. Each adversarial node is provided the value of α and δ (even though these values would not be available to the adversaries in practice), and colluding attackers are allowed to compare values.

For most attackers δ is an upper bound of where the target object could be. By knowing δ, the queried nodes that are more than δ addresses away will know that, and they do not contribute to the estimate of later nodes. If they did, the nodes would be wrong about the target location, so this represents a further advantage for the attackers that would not exist in practice. By doing this we get the absolute worst case results for the requester, and therefore a lower bound on the privacy.

In Figure 7 we illustrate the *minimum* achieved privacy ratios we get across $k = 500$ experiments when we keep $\alpha = 0.25$ and $\delta = 2^m/4$ and we vary the fraction of colluding nodes from $f = 0\%$ to $f = 50\%$. We notice that regardless of the fraction of attackers, the privacy ratio does not drop below $\alpha = 0.25$, and is in fact much higher than α in most runs, sometimes as high as 0.7. This confirms what we proved in Section VII, namely that IRIS is an (α, δ)-*private* algorithm. We also observe that the greater the fraction of colluding adversaries, the more frequently we get smaller values of the privacy ratio, i.e., the histogram move to the left. However, even for large values of f we remain above α at all times.

Attacker Advantage: To demonstrate the knowledge gain a non-colluding attacker gets from being used as an intermediate node, we execute IRIS with $\alpha = 0.75$ and $\delta = 2^m/128$, and the fraction of *colluding* adversaries $f = 0$. We then calculate the distance between an intermediary node and the target (and to the randomly picked point R_i), for every queried node, and we execute 500 experiments.

In Figure 8, we plot a histogram of the normalized distance to the target (and the random point R_i). The distance is normalized so it corresponds to a percentage of the δ parameter, since δ is an upper bound on the distance for almost all nodes. We observe that the distance to the target follows a uniform distribution, *i.e.*, every node in the interval $[O_p - \delta, O_p]$ is equally likely to be the target. For the position of the random point, we observe a right skewed distribution. This occurs because as the distance between the queried node and the target gets smaller, the probability of getting a high value for R_i tapers off. Thus, for a uniformly distributed distance to the target we have more lower than higher values for R_i. A uniform distance to the target is ideal, since it means that a non-colluding intermediate node has effectively no information about the location of the target, other than it is likely to be at most δ addresses away.

In Figure 9, we validate out implementation of Iris by plotting the probabilities we get from experiments (in blue) against the probability expressions we get from Equations (5), (6), (7) and (8) (in yellow). We observe that for Figures 9(a) and 9(c) the data from our experiments fully confirm our calculations. For Figures 9(b) and 9(d) the plots follow the equation's trend, however, they are higher than expected. This bias occurs due to the very common low values we get, that is reflected when we simultaneously examine more than one value for O_p. In Figure 9(b), we observe that the deviations are getting smaller as x gets bigger. In Figure 9(d), we observe that the deviations are maintained as the x gets bigger. This happens because for this case we examine a range for values for x, thus, any bias in smaller values is inherited to the bigger ones. Figure 9 illustrates that IRIS data from our simulation agrees with the calculated probabilities, or are lower bounded by them. This acts as a sanity check on the code used for our simulations and confirms our results described above.

IX. RELATED WORK

In this section, we provide background on decentralized privacy schemes with a focus on the ones applied to Chord. We start by elaborating on already proposed privacy metrics before describing other privacy architectures and examining how they compare to IRIS.

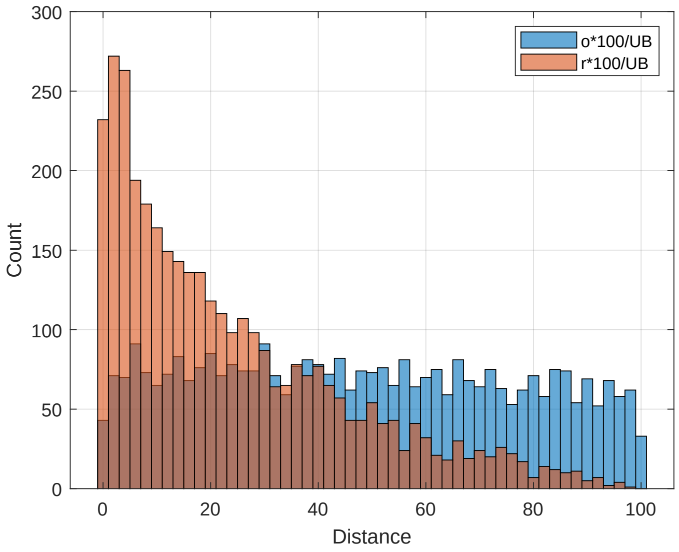


Fig. 8. The distance that queried nodes have to the target and to the randomly picked point expressed as a percentage of the δ value.

A. Privacy Metrics

In the literature, many works have studied the privacy guarantees of decentralized systems [47]. To measure the privacy provided in structured P2P architectures some works extract an anonymity score based on the size of the anonymity set, either solely [15], [30] or by normalizing it by the best possible value [6]. In both cases, the calculated score changes between nodes according to the distance the node has to the target of the request. Adopting such a metric to measure the query privacy that is provided as underlined in Section V does not reflect the average privacy achieved but the worst-case scenario. In [6], the authors also propose another anonymity metric based on entropy that takes into consideration the probabilistic advantage an attacker can have in identifying initiators. The authors underline that computing the probability of the distribution of events given an observation can be difficult to apply to complex, dynamic systems. In [38] to calculate the degree of privacy the authors use the rate between the posterior and the prior entropy after and before a compromise has occurred. (α, δ)-*privacy* builds on the last metric, but comparing to that it necessitates no demanding calculations by replacing the entropy with the size of the possible set in which the target can belong.

B. Privacy Architectures

Structured P2P architectures have been used in anonymous communication systems such as TOR as a scalable way to select the nodes to build anonymous circuits. This led many works [8], [26], [17], [49], [50], [29], [9] to focus on the security of routing, *i.e.*, guarantee an unbiased and correct `retrieve` process. IRIS tries to enhance the privacy aspect that is inherently built in Chord so as privacy guarantees can be achieved without demanding other infrastructures.

A major line of research focused on enhancing the anonymity of the sender and the receiver in structured P2P architectures [15], [39], [49], [33], [24], [27], [28], [32]. IRIS focuses on a different problem, *i.e.*, hiding the information that is queried from the intermediate nodes that participate in the routing while allowing authentication for the participating nodes.

There is a more limited bibliography regarding the deterrence of user profiling in structured P2P architectures. The work in [14] split the data and publish every share under a different overlay address, guaranteeing privacy against an adversary that can capture a small set of shares. Other works [3], [22], [51] organize nodes in quorums; the client uses threshold cryptography to obtain the index of the wanted item without revealing the item and without the individual quorum nodes knowing which item was extracted. More recently, Peer2PIR [23] applies private information retrieval (PIR) techniques to limit privacy leakage on peer routing, provider advertisements and content retrieval in the IPFS network, which is based on the Kademlia [21] DHT. IPFS is also in the process of performing a privacy upgrade by implementing Double Hashing [25]. Double Hashing has the requester query for a prefix of the target identifier and receive the records corresponding to any object that matches this prefix, thus guaranteeing k-anonymity, where k is the number of objects that match this prefix. These schemes propose changes in the overlay structure and operations that have to be followed by all the nodes in the network. IRIS does not demand global changes.

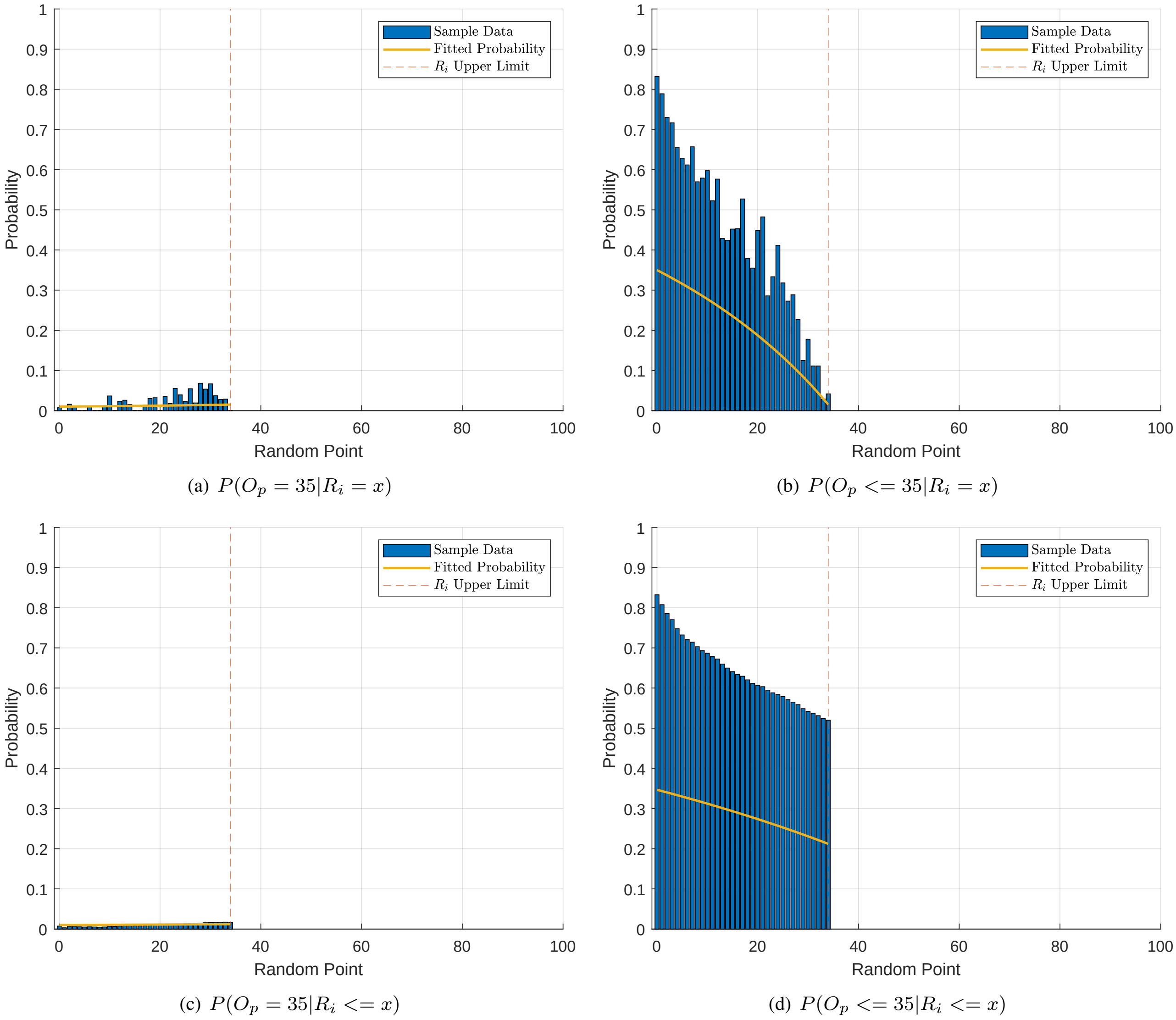
X. DISCUSSION

In this section we provide further guidance on the selection of the α and δ parameters, and discuss limitations of IRIS and its extension to other DHTs.

A. Selection of α and δ Parameters

The α and δ parameters must be selected to be in the ranges $[0, 1)$ and $[0, 2^m - 1]$, respectively. Conceptually, the bigger the values, the better the provided privacy. However, as shown in Figure 6 there is a trade-off between IRIS's privacy and performance, *i.e.*, the gain that IRIS provides comes at a cost of increased steps to reach the target. Thus, it makes sense to consider what a good selection of the privacy parameters look like; one that will guarantee a sufficient privacy level without sacrificing too much performance.

Staring with the δ parameter, we see in Figure 6(b) that its value does not significantly affect the number of iterations until convergence. This parameter specifies a minimum distance to the target, and thus directly controls the anonymity we get against the first queried node in a group of colluding nodes. This is because such a distance removes the link between the choice of intermediate node and the target itself, that vanilla chord would have. Intermediate nodes can no longer make assumptions about the location of the target, based on the fact that they are being used as intermediaries. We should pick



(a) $P(O_p = 35|R_i = x)$

(b) $P(O_p <= 35|R_i = x)$

(c) $P(O_p = 35|R_i <= x)$

(d) $P(O_p <= 35|R_i <= x)$

Fig. 9. *Probabilities for* $O_p = 35$*:* The probabilities we get from our prototype implementation.

a value that contains enough network objects to constitute a good anonymity set.

As an example consider the Tor network. In Tor we have roughly $900,000$ onion addresses [35]. Assuming a space of 2^{23} addresses that are uniformly distributed, on average every 9th address will be a valid onion address. To achieve an anonymity degree of k, we must set $\delta = k * 9$.

The selection of the α parameter has a more dominant effect on performance. This is shown in Figure 6(a) but we can see it more clearly in Figure 10, which illustrates the number of steps IRIS needs to converge to the target, across 100 experiments altering α with a step of 0.025 in the interval $[0, 1)$, keeping $\delta = 1/16 * 2^{23}$ and $f = 0$.

We observe that the number of steps IRIS needs to converge increases exponentially with respect to α, starting at 10, which is the number of steps that vanilla Chord achieves. For $\alpha < 0.7$ the increase in the steps looks almost linear, however above that, the number of steps starts to increase drastically. We still get stronger and stronger privacy guarantees, but with high performance overheads. For that reason a good practical choice of α is around 0.7 which will roughly double the steps (and search time) of vanilla chord.

Note that each search can be done with a different choice of parameters, so sensitive searches might need $(.7, 500k)$-privacy, where as normal ones can use no privacy at all.

B. Other P2P Architectures

IRIS can be directly applied to any existing system that implements the Chord protocol, e.g., [11], [18], [43], [46], [1], [34], [20], [10], [41]. IRIS can be used without any support from the other nodes in the network, even when all other nodes in the network use the vanilla Chord protocol, which makes it easy to adopt for privacy conscious clients.

In addition to Chord, several other DHTs exist in the literature, such as Kademlia [21], Tapestry [52], Pastry [40] and

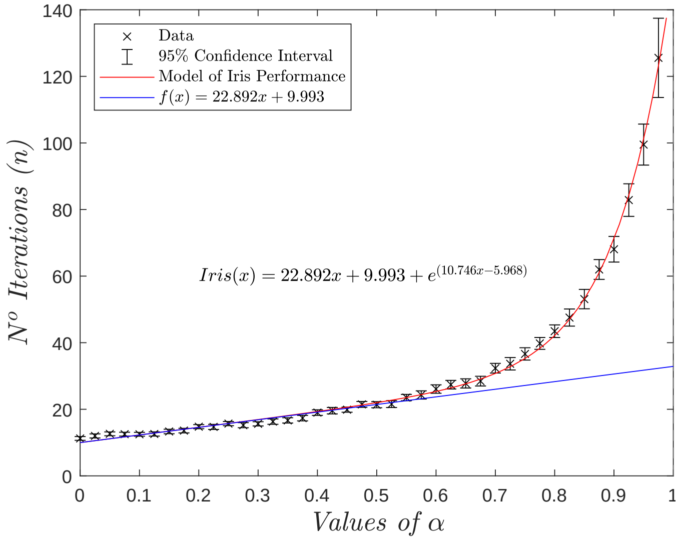


Fig. 10. The number of steps that are needed to reach the target for different values of the α parameter.

CAN [37]. These are, in principle, similar enough to Chord to provide a way that allows nodes to query their neighbours and choose the next hop when routing a message [16]. The general schema that these protocols offer intuitively suggests that IRIS can be applied to these systems as well. However, the differences that exist between them, such as the distance metric they define, may affect the provided guarantees. For example Chord uses the distance between node addresses whereas Kademlia uses an XOR metric. We leave a detailed analysis of the application of IRIS to other architectures as future work.

C. Limitations

As the requester progressively queries nodes that are closer and closer to the target, intermediate nodes will get a more and more precise range of possible targets if they are colluding with previously chosen intermediate nodes. This is an unavoidable consequence of our very strong threat model, but it is accounted for in the definition of (α, δ)-privacy and even in the worst case, the additional information each adversarial node gets is bounded by $1 - \alpha$.

XI. CONCLUSION

In this paper, we study the privacy guarantees of a search request when using Chord. To reason about the query privacy that a privacy-preserving mechanism provides, we introduce a new notion called (α, δ)*-privacy*. This privacy notion allows to measure the privacy level of a request even in the presence of strong colluding adversaries. We further design IRIS, an algorithm that replaces the regular `retrieve` algorithm in Chord to allow a requester to conceal the target of a query from the intermediate nodes that take part in the search. By performing a thorough security analysis we prove IRIS to be both correct and (α, δ)*-private*. We also perform an empirical analysis using simulations to evaluate the privacy levels that IRIS achieves and to study the trade-offs our proposal introduces between the achieved query privacy and performance. The results confirm our theoretical analysis and indicate a modest communication overhead that can be tuned by the requester based on the privacy level each query demands.

ACKNOWLEDGMENTS

We want to thank the paper and the artifacts' anonymous reviewers for their time and the valuable feedback they have provided us.

REFERENCES

[1] Sameer Ajmani, Dwaine E. Clarke, Chuang-Hue Moh, and Steven Richman. ConChord: Cooperative SDSI certificate storage and name resolution. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, number 2429 in Lecture Notes in Computer Science, pages 141–154, March 2002.

[2] Angeliki Aktypi, Dimitris Karnikis, Nikos Vasilakis, and Kasper Rasmussen. Themis: A secure decentralized framework for microservice interaction in serverless computing. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, New York, NY, USA, 2022. ACM.

[3] Michael Backes, Ian Goldberg, Aniket Kate, and Tomas Toft. Adding query privacy to robust dhts. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, AsiaCCS '12, page 30–31, New York, NY, USA, 2012. Association for Computing Machinery.

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

[5] Juan Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

[6] Nikita Borisov and Jason Waddle. Anonymity in structured peer-to-peer networks. Technical report, Computer Science Division, University of California, 2005.

[7] Kevin R.B. Butler, Sunam Ryu, Patrick Traynor, and Patrick D. McDaniel. Leveraging identity-based cryptography for node id assignment in structured p2p systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1803–1815, 2009.

[8] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, dec 2003.

[9] Giuseppe Ciaccio. Improving sender anonymity in a structured overlay with imprecise routing. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies*, pages 190–207, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[10] Russ Cox, Athicha Muthitacharoen, and Robert T. Morris. Serving dns using a peer-to-peer lookup service. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 155–165, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[11] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, page 202–215, New York, NY, USA, 2001. Association for Computing Machinery.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 21, USA, 2004. USENIX Association.

[13] Edward Eaton, Sajin Sasy, and Ian Goldberg. Improving the privacy of tor onion services. In *International Conference on Applied Cryptography and Network Security*, pages 273–292. Springer, 2022.

[14] Benjamin Fabian, Tatiana Ermakova, and Cristian Muller. Shardis: A privacy-enhanced discovery service for rfid-based product information. *IEEE Transactions on Industrial Informatics*, 8(3):707–718, 2012.

[15] Michael J Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 193–206, New York, NY, USA, 2002. ACM.

[16] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 381–394, New York, NY, USA, 2003. Association for Computing Machinery.

[17] Apu Kapadia and Nikos Triandopoulos. Halo: High-assurance locate for distributed hash tables. In *Proceedings of the 16th Network and Distributed System Security Symposium*, volume 8 of *NDSS '08*, page 142. Citeseer, 2008.

[18] NKN Lab. NKN: A scalable self-evolving and self-incentivized decentralized network. Whitepaper, 2018.

[19] Storj Lab. Storj: A decentralized cloud storage network framework. Whitepaper, 2018.

[20] T Labbai and S Jothi Prasanna. T2wsn: Titivated two-tired chord overlay aiding robustness and delivery ratio for wireless sensor networks. *Journal of Theoretical & Applied Information Technology*, 91(1), 2016.

[21] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65, Cham, 2002. Springer.

[22] Miti Mazmudar, Stan Gurtler, and Ian Goldberg. Do you feel a chill? using pir against chilling effects for censorship-resistant publishing. In *Proceedings of the 20th Workshop on Privacy in the Electronic Society*, WPES '21, page 53–57, New York, NY, USA, 2021. Association for Computing Machinery.

[23] Miti Mazmudar, Shannon Veitch, and Rasoul Akhavan Mahdavi. Peer2pir: Private queries for ipfs, 2024.

[24] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with torsk. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 590–599, New York, NY, USA, 2009. Association for Computing Machinery.

[25] Guillaume Michel. Ipip-373: Double hash dht spec. Technical report, IPFS, 2023.

[26] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S. Wallach. Ap3: Cooperative, decentralized anonymous communication. In *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, EW '04, page 30–es, New York, NY, USA, 2004. Association for Computing Machinery.

[27] Prateek Mittal and Nikita Borisov. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 161–172, New York, NY, USA, 2009. ACM.

[28] Prateek Mittal, Matthew Caesar, and Nikita Borisov. X-vine: Secure and pseudonymous routing using social networks. In *Proceedings of the 2012 Network and Distributed System Security Symposium*, NDSS '12, 2012.

[29] Arjun Nambiar and Matthew Wright. Salsa: A structured approach to large-scale anonymity. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 17–26, New York, NY, USA, 2006. Association for Computing Machinery.

[30] C.W. O'Donnell and V. Vaikuntanathan. Information leak in the chord lookup protocol. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, pages 28–35, Aug 2004.

[31] Esther Palomar, Juan M. Estevez-Tapiador, Julio C. Hernandez-Castro, and Arturo Ribagorda. A p2p content authentication protocol based on byzantine agreement. In *Emerging Trends in Information and Communication Security*, pages 60–72, Berlin, Heidelberg, 2006. Springer.

[32] Andriy Panchenko, Asya Mitseva, and Sara Knabe. Whisperchord: Scalable and secure node discovery for overlay networks. In *2021 IEEE 46th Conference on Local Computer Networks*, LCN '21, pages 170–177, 2021.

[33] Andriy Panchenko, Stefan Richter, and Arne Rache. Nisan: Network information service for anonymization networks. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 141–150, New York, NY, USA, 2009. Association for Computing Machinery.

[34] Giuseppe Pirrò, Domenico Talia, and Paolo Trunfio. A dht-based semantic overlay network for service discovery. *Future Generation Computer Systems*, 28(4):689–707, 2012.

[35] Tor Project. Tor metrics - onion services. https://metrics.torproject.org/hidserv-dir-v3-onions-seen.html, 2024. Accessed: 2024-10-08.

[36] Bernd Prünster, Dominik Ziegler, Chrisitan Kollmann, and Bojan Suzic. A holistic approach towards peer-to-peer security and why proof of work won't do. In *Security and Privacy in Communication Networks: 14th International Conference, SecureComm 2018, Singapore, Singapore, August 8-10, 2018, Proceedings, Part II*, pages 122–138. Springer, 2018.

[37] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[38] Souvik Ray and Zhao Zhang. An information-theoretic framework for analyzing leak of privacy in distributed hash tables. In *Proceedings of the 7th IEEE International Conference on Peer-to-Peer Computing*, P2P '07, pages 27–36, Sep. 2007.

[39] Marc Rennhard and Bernhard Plattner. Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES '02, page 91–102, New York, NY, USA, 2002. Association for Computing Machinery.

[40] Antony Rowstron. Pastry: Scalable, distributed object location and routing for large-scale, persistent peer-to-peer storage utility. In *IFIP/ACM International Conference on Distributed Plarforms*, 2001.

[41] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using sip. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '05, page 63–68, New York, NY, USA, 2005. Association for Computing Machinery.

[42] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *Peer-to-Peer Systems*, pages 261–269, Berlin, Heidelberg, 2002. Springer.

[43] Emil Sit, Robert Tappan Morris, and M Frans Kaashoek. Usenetdht: A low-overhead design for usenet. In *NSDI*, pages 133–146, 2008.

[44] Mudhakar Srivatsa and Ling Liu. Vulnerabilities and security threats in structured peer-to-peer systems: A quantitiative analysis. In *Proceedings of the 20th Annual Computer Security Applications Conference*, volume 10 of *ACSAC '04*, pages 252–261, USA, 2004. IEEE Computer Society.

[45] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[46] Jeremy Stribling, Jinyang Li, Isaac G Councill, M Frans Kaashoek, and Robert Tappan Morris. Overcite: A distributed, cooperative citeseer. In *NSDI*, volume 6, pages 11–11, 2006.

[47] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. Systematizing decentralization and privacy: Lessons from 15 years of research and deployments. *Proceedings on Privacy Enhancing Technologies*, 4:404–426, 2017.

[48] Dan S. Wallach. A survey of peer-to-peer security issues. In *Software Security — Theories and Systems*, pages 42–57, Berlin, Heidelberg, 2003. Springer.

[49] Peng Wang, Ivan Osipkov, N Hopper, and Yongdae Kim. Myrmic: Secure and robust dht routing. Technical report, University of Minnesota, 2006.

[50] Qiyan Wang and Nikita Borisov. Octopus: A secure and anonymous dht lookup. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pages 325–334, June 2012.

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

[52] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

Appendix A
Artifact Appendix

In this work, we introduce Iris, an algorithm that allows nodes that participate in authenticated Chord P2P networks to perform queries without revealing the target of their search to nodes other than the one holding the information. The provided artifacts include the source code of our implementation of the Iris and the Chord algorithms and the code to execute the experiments we have performed to support our claims (together with the datasets we have produced). For completeness, we also include the scripts used to create our plots. The provided artifacts allow peers to reproduce the experiments we present in the paper and build upon them, inspiring further research and development in this area.

A. Artifact Structure

In the repository we provide the code and the data we use in the paper. As a fist step, users can run the code to execute the Iris algorithm, and to collect execution data. Subsequently, the users can run the provided plot scripts to recreate the figures used in the paper. The network generation is randomized, so to fully reproduce the results we present in the paper, we share the data we generated and used for our plotting.

In Figure 11 we provide the schema of the coding files in the repository. The majority of the work is done in these four files:

- `Id_Space_Linear.m` creates a circular id space with a number of participating nodes placed uniformly at random. A fraction of them, *i.e.*, $0, \frac{1}{2}, \frac{1}{3}, \frac{1}{8}, \ldots$, are chosen as colluding adversaries.
- `Iris.m` implements the Iris algorithm as described in Section VI of the paper. This function performs an iterative search for a target object O_p, initiated by a requester N_r keeping the privacy parameters α and δ constant during the search.
- `Privacy.m` calculates the loss of privacy at every node that is queried in a search. This loss is proportional to the ratio of the *posterior* and the *prior* knowledge about the target object, as specified in Section V of our paper.
- `Chord_Lookup.m` implements the Chord `lookup` protocol, as described in Section II of our paper. Given a specified target object, the code returns the address of the next node to be queried.

B. Set Up

1) Access: The complete artifacts and most recent version of the code can be accessed at https://github.com/angakt/iris. A snapshot of the release (based on which the artifacts evaluation has been performed) has also been uploaded at Zenodo and can be accessed at https://doi.org/10.5281/zenodo.14251874.

2) Hardware Dependencies: Any computer that can run Matlab or GNU Octave. For our implementation we use a computer with an Intel Core i7-4820K processor and 16GB installed RAM memory.

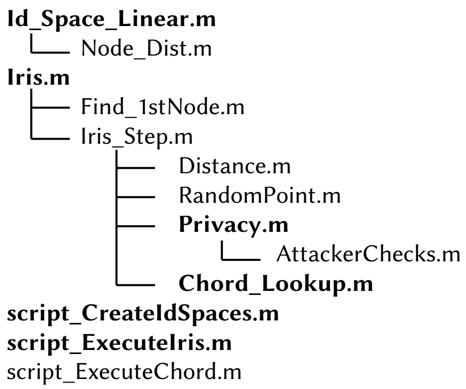


Fig. 11. *The schema of the Iris code base.*

3) Software Dependencies: The code can be executed on any operation system that supports Matlab or Octave, e.g., Windows, MacOS, or Ubuntu. We test our code on Windows 10, Education edition, version 22H2, running the Matlab version R2023a with an academic licence. However, as the code does not use any special libraries, GNU Octave, an open-source alternative to Matlab, is also sufficient to run the scripts. For convenience we link to a GNU Octave docker image available at https://github.com/gnu-octave/docker

C. Claims

The provided code is used to evaluate the proposed algorithm, Iris. More specifically the experiments performed help us attest:

C1 Iris's correctness: the algorithm converges to the target address.

C2 Iris's privacy guarantees: the algorithm is (α, δ)-*private*, *i.e.*, the privacy ratio against any queried node and colluding adversary does not drop below α, and the algorithm does not provide an advantage to the attacker's guess.

D. Execution

In the paper we perform five different experiments. The first two experiments presented in Figure 6 of the paper, focus on the evaluation of the performance cost that is introduced by Iris, *i.e.*, the extra hops that a query needs to perform. The third experiment presented in Figure 7, attests that Iris is an (α, δ)-*private* algorithm, whereas the last two experiments illustrated in Figure 8 and in Figure 9 examines the attacker's advantage when executing Iris.

1) Network Generation: The preliminary step for all the experiments is the creation of a number of different networks. These networks are generated with the `script_CreateIdSpaces.m` script.

The script creates 500 different networks of 1000 nodes each with 2^{23} number of addresses. These three parameters are hardcoded in the script but can be changed based on user's needs by changing lines 7, 11 and 12, respectively.

After executing this command 500 mat files, each one containing one initialised network, are created and saved under the folder `./experiments/networks/1000_nodes/`. After completing this preliminary step we can proceed with the execution of the experiments.

2) Experiments: For all the experiments we make use of the `script_ExecuteIris.m` script. This script allows us to run IRIS a specified number of times, specifying the α and δ parameters, and the fraction of colluding adversaries, across every set of experiments.

For every execution, the script uses a different network of 1000 nodes by loading a new network file from the `./experiments/networks/1000_nodes/` folder. For every execution the script selects the address of the target and the requester uniformly at random, checking that the requester is not among the colluding nodes.

The experiment parameters are embedded in the script thus for every new experiment a few lines needs to be changed to generate the required data for a particular experiment.

To reproduce our experiments and results, the following lines need to be changed:

- line 12 specifies the number of performed experiments of every set.
- lines 28, 29 and 34 specify the number of participating nodes in the network, the number of the id space addresses and the folder under which the mat file with the initialised network is saved, respectively.
- line 42 defines the α parameter.
- line 47 defines the δ parameter.
- line 53 defines the fraction of colluding nodes. (Recall that the colluding nodes are specified in the attackers variable in the mat file of the address space.)
- line 131 defines the name of the file to be saved with the experiments data.

To avoid an error-prone reproducibility of the performed experiments, we provide the parametrized `script_ExecuteIris.m` scripts that are to be used for every experiment. In the next section, we report their use together with further details regarding the execution of the experiments.

3) Graphs: Finally, to reproduce the graphs used in the paper, the data from the experiments can be plotted with the scripts found in the `./experiments/results/` folder. These are standard plots in either Matlab or R and we do not consider these part of our contribution, but we include them for completeness.

E. Evaluation

[Preparation]

All the experiments in the paper were executed using networks with 1000 nodes placed on an address space with 2^{23} addresses.

1) Experiment (E1): [Figure 6(a)] [1 human-minute + 1 compute-minute]: In this experiment we examine IRIS's performance for different values of the α parameter, to support our first claim.

[Execution]

1) Run the script `results\fig_DistancesPerAlpha\script_ExecuteIris.m`, this will execute IRIS setting α equal to 0.25, 0.35, 0.5 and 0.75, producing 4 csv and 4 mat files. For each value of the α parameter we execute 100 experiments. Apart from α the rest parameters remain stable, $\delta = 1/16 * address\_space$ and $f = 0$.

2) Run the script `results\fig_DistancesPerAlpha\script_ExecuteChord.m`, this will produce 1 csv file (data_a1.csv) that contains the results when executing Chord using for comparison reasons the targets and the requesters of one of the other mat files.

[Graph]

Run the `results\fig_DistancesPerAlpha\script_PlotDistancesPerAlpha.m` to plot the average distances to the target for each α value. The plot needs to be executed in the same folder with the data produced above.

2) Experiment (E2): [Figure 6(b)] [1 human-minute + 1 compute-minute]: This experiment is also related to our first claim examining IRIS convergence for different values of the δ parameter.

[Execution]

1) Run the script `results\fig_DistancesPerDelta\script_ExecuteIris.m`, this will execute IRIS setting δ equal to 1/4, 1/8, 1/16 and 1/32 of the address space, producing 4 csv and 4 mat files. For each value of the δ parameter we execute 100 experiments. Apart from δ the rest parameters remain stable, $\alpha = 0.35$ and $f = 0$.

2) Run the script `results\fig_DistancesPerDelta\script_ExecuteChord.m`, this will produce 1 csv file (data_a1.csv) that contains the results when executing Chord using for comparison reasons the targets and the requesters of one of the other mat files.

[Graph]

Run the `results\fig_DistancesPerDelta\script_PlotDistancesPerDelta.m` to plot the average distances to the target for each δ value. The plot needs to be executed in the same folder with the data produced above.

[Preparation]

For the experiments in Figures 7, 8 and 9 we alter IRIS so as to focus solely on the nodes that have a correct estimation regarding the target. Thus, we need to comment lines 27-31 and uncomment lines 35-40 in the `Iris.m` file. The next three experiments support our second major claim.

3) Experiment (E3): [Figure 7] [1 human-minute + 5 compute-minutes]:

[Execution]

1) Run the script `results\fig_PrivacyPerAttackers\script_ExecuteIris.m`, this will execute IRIS setting the f value equal to 0, 1/2, 1/3, 1/6 and 1/8, producing 5 mat files. For each f value we

execute 500 experiments. Apart from f the rest parameters remain stable, $\alpha = 0.25$ and $\delta = 1/4 * address\_space$.

2) Run the script `results\fig_PrivacyPerAttackers\script_FindMinPrivacyRatio.m`, the script loads the privacy data of the 500 experiments of each f value and finds the min privacy ratio of every experiment saving the data to 5 csv files.

[Graph]

Run the script `results\fig_PrivacyPerAttackers\script_PlotMinPrivacyRatioPerAttackers` to plot the minimum acquired privacy ratios as histograms.

4) Experiment (E4): [Figure 8] [1 human-minute + 1 compute-minute]:

[Execution]

1) Run the script `results\fig_Probabilities\fig_DistancesNormalizedByDelta\script_ExecuteIris.m`, this will execute IRIS 500 times with $\alpha = 0.75$, $\delta = 1/128 * address\_space$ and $f = 0$, producing 1 mat file.

[Graph]

Run the `results\fig_Probabilities\fig_DistancesNormalizedByDelta\script_PlotDistancesNormalizedByDelta.m` to plot the histogram of the results.

5) Experiment (E5): [Figure 9] [1 human-minute + 1 compute-minute]:

[Execution]

1) The script `results\fig_Probabilities\fig_DistancesNormalizedByDelta\script_PlotDistancesNormalizedByDelta.m` from the previous step, produces 2 csv files with the distances the queried node has to the target and to the randomly picked address. If we do not want the plotting but only to extract the two csv files that are necessary for the fifth experiment, we have to comment lines 28-45.

[Graphs]

To plot the probabilities we execute the scripts in the `results\fig_Probabilities\fig_ConditionalProbabilities` folder. Each script corresponds to one subfigure. We can alter the examined x value by changing line 9.