Adding Security and Privacy Guarantees in Structured Peer-to-Peer Networks

Angeliki Aktypi

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2023

To my parents, Evrydiki and Anastasios
whose love, faith and support are the springboards to the pursue of my dreams.

Acknowledgements

First and foremost I want to thank my supervisor, Prof. Kasper Rasmussen, for his guidance throughout my doctoral studies and the time he devoted to reading between the lines of my drafts. I would also like to thank Prof. Niki Trigoni and Prof. Ivan Martinovic for their valuable feedback during my Transfer and Confirmation and Prof. George Danezis for agreeing to stand as the external examiner of my thesis. I am grateful to the Engineering and Physical Sciences Research Council (EPSRC), the Oxford Center for Doctoral Training (CDT) in Cyber Security, British Telecom (BT), Linacre College, the CONCORDIA EU project, and the Oxford Department of Computer Science for their financial support that helped me travel to conferences, attend courses, and cover university fees and living expenses. I want to thank Prof. Paul Kearney, who supported my research proposal and gave me feedback on manuscripts and Prof. Sotiris Ioannidis for welcoming me to his team at ICS-FORTH during my time abroad. Special thanks to Prof. Jason Nurse for his helpful feedback and support during my first steps as a researcher and to Asclipias Roumbou for her helpful insights on my admission application and for proofreading my manuscripts. I couldn't have come this far without the support of my friends, Athina Siapera, Ameerah Anathalee, Oana-Maria Camburu, Munir Geden, Andikan Otung and Ilias Giechaskiel, who helped me see light at some dark hours of this journey. I want to thank also Kyriaki Marinou and Charalambos Antoniades for making Oxford feel home away from home, Antonios Bastas for highlighting the things that matter, my CDT cohort for making this not just another lonely DPhil, my OxSecure mates for sharing an altruistic dimension of our acquired knowledge, the RHB 303 companion Youqian Zhang and Lukas Helgas—the msc latex guru who helped me tame my protocol figures, for exploring Oxford's gastronomic heritage during breaks, and my research group for their helpful feedback on my presentations and our interesting paper discussions. I would also like to thank Prof. Kubra Kalkan and Prof. Nikos Vasilakis for joining me in exploring my research directions, Prof. Michael Goldsmith and Prof. Andrew Martin, for their advice during my first training year at the CDT program and the CDT administration team, David Hobbs, Maureen York and Janet Sadler, for always having their office door open and being keen to help. I left for the end my sincerest gratitude to my beloved parents, Evrydiki and Anastasios, who have always been by my side tirelessly; thank you for believing in me even when my faith was challenged and for being my role figures, teaching me the value of hard work and perseverance.

Abstract

Peer-to-Peer (P2P) networks are built in the application layer, forming a virtualised abstraction of the underlying infrastructure. In these networks, peers are self-organised in a logical structure where they communicate ad-hoc, acting as service consumers (clients) and service providers (servers). Several P2P networks have been proposed in the last few decades with purposes ranging from file sharing to instant messaging. However, despite P2P's positive features, such as scalability and robustness, the challenging provision of security and privacy guarantees burdens their real-world adoption as a general-purpose communication basis, on top of which different applications can be built and interact. This thesis tries to address this limitation.

We design SeCaS, a framework that deals with the problem of holistic discovery and secure sharing of the available device resources in a personal network. SeCaS, proposes a method to identify heterogeneous services compatible with a Distributed Hash Table (DHT) scheme. It also provides four protocols that guarantee message accountability and facilitate authorisation, which any structured P2P network can leverage.

Guaranteeing authentication in a decentralised setting is a challenging problem; we solve this by proposing Themis. This decentralised and secure transport layer can ease application development in any environment requiring point-to-point interaction. Themis presents a suite of two protocols that establish a notion of decentralised identity verification and a series of actions related to the communication and the management of nodes—*e.g.*, `store`, `find`, and `join`, forming a fully decentralised authentication solution. We underline the benefits that the adoption of Themis can bring by exemplifying its application as a secure service mesh communication network for use in data centres and companies that need dynamic load balancing and extensibility.

Acknowledging privacy concerns that come with an open-access platform, where many actors can participate and query for registered data, we define a new privacy notion that allows reasoning about the search privacy offered by a privacy-preserving mechanism in Chord, a popular DHT, even in the presence of a strong colluding adversary. We then propose Iris, a privacy-preserving object search algorithm, which allows nodes using the Chord protocol to use the network without allowing other peers (or external attackers) to track their activity or search patterns.

Overall, this DPhil thesis provides practical solutions that enable secure and private communication between entities organised in structured P2P networks to support their application-agnostic adoption in today's emerging technological areas, such as the Internet of Things (IoT) and Serverless Computing. In this way, it contributes towards an alternative to the centralised communication model that applications usually adopt, which is both secure and scalable.

Contents

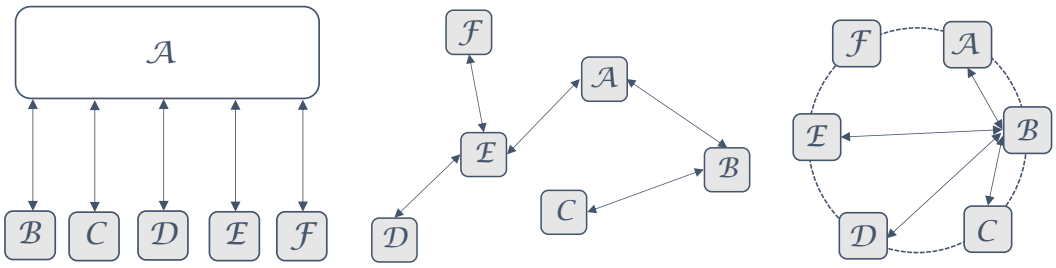
List of Figures

1 Introduction

Contents

Peer-to-peer (P2P) networks are built in the application layer and form a virtualised abstraction of the underlying infrastructure [1]. Contrary to the client-server communication model, which follows a centralised scheme where a central entity mediates between the communication of all nodes, in P2P networks, nodes communicate in an ad-hoc manner, with every node acting as a client but also as a server with the role of facilitating the communication among other nodes from the network. Several such networks have been proposed in the past and include both unstructured networks like Gnutella [2] and FastTrack [3], where nodes and, more precisely, the connections between them are not organised in a specific architecture, as well as structured networks like Chord [4] and Kademlia [5], in which peers are self-organised in a logical structure.

Figure 1.1 illustrates the network architectures the nodes form, when they follow a client-server, an unstructured and a structured communication model. To understand the differences between them lets inspect how a request from node B



(a) Client-Server (b) Unstructured P2P (c) Structured P2P

Figure 1.1: *Communication Models:* The networks the nodes form when following a centralised, and an unstructured and structured decentralised architecture.

to node F resolves between the three different communication schemes. In the centralised architecture depicted in Figure 1.1(a) all the requests are forwarded to server A, *i.e.*, the central entity that keeps the communication information such as the IP address of all the nodes in the network. Node B contacts server A that sends back the information of node F enabling node B to establish with node F a direct communication.

In P2P networks the nodes follow a decentralised communication, forwarding their requests to other nodes rather than to a central server in the network. The difference between unstructured and structured networks is that in the latter nodes are strictly placed on a virtual overlay that allows for a straightforward definition of the amount of distance between them. Leveraging this overlay, the requests are routed so as on every hop the distance to the receiver to become smaller. In unstructured P2P networks, nodes are not tied to a specific structure. The nodes form arbitrary connections to one another and they forward every request they receive randomly to other nodes in the network they know. To come back to our example, in an unstructured P2P network as depicted in Figure 1.1(b) node B initially forwards the request to node A and C, subsequently A contacts E, that eventually knows F and thus the requests resolves. On the other hand, if nodes were organised in a structured P2P network as shown in Figure 1.1(c), B would forward the request to node E, *i.e.*, the node that B knows that is closer to node F, that returns the communication information of node F. We elaborate more on how structured P2P networks work in Chapter 2.

Structured P2P networks offer significant advantages for applications involving numerous heterogeneous entities that need to discover and interact with one another to accomplish specific tasks. For instance, in a smart city scenario, traffic lights must communicate with road and car sensors to effectively manage traffic flow. In such applications, the ability to distribute and retrieve data in a scalable manner is crucial. Since nodes can join or leave the network, it is essential to be able to update the stored data rather than keep an entire history of all records.

Structured P2P networks provide a decentralised and scalable solution to these applications, allowing nodes to find the necessary information to communicate with peers, such as their IP addresses and port numbers, and to locate stored data and attribute tags within the network. These networks offer a decentralised key-value store interface, distributing the data among participating nodes. A deterministic algorithm defines which node is responsible for a specific key. At the same time, a routing scheme enables nodes to reach other nodes in the network without requiring a global view of the entire system within a limited number of hops. In structured P2P networks, the absence of a central entity mediating all connections eliminates bandwidth bottlenecks and single points of failure. Additionally, the resource discovery mechanism they provide, which operates with logarithmic complexity, can support many connections.

Beyond their scalability, these networks accommodates participating nodes' dynamic joining and leaving. This is a common case in dynamic environments such as the Internet of Things (IoT), where nodes may go offline due to intermittent connections, and Serverless Computing, where various microservices can be created or shut down in response to changes in application load. The different protocols they incorporate are designed to manage such network fluctuations; for example, the *leave* protocol informs other nodes of a node's departure while imposing minimal communication costs on the network. However, even without proper announcements of departures, it will take a few iterations for the network to stabilise again, whereas the replication mechanisms they support can prevent the loss of stored data.

Structured P2P networks have been leveraged by a variety of applications, ranging from file sharing, *e.g.*, BitTorrent [6], to instant messaging, *e.g.*, Peerchat [7]. They have also been used as a communication basis to ease the development of applications that demand decentralisation and scalability [8–11]. Our main objective is to add fundamental security and privacy properties in structured P2P networks that will be inherited by any application that uses them and act as the building blocks for further application-specific guarantees to be built on top. Adding security and privacy guarantees in structured P2P networks—not initially a design objective, is challenging due to their decentralised nature [12, 13]. In a centralised communication architecture, its design, despite the single point of failure, the privacy concerns and the dependency on a single communication link, provides a unique entity with which each node needs to exchange information. In a decentralised network, every node needs to exchange information with other nodes from the network, some of which can be malicious. Thus, in P2P networks, compared to the client-server approach, the security problem multiplies; the nodes, rather than relying on a single entity, need to protect against several others.

We aim to build infrastructures that are robust against adversaries that have under their control a certain fraction of the participating nodes in the network, and are active in the network, being capable of eavesdropping the communication links between nodes and to insert, delete and modify messages. We want to provide strong guarantees that allow nodes to determine with whom they are communicating and to provide attribution of the messages they receive. We assume no trust between the communicating entities, considering that the adversary can try to perform an attack at any point in time regardless of her previous behaviour. Allowing nodes to determine who sent what provides the means to detect any malicious action, leaving to the application the choice to decide the action that will be invoked when this is detected, *e.g.*, removing the node from the network. To provide the desired security properties, we rely on cryptographic primitives such as cryptographic hash functions, signature schemes and message authentication codes (MAC).

We want to provide *authentication* both to the peers and the available services to allow nodes to define what resource/service they make available and to which other peer in the network. Authentication is an essential feature for a general-purpose P2P overlay because, in this case, contrary to the case of the P2P overlays that were dedicated to one application, the peers joining the overlay may want to share a specific service only with some of the participants in the network. We also want to provide *integrity* to the exchanged messages to hold peers accountable for their behaviour.

Strong authentication and integrity allow us to maintain nodes accountable for their actions, at the same time, however, can allow tracking of the activity of nodes in the network. We need to provide *confidentiality* in the communication between two peers to prevent other peers from learning the specific messages they exchange. Still, confidentiality does not prevent the nodes that participate in the routing of a query from learning what is searched for. A way to address this issue can be by applying anonymity techniques that try to conceal the identity of the nodes, *i.e.*, breaking authentication. Our goal, however, is to provide solutions that can be employed independently but can also work collaboratively. Thus, to keep authentication, we focus on query privacy techniques that give the nodes control over keeping their activities hidden or known to exclusive peers in the network while they are authenticated.

1.1 Contributions

This thesis starts by introducing a general background and terminology on P2P overlays in Chapter 2. Then in Chapter 3, it proposes SeCaS, a Secure Capability Sharing framework that can leverage any structured P2P network to enable a secure collaborative operation between IoT devices. SeCaS brings: (i) a capability representation, which permits each device in a structured P2P network to specify what services they offer and can be used as a common language to search for and exchange services, and (ii) a set of four protocols that provides message authenticity in network device communication and allows a private collaboration among them.

SeCaS deals with the problems of service identification and message integrity/confidentiality but does not solve the problem of node authentication. Relying on a root of trust, *e.g.*, a certificate authority, to provide peer authentication can be considered a limitation of an open, general-purpose and secure transport layer infrastructure. Themis presented in Chapter 4, comes to address this problem by introducing a general framework for secure peer-to-peer communication built on a notion of decentralised identity management that allows any node to verify the identity of a communication partner without the need for centralised certificate authority. Themis offers confidential and authenticated communication between any two endpoints in a way that allows applications to inherit these properties and build more complex security solutions on top. The unique characteristics of Themis make it flexible enough to be used in any decentralised application, such as a secure service mesh application in Serverless Computing.

Themis brings us closer to applying structured P2P networks as a general-purpose communication basis to ease application development in open networks. However, providing a complete decentralised authentication mechanism in networks where everyone can join is a fundamental problem but only one of many. The fact that the requests from nodes are resolved by asking other nodes from the network, while beneficial, as there are no single points of failure, allows tracking the activity of peers in the network. Iris introduced in Chapter 5 comes to address this privacy concern. Chapter 5 initially introduces a new privacy notion, (α, δ)-*privacy*, that allows the evaluation of the privacy guarantees provided in Chord, a widely used structured P2P network, even against a strong colluding adversary, and then proposes Iris, a mechanism that permits nodes participating in the Chord network to perform queries without revealing their query target to the intermediate nodes that take part in the routing.

Overall, this thesis contributes towards the application of structured P2P networks as a general-purpose platform, on top of which different applications can be deployed by addressing open (Chapter 6) fundamental security and privacy challenges that such a venture imposes (Chapter 7).

1.2 Publications

The work presented in this thesis is structured around three peer-reviewed conference publications:

- Angeliki Aktypi, Kasper Rasmussen. *Iris: Dynamic Privacy Preserving Search in Authenticated Chord Peer-to-Peer Networks.* In 32nd Annual Network and Distributed System Security Symposium (NDSS '25). February, 2025. [14]

- Angeliki Aktypi, Dimitris Karnikis, Nikos Vasilakis, Kasper Rasmussen. *Themis: A Secure and Decentralized Framework for Microservice Interaction in Serverless Computing.* In 17th International Conference on Availability, Reliability and Security (ARES '22). ACM. August, 2022.[15]

- Angeliki Aktypi, Kubra Kalkan, Kasper Rasmussen. *SeCaS: Secure Capability Sharing Framework for IoT Devices in a Structured P2P Network.* In 10th Conference on Data and Application Security and Privacy (CODASPY '20). ACM. March, 2020. [16]

During my doctoral studies at Oxford, I published also the following research:

- Angeliki Aktypi, Jason R.C. Nurse, Michael Goldsmith. *Unwinding Ariadne's Identity Thread: Privacy Risks with Fitness Trackers and Online Social Networks.* In 1st International Workshop on Multimedia Privacy and Security in conjunction with the 24th ACM Conference on Computer and Communication Security (MPS CCS '17). ACM. October, 2017. [17]

This work studies privacy risks in decentralised environments, in particular, the potential exposure of user identity that is caused by correlating information that users share online and personal data that are stored by their fitness-trackers, yet it focuses more on the human related aspect rather than the technical details of this exposure. For this reason, it is not discussed further in this thesis.

1.3 Statement of Authorship

I confirm that the work presented in this thesis has been performed solely by myself under the supervision of my supervisor Prof. Kasper Rasmussen, except where explicitly stated. In particular, the work presented in Chapter 3, *i.e.*, the SeCaS framework, was done in collaboration with Prof. Kubra Kalkan. While I defined the capabilities and designed the protocols, the security analysis of the protocols was done in cooperation with Kubra. Themis, described in Chapter 4, was done in collaboration with Mr Dimitris Karnikis and Prof. Nikos Vasilakis. While the main core idea of the protocols and their security analysis was done by myself, the implementation of the use case example was done in conjunction with Dimitris and Nikos, using one of their prior works for Themis's deployment, the Atlas infrastructure.

2
Background

Contents

In this chapter, we introduce the reader to peer-to-peer (P2P) networks, presenting the different categories that exist and their key differences. We further provide background information on the distributed hash table (DHT) concept, the general scheme followed by nodes in structured P2P networks to publish and lookup for data objects. SeCaS introduced in Chapter 3 relies on this general abstraction, making it compatible with any structured P2P network. Themis, presented in Chapter 4, our proposed secure by-design structured P2P also follows a DHT scheme. We finally present background on Chord, one of the most common structured P2P in the literature. Explaining in detail the fundamentals of its operation is necessary to understand Iris proposed in Chapter 5, our proposed algorithm that allows privacy-enhanced queries in Chord. We model how Chord works, aiming to provide the backbone on which Iris builds upon and enhances.

2.1 P2P Networks

P2P systems build and maintain overlay networks at the application layer, assuming the presence of an underlying network which assures connectivity among nodes [18]. They provide a distributed communication environment where nodes, referred to as peers, can interact with each other by providing and, at the same time, consuming resources/services offered by other nodes, without the mediation of a central server. Their main features include: self-organisation, robustness and scalability. The characteristics mentioned above offer an absence of bottlenecks and a single point of failures, rendering P2P networks an alternative to the traditional client/server communication paradigm.

Compared to other distributed communication networks, they present some fundamental differences. P2P overlays focus on integrating simple entities (e.g., personal computers, connected fridges) typically encountered in the Internet of Things (IoT) ecosystem. These entities present neither abundance nor constraints regarding their resources (i.e., energy, memory, computation) [19]. Hence, concerning node limitations, they are considered somewhere between Grid computing and wireless sensor networks (WSNs) or mobile ad-hoc networks (MANETs). The former focuses more on powerful and diverse nodes [20] whereas in the latter, energy is considered a scarce resource for the nodes; thus, they focus on minimizing power consumption [21].

Concerning the assumed underlying communication infrastructure, in P2P networks, the links are not considered to be unreliable or have bandwidth restrictions. These problems are encountered in delay tolerant networks (DTNs), which focus on coupling with unpredictable end-to-end delays resulting from intermittent connectivity [18]. Even if the communication links are considered to be reliable, in P2P overlays nodes can reach each other without a direct link between them. This is usually the case in Mesh networks.

Hence, free from node constraints and link restrictions, the most basic concern of P2P networks is the effective and secure content dissemination and retrieval among a set of nodes of significant size. Based on the resource discovery technique

they follow to achieve this objective, they can be classified into two main groups, the *structured* and *unstructured* ones. In *structured* P2P (e.g., Chord [4], CAN [22], Pastry [23], Tapestry [24], Kademlia [5]) the application-specific data is placed at specific positions according to a globally followed rule. The mapping between contents/files and their location is typically provided by a DHT scheme that associates given keys to specific peers. We further analyse the operation of the DHT scheme in the following subsection. In *unstructured* P2P (e.g., eDonkey [25], Freenet [26], Gnutella [2]), there is no predefined rule that imposes the position where contents have to be placed. The absence of such a structure forces nodes to adopt approaches like flooding, random walks, or expanding-ring search to discover the data items. Comparing the two different classes, *structured* P2P provide guaranteed search results efficiently. However, this is carried out with a higher deployment cost than the *unstructured* ones. Hence, the former can be considered as a preferable option for scaled environments, whereas the latter can be more appropriate for more restricted area networks.

Based on the degree of centralisation in the way peers are organised across the networks, they can be taxonomised into four different groups [27–29]. In *centralised* P2P (e.g., Napster [30], SETI@home[31]), there is an indexed centralised server that manages files and maintains a database of its users. In *decentralised* P2P (e.g., Chord [4], Kademlia [5]) each node acts as an index server by searching and storing resources locally and as a router by relaying queries between peers. Finally, *hybrid* P2P networks (e.g. GAB [32], HybridFlood [33]) try to conjugate both centralised and decentralised architectures by benefiting from the efficiency of each combining technique, while overcoming their inherited drawbacks. A special subgroup of such networks is the *super-to-peer* P2P networks (e.g., FastTrack[3], Gnutella2 [34]) where peers are organised in different clusters and a few peers in the overlay, namely super-peers act as the clusters' heads, by acting on behalf of the peers assigned to them. Regardless of the way peers are organised, they can communicate directly with each other without any intermediate entity.

2.2 DHT Scheme

In structured P2P networks, a node can efficiently query and retrieve application-specific data across a broad set of peers, since the nodes are organised in a tightly controlled graph structure. The delivery of messages follows a key-based routing that allows data localization by exchanging messages typically with a lower bound of $\mathcal{O}(\log \mathrm{n})$, where n is the number of nodes in the network. In particular, both the nodes and the data are assigned unique identifiers (i.e., $\mathrm{Node_{ID}}$ and $\mathrm{Object_{ID}}$, respectively) from a large ID space. The attribution of these identifiers is done either randomly or by applying a collision-resistant hash function (e.g., SHA, MD5) on a resource property (e.g., the MAC address of a node, the name of a file), generating an m-bit sequence.

A defined function maps each object to a different live node called the object's responsible or root node. Commonly, as a responsible node is chosen the one whose $\mathrm{Node_{ID}}$ is the closest to the data's identifier, in the ID space. The responsible node is in charge of maintaining a record of values, which are location information concerning the nodes on which the data can be found. The nodes achieve that by implementing a Distributed Hash Table (DHT) decentralised storage/retrieval system. DHT provides the same functionality as a traditional hash table by storing the correlation between an object and a value using a store (i.e., store(object, data)) and lookup (i.e., value = lookup(object)) method.

To join the network, a node must find an active peer called the bootstrapping node, which will permit the successful discovery of the other peers in the network. The network routes messages with a given $\mathrm{Object_{ID}}$ to its responsible node by following a forward mechanism that leads progressively closer to the data's identifier. To route messages efficiently, all nodes maintain a routing table consisting of the $\mathrm{Node_{ID}}$ and the communication address (e.g., IP) of several other nodes. For fault tolerance, replication mechanisms exist to map more than one responsible node to each object. Also, stabilization techniques are executed periodically in the background to identify peers that leave the network. These techniques hand over

the objects for which the exiting peers were responsible to other nodes, preventing the partitioning of the network.

Different protocols that construct structured P2P networks show differences mainly in the organisation and the mapping schema of the ID space, the state that each node maintains, and the distance metric they use. They can also vary on the routing style they apply when forwarding messages. In *iterative* routing style, the next-hop node is returned to the lookup initiator, who sends all the requests, whereas in *recursive* routing, the queried intermediate node forwards the request to the next node until the responsible node is reached that sends the response to the initiator [35].

2.3 Chord Protocol

Chord is one of the pioneering DHTs and has been extensively studied in the literature. It has also been implemented in various proposed applications, including the Seeks distributed web search engine [36], the Cooperative File System (CFS) [37], UsenetDHT [38], and OverCite [39]. Although Chord has not reached the same level of deployment as the Kademlia DHT, it is associated with significant infrastructures. For example, in Tor, CFS has been proposed as a means to enable efficient decentralised retrieval of introductory points for onion services [40] . More recently, infrastructures like the NKN blockchain-based network [41] have chosen to incorporate Chord into their architecture due to the verifiable path it provides between the source and the query target. In Chord, queries move in a single direction, and messages are forwarded based on routing tables constructed deterministically according to the node's position and anchor points on the address space. Chord has been acknowledged for its simplicity and performance [42]. Its straightforward design makes it a suitable choice for initial studies on the security guarantees of new proposals, whereas, its similarities with other DHT protocols offer a foundation for expanding new techniques in their design.

Chord offers a decentralised 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.

Algorithm 1 Chord's Store Algorithm

```
1: function STORE(RT_h, O_k, Data)
2:     N_n ←SELECTCLOSESTNODE(RT_h, O_k)
3:     repeat
4:         N_n' = N_n
5:         N_n ←LOOKUP(N_n', O_k)
6:     until N_n == N_n'
7:     return PUSH(N_n, O_k, Data)
```

2.3.1 Modelling Chord

In this section we model how Chord works. We describe only protocols that need to be executed to allow specific data to be stored in the network and retrieved by a requester; these are going to serve later in Chapter 5 as the backbone for Iris, the privacy-preserving algorithm we propose. 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:

Algorithm 2 Chord's Retrieve Algorithm

```
1: function RETRIEVE(RT_r, O_k)
2:     N_n ←SELECTCLOSESTNODE(RT_r, O_k)
3:     repeat
4:         N_n' = N_n
5:         N_n ←LOOKUP(N_n', O_k)
6:     until N_n == N_n'
7:     return FETCH(N_n, O_k)
```

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 centralised 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.

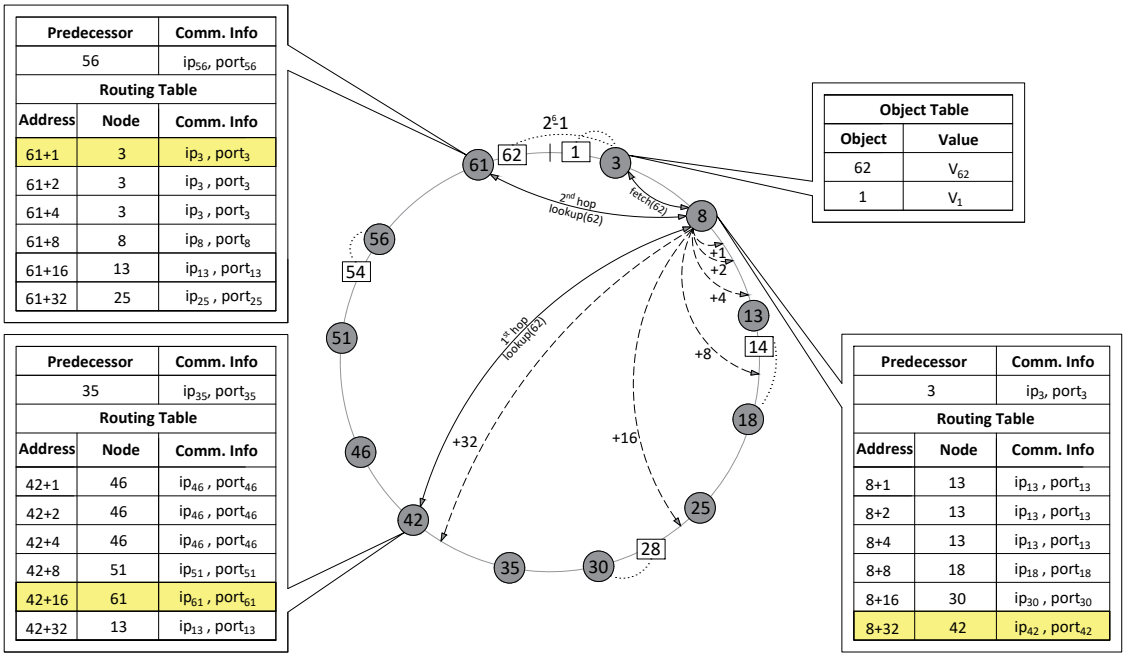


Figure 2.1: *An example of the* `retrieve` *algorithm in Chord network, executed by node* 8 *to retrieve the data associated with object* 62.

In Figure 2.1, 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.

3

SeCaS

A Secure Capability Sharing Framework

Contents

This chapter presents SeCaS, our initial approach to enhancing the security guarantees when adopting structured P2P networks. In SeCaS, we study how to enable structured P2P networks within ubiquitous environments, such as the Internet of Things (IoT), to facilitate secure collaboration among various devices with different services. We have developed a naming scheme, referred to as *capabilities*, which describes the services offered by each device. This scheme simplifies the registration and discovery of device services within a structured P2P network. We have also designed four protocols that can leverage any structured P2P network, providing critical security features for peer communication, including confidentiality and access control.

3.1 Introduction

The achievements of new technological advances and the appealing services they provide result in a proliferation of Internet of Things devices in our life. These smart embedded systems are increasingly ubiquitous and notoriously heterogeneous allowing seamless integration between the physical and the cyber world. The possession of a large variety of such smart appliances by the same owner forms the owner's IoT ecosystem, a personal network where each connected device provides specific services for the user. This ecosystem can enable the available smart devices to unite their strengths and overcome individual weaknesses (e.g., the lack of storage in one device) or provide more advanced functionalities (e.g., smart cameras to add with elderly care). To realise such potential and permit the owners to make the most out of the infrastructure they possess, a secure discovery and access control mechanism that allows device collaboration must be in place [43].

Typically, the mechanisms, which allow devices to collaborate, follow a centralised operation, where a central managerial entity (e.g., a search engine [44], a broker [45], a central hub [46]) controls the information flow and orchestrates the interactions of devices, mediating among their communication [47]. Regardless of their well-established application, the dependency of these centralized mechanisms on the availability of the connection link between the central administrative point and each

device makes them vulnerable to single-point failures as when the connection link is down, collaboration among devices cannot be achieved. From a privacy perspective, relying on an external infrastructure that aggregates and controls crucial personal information on users raises significant threats [48]; if the central server is breached the whole system is compromised, whereas users do not control the way their data are handled or by whom they are processed [49]. Also, the fact that the same entity handles all the requests can result in network bottlenecks making this solution not inherently scalable, an essential property of IoT infrastructures that have to accommodate a significant and continuously increasing number of connected devices.

These well-known issues [50–52] motivate the use of decentralised communication models to be used to allow for all of these endowed components to interact with each other by discovering the resources that are available in the network. In this chapter, we study the use of structured peer-to-peer (P2P) overlays in such a use-case. In structured P2P overlays, nodes communicate with each other in a decentralised manner without being obliged to communicate with a hub or a cloud backend, making the system more reliable and robust. The load is uniformly distributed among the network devices themselves; thus all the data are handled locally mitigating personal information leaks. The DHT routing scheme that is followed by the peers provides scalability as queries can be resolved with logarithmic complexity in the size of the network, without creating bandwidth bottlenecks or requiring the nodes to be within distance to directly communicate with each other.

However, employing structured P2P networks to enable IoT device collaboration is not a trivial task [12, 13]. First of all, the exact match query that DHT performs necessitates devices to initiate several communication sessions that are proportional to the resources they search, even if they can be closely related (e.g., they can refer to the possibility to occupy storage but of different space amount). The design of an appropriate expression format that is both (i) compatible with the exact query discovery that DHT performs and (ii) provides discovery flexibility by allowing nodes to search for the service they want with as little communication cost as possible, is needed.

Enabling device collaboration in a structured P2P manner is also a challenging task from a security point of view. Firstly, the structured P2P networks do not have an administrative entity that can invigilate peers' communication. Also, IoT topologies are highly dynamic due to the mobility of devices and their constrained resources that can lead them to fall in sleep-mode regularly. However, to accomplish such a task it is fundamental to guarantee that only authorised nodes can access the network, that only proper members of the network can access the services of other nodes and that any malicious activity or configuration error can be detected. The heterogeneity of IoT devices concerning their resources further creates the need for the proposed techniques to apply to both powerful and lightweight devices in order to provide scalability.

Our contributions are three-fold:

- We introduce *capabilities*, *i.e.*, a flexible way of representing device services, which enables collaboration among peers that are organised in a structured P2P network.
- We design *SeCaS* framework that consists of four protocols that achieve an authenticated bootstrapping, an authorized access to the *capabilities* of nodes and render peers accountable for the messages they exchange; thus allowing them to collaborate securely by discovering and exchanging services.
- We provide a complete security analysis with respect to our threat model for the four protocols and we analyse the complexity of our proposal based on the computational, memory and communication overhead that is introduced to the peers. We prove that our framework can achieve both secure collaboration and good performance for large deployed connected environments.

3.2 Applicability

SeCas protocols are based on a general key-value store interface that enables network nodes to store key-value pairs, retrieve values associated with specific keys, and

dynamically join or leave the network. While SeCaS protocols are agnostic to the implementation of the key-value store interface, in our design, we assume a general DHT scheme to allow for decentralised service discovery.

The DHT scheme efficiently balances the load among participating nodes. This distribution is independent of the number of participants in the network. As a result, when a node joins or leaves the network, only a few nodes (neighbouring nodes) are affected, eliminating the need to rehash the entire structure, as would be required in a traditional hash table. By employing a DHT scheme, peers can address their requests to other participating nodes instead of relying on a specialised directory (such as a centralised hub) or sending requests to all registered nodes (i.e., flooding). This scheme allows for efficient resolution of requests within a limited number of hops, scaling well as the number of participating nodes increases.

In our design, we assume a general DHT abstraction (API) that is followed by any structured P2P network (e.g., CAN [22], Chord [4], Pastry [23], Kademlia [5]). This API can be provided either as a library [53] or as a service [54], and it defines the following three primary methods:

- **Join**($\mathbf{ID_{NC}}, \mathbf{Node_{BS}}$): This method provides the possibility a new node NC to join the network with the help of a bootstrapping node BS. BS, which already takes part in the overlay, assigns a $Node_{ID}$ to NC, which is used as its identifier in the DHT. BS creates this $Node_{ID}$ according to an identifying property (ID) of the newcomer (e.g., its IP address). Consequently, with the help of BS and other peers in the network, the NC initialises its routing table, $Table_{ID}$.

- **Store**($\mathbf{Object_{ID}}, \mathbf{Node_{ID}}$): This method associates the unique identifier of a peer, $Node_{ID}$ with a specific Object, $Object_{ID}$ in the DHT, and stores it at the responsible node for this specified Object.

- **Lookup**($\mathbf{Object_{ID}}$): This method returns when available the location information (i.e., a list with all the nodes' identifiers, $\{List_{ID}\}$) that is associated

with the indicated $Object_{ID}$, which is stored in its responsible node; or an empty list.

For concreteness reasons and without loss of generality we use Chord [4] to indicate examples and underline the performance analysis of our proposal.

3.3 Capabilities

The purpose of our framework is to enable devices to work collaboratively in a secure context, by exchanging services. In this section, we elaborate on *capabilities*, our hierarchical service representation that allows denoting all the different services that nodes can contribute to the network. We start by underlining the objectives that need to be achieved by the proposed structure and then we present our proposal and its advantages.

3.3.1 Prerequisites

P2P networks have been widely used in different applications that were primarily dedicated to file-sharing by distributing network bandwidth [55]. However, throughout the years they have also enabled the simultaneous exchange of other resources such as processing power [56] and disk storage space [57]. Nowadays, IoT ecosystem presents a significant heterogeneity regarding the abilities of the devices that constitute it. To enable their exchange a mechanism that provides an easy way to represent them must be in place. In particular, this representation must provide a coherent registration and management of the different things that each device can do. It needs to be comprehensive and human-readable and easily extended to adapt any newly introduced device benefits.

To be able to use the DHT and take advantage of the accurate discovery of objects that it provides, the representation mechanism must be able to generate objects by hashing. Furthermore, it needs to allow searching the network in an advanced way, for example, by executing queries that provide the possibility to locate similar objects that fall under the same group. An easy way to achieve that

is by using range queries, where the value of the given attribute is between an upper and lower bound. However, in the DHT, the object to be searched is specified by its unique identifier [58], allowing for only exact-match queries. The challenge that arises is to enable the check of a range by using one and not multiple exact-match queries, as this redundancy increases the communication cost for the nodes.

3.3.2 Representation mechanism

In our study, we define *capabilities* as all the different ways in which devices can contribute services in the network [59]. The services can emanate from any resource that the device has, a software (e.g., communication interfaces), a hardware (e.g., memory) or a transducer (e.g., sensor, actuator). For example, an "Amazon Echo" device can access the web through its WiFi interface for a device that only has Bluetooth connectivity, can save data for a peer that is running out of memory or it can produce an alarm sound for a sensor device that does not have a speaker. In our definition, we use the term *capabilities* differently than it is typically used in the context of authorisation. In SeCaS, *capabilities* refer to each device's operations. The Resource Reservation Protocol, described in Section 3.5.2, utilises a token as evidence of the provider's agreement to provide a specific *capability*, as defined by our definition. This token operates on a similar principle to how capabilities are used in authorisation.

Capabilities, our representation mechanism, are described in plain-text following a hierarchical model depicted in Backus-Naur form [60] expression illustrated below. The representation of each *capability* is the name of a service with any number of refinements, separated by dots. The *Service* is used as the group label of each *capability* and refers to the functionality that is provided by a device acting as a general description of the *capability*. The *Refinements* are used in order to give more details concerning the different *capabilities*. If we regard the service as the general group within which a functionality is categorized, we can consider the refinements as the concrete identifiers of the subcategories that constitute the group. Refinements specification walks through the different subgroups of an

initial category going from something more general to something more specific. Following that different *capabilities* have a different number of refinements. For example, `Communication.TCP.HTTPS.TLS12` *capability* corresponds to the existence of a specific communication interface and the refinement specifies the exact communication module that is provided.

$$Object = hash(Capability)$$
$$\text{where } \langle Capability\rangle ::= \langle Service\rangle \{\texttt{"."}\langle Refinement\rangle\}$$

The plain-text of the *capabilities*' representation is hashed by using a collision resistance function creating the *Objects*. For each *capability* a different number of objects can be generated depending on the specified Refinements that accompany the used Service. In particular, for a *capability* that has n refinements there will exist $n+1$ associated objects inserted in the network. After the Objects have been obtained by hashing, a device can invoke the lookup operation by specifying any of the specific objects that is associated with the *capability* it is looking for.

$$Capability = Service.Ref_1 \dots Ref_n \Rightarrow Objects = \{Object_0, \dots, Object_n\},$$

$$\begin{aligned} \text{where } Object_0 &= hash(Service),\\ Object_1 &= hash(Service.Ref_1),\\ &\vdots\\ Object_n &= hash(Service.Ref_1 \dots Ref_n). \end{aligned}$$

Our proposal provides a convenient way for manufacturers to define new *capabilities* by introducing new Services or extending the Refinement list for the already created ones. It can also preserve compatibility with the way that *capabilities* are represented in already implemented protocols for automation control in smart-home environments such as Samsung's SmartThings [61], Google's Weave [62] and Apple's HomeKit [46]. In particular, the different information that they specify for each *capability* in their description (e.g., commands, attributes, status), can be included as a different refinement in our proposal.

This representation mechanism introduces a lightweight way to generate Objects that incorporate all the necessary information that needs to be defined so that a

specific *capability* can be located efficiently into a broader set; thus compatibility with the DHT is provided, and the execution of exact-match queries is possible.

The hierarchical representation offers a retrieval flexibility during the search operations. If the precise name of a *capability* is not known (i.e., compatibility issues stem from non-unified naming of *capabilities* by different vendors) or when the invoked lookup operation of a sought-after *capability* returns an empty list, nodes can choose to search for a more generalised *capability* taking advantage of the hierarchy's peel-off property. As a result, we can achieve a reduction of the misleading indicated unavailability cases when not the specified but similar objects exist in the network. For example, if a node searches for `Thermostat.TemperatureState` but this service is stored as `Thermostat.TemperatureMeasurement` by another peer, the requester can peel off the refinement and search only for `Thermostat`. Similarly, `MemoryProvision.Level2` provides a *capability* of offering storage to another device to save data and the refinement indicates the available space, depicted in different levels that correspond to different storage magnitudes. If the communicated node does not have `Level2` amount of free space, then it can define the level of free space that it has available, for example `Level1`. The above-mentioned described lookup search achieves performing general/range queries by using only one exact-match query; thus, there is no need to perform multiple exact-queries to examine a specific range that increases the bandwidth usage and the communication overhead for the nodes.

The creation of multiple hashed values proportional to the number of defined refinements for each *capability* renders more fault tolerance to the system. In particular, each *capability* has associated with it multiple objects. Hence, the disappearance from the network of a *capability* demands the failure of multiple nodes. Thus, the possibility of unintentional deficient lookup queries is restricted. The proposed representation in conjunction with the Fulfilment protocol presented in Section 3.5.4 achieves the exchange of any *capability* that can be encountered in an IoT ecosystem, despite their heterogeneity. In particular, both *capabilities* that demand one communication session (e.g., the download of a software update) or more

(e.g., the storage and retrieval of data from the memory of a device) can be satisfied. To carry and transmit all the necessary information that needs to be specified for each *capability* separately, we harness a *Data* structure. Each time this structure is decomposed into different components (e.g., commands, attributes, methods) and the associated semantics and values vary based on the defined *capability* and the kind of message (e.g., sending or responding) for which it is used.

3.4 System and Adversary Model

In this section, we introduce the system model, on which we base our work, and we specify the adversarial model against which our framework is secure.

3.4.1 System Model

SeCaS addresses an IoT infrastructure that consists of a group of connected devices, each equipped with a set of different *capabilities*, all owned by the same ownership domain, e.g., a person, an organisation; this can be an individual's home network, a hospital with various units, a factory with different divisions, or a university campus that extends across a city with multiple departments. Such an ecosystem encompasses a range of devices, from lightweight to powerful ones. However, SeCaS does not focus on devices with significant resource constraints, such as those found in Wireless Sensor Networks (WSNs). It assumes that the participating nodes can perform cryptographic operations, such as generating cryptographic signatures.

SeCaS runs on top of the transport layer (e.g., TCP, UDP), assuming that nodes can reach each other and that a reliable method exists for delivering messages exchanged between them. We consider a dynamic topology where nodes can join and leave the network at any time. To join the network, new nodes broadcast their presence to discover a bootstrapping node.

Once they join, the nodes follow a key-value scheme by using the *capabilities* representation as specified in Section 3.3 to advertise the services they offer and to discover the nodes that provide the services they search. In our design, we assume that nodes follow a DHT scheme to enable decentralised discovery and

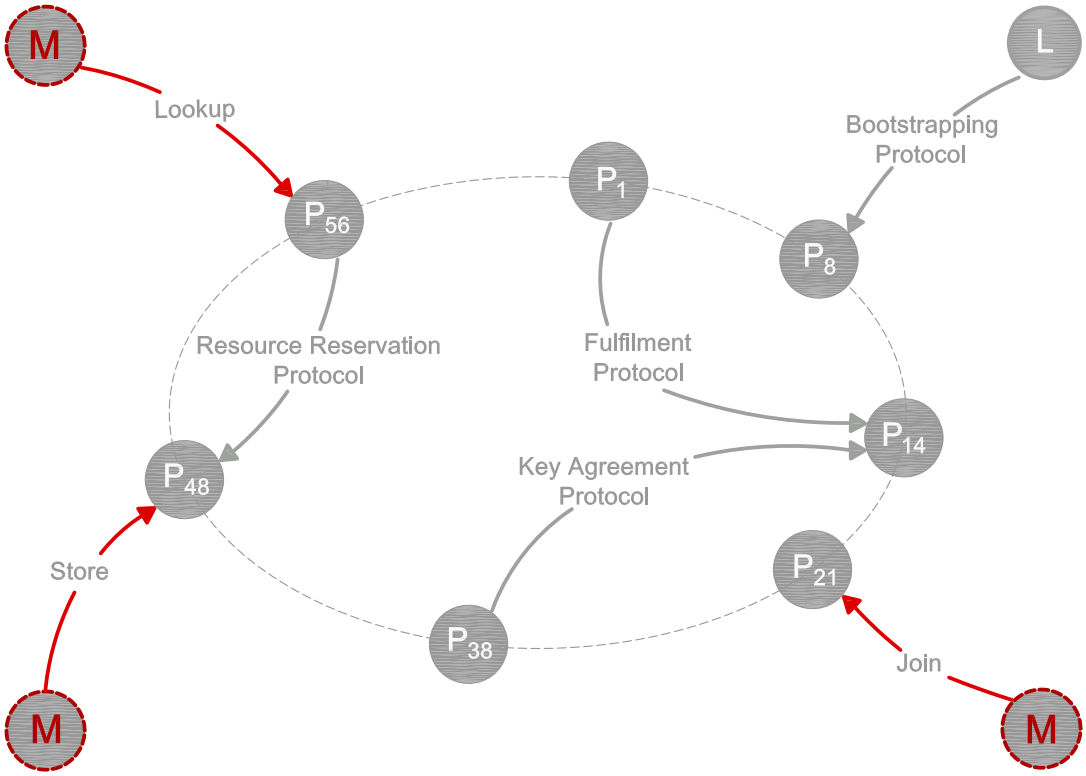


Figure 3.1: *The system and adversary model.* Nodes are organised in a structured P2P network, legitimate peers that are already a member in the network are depicted with the letter "P", whereas the legitimate newcomer is denoted with the letter "L"; malicious nodes are shown with the letter "M". Red arrows demonstrate the different attacks that we consider during the communications.

message routing. The generality of the API on which SeCaS protocols rely allows for any structured P2P network (e.g., Chord, Kademlia) to be used. Without loss of generality, we choose Chord in our system model depicted in Figure 3.1. SeCaS protocols, as defined in Section 3.5, enable devices to collaborate securely.

3.4.2 Threat Model

We consider a Dolev-Yao attacker [63] who has complete control over the communication channel. This means the attacker can eavesdrop on, manipulate, replay, and intercept all communications, as well as non-deterministically inject messages into the network. For our analysis, we exclude attacks such as denial-of-service (DoS) and jamming. In terms of compromised nodes, the SeCaS protocols guarantee strong accountability for the messages exchanged between nodes. This means that

if a node acts maliciously, there will be cryptographic proof of its actions, which can aid in identifying and removing it from the system through an out-of-band process. The goals of the attacker, depicted in red in Figure 3.1, are:

1. to join the network and obtain the rights of an authenticated node, i.e., to gain permission to use a service registered in the network, making it unavailable to a legitimate node

2. to manipulate the DHT operations, i.e., initiating fake store operations or responding erroneously to lookup requests

3. to break the confidentiality and the integrity of the communication among peers that collaborate

Our proposal is built on top of a simple DHT API, assuming no security guarantees to be provided by its functions (i.e., Join, Store and Lookup); as long as those functions are provided, the security guarantees of SeCaS remain intact regardless of the specific type of the underlying peer-to-peer network. This flexibility makes SeCaS applicable in various scenarios. To ensure membership control, SeCaS requires an authentication mechanism and a method to validate the identities of legitimate nodes. In our proposal, we assume an authentication scheme in which the network owner possesses a public-private key pair, certifying out-of-band the association between the identity and the corresponding public key of legitimate peers. Alternatively, more decentralised authentication schemas that do not require certificates can be utilised, such as self-certifying identifiers, as done in Themis (discussed in Chapter 4). In this case, the authentication scheme must be combined with a mechanism to validate dynamically the nodes that take part in the network.

3.5 Framework Protocols Description

In this section, we present the four protocols that constitute our proposed communication framework. The protocols allow devices to execute an authenticated bootstrapping at any structured P2P network and to check the authenticity of the

messages they received from the DHT. Peers that follow the protocols by using the *capabilities* representation we introduce in Section 3.3 can collaborate in a secure manner by reserving and granting access to others peers *capabilities*. For all the protocols it is true that if any of the verification steps fails, the protocol terminates with an error.

3.5.1 Bootstrapping Protocol

One device with C number of *capabilities*, henceforth referred to as Alice, that has already obtained her certificate (i.e., she is a legitimate node) and wants to join the overlay, follows the Bootstrapping protocol, depicted in Figure 3.2. After picking a random nonce N_A, Alice initiates the communication by broadcasting her certificate, the freshly picked nonce N_A and her signature on the wireless channel, waiting to hear back responses from neighbouring nodes. Upon receiving the broadcast message, one of the devices already a member of the overlay, henceforth referred to as Bob, authenticates Alice by examining the received certificate and checking the integrity of the nonce based on the provided signature. Subsequently, Bob picks a nonce N_B and replies to Alice by providing his certificate, his node unique identifier in the DHT and the freshly picked nonce N_B that he signs together with the nonce that was indicated in Alice's initiated message. Alice authenticates Bob based on his certificate and she inspects the integrity and freshness of his response based on the provided signature and the included nonces. She responds back to Bob, by signing the nonce that he picked together with his node identifier. Bob after receiving Alice's signature knows that he really talks to Alice, thus, he is safeguarded from the unnecessary execution of the *Join* operation that is invoked immediately afterwards. Thereafter the two devices execute together the Join operation provided by the DHT, which returns the unique identifier that is assigned to Alice and her initialised routing table. Finally, Bob signs Alice's identifier together with the nonce that she chose. After receiving Bob's signature, Alice accepts the node identifier that she was assigned $Node_A$, as valid and she then invokes the *Store* operation in order to update the records of each responsible node of the C *capabilities* that she can

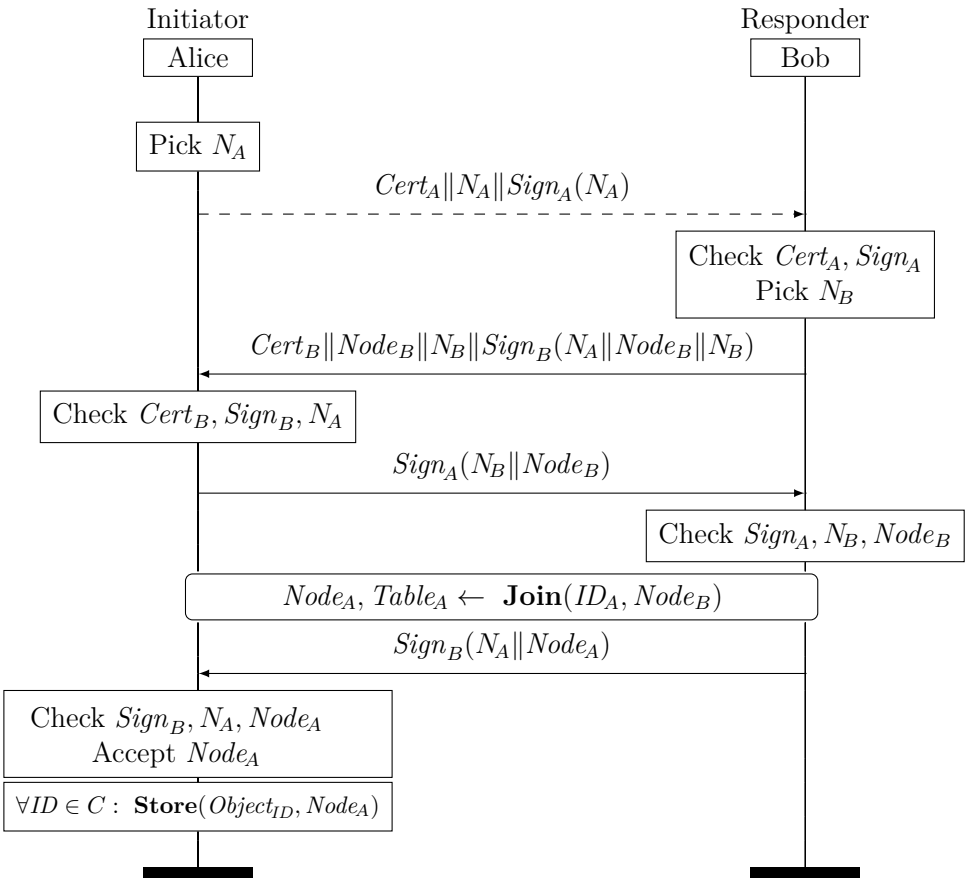


Figure 3.2: *Bootstrapping protocol.* A node that wants to be registered in the network initiates a broadcast message waiting a response from a SeCaS peer. Nodes obtain their certificates out-of-band.

contribute to the network. At the end of the protocol, a legitimate and alive node Alice has successfully joined the network and the network is both acquainted of her presence and of the *capabilities* that she can provide.

3.5.2 Resource Reservation Protocol

When a node, referred to as Alice searches for a *capability* in the overlay, she initiates the Resource Reservation protocol, illustrated as Figure 3.3. At first, Alice invokes the *Lookup* operation of the DHT specifying the unique identifier, $Object_{ID}$ of the *capability* that she is looking for. The invoked operation returns a list with all the peers, which have denoted that can provide the requested service. However, as the DHT provides no security, the obtained list is considered to be potentially

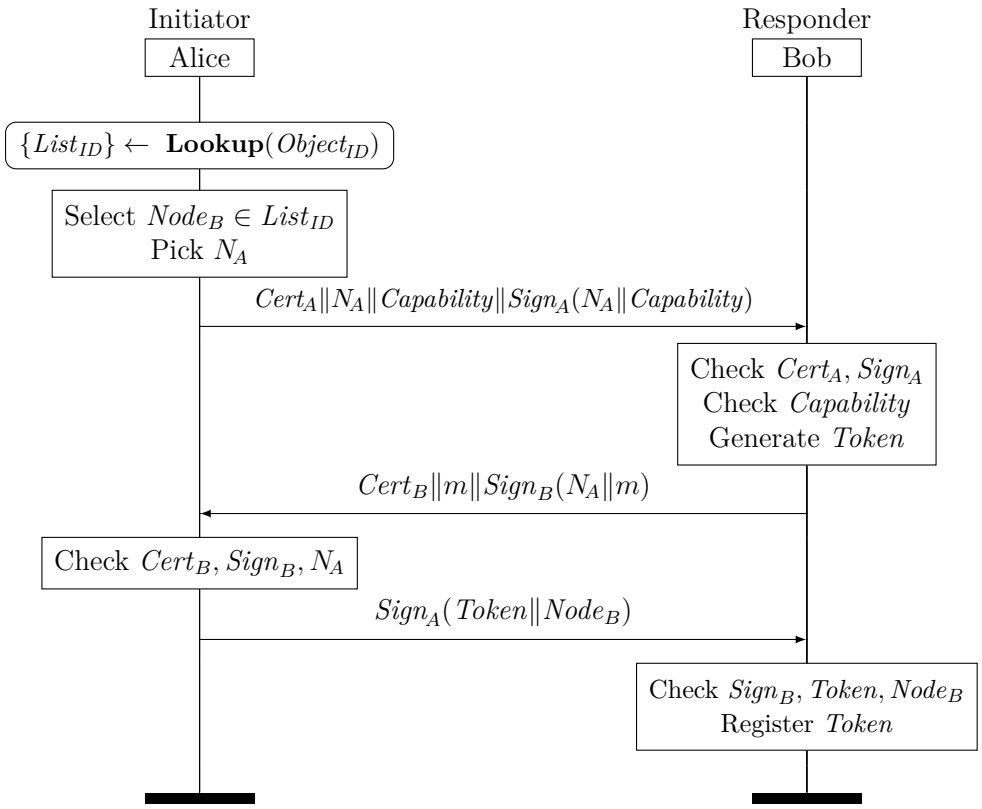


Figure 3.3: *Resource Reservation protocol.* The initiator who needs a *capability* invokes the lookup operation from the DHT specifying its $Object_{ID}$. Then she selects arbitrarily one of the available peers from the return list and verifies if he has the sought-after service. If the service is available, she reserves it.

tampered. After obtaining the list, Alice randomly selects a peer named Bob to send her request. This method improves fairness among nodes, preventing any single node from becoming a repeated bottleneck in service provision. Alice sends Bob a signed *capability* request message that contains her certificate, a freshly picked nonce, N_A and the queried *capability*, $Capability$. Bob checks the authenticity of the message by examining the received certificate and signature and inspects if he can provide the requested service. If this holds, he generates a $Token$ (i.e., a random number) that he associates with Alice's $Node_{ID}$ and the sought-after $Capability$. Subsequently, he replies to Alice's request by providing his certificate and a message m, that includes an acknowledgement and the freshly generated token, $m = Ack||Token$. He also signs message m together with the nonce, N_A that was chosen by Alice. When Alice receives Bob response, she checks its authenticity

by inspecting the included certificate and the provided signature and assures that it corresponds to her initial request based on the returned nonce. She then signs the provided token together with Bob's node identifier and sends it to him. Finally, after checking the received signature Bob registers the token as issued. In case Bob cannot provide the requested service, he replies to Alice's request by providing his certificate and a message m, that includes a negative acknowledgement, $m = Nack$. He also signs message m together with the nonce, N_A that was chosen by Alice, and the protocol terminates. If the sought-after *capability* is not currently available in the network (e.g., Bob was the only one who could provide it), Alice can take advantage of the peel-off property of the proposed *capability* representation to look for other *capabilities* that are related to the sought after one. At the end of this protocol, the requester node Alice reserves the *capability* from another peer Bob who has this *capability* in the network. The nodes that are looking for a specific *capability* after executing this protocol will know if such *capability* is really registered in the network and they will have also identify the node that is able and committed to providing it. Also, the contacted peer will be aware of the interest of another node for its specific service and will have reserved it for as long as the provided token is in effect (e.g., for a time interval T).

3.5.3 Key Agreement Protocol

Two peers jointly agree on a secret key following the Key Agreement protocol depicted as Figure 3.4. The protocol is based on the Diffie-Hellman algorithm. The predefined base g is a primitive root in the group of integers modulo p. Alice, who initiates the protocol, chooses a secret integer a and picks a random nonce N_A. She then sends to Bob her certificate, $g^a \bmod p$ and the freshly picked nonce N_A. In her message, she also includes her signature for the $g^a \bmod p$, N_A and Bob's identifier. Bob upon receiving the message checks the correspondence of the indicated certificate and the provided signature and also inspects the node identifier of the message's receiver assuring that this message is addressed to him. If the check is successful, Bob chooses, in turn, a secret integer b and constructs the shared secret

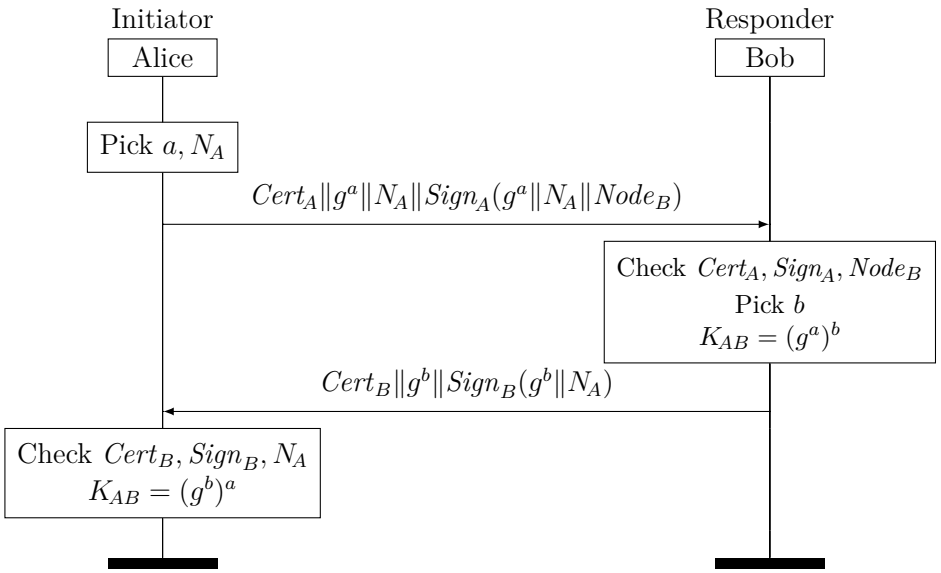


Figure 3.4: *Key Agreement protocol.* Two peers that want to establish a shared secret key follow an authenticated Diffie-Hellman key algorithm.

key $K_{AB} = (g^a)^b \bmod p$. He sends back to Alice his certificate and $g^b \bmod p$ that he signs together with the nonce N_A that she picked at the beginning. As soon as Alice receives Bob's response, she checks the correspondence of the indicated certificate with the provided signature and the returned nonce, and she then calculates the shared secret key $K_{AB} = (g^b)^a \bmod p$. If the protocol terminates, it guarantees that the secret key is only known to Alice and Bob. The resulting secret will be used in subsequent communication between the devices, enabling them to authenticate each other and exchange *capabilities* over a confidential secret channel. The protocol takes into consideration the storage consumption at each node and the dynamic character of the network. By following it, nodes can establish keys with any of the peers registered in the system if they choose to do so; thus the number of keys that each node has to store in its memory is not in direct proportion to the network size.

3.5.4 Fulfilment Protocol

When a node wants to make use of a peer's commitment to the provision of a *capability*, it initiates the Fulfilment protocol delineated in Figure 3.5. Primarily, the initiator, Alice sends the possessed token that has been obtained from the

execution of the Resource Reservation protocol, a data structure that forms her request and her node identifier. The message is sent encrypted together with its MAC, by using the secret key that is established from the completion of the Key Agreement protocol between Alice and the responder, Bob. The responder upon receiving the node's request retrieves their shared secret key, checks its integrity based on the provided MAC and then decrypts it. Subsequently, he ensures that she did not create the message in the past by inspecting the sender's included node identifier. Then, he examines if the specified token is still valid, if it is associated with the initiator's $Node_{ID}$ and if it is in accordance with the *capability* $Object_{ID}$ to which the defined data structure refers. If the checking is successful, he provides the specified *capability*. Hence, he responds back by encrypting the initially provided token, a data structure that contains his respond, his node identifier and a new generated token in case of a long lasting *capability*. This token will be used in a subsequent execution of the Fulfilment protocol to refer back to a *capability* that is still under sharing. Finally, the responder includes the MAC of his encrypted response that will then be used from the initiator for checking the integrity of the replied message. The protocol provides access control to the nodes' *capabilities*. The responder is able to control which device is going to be the beneficiary of the service provided with the help of the token. The initiator at the end of the protocol's execution will have taken advantage of the other peer's *capability* and will have an acknowledgement for the provided service. All the communication is encrypted providing confidentiality to the communication between the collaborative parties.

3.6 Security Analysis

Based on the threat model that we describe in Section 3.4.2 we now evaluate the security of the protocols of our framework by discussing the likelihood of an adversary breaking the security guarantees of each protocol to achieve his attacking goals. We will base our analysis on the following four main assumptions:

1. The signature scheme used by the protocol participants and the CA is secure, i.e., it is not possible to forge a signature without access to the private key.

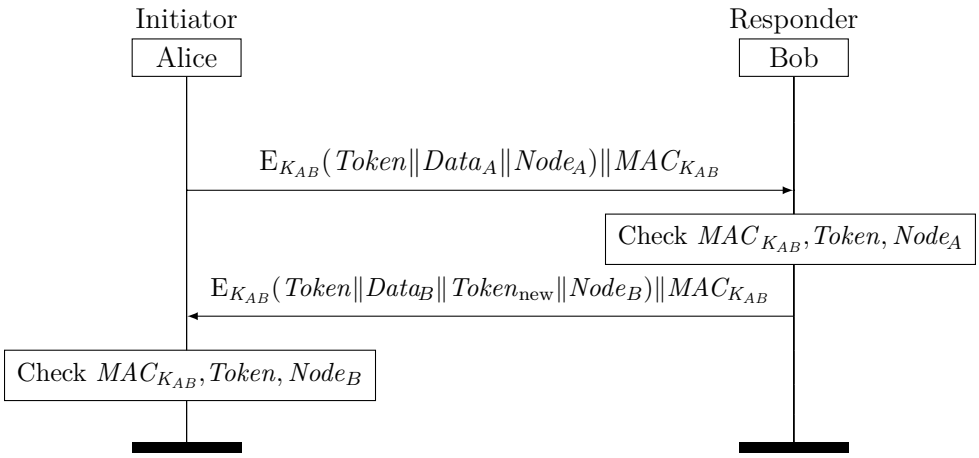


Figure 3.5: *Fulfilment protocol.* The responder grants access to the initiator to one of his services based on the indicated token. The nodes are authenticated based on the secret key they share.

2. The same nonce is only picked twice with negligible probability.

3. The same token is only picked twice by the same device with negligible probability.

4. Each device registers only one token for any given *capability* each time it is requested, along with the $Node_{ID}$ of the node to whom it was issued. The tokens are maintained for a time T (enough for nodes to execute the protocols in Figures 3.4 and 3.5). After this time the tokens are discarded.

3.6.1 Bootstrapping Protocol

The Bootstrapping protocol, Figure 3.2 provides two guarantees, one for Alice and one for Bob.

Guarantee 3.1. *If the protocol completes successfully, Alice is assigned a $Node_{ID}$ sent by Bob (a valid node), and she correctly joins the overlay network.*

Proof. For an adversary to break this guarantee and falsely convince Alice that she has been assigned a valid $Node_{ID}$, the adversary would have to successfully send Message 4, as this is the only way for Alice to get the $Node_{ID}$ that is signed by Bob. The adversary only has two options to send Message 4. He can either

craft the message or replay a previously captured message. To craft Message 4 the adversary has to produce a signature over the message content that matches a certificate signed by the CA. This means he must either forge the signature or obtain the private keys. The adversary cannot forge the signature by assumption 1, so assuming the private keys are kept private, the adversary cannot craft the message. For replay to work, the adversary has to make sure that the content of Message 4 expected by Alice, corresponds to one of the messages available to her. Because the nonce N_A selected by Alice is part of the signature of Message 4, this means that the adversary either has to force Alice to choose a nonce that corresponds to one of his captured messages, or predict the choice and force Bob to construct a valid message ahead of time. The adversary cannot influence Alice's choice of N_A and by assumption 2 previous nonces (picked by Alice or other nodes) will only be useful with negligible probability. This means he would have to predict the choice and make Bob generate Message 4 ahead of time. However, Bob will only send Message 4 (containing N_A) in response to a signed response (Message 3), and we prove subsequently that the adversary cannot forge such responses. □

Guarantee 3.2. *For Bob, this protocol guarantees that the join request is fresh and originated from Alice (a valid node).*

Proof. For an adversary to break this guarantee and falsely convince Bob that he is Alice, the adversary would have to send Message 3 successfully. The adversary has two options: he can either create or replay Message 3. To create Message 3, the adversary has to produce a signature over the message content, which is not possible according to assumption 1. To replay Message 3, the adversary has to control Bob's choice of N_B, or predict it and force Alice to generate Message 3. However, these options are not viable for similar reasons to those explained above related to the replay of Message 4. □

3.6.2 Resource Reservation Protocol

The Resource Reservation protocol, Figure 3.3 has two guarantees, one for Alice and one for Bob.

Guarantee 3.3. *At the end of the protocol, Alice obtains a token which proves that she has reserved a resource from Bob (a valid node).*

Proof. For an adversary to break this guarantee and falsely convince Alice that she has received a valid token, the adversary has to send Message 2 successfully. The adversary only has two options to send Message 2. He can either create the message or replay a previously captured message. To create the message, the adversary has to produce a signature over the message, which is not possible according to assumption 1. To replay, the adversary has to control Alice's choice of N_A, or predict it and force Bob to generate Message 2, nevertheless for similar reasons to the ones explained in Section 3.6.1 for the Bootstrapping Protocol, this cannot be achieved. □

Guarantee 3.4. *For Bob, the protocol guarantees that the request is fresh and really from Alice, i.e., that it has not been changed or submitted by someone else.*

Proof. For an adversary to make Bob register a token as "issued", he must confirm the token with Message 3 (in Figure 3.3). The adversary only has two options to send Message 3. He can either craft the message or replay a previously captured message. To craft Message 3 the adversary has to produce a signature over the message which is not possible according to assumption 1. For replay to work, the adversary has to control the choice of the token, or predict it and force Alice to generate Message 3. With the help of assumption 3 and similar reasons of the replay of messages as above, these options are not viable. □

3.6.3 Key Agreement Protocol

The Key Agreement protocol, Figure 3.4 provides a guarantee that holds for both Alice and Bob.

Guarantee 3.5. *If the protocol is completed successfully, a secret key is created which is only known by Alice and Bob.*

Proof. For an adversary to break this guarantee he has two options: he can create the secret key by using the information obtained from Message 1 and Message 2, or he can control the choice of a or b. To obtain the key, he gets the information g^a from Message 1 and g^b from Message 2. To craft the key from this information he needs to solve the discrete logarithm problem which is computationally infeasible. Assuming the adversary cannot influence Alice's choice of a or Bob's choice of b the adversary cannot create the shared key between Alice and Bob. □

3.6.4 Fulfilment Protocol

The Fulfilment protocol, Figure 3.5 provides two guarantees, one for Alice and one for Bob.

Guarantee 3.6. *If the protocol completes, Alice will successfully access one of Bob's resources.*

Proof. In order for an adversary to break this guarantee, the adversary would have to successfully send Message 2 to take advantage of the provided *capability*. The adversary can either craft the message, or replay a previously captured message. To craft Message 2 the adversary has to encrypt the message content; thus he has to obtain the secret key K_{AB} that is shared between Alice and Bob. However, as we proved in the Key Agreement Protocol, the adversary cannot obtain the secret key, K_{AB}. To replay the message, the adversary needs to make the content of the message acceptable by Alice. According to assumption 3, the same token cannot be applicable twice for the same device. Thus, Alice will understand from the token number that it is a replay message. However, there can be a case that Alice has reserved in the past a *capability* for Bob with the same token and they have also executed the fulfilment protocol, the adversary will be able to capture the encryption of the provided token and perform a replay attack for Message 1

(reflection attack). However, $Node_B$ in the message will reveal that this message is not coming from Bob. □

Guarantee 3.7. *For Bob, the protocol guarantees that he shares with Alice the resource for which he earlier provided a token.*

Proof. Alice cannot cheat and take advantage of another resource other than what she reserved based on the assumption 4. For an adversary to trick Bob to share his *capability* with him will have to send Message 1 successfully. The adversary can either craft or replay the message. To craft Message 1 the adversary has to encrypt the message content; thus he has to obtain the secret key K_{AB} which we proved that it is not possible in the previous Section 3.6.3. To replay the message, the adversary needs to make the content of the message acceptable by Alice. According to assumption 3, the token number will reveal that it is a replay message. However, there can be a similar case explained above, which can cause a reflection attack. However, $Node_A$ in Message 1 casts out this threat. □

3.7 Complexity Analysis

In this section, we provide a complexity analysis of the SeCaS framework that runs on top of a structured P2P network. To avoid duplicating existing work that implements DHT [58, 64, 65] in IoT environments, we focus on the performance delta (i.e., computational overhead, storage consumption, communication cost) that nodes need to sustain when they participate in DHT and when they follow our proposal. In our analysis, we use as a benchmark the Chord [4] protocol, for concreteness and consistency reasons. However, the referred performance is followed by the majority of the structured P2P networks.

SeCaS design incorporates several key choices to reduce the complexity of participating nodes. In the Bootstrapping protocol, which is executed only once by each node to join the network, the joining message is broadcasted. This allows

	Computational Overhead		Storage Consumption		Communication Cost	
Protocol	Alice	Bob	Chord	Introduced	Chord	Introduced
Bootstrapping	$2s + 3v$	$2s + 3v$	$\mathcal{O}(m) + \mathcal{O}(r)$	$4 \cdot k_a$	$\mathcal{O}\Big((\log n)^2\Big) + \mathcal{O}(\log n)$	3
Resource Reservation	$2s + 2v$	$1s + 3v$	–	$(g + m + t) \cdot n_t$	$\mathcal{O}(\log n)$	n_t
Key Agreement	$1s + 2v + 2e$	$1s + 2v + 2e$	–	$(m + k_s) \cdot (n_t + n_s - (n_t \cap n_s))$	–	$n_t + n_s - (n_t \cap n_s)$
Fulfilment	0	0	–	$f(Data)$	–	2

Table 3.1: Complexity overview of the computational overhead, the storage consumption and the communication cost that nodes need to sustain when following the Chord P2P protocol and the ones introduced on top of them by SeCaS Framework.

devices with more limited resources to opt out of serving as bootstrapping nodes for new devices. To guarantee freshness and authenticity, the Bootstrapping and the Resource Reservation protocols, as shown in Section 3.7.1, use four and three signatures, respectively, making them slower than other off-the-shelf protocols such as TLS or QUIC. However, these two protocols provide a different functionality compared to TLS or QUIC, which is why they can not be used. While TLS/QUIC establish a secure, authenticated channel between two interacting nodes for later communication, the Bootstrappingng protocol provides the guarantees that an alive, legitimate node can join the network correctly. The Resource Reservation allows legitimate nodes to check the availability of a *capability* and reserve it. SeCaS establishes secure channels when nodes collaborate—using the established secure channel to exchange multiple *capabilities*, rather than when they interact. By doing so, it eliminates the need to establish encrypted channels between nodes that will not collaborate but will only interact for routing purposes. Regarding collaboration among nodes, each node decides how many *capabilities* it will make available within the system.

3.7.1 Computational Overhead

Recent studies have shown that it is feasible to apply public key cryptography to sensor networks by using the right selection of algorithms and associated parameters, optimization, and low power techniques [66]. Following that, we base our analysis on the number of demanding cryptographic operations (i.e., exponentiations, signature generations and verifications) that nodes need to perform.

Apart from the hashing operation, which does not impose a significant computational burden to the nodes, Chord and in general the DHT does not demand any other cryptographic operation. Hence, based on our assumption the computational overhead for the nodes in Chord is zero.

SeCaS framework provides the security guarantees that we present in Section 3.6, by using different cryptographic operations. Starting with the Bootstrapping protocol, both Alice and Bob have to perform two signature generations over the

messages they initiate. Also, both of them have to verify signatures three times to authenticate their communicating party based on their provided certificate or to check the authenticity of the received messages. Hence, the total computational cost of the Bootstrapping protocol is four signature generations and six signature verifications. Analogously, we compute the total introduced overhead for the Resource Reservation protocol and the Key Agreement protocol, which follows a Diffie-Hellman key exchange algorithm. The Fulfilment protocol demands only symmetric encryption-decryption and message authentication code (MAC) verifications; thus following our initial assumption, its computational cost is zero. Table 3.1 summarises our results.

3.7.2 Storage Consumption

To avoid a linear search of the network [67], nodes in Chord maintain a routing table (denoted as finger table) containing at maximum m entries, where m is the number of bits of the identifiers. A finger table entry includes the finger interval and both the Chord identifier and the communication address of the relevant node; thus for storing their finger table nodes are burdened to reserve $\mathcal{O}(m)$ space in their memory.

In the DHT, nodes are responsible for storing application-specific data related to a number of different objects that are registered in the network. In Chord, assuming h registered objects in a network with n number of nodes, each node is responsible on average for $r = h/n$ objects. Each new entry causes the table and also the space that it occupies to increase linearly and in direct proportion to the number of inputs. Hence, the nodes are burdened to reserve $\mathcal{O}(r)$ space in their memory for the data that they are responsible for.

The hierarchical representation that SeCaS introduces for representing *capabilities* associates multiple objects with each *capability* based on the number of refinements that it has. Despite introducing a higher load to the total network, the load balancing feature that is inherent in DHTs such as in Chord promotes the uniform distribution of the extra objects among all the nodes in the network. By

having all the nodes contributing evenly in the maintenance of the network, we reduce the collateral damage of the failure of each node in the total network operation.

In addition to these two tables, the certificate-based authentication mechanism which we adopt for SeCaS schema here requires each participant node to store a few further properties: the public key of the owner (PK_{CA}), a pair of public-private keys (PKi, SKi) and its validated identity in the form of a certificate ($Cert_i = (Sign_{CA}(ID_i, PK_i), ID_i, PK_i)$). The amount of storage that these properties occupy on each node's memory depends on the selected cryptographic asymmetric algorithm that is used for generating key pairs and signing messages, and the selected key-length. For example, in RSA the signature size depends on the key size, the RSA signature size is equal to the length of the modulus in bytes. This means that for a k_a-*bit* key, the resulting signature will be exactly k_a-*bit* long (e.g., 2048-bit key length will result in a signature of 256 bytes). Taking the bit length of the chosen key k_a as constant, storing the four preliminary values requires $4 \cdot k_a$ bits of memory space on each node.

Further to these values, in the proposed framework each node needs to store two more tables. In the first table, each node has to maintain the tokens that provide authorised access to its *capabilities*. Each entry in this table has to indicate the string value of the *capability*, the unique identifier of the node that has reserved it and the related token. Assuming a g-*bit* string, an m-*bit* unique identifier for the DHT and a t-*bit* token in use, the amount of memory that this table requires is $(g + m + t) \cdot n_t$ bits, where n_t is the number of nodes that each node serves.

The second table, has to save the shared secret symmetric keys which the node establishes with other peers from the network. Each entry in this table will include the unique identifier of the cooperative node and the established symmetric key. The total amount of memory that this table demands depends on the bit length of the established symmetric key (e.g., 256-bit for AES algorithm [68]) and it grows linearly whenever a new key is established. Nodes share a secret key with all the nodes with which they collaborate. Considering the establishment of a k_s-*bit* symmetric key

between the nodes, the storage of this table reserves $(m+k_s)\cdot(n_t+n_s-(n_t\cap n_s))$ bits on the memory of each node, where n_s is the number of nodes that serves this node.

Nodes are further burdened to use space in their memory depending on the *Data* structure of the *capability* that is exchanged each time. The Fulfilment protocol, Figure 3.5 enables *capability* exchange by making use of this structure, which carries and transmits all the necessary information (e.g., commands, attributes, methods) for each *capability*.

3.7.3 Communication Cost

To maintain a correct mapping in the DHT when a node n_i joins the network, the responsibility of the identifier space is going to be updated, so as the newcoming node to be assigned certain identifiers previously assigned to other peers. This initiates a message exchange procedure in the network to update the routing invariants of the affected nodes. In Chord, such a join procedure demands $\mathcal{O}((\log n)^2)$ messages, in order to make sure certain identifiers previously assigned to n's successor to become assigned to n.

To locate both resources and peers across the network, nodes in the DHT use their routing table. The communication complexity achieves a balanced trade-off between the cost of maintaining a full directory and the network delay of storing one neighbour per node. The level of routing complexity in a stable n-node Chord overlay requires $\mathcal{O}(\log n)$ hops. Hence, both the store and lookup operations demand $\mathcal{O}(\log n)$ number of messages if recursive routing [69] is applied or $\mathcal{O}(2\log n)$ in case of iterative routing [70].

In SeCaS, both the Bootstrapping protocol (Figure 3.2) and the Fulfilment protocol (Figure 3.5) of our scheme, add a constant number of three and two messages, respectively, to the nodes that are communicating in the structured P2P network. The number of executions of the Resource Reservation protocol (Figure 3.3) and the Key Agreement protocol (Figure 3.4), as already explained in the previous Section 3.7.2, depends on the number of nodes that the node serves

and the number of the nodes that serve it; thus the introduced communication cost from these protocols will be of n_t and $n_t + n_s - (n_t \cap n_s)$ messages, respectively.

In SeCaS, nodes' liveliness is inspected by using cryptographic nonces and not other freshness mechanisms (e.g., timestamps); thus no time synchronisation is demanded among the different devices in the network. The communication overhead that is generated in order the devices to acquire the necessary properties (e.g., their certificate) for being authenticated is not a burden that our proposal imposes but is linked with the authentication schema that SeCaS uses; thus we do not include it in our calculations.

3.8 Summary

In this chapter, we introduce SeCaS, a framework that enables efficient and secure *capability* sharing between IoT devices. Our framework takes advantage of the self-organised and decentralised communication provided by a structured P2P network and is built to utilise the search *capabilities* such networks already possess. SeCaS ensures a number of security properties including only letting authorised nodes join the network and search for *capabilities*; message freshness; authentication of nodes and messages; and accountability to aid in troubleshooting in case of configuration errors or malicious behaviours. The security properties hold regardless of the exact type of underlying P2P network, on the only condition that three functionalities – *Join*, *Store*, and *Lookup* – are provided. We do not make any security assumptions about these functions, nor does it matter how they are implemented. The adversary is considered to have full control over the data returned by these functions. Despite this strong adversary model, we provide a complete decentralised secure *capability* discovery and resource sharing network solution. This is accomplished by first introducing a flexible way to represent device services that we call *capabilities*. *Capabilities* act as a common language to allow nodes to exchange, and search for, services without the need for a central hub. We design four protocols that utilise the underlying P2P network as a transport layer and together implement the framework. These protocols guarantee that only authorised nodes can access the

network and only proper members of the network can access the *capabilities* of other nodes. We provide a complete security analysis for all four protocols, with respect to our threat model. We additionally analyse the computational overhead, as well as the memory and communication complexity of our proposal showing that our framework scales well and does not impose a considerable cost to the participating nodes. Both our security and complexity analyses prove SeCaS to be a scalable architecture that provides fault-tolerance and addresses privacy concerns, suggesting it as a feasible alternative to the established IoT collaboration approaches that are either centralised or use flooding mechanisms.

4

Themis

An Authentication Framework for P2P Interaction

Contents

This chapter presents Themis. Themis brings us closer to an open communication basis based on structured P2P networks, eliminating the preliminary steps nodes must perform to join a network. SeCaS presented in the previous chapter,

guarantees important security properties in the communication between nodes but does provide authentication. THEMIS presented in this chapter fills this gap by proposing a decentralised authentication scheme. Any application that demands point-to-point interaction can adopt THEMIS. Here, to demonstrate the problems that THEMIS can solve, we present how we can use THEMIS to build a service mesh application utilised in Serverless Computing environments.

4.1 Introduction

Serverless Computing [71–75] is a recent approach to Cloud Computing that simplifies the development and use of cloud resources. Serverless applications are comprised of stand-alone microservices that interact with each other in a peer-to-peer (P2P) fashion: each microservice is responsible for only a small fraction of the application functionality, and can thus be invoked, be passed parameters, and scale-up independently. A *service mesh* [76, 77] is a dedicated infrastructure layer that allows communication among microservices, which can belong to different deployment clusters of container platforms, *e.g.*, Kubernetes [78].

Existing service meshes enable microservice communication by adopting a centralised architecture—microservices can authenticate and discover each other by communicating with central registries. Relying on central registries to administrate the microservice communication provides a clear and straightforward administration. However, it makes it challenging to support open cloud platforms creating vendor lock-in issues. Allowing microservices to communicate in an ad-hoc manner can facilitate the incorporation of machines/services that belong to different federated domains, whereas the lack of dependency on a central communication link can allow the deployment of serverless applications in dynamic edge, *e.g.*, IoT, and volatile, *e.g.*, disaster zones, environments [79].

This chapter presents THEMIS, a secure general-purpose abstraction overlay suitable to any application that demands point-to-point communication but particularly tailored to the service mesh infrastructures underpinning today's serverless environments. THEMIS is a framework built on a notion of decentralised identity

management to allow secure service-to-service interaction when the connection to a central registry might not always be available, addressing underlined design flaws in the security of state-of-art service meshes [80].

Themis's design achieves security, extensibility, and service discovery in microservice communication featuring a two-layer protocol architecture. Its lower-layer consists of two protocols that provide authenticity, confidentiality and integrity to the communicated messages. Its upper-layer builds on the previous protocol pairs to provide a series of P2P communication and identifier management actions. We prove the low-level protocols secure in a strong adversary model, and discuss the resulting security properties of the upper level protocols. We further describe how Themis can serve as a building platform for a service mesh application.

Our Themis prototype leverages the ubiquity of JavaScript powered runtime environments in serverless platforms, and is available to modern applications as an open-source library. We evaluate Themis on eight end-to-end serverless applications, as well as a number of microbenchmarks targeting standalone serverless operations. The key take-away from the evaluation is that the performance overhead of Themis (compared to an insecure baseline implementation) is minimal, while providing the security benefits described earlier. Our contributions can be summarised as follows:

- *Security Protocols*: We propose two novel protocols for key agreement and subsequent secure communication that leverage the self-certifying identities of the participating nodes. This way we do not depend on a centralised root of trust like a traditional certificate authority.

- *Model and Proofs*: We provide a detailed security analysis of the security guarantees of our protocols. Specifically, we prove that they achieve authentication, confidentiality and integrity for all the exchanged messages.

- *High-level Operations*: We specify five key operations, *i.e.*, `find`, `store`, `join`, `update` and `leave` that nodes execute to maintain a structured overlay organisation. Leveraging these building blocks Themis achieves service discovery and extensibility in a fully decentralised manner. We also elaborate on the security

properties that hold against several common classes of attacks that target this P2P structure.

- *Open-source Implementation*: We implement THEMIS in about 3.3K lines of JavaScript, as a pluggable application library called `Themis` and built atop QuickJS—a small and embeddable Javascript engine. The `Themis` library takes care of object initialization, communication, and serialization, leveraging a JavaScript port of the NaCl Networking and Cryptography library. We provide an open-source release of THEMIS's implementation at github.com/atlas-runtime/themis.

- *Empirical Evaluation*: We evaluate THEMIS's characteristics across eight serverless applications as well as targeting microbenchmarks scaling between 1–1,000 nodes. THEMIS's security benefits come at a small-to-imperceptible cost in terms of runtime performance and only modest cost in terms of lines of code changed.

4.2 Themis Architecture

This section elaborates on THEMIS's design and gives a high-level description of its architecture.

4.2.1 Design Goals

THEMIS is a secure P2P communication scheme, designed to suit the implementation of a large-scale, multi-cloud, and open service mesh, by achieving:

Security: The multi-tenant nature of a service mesh, *i.e.*, allowing multiple applications to share resources of the same machines simultaneously, demands fine-grained security guarantees. The security mechanism in place must allow for *confidentiality*, *i.e.*, that the data being transferred among two parties remains hidden, so as services coexisting on the same network cannot eavesdrop on others communication. To realise an open service mesh in which different providers can take part, the joining process must be relaxed from setup burdens such as communicating with a central authority registry. However, easing the joining procedure allows

both malicious and honest nodes to coexist on the same network. For this reason, the service mesh must provide strong *accountability* for the exchanged messages so as attribution of malicious behaviour to be performed. To achieve accountability, both *authentication*, *i.e.*, the parties exchanging information are who they claim to be and *integrity*, *i.e.*, the data being transferred has not been forged or tampered with, must be guaranteed. These can be the basic underneath blocks for other security primitives to be built on top, *e.g.*, authorisation.

Extensibility: Each service in a serverless application is created and killed independently, based on the usage demand. A service mesh needs to inherently provide *scalability* to serve the high replication of services. To avoid vendors lock-in issues and privacy concerns, service meshes must be *open*, *i.e.*, allowing instances to be added and removed flexibly and quickly, and be hosted on both commercial clouds and client in-house premises.

Service Discovery: Service instances must be able to discover each other based on the business logic they implement. Instances can implement different parts of the workflow of the same application or can provide *observability* functionalities to the serverless infrastructure by collecting metrics of the internal state of the system. Deploying the discovery mechanism in a centralised way limits the resilience of the serverless infrastructure against a region outage and restricts the nature of serverless applications, *e.g.*, disaster management. A decentralised service discovering mechanism must allow inherently *load balancing* among the available instances that implement the same service, *fault tolerance* by allowing instances to redirect their request to a service instance with a healthy state, and *canary release* for testing new versions of services.

4.2.2 Overview

Themis's design features a two-layer protocol architecture. The first, lower-layer described in Section 4.4, forms a core communication protocol pair that offers confidentiality, integrity, and authentication without depending on a centralised point of authority. This layer is comprised of two protocols: an authenticated key

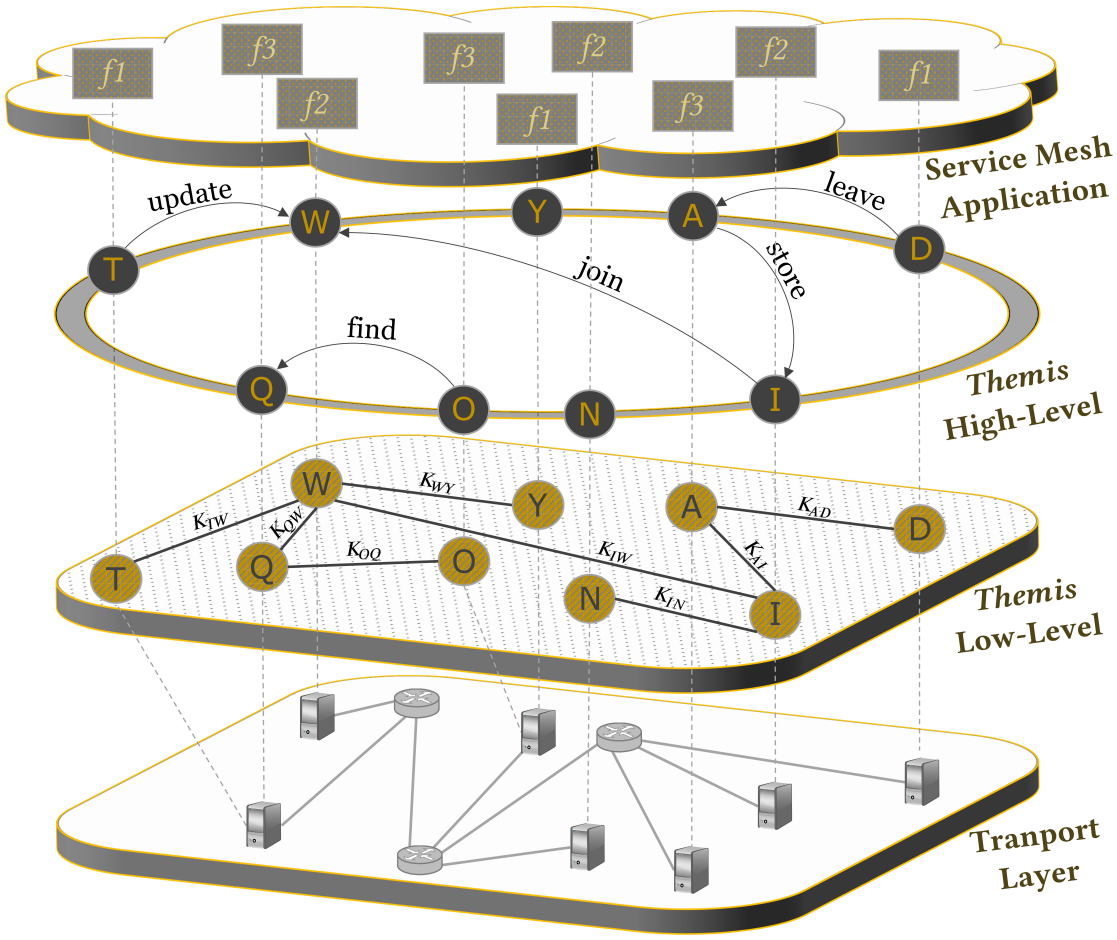


Figure 4.1: *THEMIS* sits on top of the transport layer, organising peers on a ring topology.

agreement protocol for setting up a secure communication channel between two nodes, and a protocol for direct communication between the two authenticated nodes. The key agreement protocol is somewhat comparable to the mTLS handshake protocol; the communication protocol leverages symmetric cryptographic primitives to ensure secure communication. Combined, the two protocols provide specific security guarantees to the communication channel established between two nodes, which we prove in Section 4.4.2. They effectively bind a symmetric key to a self-certifying identity so that any message sent over the secure channel that the key enables can be cryptographically bound to an identity on the network.

The second, upper-layer described in Section 4.5, builds on the previous protocol pair to provide a series of actions related to identifier management. Examples

of actions include `store`, which allows nodes to associate their overlay identifier with a data identifier, *i.e.*, an object, and `update`, which ensures correct workload distribution among network participants. These actions are typical in P2P overlay systems such as Chord [42]. This layer leverages the guarantees provided by the lower-layer to enhance security properties on nodes' P2P communication underpinning a fully decentralised serverless infrastructure; we analyse these properties in Section 4.5.2. This layer also incorporates several tunable parameters that depend on deployment specifics. For example, a redundancy factor allows several copies of a single identifier-to-node mapping; a freshness factor allows the network to self-calibrate mapping staleness. We present how these parameters can be used to provide a complete service mesh architecture in Section 4.6.

4.3 System & Adversary Model

In this section, we provide the system and adversary model of Themis. While Themis design was motivated by the service mesh application, it is equally applicable to any decentralised application.

4.3.1 System Model

Our system consists of a set of nodes that interact to exchange application-specific data. Each node can represent a machine or a service that needs to communicate, for example, to build a serverless application. Nodes may be physically located (and controlled by) different operators. The data-objects represent a specific capability or piece of data that nodes wish to make public to other network members. We assume that nodes know the name of the data they are looking for. Themis is agnostic to how data is named.

Themis is built on top of the transport layer, assuming that any node can reach any other node. It follows a layered architecture depicted in Figure 4.1, which consists of a lower-layer and an upper-layer described in Sections 4.4 and 4.5, respectively. Every node that joins Themis participates in the structured P2P network that Themis defines in its upper-layer. This structure allows nodes to

discover each other and the information they seek from the network in a decentralised manner by storing the identifiers of the nodes that initiate a store operation for that data. We assume that each node has sufficient computational power to perform the cryptographic calculations needed to establish and use the secure communication channel defined in THEMIS lower-layer, as well as enough storage space and energy to keep state and forward messages for the overlay network. In THEMIS, nodes can be grouped into different networks distinguished by a network identifier. To join a network, nodes need to communicate with a member of that network in order to bootstrap communication. Each node has an identifier that uniquely identifies them within the system; the node generates and stores a cryptographic key-pair, using the public key to construct its identity, in this way, nodes can be authenticated and link the messages they send within THEMIS.

4.3.2 Adversary Model

THEMIS considers a Dolev-Yao attacker who has complete control over the communication channel, allowing her to listen to and initiate connections with other nodes in the network. The attacker participates in the network and can control multiple nodes simultaneously. However, she does not have physical access to the machines, so she cannot retrieve any keys stored locally in the nodes' memory. Additionally, the attacker cannot compromise benign nodes or undermine the security properties of the underlying cryptographic schemes used by THEMIS's protocols, such as hashing, signatures, and Message Authentication Codes (MACs). The attacker's goal is to violate the confidentiality, integrity, and authentication of the messages exchanged over THEMIS.

The goal of THEMIS is to establish service identities and bind them to a secure channel for future communications. It does not assume any security guarantees from the transport layer and operates under the assumption that participants cannot establish trust in a third party, such as a certificate authority. The nodes do not know in advance which other nodes they will interact with, making out-of-band certificate installation or pre-shared secrets techniques impractical.

Themis does not address attacks aimed at disrupting node communication, including denial of service (DoS) or jamming. Furthermore, Themis does not specify a particular authorisation mechanism; however, the strong authentication, integrity, and confidentiality guarantees it provides enable the development of additional security features, such as access control policies tailored to the specific needs of each application.

4.4 Themis's Low-Level Architecture

In this section we initially describe the protocols that constitute Themis's lower-layer and we further analyse the security guarantees they provide.

4.4.1 Low-Level Protocols

All messages exchanged between nodes in Themis are sent over secure channels established by our two custom protocols described in this section. Together, these protocols ensure each participating node a strong cryptographic binding between a peer's public key and a shared key, as well as the confidentiality and authenticity of the messages sent through the secure communication channel created by this key. The first protocol facilitates an authenticated key agreement between any two network identities without relying on a centralised Public Key Infrastructure (PKI). The second protocol employs the established symmetric key to provide message integrity, confidentiality, and authentication. Since the identities of the peers in the network are derived from their public keys, this strong binding effectively enables peers to associate keys with specific identities, assuring that they communicate with the same node every time. Consequently, a malicious node can only send messages using the public key it possesses, which effectively assigns it a unique identifier in the network.

Authenticated Key Agreement

Authentication in this context means that all messages can be attributed to exactly one identity. The identity of a node is the hash of its public key and the name

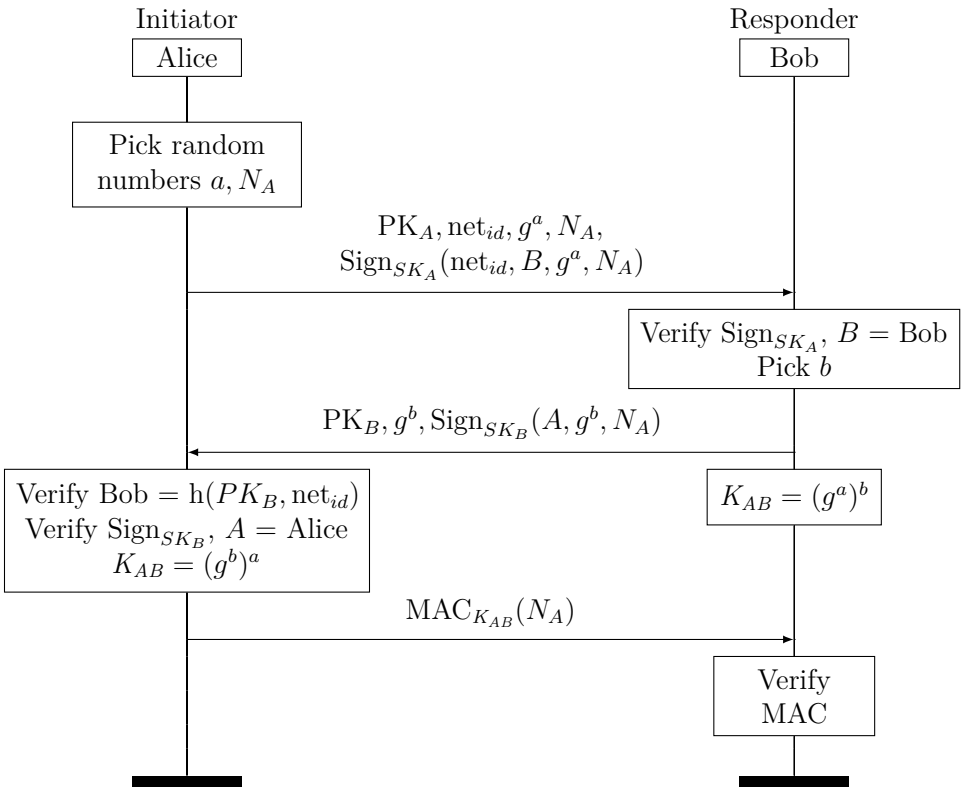


Figure 4.2: *Authenticated Key Agreement protocol.* Alice and Bob establisha shared symmetric key to be used for subsequent communication.

of the network. That means that Alice can freely pick a public/private key-pair (PK_A, SK_A) but the hash of the public key determines Alice's identity on the network net_{id}, i.e., $Alice = \mathrm{h}(PK_A, \mathrm{net}_{id})$.

The protocol is shown in Figure 4.2. Alice picks a Diffie–Hellman exponent a and a fresh nonce N_A. She sends g^a and N_A to Bob along with PK_A and net_{id}. Everything, including Bob's identifier B, is signed with SK_A. When receiving a new message, Bob first verifies that the signature is valid. He then inspects the identifier B that is signed by Alice to check that he was the intended recipient. Bob then picks his own Diffie–Hellman exponent b and sends g^b back to Alice along with his own public key, signed by the corresponding private key. Bob includes N_A from the first message to allow Alice to confirm freshness, and Alice's identifier A to allow her to verify that the message was meant for her. Bob then computes the key K_{AB}. When Alice receives message 2 she verifies that the hash of the public key PK_B corresponds to the identity she was intending to communicate

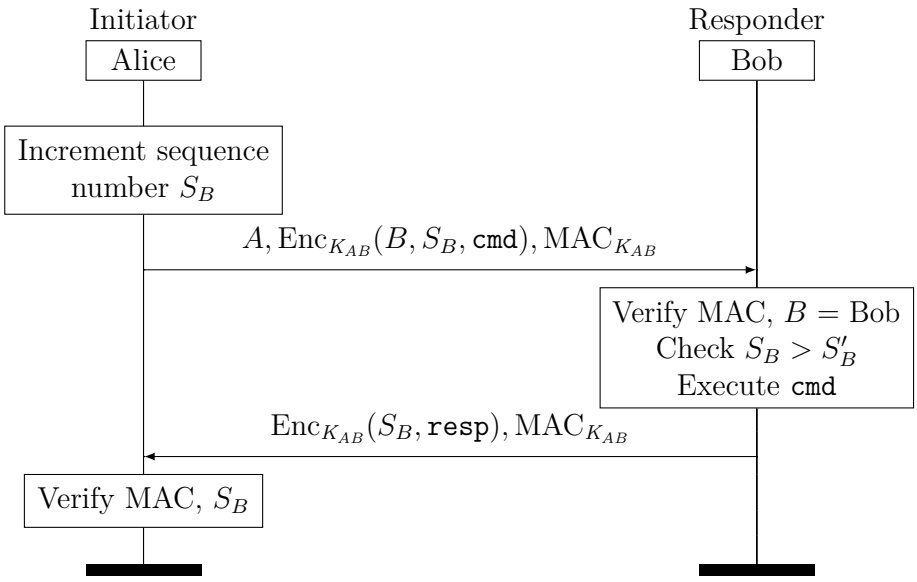


Figure 4.3: *Secure Communication Protocol.* Alice and Bob communicatesecurely using a symmetric key established with the Authenticated Key Agreement protocol.

with. If that is the case she checks that the signature is valid and computes the new shared symmetric key $K_{AB} = (g^b)^a$. To prove to Bob that she knows the key, Alice sends him a MAC of N_A created with K_{AB}.

If the protocol terminates without errors, **it guarantees that Alice and Bob share the same secret key**. The resulting secret key will be used in subsequent communication between the devices, enabling them to authenticate each other. Both Alice and Bob store K_{AB} together with the identifier of the other party. Hence, the number of keys that each node has to store is proportional to the number of identifiers with which it chooses to communicate.

Secure Communication

The Secure Communication Protocol is used for all communication between nodes in the network (except for key establishment). Specifically, all the procedures used to maintain the P2P network, *i.e.*, `join`, `update` and `leave`, as well as the messages exchanged to execute a `find` or `store` operation, use this protocol to provide confidentiality, integrity and message authentication.

The protocol is shown in Figure 4.3. Alice, who wants to send the command

`cmd` to Bob, first increments the sequence number S_B she maintains for her communication with Bob. She then encrypts using the symmetric key she shares with Bob the command along with Bob's identity and the sequence number. In her message, she also includes her identity A to allow Bob to retrieve the correct symmetric key and a MAC of everything. When receiving the message Bob will verify the MAC using the already established key, and if the MAC is valid he decrypts the message. Bob ensures that the identity in the encrypted message is his, and that the sequence number S_B is higher than the previous sequence number he received from Alice. If so, he can process the command. When a response `resp` is ready, Bob encrypts it with K_{AB} together with S_B, he calculates the MAC of the encrypted message and send them back to Alice. When Alice receives message 2, she verifies that the MAC is valid and that the sequence number corresponds to the one she sent in message 1.

If the protocol terminates without errors, **it guarantees confidentiality and integrity of the command and response**.

4.4.2 Security Analysis

In this section we prove the security guarantees provided by our protocols presented in the previous section (Sec. 4.4.1). For our analysis we assume an attacker under the threat model as described in Section 4.3.2.

Authenticated Key Agreement

Guarantee 4.1. *If the Authenticated Key Agreement Protocol terminates without errors and the decisional Diffie–Hellman (DDH) assumption holds in the underlying group, the key K_{AB} is known only to Alice and Bob.*

Proof. The only terms in the Authenticated Key Agreement protocol that are related to the key are the terms $a, b, g^a, g^b, MAC_{K_{AB}}$ and of course the key itself. Of these a and b are never visible to the adversary as they are picked fresh during the protocol, and remain internal to Alice and Bob, respectively. By the assumption that all the underlying cryptographic primitives are secure the adversary cannot

obtain the key from the MAC. That leaves g^a and g^b. If the adversary has a method of getting K_{AB} from these values then he can use that method to break the Diffie-Hellman assumption, which is a contradiction. □

Guarantee 4.2. *If the Authenticated Key Agreement Protocol terminates without errors both Alice and Bob will be in possession of the same key.*

Proof. We prove this for Alice and Bob individually, starting with Alice. If the protocol completes successfully, Alice knows that she shares K_{AB} with Bob. For an adversary, Eve, to break this guarantee and convince Alice to assign another key K_{AE} to her communication with Bob, Eve would have to successfully send message 2, as this is the only way for Alice to obtain Bob's Diffie-Hellman contribution. Eve has two options to send message 2: she can either craft the message or replay a previous captured message.

In order to craft message 2 the adversary has to find another key pair where the public key obeys the following $B = \mathrm{h}(PK_E, \mathrm{net}_{id})$. This means that Eve has to break the second preimage resistance property of the underlying cryptographic hash function, which again contradicts the adversary model. Without the ability to change the keys for Bob's signature, the adversary has to either forge Bob's signature or obtain his private key. Both again contradict the adversary model. That only leaves the option to replay a previous message. For replay to work, Eve has to make sure that the content of message 2 expected by Alice, corresponds to one of the messages available to Eve. Because the nonce N_A selected by Alice and Alice's identifier are both part of the signature in message 2, it means that Eve cannot reuse a message from a previous session or one that Bob sent to another node, to attack Alice. Eve would have to force Alice to chose a nonce that corresponds to one of Eve's captured messages; however, according to our threat model, Eve cannot influence Alice's choice of N_A.

We now prove the same for Bob. By the same argument as above the adversary cannot forge Alice's signature on message 1. However replay is a different story. Bob does not have a way to readily check if message 1 is a fresh message from Alice

or indeed a replay from Eve, so Eve can initiate a protocol run. However, in order to successfully fool Bob and make him register a different symmetric key for Alice, Eve has to confirm the new key in message 3. By Guarantee 4.1, Eve does not know the symmetric key even if she sent the first message. That means that she has to produce a valid MAC of N_A without knowing the key. For a secure MAC scheme this is not possible, and guessing either the key or the mac-value itself is only possible with negligible probability. □

Secure Communication Protocol

Guarantee 4.3. *As long as the symmetric key K_{AB} remains known only to Alice and Bob, message confidentiality and integrity is preserved for any command and response sent by Alice and Bob respectively.*

Proof. Both the command and response are solely sent encrypted with K_{AB}. The only way to break confidentiality of the command or response is to break the confidentiality of the encryption function. This contradicts the threat model which states that all underlying primitives are secure. In order to violate message integrity the adversary would have to recreate the MAC of the message without knowing the MAC-key. This again contradicts the threat model which states that all underlying primitives are secure. □

Guarantee 4.4. *Any command received by Bob can be attributed to Alice and any response Alice receives can be attributed to Bob. In other words, message authentication is guaranteed.*

Proof. This guarantee has to be shown for Alice and Bob individually. To break the guarantee for Alice the adversary would have to manipulate message 2. By Guarantee 4.3 message 2 cannot be crafted, so that only leaves replay. To successfully convince Alice that a false response came from Bob, the adversary has to replay a valid message containing the same sequence number that Alice used at the start of the protocol. The sequence number is incremented before each new transmission by Alice so two packets from different sessions will never use the same number. The

only message that uses the same sequence number is the one just sent by Alice, but the encryption contains the intended receiver so it cannot be used to replay (reflect) message 2. The only message the adversary can replay is the true message sent by Bob which is not an attack.

To break the guarantee for Bob the adversary must manipulate message 1. By Guarantee 4.3 message 1 cannot be crafted, so that only leaves replay. To succeed the adversary must replay a message containing a sequence number that is bigger than the biggest one Bob has yet received. That means the adversary can only drop messages but never deliver them out of order. In fact the adversary cannot use any old messages that was sent prior to the last message received by Bob. □

4.5 Themis's High-Level Architecture

In Section 4.4.1 we presented the two protocols that construct THEMIS and in Section 4.4.2 we proved the security properties they provide. In this section, we elaborate on how THEMIS builds a secure structured P2P platform.

4.5.1 High-Level Protocols

To enable nodes to connect with other nodes in the network and get the information they seek in a decentralized manner, THEMIS follows a DHT architecture implementing five key operations, `find`, `store`, `join`, `update` and `leave`. From these operations, `find` and `store` support the identifier-to-object mapping whereas `join`, `update` and `leave` support maintenance procedures for P2P organisation. Every peer initially executes THEMIS's *Authentication Key Agreement* protocol depicted in Fig 4.2 with each other node it wants to communicate. This handshake allows nodes to establish a secure communication channel based on a secret symmetric key. To execute the P2P operations, nodes send messages using THEMIS's *Secure Communication* protocol depicted in Fig 4.3. The sender and the receiver assign the `cmd` and `rsp` variables according to the respective P2P operation as illustrated in Table 4.1. In the following paragraphs we describe the P2P operations.

Operation	Sender (`cmd`)	Receiver (`resp`)
Find	(`find`, $identifier$)	$value$ **or** id
Store	(`store`, obj)	ack
Join	(`join`, net_{id})	id
Update	`update`	id **or** $(id, ObjTable)$
Leave	(`leave`, $ObjTable$, id)	ack

Table 4.1: *THEMIS's High-Level Messages.* The sender and receiver assign `cmd` and `rsp` in the Secure Communication Protocol according to the operation they want to execute.

Find: This operation allows nodes to reach specific identifiers, *i.e.*, m-bit addresses from a finite address space 2^m, organised on a ring topology in a clockwise order and calculated at random using a *cryptographic hash function* $h(m)$. The identifier that is specified each time by the initiator can refer to a node $node_{id}$ or to an application-specific data $object_{id}$. When nodes receive a `find` request, they try to resolve it by examining if this identifier is included in their routing table or in their object table. In its routing table, *i.e.*, RT_{id}, every node maintains m entries with the details of other nodes that succeeds it on the ring. For a node with $node_{id} = n$, each entry i contains the details of the closest node with $node_{id} \geq n + 2^{(i-1)}$, with $i = 1$ being the successor of n. The way the routing table is constructed allows identifiers to be found in at most m steps. Further to these m entries, nodes save in their routing table the details of their predecessor, *i.e.*, the closest node that precedes them along the ring. The object table, *i.e.*, OT_{id}, is the table where nodes store mappings between $node_{id}$ and $object_{id}$ for the objects they are responsible for. The successor of every object, *i.e.*, the closest node for which it applies $object_{id} \leq node_{id}$, becomes the responsible for this object node. The `find` operation for a $node_{id}$ terminates to a node that has this $node_{id}$ in its RT_{id}, returning the associated communication address of the specified $node_{id}$. In the case of an $object_{id}$, `find` terminates to the responsible for this object node, who will return the value it stores for this object, *i.e.*, a list of identifiers $\{node_{id}\}$ of the nodes who have associated themselves with this object. In case the receiver of the request does not have this identifier neither in its RT_{id} nor in its OT_{id}, it will return the $node_{id}$

(and the communication address) of the closest entry from its RT that precedes the sought-after identifier. To allow the initiator to monitor how her request is progressively resolving, we adopt an iterative routing where all the traffic is handled by the requesting node; the responder will return the closest node to the initiator, who will then initiate a new operation with this new node.

Store: This operation provides the possibility for each node to associate its $node_{id}$ with a specific $object_{id}$. The node who wants to be associated initially starts a `find` operation for this $object_{id}$; this will provide the possibility to the initiator to retrieve the responsible node for this object. As soon as the node has this information, it will contact the responsible node specifying the object with which it wants to be associated. The responder will then amend the list for this object with the $node_{id}$ of the initiator. Themis achieves fault tolerance by adopting a replication mechanism that maps application-specific data to multiple objects. In particular, every descriptor that is defined by the naming mechanism in use is hashed together with a counter number $object_{id} = h(name_{id}||r)$ where $r \in [0, k]$; thus, for a single application-specific data nodes create and store $k + 1$ objects. The value k is a network constant agreed among the participant nodes, which determines an upper bound of the redundant objects that will be stored on the network. However, nodes are free to decide the value k they will use for every application-specific data. The responsible nodes drop the values they store in between specific time intervals t. For this reason, peers contact the responsible nodes periodically, to make sure that their associations are maintained registered on the network.

Join: To learn the first node who succeeds them in the network, nodes execute the *join* operation with a bootstrapping node, a node which they know and which is already a member of this network. The bootstrapping node will initiate a `find` operation specifying the newcomer's $node_{id}$ as the sought-after identifier. Based on the routing procedure explained above, the `find` operation will return at the end the first successor of this identifier in the network. The responder will return to the initiator its first successor who upon receiving its response will save it as its first successor in its RT_{id}.

Update: To maintain a consistent mapping between nodes and objects, nodes periodically execute the *update* operation in specific time intervals. During this operation every node will contact its first successor to check if it is actually the node who succeeds it in this network. The responder, upon receiving the initiator's request will retrieve his predecessor and check if his predecessor is also a predecessor of the initiator. If this is the case, the responder saves the initiator as his predecessor and checks to see if the $node_{id}$ of the initiator is closer than his identifier to any of the objects for which he is responsible. If there are such objects, he will pass them with his response to the initiator who will now become the responsible for these objects node. If the predecessor of the responder is between the responder and the initiator identifiers, the responder will send back to the initiator his predecessor $node_{id}$ as this is her correct first successor who she needs to communicate.

Leave: In THEMIS nodes maintain in their OT_{id} the associated values for all the objects for which they are responsible. The *Leave* operation allows for node departures from a network to occur without any loss of associations. The initiator who wants to leave the network communicates her first successor to pass the information related to the objects for which she is responsible together with the $node_{id}$ of her predecessor. The responder will store and become now responsible for these objects. He will also update his predecessor so as to point to the initiator's predecessor. In THEMIS nodes are free to leave and join at any time. Nodes can abandon any network in which they participate without notifying other nodes, in this case any node who will try to contact them will receive a timeout and the node will be presumed dead, and all the associated values for the objects for which they were responsible will be lost. The rest nodes of the network can retrieve the lost associations by initiating a find request for one of the k remaining objects.

4.5.2 P2P Attacks

In this section we elaborate on the security properties of the P2P operations against common attacks of DHT networks [13].

Sybil attack: The self-certifying identifiers that Themis provides allow every device to participate in the network without the need for any initial communication. They also allow devices to create as many identities as they want, known as a Sybil attack, which can be problematic in certain applications. Themis does not consider this behavior an attack since the properties of message linkability and authentication still apply for every identity. Some applications might want to use different identities to provide different services—even though these are hosted by the same device. Our solution makes that possible. For applications where Sybil attacks are problematic, Themis provides the application layer with enough information, *i.e.*, the communication address of each identity, to implement checks for which identities are allowed to run on which devices.

Eclipse (Routing Table Poisoning) attacks: In an Eclipse attack an attacker tries to isolate a node from honest peers by placing malicious nodes as its neighbors. It is possible for an attacker to insert incorrect information into the routing table of a device if that device happens to contact an adversary. However, because of the circular structure of the addresses, it is always possible to check if a `find` request is making progress towards its goal. If progress is not being made the device can make sure not to contact the same identity again. As long as there are still honest nodes present in the routing table, the device will eventually ask one of them and get useful information. A coordinated attack against a specific device could result in DoS, but that falls outside of our threat model (Sec. 4.3.2).

Storage attacks: Themis enables nodes to store an association between an application-specific data object, *e.g.*, a service name, and its identity in the network. Specifically the association is kept by $k + 1$ different nodes, and some of those nodes might decide to drop the association, or make up an association that does not exist (ghost objects). If that happens other devices that search for an object will either not find it or find a ghost object that matches their search. This is a nuisance but will never result in a violation of any of our security guarantees. Any object is stored in the network $k + 1$ times so if an association is drooped by a few nodes the other objects will still be available. Furthermore, it is the responsibility

of any node that stores data in the network to regularly check and re-store the data if objects are missing, so such an attack has limited effects. If a ghost object is returned from a `find` operation, it will result in a connection attempt to the identity pointed to by the ghost object. That identity can either signal that the association is unknown, in which case the attack resulted in a single unnecessary connection and nothing else. If the ghost object points to an attacker that then provides a service, that is not an attack on THEMIS. This is equivalent to an attacker announcing a service under a specific name and then providing that service. The service itself might be malicious, but that is a problem for the application layer as THEMIS does not know or care what data is transferred, only that the identity of the communicating partners cannot be falsified.

4.6 Themis Service Mesh

We now elaborate on how THEMIS can be used to assist programmers tasked with the development of serverless applications by taking care of all the low-level details of a secure and scalable communication across serverless nodes. Let's assume an example application, *AppAuth* consisting of two microservices: an ingress point, *i.e.*, `handler`, and a processing back-end, *i.e.*, `check`. When the `handler` microservice receives a web `request` carrying a user`name` and a `pass`word, uses `check` to fetch a dictionary of known users' credentials, validates the received pair, and then handles it appropriately by sending a response message and associated code.

The programmer starts by creating a mesh with a specific network identifier to group the microservices that need to communicate, specifying a bootstrapping node. In our example, microservices `join`s a service mesh with identifier `twiitr` and bootstrapping point the route `/main`. Microservices are placed on a ring topology based on each microservice's node identifier, following a key-based routing scheme. THEMIS allows *service discovery* by creating mappings between the node identifiers and the microservices. A node first creates a cryptographic public-private key pair and generates its identifier $B = h(PK_B, twiitr)$. It then calculates an object identifier for the microservice it implements, by hashing the service's name interface,

e.g., $Object_B = h(check)$. A mapping is created by initiating the `store` operation that informs the responsible node for $Object_B$ to amend the object's value with its node identifier B. Nodes that request `check`, for example node A, retrieve B by initiating a `find` operation for $Object_B$. In fact, the `find` operation returns to the requester a list of node identifiers that have executed the `store` operation specifying $Object_B$. There is one node responsible for every object identifier. Themis adopts a redundancy mechanism having one microservice to be associated with multiple object identifiers. In our example, assuming a redundancy factor $k = 2$, the `check` microservice will be associated to three sibling object identifiers, $Object_B$, $Object_{B1} = h(check, 1)$ and $Object_{B2} = h(check, 2)$, for which node B initiates three separate `store` operations. Node A can retrieve B by initiating a `find` operation for any of the three object identifiers. The returned list and the redundancy factor provide *fault tolerance*; in case of failing nodes the requester can select another node from the list or initiate a `find` operation for a sibling object, until a node with a healthy state is found. Nodes select randomly over the returned list the node to communicate; thus, the load is evenly distributed between the replicated nodes. Further to the dynamic *load balancing* property that this technique provides, it also enables *canary releases*. As nodes are evenly selected, programmers can deploy a new version of a microservice progressively on different machines based on their nodes identifiers.

In a centralised architecture configuration functionalities, *e.g.*, authentication and discovery, are implemented in whole or in part by control plane registries which data plane proxies need to communicate to operate. Themis couples the data and the control plane implementing a fully decentralised application. In our example the responsible node for $Object_B$, *e.g.*, node C, fulfils the task of a central service discovery registry redirecting A to B. Themis leverages the DHT to route messages among peers; thus, it inherits its *scalability* property to accommodate the high rate of replication services. Abolishing managerial registries provides *openness* that facilitates migration off commodity serverless platforms, particularly beneficial to enable access and processing of data hosted on edge devices. Decreasing

programmers lock-in can allow them to invent their own configuration services, *e.g.*, *observability*, exposed through special service-name interfaces that application microservices can discover following the same discovery mechanism explained above. The low-level protocols of Themis render nodes *accountable* for the messages they exchange. Using every message as a node behaviour trace, faulty or malicious activity can be identified and resolved.

4.7 Implementation

We have implemented a prototype of Themis in about 3.3K lines of JavaScript, chosen due to its ubiquity in serverless environments. Our implementation is open-sourced and available on GitHub at github.com/atlas-runtime/themis. Themis is embedded in Atlas [81], a runtime environment for automated offloading and scale-out of JavaScript programs.

A thin command-line wrapper allows the construction of virtual nodes for testing and experimentation, as Unix processes running on QuickJS [82]. QuickJS is an embeddable Javascript engine that supports the ES2020 specification including modules, asynchronous generators, and proxies. The runtime environment is small, offering about 210 KiB of x86 code for a simple hello world program.

Themis uses the built-in EcmaScript object type to maintain an in-memory mapping between strings to identifiers (along with their timeouts and debug metadata). In terms of cryptographic support, Themis offloads operations to NaCl [83], a high-speed constant-time cryptographic library compiled to JavaScript using Emscripten (adding another 2K LoC). NaCl offers high performance cryptographic primitives, avoids calls to dynamic memory allocation functions such as `malloc` and `sbrk`, and uses small amounts of stack space. Upon bootup, a service generates a new secure public-private key pair. For the communication between two services, NaCl provides primitives that combine a receiver's public key with the sender's private key to derive a common symmetrical key. Subsequently, this key is used to symmetrically encrypt and authenticate all messages (in Themis, serialized buffers) to perform the P2P actions defined in the upper-layer of Themis.

Themis is accessible as a software module (library), implantable into (often, pre-existing) programs using a conventional `import` statement. Upon import, the Themis library (1) checks for the existence of a public-private key pair, and if non-existent generates a fresh one; (2) loads and binds a listener of the chosen transport protocol on a pre-specified port (consecutive ports for multiple nodes running on a single physical host); (3) returns the `Themis` object, which is used to access `Themis`'s interfaces.

4.8 Evaluation

In this section, we evaluate Themis, trying to understand the overhead it introduces to the applications built on top, investigating the following questions: (Q1) What are the performance and scalability characteristics of serverless applications that implement the security protocols that constitute Themis lower-layer, and how do they compare to Themis-less versions (Sec. 4.8.1)? (Q2) How do different Themis operations perform in the limit, and how do they compare with their insecure counterparts (Sec. 4.8.2)? To answer these questions, we perform experiments across two distinct environments. For Q1, we use as benchmarks eight serverless applications, outlined in Table 4.2 (alphabetical order), whereas for Q2, we use synthetic microbenchmarks that stress individual Themis operations.

Result Highlights: Across all eight serverless applications, the security benefits of Themis come at an imperceptible throughput overhead (1.24%, on average) and a small latency overhead ($< 4\%$ in almost all the benchmarks), more pronounced in the context of low-latency serverless applications. In the worst case, the application startup overhead introduced by Themis is 356.52%. However, it remains under $1s$ and is a one-off cost amortised across the long execution times typical for serverless applications. The synthetic microbenchmarks indicate that the best-possible security overhead Themis introduces on the `find` DHT operations ranges from 2.1% to 31.6% with the fastest `find` request (involving one hop) taking around 300ms. The total execution time for each operation is expected to be higher for real-world scenarios where nodes participating in the DHT operations can be located in various

places around the globe, resulting in each hop in the network spanning significant distances. However, aside from the decentralisation cost inherited by the DHT scheme, the overhead introduced by Themis to provide its security guarantees ranges only a small percentage of the total execution time for the operations.

Experimental Setup: On the hardware side, for Q1 we use an 8-node distributed cluster on Microsoft's Azure Cloud. It amounts to eight DC1s v.2 machines equipped with Intel Xeon E-2288G CPUs and 4GB of memory, and running Ubuntu 18.04 LTS with kernel version 5.4.0-147. We use this environment to get insights into the performance penalties introduced to applications built on top of Themis due to its lower-layer security protocols. For Q2, we use a large-scale multiprocessor equipped with an 128-core Intel Xeon E7-8830 processor at 2.13GHz, 512GB of memory; it is running Debian 4.19.160-2 with kernel version 4.19.0-13. This environment uses a DHT implementation that adheres to the vanilla Chord [42] protocol for Themis's upper-layer, and it is utilised to launch hundreds of micro-services, avoiding the non-determinism introduced by network operation and allowing us to zoom into Themis-inherent overheads. On the software side, we use QuickJS u-2021-03-27. We launch multiple (virtual) `Themis` nodes as operating-system processes on each physical node. Each virtual `Themis` node has its own copy of the runtime environment, listens on a separate (ip, port) pair, and accepts events in its own event queue. Except when noted otherwise, we report averages over 1K runs.

4.8.1 Q1: End-to-End Performance

To understand the performance and scalability overheads introduced by Themis, we compare the Themis-augmented benchmark applications against their non-secure counterparts—*i.e.*, ones that do not incorporate Themis' security components. Table 4.2 presents the results of Themis's end-to-end performance evaluation across five metrics: startup time, execution time, throughput, latency and duration. Each measurement cell contains three numbers—a tuple of the form (Δ, V, T) where Δ is the percent difference between the vanilla and Themis-augmented versions of

the serverless application, and V and T is the absolute measurement corresponding to the vanilla and the secure, Themis-augmented implementation, respectively.

Regarding startup time, *i.e.*, , the duration cost of reaching a stable state—registering all relevant services and creating encrypted communication channels between them—Themis introduces an average overhead 0.16s. The total execution time, *i.e.*, the end-to-end time to run the full load, shows minimal differences (under 1% in most of the cases). Similarly, the request throughput, *i.e.*, the number of requests a serverless application handles per second, shows identical performance for the majority of the evaluated applications. Themis has a higher impact on the application Latency, the average time to execute a single request. Themis introduces a maximum of 13% latency overhead to each request, whereas the execution overhead of each request ranges between 0–7.75%.

4.8.2 Q2: Individual Operator Performance

To understand Themis's performance of `find` and `store` operations across different scales, we perform two experiments where we insert and retrieve 1M 16-byte data objects at a constant rate of 50K objects per second. In the first experiment, the objects are configured to resolve to the node receiving the request to store or retrieve the object—*i.e.*, the node does not need to forward the request to other nodes. By resolving the request locally, *i.e.*, excluding multiple hops of serialization, inter-process communication, and context switching, the throughput and latency results show the best-possible performance achieved by our runtime implementation—*i.e.*, Themis's practical limits due to the implementation's runtime environment. The resulting throughput averages 10223 and 9332 operations per second for `find` and `store`, respectively; the resulting latency averages 331ms and 338ms for `find` and `store`, respectively.

In the second experiment, we focus on the throughput as a function of the number of nodes. Object identifiers are now randomly generated, and thus are expected to hit all the nodes in the network with uniform probability. Fig. 4.4 shows the throughput and the latency of the `find` operation as a function of the

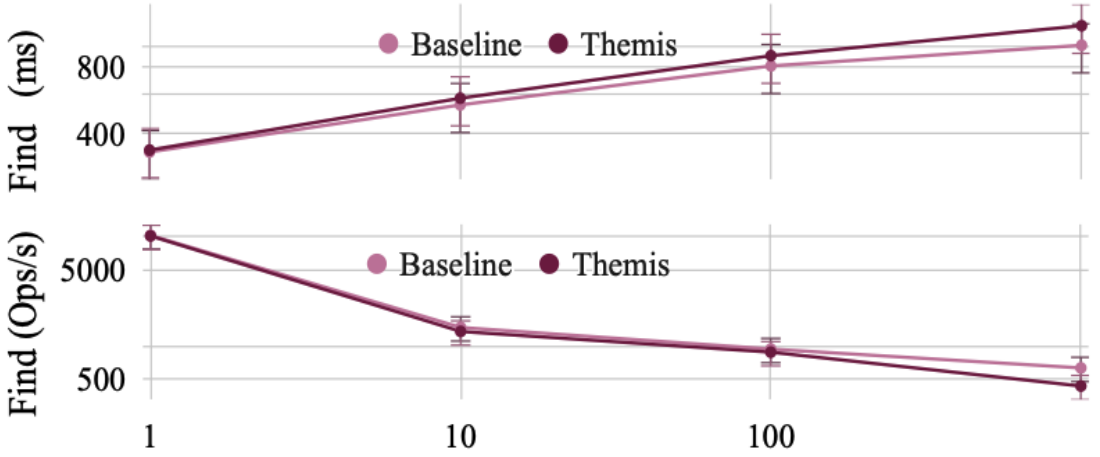


Figure 4.4: *Operation* `find`. The plots show the throughput (bottom) and the latency (top) of the `find` operation, as a function of the number of nodes, on a constant operation workload of 50K peer-to-peer operations per second.

number of nodes. The overhead of adding security ranges between 2.1–31.6% and depends critically on the percentage of nodes that have already performed the key agreement protocol. For low numbers of nodes (left side of plots), the majority of nodes have performed the key agreement protocol, whereas for high numbers of nodes (right side of plots), the majority of nodes have not.

To understand the performance of the `join` operation, we have 500 nodes contact ten bootstrap nodes in round-robin fashion. Starting these 500 nodes sequentially takes 362.466s (an average of 720.5ms/node). If, however, we spawn 500 nodes in parallel, we get a total of 15.403s (an average of 30ms/node). A part of this overhead includes operating system overheads such as V8 process creation, which averages about 320ms per node. Other overheads arise from the identify (public-private key pair) creation and the authenticated key agreement protocol, averaging about 52ms.

To understand the overhead of `leave`, we launch a series of nodes with a startup configuration that runs a `join` followed by a `leave` command when the `join` command completes. We run this sequentially in a loop where we spawn a node only after the previous node has shutdown; on average, "blinking" a node, *i.e.*, have a node `leave` right after `join`ing, takes a total of 780ms. Much of this time is spent in system-level overheads, most of which is from (i) importing multiple

library source files and (ii) binding to various network interfaces. The overhead of `leave` amounts to less than 50ms.

4.9 Summary

This chapter presents Themis, a framework for secure P2P communication that is general enough to be usable in a variety of scenarios that demand point-to-point interaction. As an example, here we shown how Themis can serve as a platform for implementing a secure service mesh communication network for use in data centres and companies that need dynamic load balancing and extensibility. Themis consists of two layers. Its lower-layer provides a secure communication protocol, similar to mTLS in many ways but with a strong emphasis on distributed identity management. We provide a thorough security analysis that proves the claimed security guarantees, *i.e.*, confidentiality, message integrity, and message authentication/linkability. We emphasise how any additional guarantees can be built on top, leveraging its core security properties. Its upper-layer consists of a set of actions that offer a fully functional P2P network.

	Startup Time [s]		Exec. Time [s]		Throughput [req/s]		Latency [s]		Duration [s]	
	%Δ	T / V	%Δ	T / V	%Δ	T / V	%Δ	T / V	%Δ	T / V
DecisionTree	19.23	0.31 / 0.26	0.20	176.25 / 175.90	0.00	0.68 / 0.68	0.91	57.59 / 57.07	0.00	1.45 / 1.45
K-Means Clustering	45.86	0.31 / 0.21	0.23	245.70 / 245.14	0.00	0.49 / 0.49	0.23	92.19 / 91.98	0.00	2.02 / 2.02
Knn	356.52	1.05 / 0.23	0.48	731.69 / 728.20	0.00	0.16 / 0.16	0.58	336.93 / 334.98	0.33	6.03 / 6.01
LinearRegression	43.48	0.33 / 0.23	7.97	169.10 / 156.62	7.79	0.71 / 0.77	12.99	53.48 / 47.33	7.75	1.39 / 1.29
NLP	12.50	0.45 / 0.40	0.95	129.19 / 127.98	1.06	0.93 / 0.94	3.94	34.30 / 33.00	0.95	1.06 / 1.05
NaiveBayes	42.86	0.30 / 0.21	0.39	124.62 / 124.14	1.03	0.96 / 0.97	2.80	31.89 / 31.02	0.00	1.02 / 1.02
RandomForest	30.00	0.26 / 0.20	0.22	120.40 / 120.13	0.00	1.00 / 1.00	2.69	29.74 / 28.96	0.00	0.99 / 0.99
Unweighted Shortest-Path	0.00	0.04 / 0.04	0.13	165.46 / 165.24	0.00	0.73 / 0.73	0.37	52.23 / 52.04	0.00	1.36 / 1.36

Table 4.2: *End-to-end performance evaluation.* For each measurement we present three values: The performance of Themis T, the performance of a vanilla implementation V, and the increase in percent %Δ.

5 Iris

Dynamic Privacy-Preserving Search in Authenticated Chord Networks

Contents

The previous two chapters deal with security challenges that structured P2P networks encounter, contributing towards their application-agnostic adoption and

use as an open network that nodes can join easily. This chapter addresses privacy challenges in authenticated Chord P2P networks. Our primary goal is to examine how we can prevent the tracking of the activity of nodes in the network. Iris, our proposed privacy-preserving mechanism allows us to keep all the security properties already achieved in the previous chapters, permitting the user to determine the trade-off between the privacy gain and the performance cost of a query while maintaining compatibility with the existing Chord protocol. To better capture the privacy achieved by the iterative nature of the search (to resolve a query, the requester needs to ask a number of intermediate nodes if they have the target information), we propose a new privacy notion inspired by k-anonymity. (α, δ)-*privacy*, the new privacy metric we introduce, 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.

5.1 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 [4] 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) [37], UsenetDHT [38], OverCite [39] and ConChord [84]. It has also been proposed in the literature as a resource service discovery mechanism in grid computing [85] and WSN [86] and an alternative to the traditional centralised design for DNS [87] and telephony [88] systems. More recently, CFS has been proposed by the Tor project [40] 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) [41] 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 [89].

Although privacy was not initially a design objective in Chord, achieving it is clearly desirable in many (most?) situations [12, 90, 91]. 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 [92, 93]. 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., [15, 94–96]. Existing P2P applications such as CFS [37], NKN [41], the Inter Planetary File System (IPFS) [97] and the Storj distributed storage platform [98], 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) 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 5.5.

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 [99, 100] or to the data structure [101], 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 behaviour) 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.

5.2 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.

5.2.1 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.

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 2.3.1, 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.

5.2.2 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.

5.3 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 2.3.1. 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., [15, 94, 102], 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.

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 5.4, for parameters α and δ chosen by the requester.

5.4 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 parametrised 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

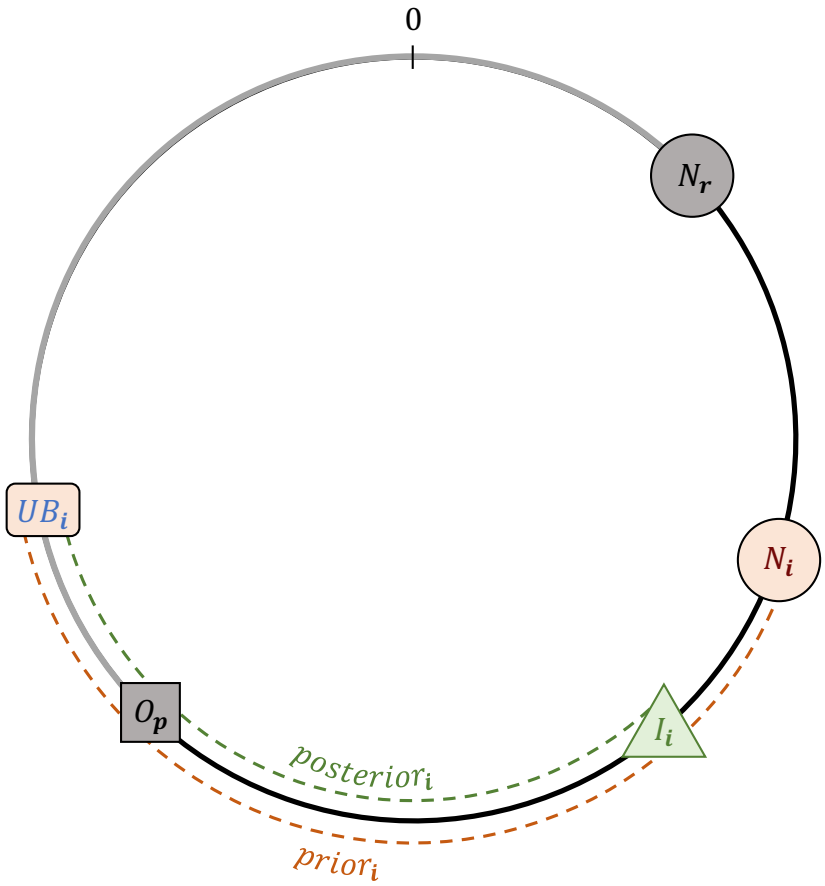


Figure 5.1: *The privacy metric.* The orange dashed line indicates the $prior_i$ range of N_i against N_r. The green dashed line shows the $posterior_i$ range of N_i after knowing I_i. Both ranges are computed based on UB_i, *i.e.*, the upper bound of node N_i's estimate regarding the range in which belongs the actual target of node N_r.

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.

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 5.5 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 5.1, 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.

5.5 Iris

In this section, we describe IRIS, the mechanism we develop to allow for (α, δ)-*private* queries in Chord. Table 5.1 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.

Table 5.1: List of symbols and notation used in this chapter

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 5.4
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

5.5.1 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

Algorithm 3 IRIS's Retrieve Algorithm

```
1: function IRIS(RT_r, O_p, α, δ)
2:     N_i ←SELECTSTARTNODE(RT_r, O_p, δ)
3:     repeat
4:         N_i' = N_i
5:         R_i ←RANDOMADDRESSBETWEEN(N_i, O_p)
6:         I_i ←LERP(R_i, N_i, α)
7:         N_i ←LOOKUP(N_i', I_i)
8:     until N_i == N_i'
9:     return FETCH(N_i, O_p)
```

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.

5.5.2 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: `iris`$(RT_r, O_p, \alpha, \delta) \rightarrow$ *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.

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

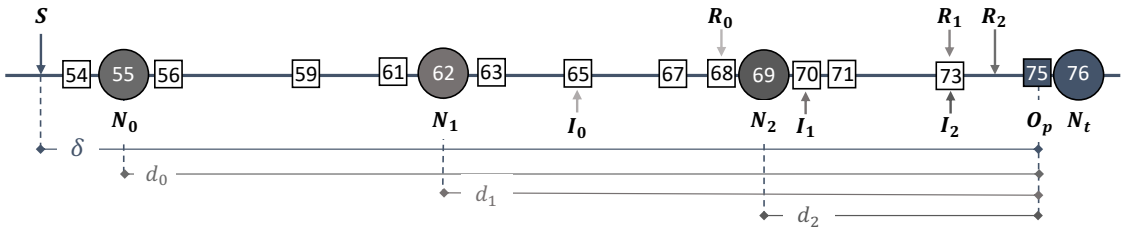


Figure 5.2: *IRIS's application example.* The requester targeting object $O_p = 75$ selects $\delta = 22$ and $\alpha = 0.25$, and queries back to back nodes for identifiers chosen in the interval $[53, 75)$. In every iteration the interval degrades, converging at the end to node $N_t = 76$.

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 5.2. Let's assume a requester node 44 wants to find the node that stores the values of a service by the name $'secret'$. 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.

5.6 Security Analysis

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

5.6.1 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 5.5.2 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 5.2, 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 (5.1). Because $\alpha \in [0, 1)$, we have that $\lim_{n\to\infty} d_n = 0$, thus, the algorithm converges on the target.

$$d_n = d_0 \cdot \left(\frac{\alpha + 1}{2}\right)^n \tag{5.1}$$

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:

$$\begin{aligned} d_n = \nu \Rightarrow (\alpha+1)^n \cdot \frac{d_0}{2^n} &= \nu \\ \Rightarrow n &= \frac{log_\alpha d_0 - log_\alpha \nu}{log_\alpha 2 - log_\alpha(\alpha+1)} \end{aligned} \tag{5.2}$$

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 [103]. Assuming N active nodes, the distances between consecutive active nodes can be modelled 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 modelled 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

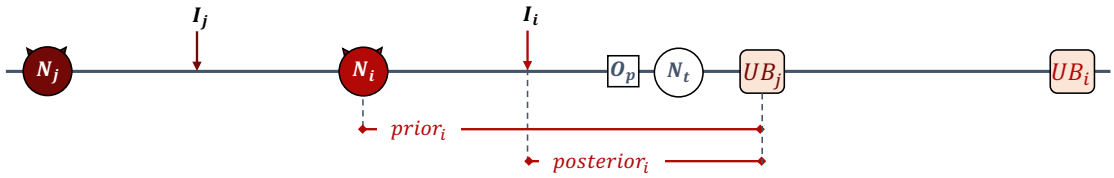


Figure 5.3: *A colluding adversary.* Assuming that N_j is the first asked colluding adversary, every other colluding node that the requester queries can use UB_j instead of UB_i in their calculation to infer the target.

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 [103], to obtain the minimum false positives and false negatives, γ must be equal to $\gamma = 1/f$.

5.6.2 Query Privacy

Here we prove that Iris is an (α, δ)-*private* algorithm following the definition introduced in Section 5.4. We start by analysing 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.

Lone Adversary

Let us now consider a lone adversary that controls only node N_i. Recall from Section 5.5.2 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:

$$\begin{aligned}
\frac{posterior_i}{prior_i} &= \frac{|UB_i - I_i|}{|UB_i - N_i|} = \frac{|UB_i - R_i| + |R_i - I_i|}{|UB_i - R_i| + |R_i - N_i|} \\
&= \frac{|UB_i - R_i| + \alpha \cdot |R_i - N_i|}{|UB_i - R_i| + |R_i - N_i|} \\
&= \alpha \cdot \frac{\frac{|UB_i - R_i|}{\alpha} + |R_i - N_i|}{|UB_i - R_i| + |R_i - N_i|} \\
&= \alpha \cdot c_a
\end{aligned} \tag{5.3}$$

In Equation (5.3), 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.

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 5.3, 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:

$$prior_i = \delta - |N_i - N_j| \tag{5.4}$$

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 (5.4), 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 (5.3) 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.

5.6.3 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. 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 5.5.2, 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 5.4 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 5.4(a) and in Figure 5.4(c), we examine what are the possible case(s) for the target to be equal to value o whereas, in Figure 5.4(b) and in Figure 5.4(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:

$$P(O_p = o | R_i = x) = \frac{1}{\delta - x} \tag{5.5}$$

$$P(O_p <= o | R_i = x) = \frac{o - x}{\delta - x} \tag{5.6}$$

$$P(O_p = o | R_i <= x) = \frac{x}{x(\delta - x) + \frac{x(x-1)}{2}} = \frac{2}{2\delta - x - 1} \tag{5.7}$$

$$P(O_p <= o | R_i <= x) = \frac{x(o - x) + \frac{x(x-1)}{2}}{x(\delta - x) + \frac{x(x-1)}{2}} = \frac{2o - x - 1}{2\delta - x - 1} \tag{5.8}$$

From Equations (5.5) to (5.8) 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.

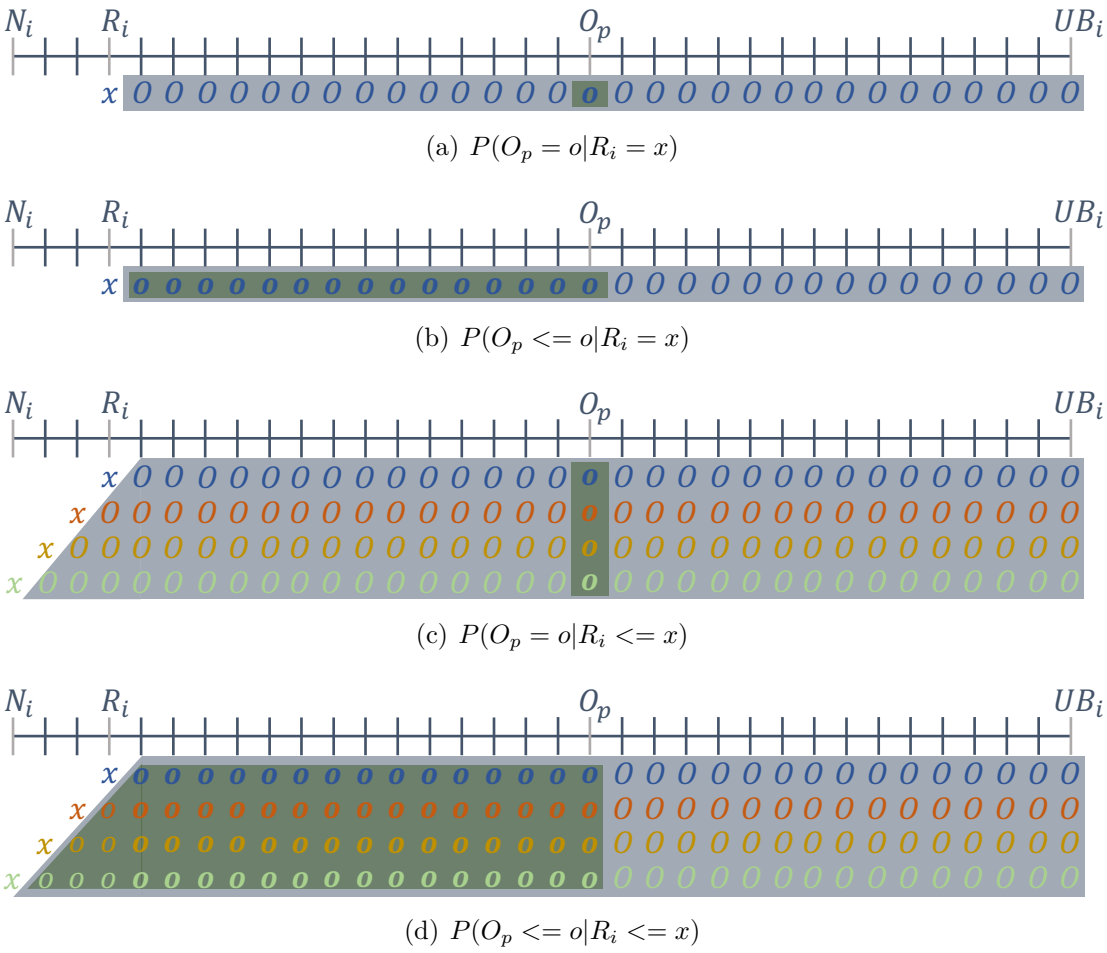
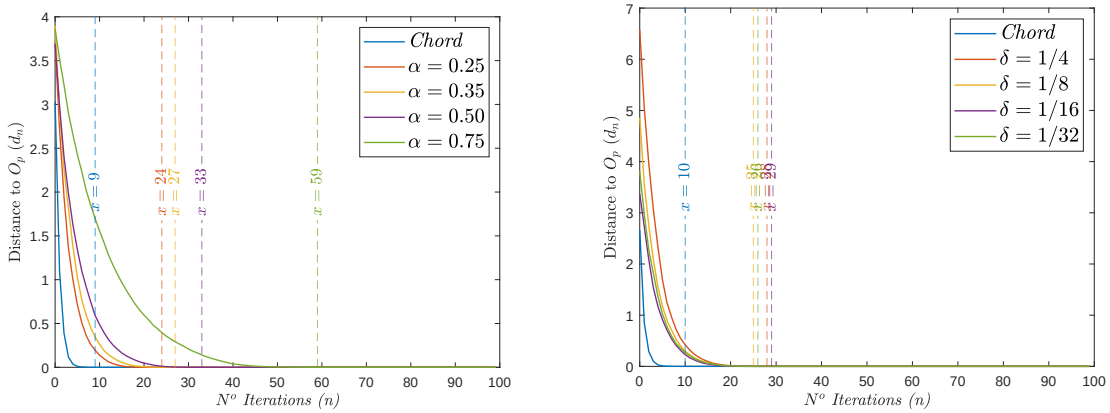


Figure 5.4: *Probability Calculation:* We mark with O the addresses that O_p can obtain and with o and x the explicit value(s) of O_p and R_i, we examine.

5.7 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 analyse how they affect performance and correctness. We simulate different fractions of adversaries in the network and analyse 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



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

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

Figure 5.5: *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.

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.

5.7.1 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.

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. A detailed description regarding the provided material can be found in Appendix A.

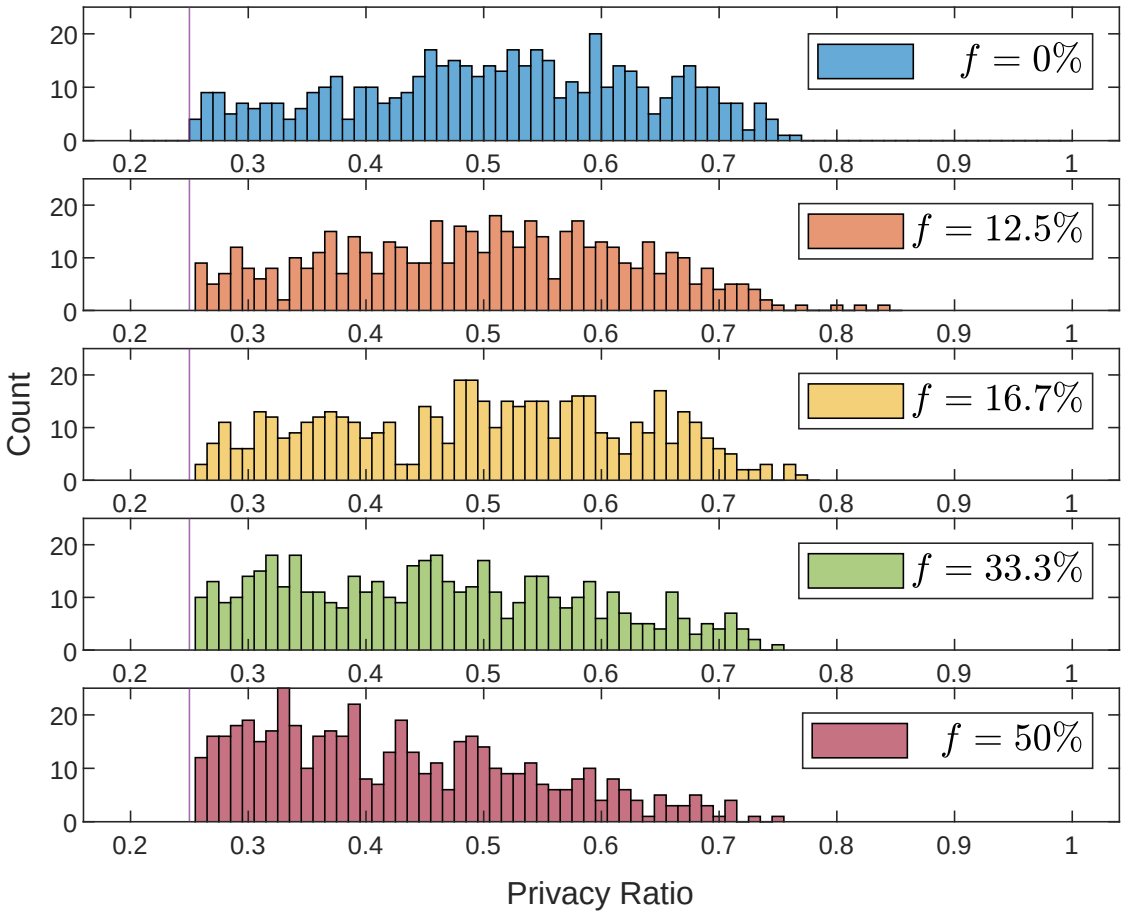


Figure 5.6: The posterior and prior knowledge rate for different fractions of colluding adversaries.

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.

5.7.2 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 5.5(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 5.5(b). We observe only minor difference in the average number of hops, which matches the prediction produced by Equation (5.2) 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 5.4, we execute IRIS by varying the fraction of colluding adversaries. Following the analysis in Section 5.6.2, 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 5.6 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 5.6, 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 5.7, 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 5.8, we validate out implementation of Iris by plotting the probabilities we get from experiments (in blue) against the probability expressions we get from Equations (5.5), (5.6), (5.7) and (5.8) (in yellow). We observe that for Figures 5.8(a) and 5.8(c) the data from our experiments fully confirm our calculations. For Figures 5.8(b) and 5.8(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 5.8(b), we observe that the deviations are getting smaller as x gets bigger. In Figure 5.8(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 5.8 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.

5.8 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.

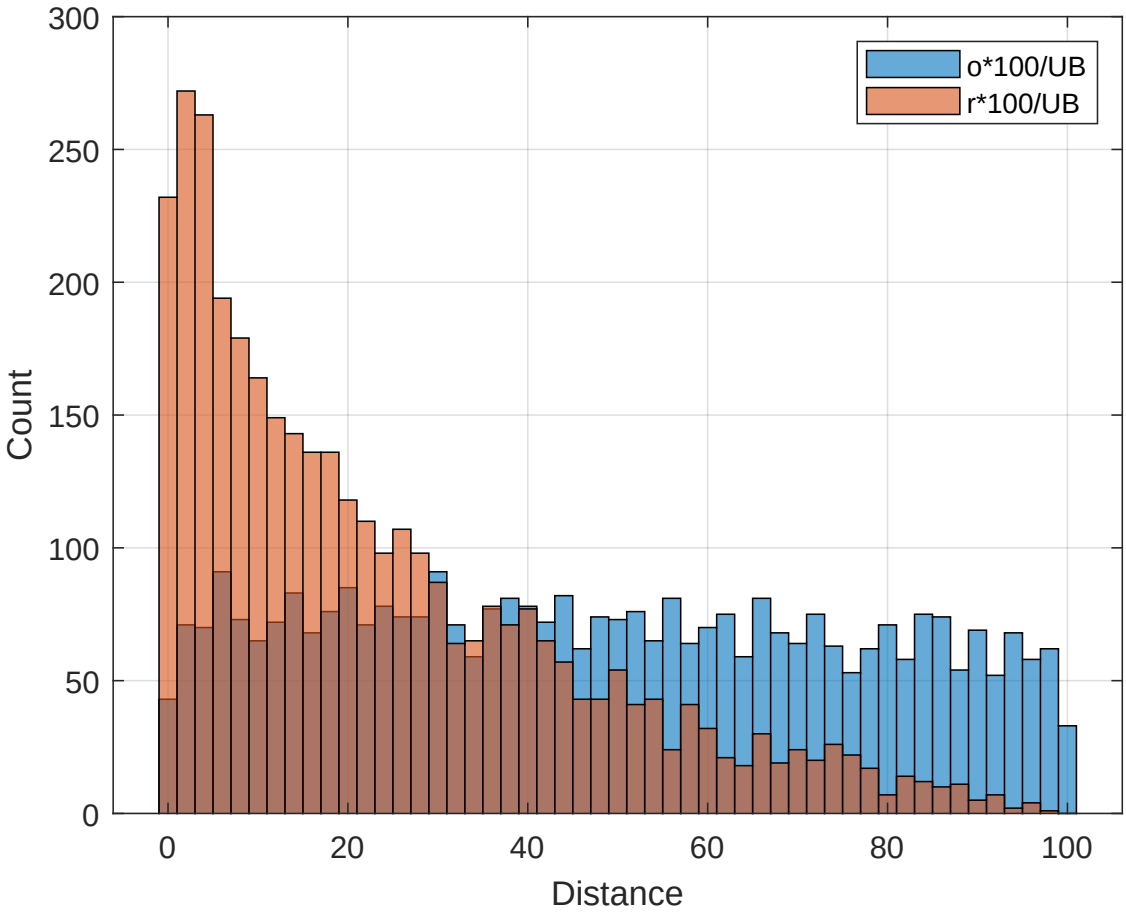
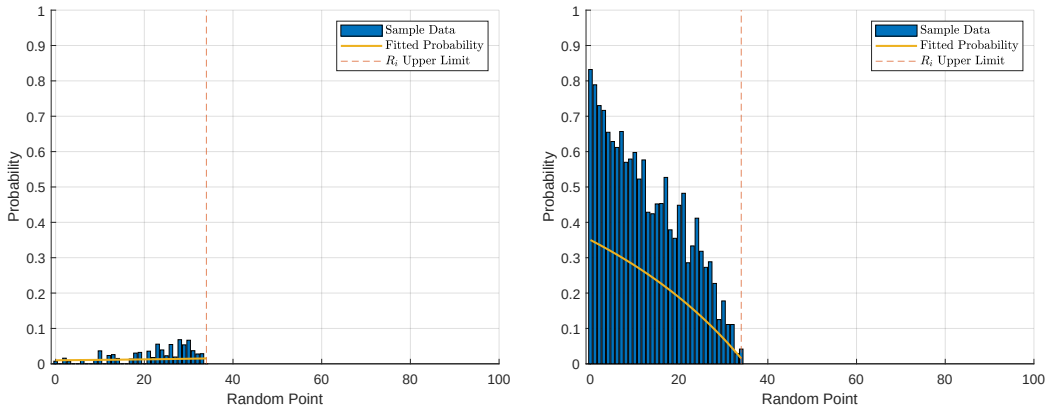


Figure 5.7: The distance that queried nodes have to the target and to the randomly picked point expressed as a percentage of the δ value.

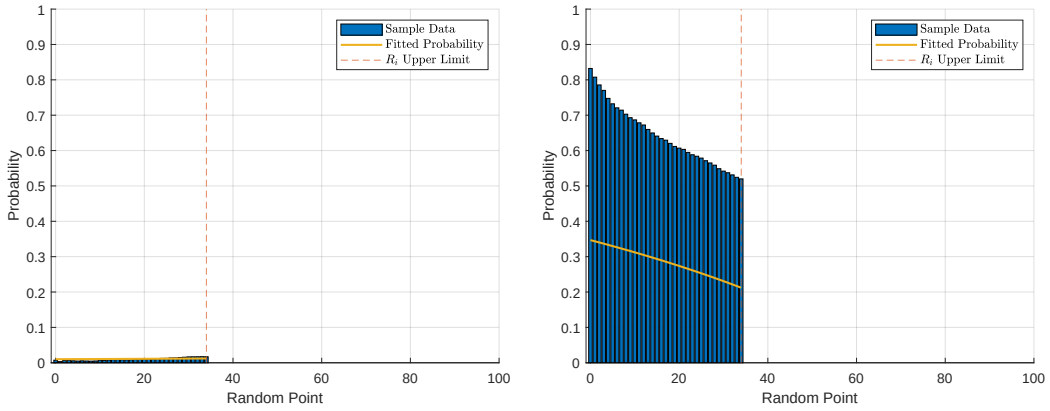
5.8.1 Selection of α and δ Parameters

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 5.5 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 5.5(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



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

Figure 5.8: *Probabilities for $O_p = 35$:* The probabilities we get from our prototype implementation.

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 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 [104]. 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 5.5(a) but we can see it more clearly in Figure 5.9,

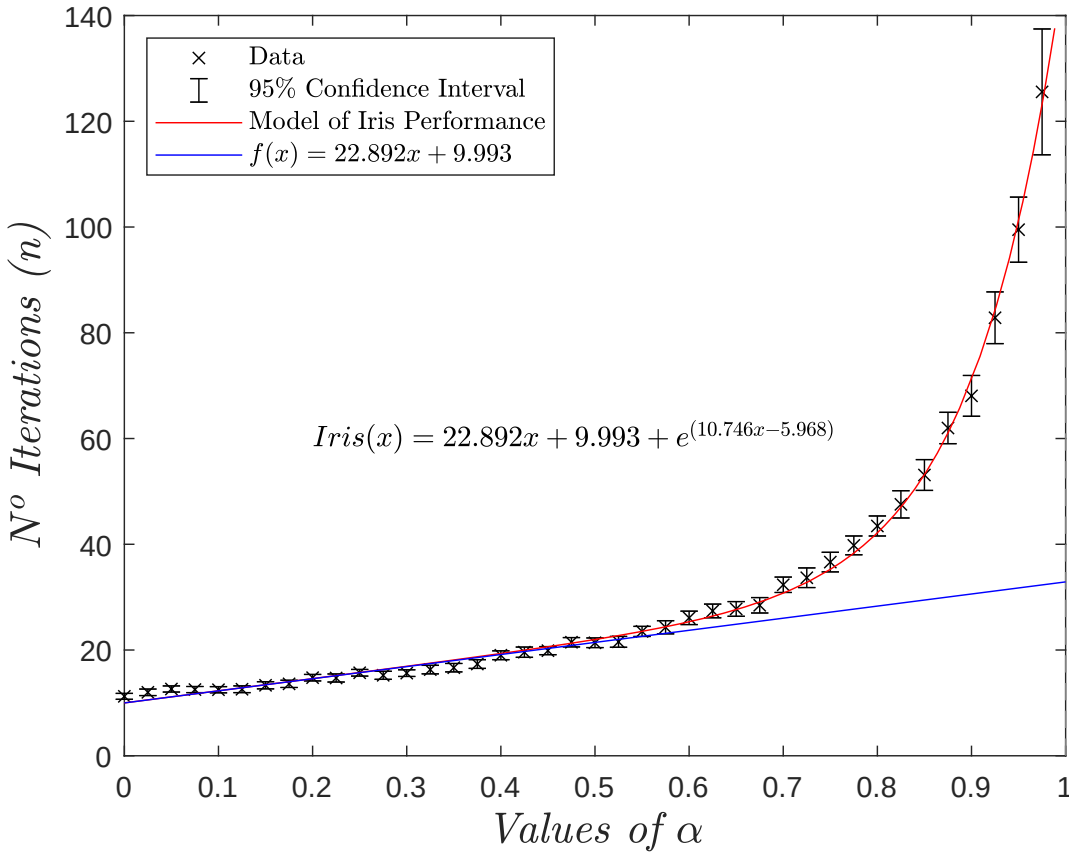


Figure 5.9: The number of steps that are needed to reach the target for different values of the α parameter.

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.

5.8.2 Other P2P Architectures

Iris can be directly applied to any existing system that implements the Chord protocol, e.g., [37–39, 41, 84–88]. 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 [5], Tapestry [24], Pastry [23] and CAN [22]. 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 [105]. 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.

5.8.3 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 $1 - \alpha$.

5.9 Summary

In this Chapter, 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.

6
Related Work

Contents

This chapter focuses on related work that addresses security challenges in structured P2P networks and intends to position our proposals against them. We start with a brief introduction highlighting the use of structured P2P networks nowadays, which demonstrates the importance of further pushing the bar of their development. Structured P2P networks underpin many newly introduced blockchain technologies; thus, their security and privacy guarantees are of utmost importance. We continue with general related work focusing on the three main categories of attacks in P2P networks. We then discuss proposals that can enable a holistic and secure search in structured P2P networks. Subsequently, we discuss authentication schemes applied on structured P2P networks. Finally, we present already proposed

privacy metrics applied on decentralised networks, focusing on the ones used on Chord, and we describe other privacy decentralised architectures already proposed in the literature.

6.1 Structured P2P Networks in 2020s

Structured P2P networks were introduced in the early 2000s, providing an efficient decentralised key-value store interface. Over the years, various platforms have utilised these networks to address node discovery and routing challenges. Notable examples include file-sharing applications such as Dat [106], BitTorrent [55], and IPFS [97], decentralised web search engines like Seeks and YaCy [36], and anonymous communications, such as I2P [107]. Additionally, they have been applied in mesh networks like Cjdns [8] and Yggdrasil [9], as well as in communication stacks such as libp2p [11] and GNUnet [10].

Nowadays, there is a push towards more decentralised infrastructures that empower users to regain control over their data and reduce the dominance of large corporations in existing deployments. New technologies, such as blockchain-based systems, have emerged to realise such a vision, which, based on a consensus algorithm [108], allow independent nodes to form a ledger of accepted blocks without a trusted third party getting involved. Structured P2P networks and the applications that leverage them provide essential functionalities—such as lookup, routing, and node discovery—to blockchain systems and decentralised applications (DApps).

The libp2p communication stack was initially developed to support IPFS, a content-addressable protocol that allows nodes to locate and retrieve files from the network. It has since evolved into a standalone product widely used by other systems, including the Ethereum Global Network [109], Swarm [110], and Filecoin [111], the latter two being initiatives aimed at incentivising storage services. BitTorrent, a widely adopted file-sharing protocol, began with a centralised tracker for node discovery. It later transitioned to a decentralised architecture with the adoption of Kademlia-based distributed hash tables (DHTs), specifically the Mainline DHT [112] and Vuze DHT [113]. Today, BitTorrent has evolved into a suite of diverse

products, enabling the BitTorrent File System (BTFS), which offers distributed file storage, and the BitTorrent Chain (BTTC), which facilitates cross-chain blockchain interactions. Storj [114], a pioneering decentralised storage network, employs several S/Kademlia [115] mechanisms—an adapted version of the Kademlia protocol—to enable secure routing and node lookups. On the other side, the New Kind of Network (NKN) [41] implements the Chord protocol for routing and node discovery.

These examples highlight that structured P2P networks remain a crucial component of many modern technologies, emphasising the ongoing need to enhance their security and privacy guarantees.

6.2 Security in Structured P2P Networks

Structured P2P present many weaknesses that are correlated with the associated protocols of the DHT scheme. Such attacks can be classified into three different groups: the Sybil attacks, the eclipse attacks and the routing & storage attacks. In the following subsections, we cite the existing in the literature proposals that recommend defences against the aforementioned attacks. We also discuss both their advantages and disadvantages.

6.2.1 Sybil attack

The open-access character of P2P overlays provides the possibility for attackers to introduce a large number of bogus nodes to the network. These nodes can subvert protocols based on redundancy. This is normally referred to as a Sybil attack. An effective defence against the Sybil attack involves using digital certificates, provided that each node is associated with one certificate and that there is a way to discern if an entity requesting a certificate is a Sybil attacker or not.

To issue and distribute the provided certificates to the participant nodes in the network, a few proposals assume the existence of a single certifying authority (CA) [116]. The CA is paired with the nodes as part of an initialised configuration phase out-of-band . This architecture is suitable for networks which scale to a limited extent, such as topologies, that encompass devices belonging to the same ownership

domain (e.g., owned by the same person or organisation), which are distributed within a distinct geographic area (e.g., a home or a commercial establishment). Despite its easiness and simplicity, this approach is considered problematic when applied to open-access networks that scale to an extended level and incorporate nodes presenting high mobility and intermittent connections (e.g., smart cities). This is because it introduces a single point of failure as it necessitates a link between the CA and the joining node that might not be available.

To overcome the obstacles posed by a single CA scheme regarding availability, schemes that make use of distributed CA based on threshold cryptography have been proposed [117]. In these approaches, a subset of the participating nodes can jointly create a digital signature to issue certificates in the presence of malicious nodes. These schemes present scalability and performance problems. These occur, firstly, due to the dynamic population in P2P networks, where nodes continuously join and leave (churn). Secondly, because of the communication overhead that is imposed on each node to obtain one signed certificate.

Different approaches that authenticate nodes through a consensus procedure without relying on certifying authorities have also been proposed in the literature [118, 119]. These schemes provide Byzantine Fault Tolerance to the network by guaranteeing correct authentication for nodes if no more than a specific portion of the total parties are malicious or faulty. The significant constraint of these schemes is the fact that they can provide their security primitives when the majority of peers in the network follow the protocols correctly. This assumption defines a strong adversary model for the system that limits its application to P2P networks, which follow an 'open-access' policy and incorporate devices that commonly present failures.

Alternative schemes employ reputation-based mechanisms [120]. In these proposals, nodes decide whether to collaborate with other peers in the network based on their ranking. Peers acquire their ranking by the feedback they get from other nodes in the network. However, nothing prevents a node from acting maliciously despite any good reputation that it might have constructed based on the correct way it was operating in the past. Finally, other schemes to defend

against Sybil attackers try to distribute queries across the network [121] or leverage social graphs for routing messages [122].

In this thesis, we do not address Sybil attacks; that is, we do not build any specific mechanism to identify or prevent attackers from operating Sybil identities. In SeCaS, we rely on an out-of-band mechanism to identify the benign nodes in the network. In Themis, the peers can create as many identifiers as they want in the network. This can be beneficial in some scenarios, e.g., when a machine runs services that must be associated with separate applications with distinct identifiers, as in the Serverless Computing paradigm. In scenarios where this is considered problematic, the strong message linkability and authentication primitives, as well as the communication address of each identity Themis provides, can be leveraged to support the design of a Sybil resistance mechanism. In Iris, we take a thorough approach, considering a strong adversary that can operate several Sybil identities in the network. The privacy metric we introduce allows us to reason about the privacy of an iterative lookup process even when Sybil adversaries take part in the lookup.

6.2.2 Eclipse attack

Nodes in P2P overlays have a partial view of the network through links that maintain with other peers in their routing tables. In an eclipse attack, the attacker tries to corrupt the routing tables of honest nodes by filling them with references to malicious nodes. Possible ways to exploit such an attack is by updating the routing tables with incorrect entries or by subverting the network proximity measurement mechanism.

A basic defence against eclipse attack is to constrain the identifiers of nodes that can be used in routing tables, as is done in Chord [4]. This is a valid mitigation only if node identifiers are random and stable, and malicious nodes are spread over the identifier space. This can be achieved either by using a central authority [116] or by introducing periodic churn every time a node joins the overlay [123].

Hildrum et al. [124] proposed the use of redundant routing table entries to prevent eclipse attacks, expecting that some entries will be honest and thus sufficient to allow successful delivery of messages. Castro et al. [116] suggested the use of two

routing tables: one optimised with network measurements and the other constrained and used in case of a test failure. Both Condie et al. [125] and Awerbuch et al. [123] improve this method by periodically resetting the optimised table entries and continuously introduced churn. Finally, Sign et al. [126] proposed the control of the in-degree and out-degree of overlay nodes via anonymous auditing. In general, the outlined proposals show that providing mitigations against the eclipse attack involves a tradeoff between performance and complexity.

6.2.3 Routing & storage attacks

Routing & storage attacks cover various attacks where malicious nodes do not follow the routing and storage protocols correctly. Possible attacking scenarios include the interruption of the propagation of messages by routing them to incorrect or malicious nodes and the provision of corrupted responses to lookup requests.

Assuming that nodes are not able to select their location in the identifier space, data replication can achieve an adequate defence against storage attacks. The various approaches in the literature, follow different rules concerning where to store replicas across the network. On one hand, replicas are stored in nodes that are numerically close in the identifier space [116, 124], simplifying their maintenance in case of mutable data. On the other hand, replicas are spread over the identifier space at random or equally spaced locations [24, 127], requiring malicious nodes to have greater control of the network to perform successful attacks.

Concerning routing attacks there exist many studies in the literature that tackle the problem by applying different strategies [128]. Firstly, redundancy-based strategies send multiple messages throughout the network to augment the probability of reaching the responsible peer. One way to implement these strategies, is through multiple paths [116] that use several paths to send a message from a source to a destination. Another way, is through wide paths [124] that attempt to send messages to a group of peers (quorums) in each step. Multiple paths are a better match for replicas spread over the identifier space, while wide paths are suitable to replicas stored at numerically close nodes. Secondly, different approaches [129, 130]

provide secure routing based on misbehaviour node detection, where the requester can detect malicious nodes in each hop, thus, verifying some invariant of the system. Finally, reputation systems [131] and social networks [132] have also been used in order to provide knowledge and improve the routing process.

6.3 Secure Holistic Service Discovery

The distributed and scalable nature of structured P2P networks led to their adoption as a service discovery technique in different networks [8–11]. SeCaS and Themis, introduced in Chapters 3 and 4, respectively, propose mechanisms that enhance the security guarantees provided when leveraging DHT schemes, thereby supporting their use as a foundation for general-purpose decentralised application deployment. Similar to other approaches, such as libp2p [11], our designs assume that in the DHT, nodes do not store the actual data but only the node identifiers of those nodes that claim to hold that data. The actual data is maintained locally in the memory of the peer that generates it. This strategy broadens the range of potential applications that can be built on top of a DHT, as it eliminates restrictions related to the data types of applications utilising it for deployment.

To enable the registration of application-specific data in the DHT—allowing them to be represented as addresses in the address space—IPFS employs a content-addressed scheme that creates objects based on the cryptographic hash of the content [133]. This allows peers to verify the integrity of the data. Our proposed naming scheme for data registration in the DHT, termed *capabilities*, is designed for the Internet of Things (IoT) domain but is versatile enough to be applied in various applications. It emphasises a more flexible way to search for data in the DHT, which operates with exact match queries. Additionally, the ability to register multiple objects per data point introduces a redundancy factor that enhances data availability. *Capabilities* provide a way to represent all the services that nodes can contribute to the network. Various methods have been proposed in the literature to denote the competence of devices within IoT ecosystems. Semantic ontologies have been introduced to address challenges related to interoperability and heterogeneity [134–

136]. Despite the effectiveness of semantic models in supporting resource and service discovery in IoT environments, the multiple annotations they use to describe specific services do not align well with the exact term searches of DHT schemes. To facilitate collaboration among IoT devices in industry settings, companies have adopted more programmable approaches to coordinate and control devices within home automation platforms [46, 61]. In these frameworks, devices are abstracted on their underlying commands and attributes. While these structures achieve a detailed representation of device capabilities, they explicitly target specific services. This limitation means that they do not allow for a more advanced search capability—such as moving from a general to a specific service or vice versa. In contrast, our proposal benefits from improved retrieval flexibility during search operations.

The flexibility provided by *capabilities* contributes towards the adoption of structured P2P networks in ubiquitous environments, such as the Internet of Things (IoT) and Serverless Computing, that consist of nodes that offer diverse services.

IoT Devices Collaboration: SeCaS leverages a DHT scheme to achieve scalability while avoiding spatial constraints and network flooding. These limitations are encountered in other distributed resource discovery frameworks that rely on the broadcast communication channel [137] and social links [138] to advertise available services among IoT devices. Compared to other proposals that also rely on a structured P2P overlay to enable resource discovery in IoT environments [64, 65, 139], SeCaS guarantees secure and reliable collaboration among devices by solely using the DHT structure without requiring peers to be organised into different neighbourhoods or introducing additional organisational layers to the system architecture. With its *capability* representation and implemented protocols, SeCaS is the first to leverage P2P networks for the safe discovery and exchange of services provided by numerous IoT devices belonging to the same ownership domain, all without increasing the deployment cost of the system.

Serverless Computing Mesh: Applying a P2P architecture for secure communication between services in serverless computing environments is a relatively new area of research. Existing service meshes [140–144] follow a centralised architecture where

dedicated registries form the control plane, coordinating the microservice proxies in the data plane. Consul [145], on the other hand, adopts a more decentralised approach by utilising the Raft consensus protocol, which allows multiple servers that maintain all the information regarding the state of a cluster to elect a leader that processes all the queries. The study in [79] also relies on a service registry to store data about registered services; the registry operates like a DNS server, resolving human-readable names to service hashes that can be queried through a DHT structure. In contrast, Themis provides a holistic decentralised design where information regarding nodes and the application-specific data they offer, *i.e.*, microservices, are stored directly on the DHT by the nodes themselves. Themis leverages the DHT so as to remove the need for a service registry and allow nodes to discover one another and the services they need in a completely decentralised manner.

SeCaS and Themis leverage the DHT to facilitate service discovery among IoT devices and microservices, primarily addressing the security challenges that arise from this design choice. The architecture of SeCaS is built around four main pillars: authorised membership, access control, secure channel establishment, and private communication. Its security model focuses on collaboration rather than mere node interaction. This distinguishes SeCaS from Themis, which, similar to TLS, aims to establish a secure channel for node interaction. SeCaS guarantees both message authentication and freshness for nodes that communicate within the DHT and only establishes encrypted channels between nodes that share *capabilities*. This approach avoids the unnecessary creation of encrypted channels between nodes that solely interact for routing purposes. Another key distinction between SeCaS and Themis is that SeCaS offers an access control mechanism. This is achieved by managing who can join the network and by allowing individual nodes to control access to their resources using tokens. In contrast, Themis accommodates a decentralised network where devices from different domains must interact over a secure, authenticated channel, leaving it up to the applications that use this channel to implement additional security features like membership and access control.

6.4 Authentication Schemes

The distributed nature of structured P2P networks, *e.g.*, Chord [4, 42], Kademlia [5], Can [22], Tapestry [24], exposes them inherently to important attacks [13, 146]. To solve the problem of authentication in distributed systems, many proposals in the literature use a single root of trust, commonly delegated by a certificate authority (CA) [16, 116, 147]. The nodes, as a part of a configuration procedure, obtain from the CA a certificate that effectively binds the node's public key and its respective identity in the overlay. Identity-based cryptography (IBE) has also been used to provide authentication in P2P networks [148–151]; a trusted third party, called the private key generator (PKG), generates a private key that corresponds to the public parameter that is used as the node's identifier. Other peers in the network can encrypt the messages they want to send to the respective node based on its identifier. Each peer to be able to decrypt the messages has to obtain from the PKG the corresponding to its identifier secret key.

Other works follow a more distributed approach to provide authentication. Schemes such as [117, 152], do not rely on a single CA. Threshold cryptography is applied to allow a subset of the participating nodes to jointly create a digital signature. The created digital signature is used as a certificate in the presence of malicious nodes. Different approaches that authenticate nodes through a consensus procedure have also been proposed in the literature [118, 119]. These schemes provides Byzantine Fault Tolerance, allowing a peer to authenticate another by consulting a group of peers in the network. Alternative schemes employ reputation-based mechanisms [120, 153]. In these proposals, nodes decide whether to collaborate with other peers in the network based on their ranking. Peers acquire their ranking by the feedback they get from other nodes in the network.

Requiring a preliminary pairing step for every node with a third party that can act maliciously, can create challenges in open, large-scale networks. In these environments, devices owned by different entities may interact, making it difficult to agree on a common root of trust. Additionally, nodes are expected to communicate with one another in dynamic peer-to-peer (P2P) manner, and a connection to a

central server may not always be available. This is why THEMIS offers authentication without requiring any prior registration for nodes before they join the network. THEMIS enables nodes to independently generate a key pair and establishes an identity based on the hash of the node's public key (self-certifying identities [154]). Its lower-level protocols effectively bind a symmetric key to this identity, enabling secure communication between nodes. THEMIS to provide authentication, does not require nodes to communicate with anyone other than the specific node they wish to reach, thus minimising the communication overhead needed to establish an authenticated connection. Furthermore, THEMIS does not make any assumptions about the behaviour of the nodes. It ensures strong linkability between a node's identity and the messages sent, meaning that nodes can be held accountable at all times, and any malicious actions can be traced back to them.

The self-certifying identities that THEMIS uses, obliviates the need for a certificate to associate the identity that a peer has with a specific public key, as done in protocols like TLS [155]. The certificate can be issued by a Certificate Authority (CA), or be self-signed by the peer, as done in [156] and in [157]—that embeds in the initial handshake a signed identity payload to authenticate the static public key of the Noise_XX [158] protocol. The lack of a certificate indicates that there is no strong connection between a network identity and a real-world identity. This allows every node to claim any unique identity they desire. However, THEMIS guarantees that only nodes in control of a specific identity—meaning they possess the secret key linked to the corresponding public key—can associate the messages they send with that identity. This design provides a decentralised and scalable solution for identity management that is a key requirement for large-scale and open deployments.

Self-certifying identities have been utilised in P2P architectures in the past. Works such as those by Baumgart et al. [95] and Prunster et al. [96] focus on defending against eclipse and Sybil attacks and propose the use of a hash of a public key to generate a node's identity within the network. This method ensures the integrity of messages exchanged between nodes in the overlay. THEMIS builds on these works by providing additional security guarantees, including confidentiality

and message authentication/linkability. THEMIS enables nodes to establish a secure communication channel based on a symmetric key. This approach significantly reduces the computational burden on peers compared to the continuous need for asymmetric cryptographic computations.

Unlike other protocols [159, 160] that allow nodes to establish a secure communication channel, THEMIS does not require peers to be synchronised. Compared to [157], which also allows nodes to create a secure channel with a specific node in the overlay, THEMIS offers strong authentication for all exchanged messages. This feature allows peers to identify both the sender and recipient of messages at every step of the communication process. In SECIO [161], an earlier secure transport protocol for libp2p [11], peers utilise self-certifying identities without relying on traditional certificates. However, THEMIS provides stronger security guarantees than SECIO, such as allowing key confirmation and resisting identity-misbinding attacks.

6.5 Privacy Metrics

In the literature, many works have studied the privacy guarantees of decentralised systems [162]. Several metrics have been proposed to measure privacy [163]. The selection of a good privacy metric necessitates careful thinking of the specific use case and the kind of property the metric measures, e.g., uncertainty, indistinguishability, information loss/gain.

In the case of structured P2P networks, when measuring the privacy of a query, we need to consider the routing policy that is applied, which states that to guarantee convergence on a bounded number of hops, the requester needs to get closer to the target at every step. This rule affects who the requester will query and for what, resulting in adversaries having different estimates regarding the target, even if the requester applies an algorithm to conceal the real target.

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 [92, 164] or by normalising it by the best possible value [165]. Using, however, a metric that quantifies the uncertainty of the adversary results in the calculated score to

change between nodes, according to the distance the node has to the target of the request. Adopting such a metric or other metrics such as differential privacy [166, 167], which measure the possibility of the adversary to distinguish between two adjacent datasets, to measure query privacy in structured P2P networks such as Chord networks does not reflect the average privacy achieved but the worst-case scenario, giving a wrong flavour regarding the privacy achieved for the query in total, i.e., for all the iterations.

In [165], the authors 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 [168] to calculate the information leakage from compromised routing tables in DHT networks the authors use the rate between the posterior and the prior entropy after and before a compromise has occurred. (α, δ)-*privacy*, introduced in Section 5.4, builds on the last metric by replacing entropy with the size of the possible set to which the target can belong, which eliminates the need for complex calculations. This approach aims to provide adequate granularity to reason about the privacy achieved throughout a complete query.

6.6 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 [99, 103, 169–173] 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 [92, 93, 122, 171, 174–177]. 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 [101] 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 [100, 178, 179] organise 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 [180] 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 [5] DHT. IPFS is also in the process of performing a privacy upgrade by implementing Double Hashing [181]. Double Hashing has the requester querying for a prefix of the target identifier and receiving 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.

7 Conclusion

Contents

Structured P2P networks provide a decentralised and scalable infrastructure that can ease application development in many different environments, such as the Internet of Things and Serverless Computing. Regardless of the specific application they serve, these networks offer to the participating nodes the possibility to discover other peers, by allowing them to obtain their communication information (*e.g.*, their IP and port number), and to search for data or attribute tags stored in the network. This thesis provides solutions to security and privacy problems that structured P2P networks encounter, to guarantee a secure and private operation for any application that adopts them in its design.

7.1 Review of Principal Contributions

To allow structured P2P networks to be used by different applications, we need to establish an identity for the objects that can be searched in the network and guarantee accountability for the exchanged messages among nodes. SeCaS, presented

in Chapter 3, accomplishes a holistic and secure object search by initially proposing a way to name heterogeneous services compatible with the way the structured P2P networks operate, and a series of protocols that guarantee message accountability and facilitate authorisation. SeCaS key contributions allow the secure leverage of any structured P2P network in the IoT ecosystem, allowing device owners to take full advantage of the capabilities of their devices through synergy and giving them back control over their data, as they do not need to rely on a central managerial entity—that third parties often administrate—to mediate between their device communications.

SeCaS, similar to other works, to provide authentication relies on a route of trust that nodes establish out-of-band, such as a certificate authority. This authentication method is an easy-to-implement approach when we deal with an one ownership domain. However, it gets challenging when devices of many different owners that need to agree on a common third party participate in the network or when communication with an external entity cannot be established. Themis framework, discussed in Chapter 4, tries to address this problem. Our framework provides a notion of decentralised identity verification, together with a series of actions related to node communication and management—*e.g.*, `store`, `find`, and `join`, forming a thorough, decentralised and secure communication solution in a variety of scenarios that demand point-to-point interaction. A use case example is a secure service mesh communication network needed in data centres and companies that require dynamic load balancing and extensibility in which we describe precisely how Themis can be applied and the performance it can achieve.

One downside of peers having verifiable identities for authentication and secure communication is that it enables tracking of peer activities. In structured P2P networks such as Chord, the peers must disseminate the search term to other network peers to perform a search. Techniques such as obfuscation do not work with the binary search that these networks follow. To address privacy concerns while supporting strong authentication, *i.e.*, allowing each peer to have a "valid" verifiable identity, in Chapter 5 we propose Iris, a privacy-preserving object search

algorithm that allows nodes to use the network without allowing other peers to track their activity or search patterns. We also introduce (α, δ)-*privacy*, a notion allowing us to reason about the privacy that is achieved by privacy-enhancing algorithms in Chord, including our own, even in the presence of a strong colluding adversary.

7.2 Discussion & Future Work

Our contributions bring us closer to enabling the use of structured P2P networks in an application-agnostic fashion that provides by design security and privacy guarantees. The *capabilities* naming scheme we introduce in Chapter 3, decouples the adoption of structured P2P networks by one specific application, allowing us to identify and search heterogeneous services flexibly. In this way, we give control back to the users over their data by eliminating the need for a trusted third party in the application design. Iris also puts the users at the centre of decision-making by allowing them to define the level of privacy they want each time for their query.

The protocols that constitute SeCaS and Themis address security problems in peer-to-peer communication. By guaranteeing security primitives at this level, we facilitate the design of applications built on top, which inherit the security they provide. In this way, the development of secure applications can focus more on the security properties related to the specific task they serve, such as resistance against Sybil attacks. SeCaS and Iris allow leveraging existing P2P networks. In this way, we can facilitate the improvement of already deployed infrastructures that need to be revised to incorporate security and privacy primitives in their design. With Themis, we move closer to an open-access global network where peers can be authenticated without being obliged to perform any preliminary step to be assigned an identity.

Our solutions solve many security and privacy problems in the design of structured P2P networks; there are still, however, opportunities for further enhancements. With Iris we solve an important privacy problem in structured P2P networks, *i.e.*, preserve the privacy of the search patents of peers. Yet, there is room to provide a more private decentralised infrastructure when structured P2P networks are adopted. The ad-hoc manner adopted by these networks results in the leakage of

the routing table of peers to the rest of the nodes; when a peer cannot resolve a request, he relays the requester to another node after advising his routing table. In this way, the routing information is revealed to the requester, who can leverage this information to infer the relationships between peers.

Future steps can work towards the development of a mechanism that guarantees routing privacy, that is, the requester will not be able to track relationships between nodes based on the nodes they have on their routing tables. The mechanism will have to guarantee that even if nodes chose to maintain in their routing table specific peers the rest of the peers in the network will not be able to infer the persistent peers connections based on the responses they get. Towards this objective, the different networks inherently built-in Themis can be leveraged to allow peers to group identities—addresses from the address space—they control and guarantee that their association can be determined only by specific peers.

Even if there are still steps we can take (and there always will be) to improve the security and privacy of structured P2P networks, this thesis makes an essential step towards their adoption in today's emerging technological areas in a more private and secure context. Our practical solutions contribute to a more private and secure decentralised future, by redefining how entities communicate.

Appendices

A

Iris Implementation and Benchmarks

Contents

In Chapter 5, 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. Here we provide more details on the artifacts of this work that 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, in the provided artifacts repository we also include the scripts used to create our plots. The provided artifacts allow peers to reproduce the experiments we present and build upon them, inspiring further research and development in this area.

A.1 Artifacts

A.1.1 Structure

In the repository we provide the code and the data we use in Chapter 5. 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. The network generation is randomised, so to fully reproduce our results we share the data we generated and used for our plotting.

In Figure A.1 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 5.5. 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 5.4.

- `Chord_Lookup.m` implements the Chord `lookup` protocol, as described in Section 2.3.1. Given a specified target object, the code returns the address of the next node to be queried.

A.1.2 Set Up

Access

The complete artifacts and most recent version of the code can be accessed at https://github.com/angakt/iris. The artifacts have gone through an artifacts evaluation process for the NDSS 2025 Symposium, being awarded the *available*,

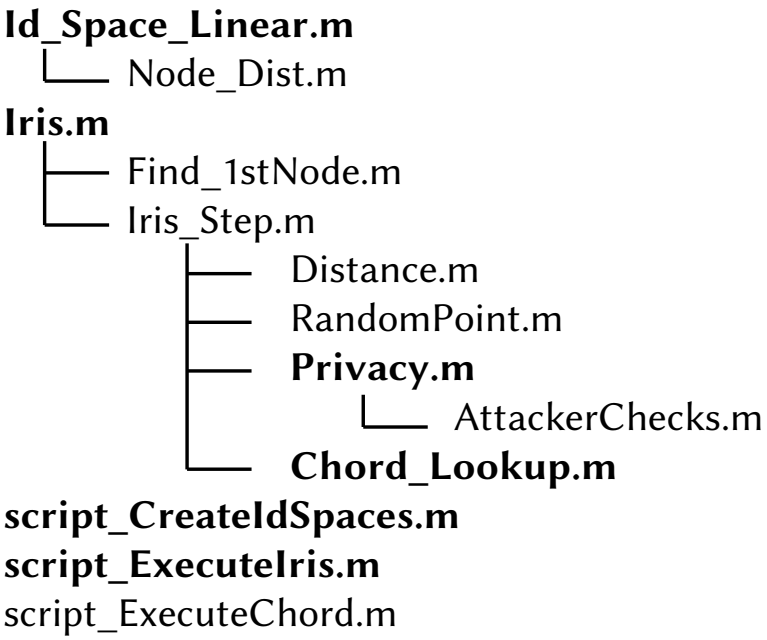


Figure A.1: *The schema of Iris's code base.*

functional and *reproduced* badges. A snapshot of the release based on which the artifacts evaluation has been performed has been uploaded at Zenodo and can be accessed at https://doi.org/10.5281/zenodo.14251874.

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.

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

A.1.3 Claims

The provided code is used to confirm empirically the influence of different values of the privacy parameters in Iris's convergence, the distribution of the privacy ratio among the queried nodes and the information a non-colluding attacker can deduce when being queried. The experiments performed help us get insights on:

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.

A.1.4 Execution

In Chapter 5, we perform five different experiments. The first two experiments presented in Figure 5.5, 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 5.6, confirms that Iris is an (α, δ)-*private* algorithm, whereas the last two experiments illustrated in Figure 5.7 and in Figure 5.8 examines the attacker's advantage when executing Iris.

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.

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.

Graphs

Finally, to reproduce the graphs, 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.

A.1.5 Evaluation

[Preparation]

All the experiments in Chapter 5 were executed using networks with 1000 nodes placed on an address space with 2^{23} addresses.

Experiment (E1)

[Figure 5.5(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.

Experiment (E2)

[Figure 5.5(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 5.6, 5.7 and 5.8 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.

Experiment (E3)

[Figure 5.6] [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.

Experiment (E4)

[Figure 5.7] [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.

Experiment (E5)

[Figure 5.8] [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.

References

[1] Apostolos Malatras. 'State-of-the-Art Survey on P2P Overlay Networks in Pervasive Computing Environments'. In: *Journal of Network and Computer Applications* 55 (2015), pp. 1–23.

[2] Wikipedia Contributors. *Gnutella - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Gnutella&oldid=850434913`. [Online; accessed 05-October-2023]. 2018.

[3] Wikipedia contributors. *FastTrack - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=FastTrack&oldid=1122779074`. [Online; accessed 05-October-2023]. 2022.

[4] Ion Stoica et al. 'Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications'. In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160.

[5] 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*. Cham: Springer, 2002, pp. 53–65.

[6] Wikipedia Contributors. *BitTorrent - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=BitTorrent&oldid=855276731`. [Online; accessed 05-October-2023]. 2018.

[7] Forrest Pieper, Will Drevo and Colin Taylor. 'Peerchat: A Distributed, P2P Communication Network based on Kademlia'. In: (2014).

[8] cjdns Authors. *cjdns Whitepaper*. 2016. URL: `https://github.com/hyperboria/docs/blob/master/Whitepaper.md`.

[9] Yggdrasil Authors. *Yggdrasil Network*. 2023. URL: `https://yggdrasil-network.github.io`.

[10] Christian Grothoff. *The GNUnet System*. HDR. 2017. URL: `https://inria.hal.science/tel-01654244v1/file/habil.pdf`.

[11] Libp2p Authors. *Libp2p*. Protocol Labs. Oct. 2023. URL: `https://libp2p.io`.

[12] Dan S. Wallach. 'A Survey of Peer-to-Peer Security Issues'. In: *Software Security — Theories and Systems*. Berlin, Heidelberg: Springer, 2003, pp. 42–57.

[13] Guido Urdaneta, Guillaume Pierre and Maarten Van Steen. 'A Survey of DHT Security Techniques'. In: *ACM Computing Surveys* 43.2 (2011), pp. 1–49.

[14] Angeliki Aktypi and Kasper Rasmussen. 'Iris: Dynamic Privacy Preserving Search in Authenticated Chord Peer-to-Peer Networks'. In: *Proceedings of the 32nd Annual Network and Distributed System Security Symposium*. NDSS '25. 2025.

[15] Angeliki Aktypi et al. '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. Vienna, Austria: Association for Computing Machinery, 2022.

[16] Angeliki Aktypi, Kubra Kalkan and Kasper B. Rasmussen. 'SeCaS: Secure Capability Sharing Framework for IoT Devices in a Structured P2P Network'. In: *Proceedings of the 10th ACM Conference on Data and Application Security and Privacy*. CODASPY '20. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 271–282.

[17] Angeliki Aktypi, Jason R.C. Nurse and Michael Goldsmith. 'Unwinding Ariadne's Identity Thread: Privacy Risks with Fitness Trackers and Online Social Networks'. In: *Proceedings of the 1st Workshop on Multimedia Privacy and Security*. MPS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1–11.

[18] Angela Sara Cacciapuoti, Marcello Caleffi and Luigi Paura. 'Mobile P2P: Peer-to-Peer Systems over Delay Tolerant Networks'. In: *Delay Tolerant Networks: Protocols and Applications*. CRC Press, 2011, pp. 1–35.

[19] Stefan Kraxberger and Udo Payer. 'Security Concept for Peer-to-Peer Systems'. In: *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*. ACM. 2009, pp. 931–936.

[20] A Cabani et al. 'Distributed Computing Systems: P2P versus Grid Computing Alternatives'. In: *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*. Springer, 2007, pp. 47–52.

[21] J Antonio Garcia-Macias and Javier Gomez. 'MANET versus WSN'. In: *Sensor Networks and Configuration*. Springer, 2007, pp. 369–388.

[22] Sylvia Ratnasamy et al. 'A Scalable Content-Addressable Network'. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2001, pp. 161–172.

[23] 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.

[24] Ben Y. Zhao et al. 'Tapestry: A Resilient Global-Scale Overlay for Service Deployment'. In: *IEEE Journal on Selected Areas in Communications* 22.1 (2004), pp. 41–53.

[25] Wikipedia Contributors. *EDonkey2000 - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=EDonkey2000&oldid=840158081`. [Online; accessed 05-October-2023]. 2018.

[26] Wikipedia Contributors. *Freenet - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Freenet&oldid=845853900`. [Online; accessed 05-October-2023]. 2018.

[27] Choon Hoong Ding, Sarana Nutanong and Rajkumar Buyya. 'Peer-to-Peer Networks for Content Sharing'. In: *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*. IGI Global, 2005, pp. 28–65.

[28] Esther Palomar et al. 'Security in P2P Networks: Survey and Research Directions'. In: *International Conference on Embedded and Ubiquitous Computing*. Springer. 2006, pp. 183–192.

[29] Nima Jafari Navimipour and Farnaz Sharifi Milani. 'A Comprehensive Study of the Resource Discovery Techniques in Peer-to-Peer Networks'. In: *Peer-to-Peer Networking and Applications* 8.3 (2015), pp. 474–492.

[30] Wikipedia Contributors. *Napster - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Napster&oldid=851377951`. [Online; accessed 05-October-2023]. 2018.

[31] Wikipedia Contributors. *SETI@home - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=SETI@home&oldid=1178006979`. [Online; accessed 05-October-2023]. 2023.

[32] Matei Zaharia and Srinivasan Keshav. 'Gossip-based Search Selection in Hybrid Peer-to-Peer Networks'. In: *Concurrency and Computation: Practice and Experience* 20.2 (2008), pp. 139–153.

[33] Hassan Barjini, Mohamed Othman and Hamidah Ibrahim. 'An Efficient Hybridflood Searching Algorithm for Unstructured Peer-to-Peer Networks'. In: *International Conference on Information Computing and Applications*. Springer. 2010, pp. 173–180.

[34] Wikipedia Contributors. *Gnutella2 - Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Gnutella2&oldid=805351050`. [Online; accessed 05-October-2023]. 2017.

[35] Gerald Kunzmann. 'Recursive or Iterative Routing? Hybrid!' In: *Kommunikation in Verteilten Systemen (KiVS)*. Gesellschaft für Informatik eV. 2005.

[36] Michael Herrmann et al. 'Censorship-resistant and privacy-preserving distributed web search'. In: *14-th IEEE International Conference on Peer-to-Peer Computing*. 2014, pp. 1–10.

[37] Frank Dabek et al. 'Wide-area cooperative storage with CFS'. In: *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*. SOSP '01. Banff, Alberta, Canada: Association for Computing Machinery, 2001, pp. 202–215. URL: `https://doi.org/10.1145/502034.502054`.

[38] Emil Sit, Robert Tappan Morris and M Frans Kaashoek. 'UsenetDHT: A Low-Overhead Design for Usenet'. In: *NSDI*. 2008, pp. 133–146.

[39] Jeremy Stribling et al. 'OverCite: A Distributed, Cooperative CiteSeer'. In: *NSDI*. Vol. 6. 2006, pp. 11–11.

[40] 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. San Diego, CA: USENIX Association, 2004, p. 21.

[41] NKN Lab. *NKN: A Scalable Self-Evolving and Self-Incentivized Decentralized Network*. Whitepaper. 2018. URL: `https://nkn.org/wp-content/uploads/2020/10/NKN_Whitepaper.pdf`.

[42] Pamela Zave. 'Reasoning About Identifier Spaces: How to Make Chord Correct'. In: *IEEE Transactions on Software Engineering* 43.12 (2017), pp. 1144–1156.

[43] Juan A Holgado-Terriza and Sandra Rodriguez-Valenzuela. 'Services Composition Model for Home-Automation Peer-to-Peer Pervasive Computing'. In: *Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE. 2011, pp. 529–536.

[44] Soumya Kanti Datta, Rui Pedro Ferreira Da Costa and Christian Bonnet. 'Resource Discovery in Internet of Things: Current Trends and Future Standardization Aspects'. In: *World Forum on Internet of Things (WF-IoT)*. IEEE. 2015, pp. 542–547.

[45] Andrew Banks and Rahul Gupta. 'MQTT Version 3.1. 1'. In: *OASIS Standard* 29 (2014). URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.

[46] Apple. *HomeKit*. 2023. URL: https://developer.apple.com/documentation/homekit (visited on 05/10/2023).

[47] Fen Zhu, Matt W Mutka and Lionel M Ni. 'Service Discovery in Pervasive Computing Environments'. In: *IEEE Pervasive Computing* 4 (2005), pp. 81–90.

[48] Yuan Tian et al. 'SmartAuth: User-Centered Authorization for the Internet of Things'. In: *USENIX Security Symposium*. USENIX Association. 2017, pp. 361–378.

[49] Wikipedia contributors. *Facebook-Cambridge Analytica Data Scandal — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=FacebookâĂŞCambridge_Analytica_data_scandal&oldid=892459300. [Online; accessed 05-Oct-2023]. 2019.

[50] Earlence Fernandes, Jaeyeon Jung and Atul Prakash. 'Security Analysis of Emerging Smart Home Applications'. In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 636–654.

[51] Mahmoud Ammar, Giovanni Russello and Bruno Crispo. 'Internet of Things: A Survey on the Security of IoT Frameworks'. In: *Journal of Information Security and Applications* 38 (2018), pp. 8–27.

[52] Omar Alrawi et al. 'Sok: Security Evaluation of Home-Based IoT Deployments'. In: *IEEE S&P*. 2019, pp. 208–226.

[53] Frank Dabek et al. 'Towards a Common API for Structured Peer-to-Peer Overlays'. In: *Peer-to-Peer Systems II – International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2003, pp. 33–44.

[54] Brad Karp et al. 'Spurring Adoption of DHTs with OpenHash, a Public DHT Service'. In: *International Workshop on Peer-to-Peer Systems*. Springer. 2004, pp. 195–205.

[55] Bram Cohen. *The BitTorrent Protocol Specification*. 2008.

[56] David P Anderson et al. 'SETI@ Home: An Experiment in Public-Resource Computing'. In: *Communications of the ACM* 45.11 (2002), pp. 56–61.

[57] John Kubiatowicz et al. 'Oceanstore: An architecture for global-scale persistent storage'. In: *ACM SIGARCH Computer Architecture News*. Vol. 28. 5. ACM. 2000, pp. 190–201.

[58] João Pedro Fernandes Alveirinho. 'Resource Location in P2P Systems'. MA thesis. Universidade Técnica de Lisboa, Oct. 2010.

[59] Jack Sturgess, Jason R. C. Nurse and Jun Zhao. 'A Capability-Oriented Approach to Assessing Privacy Risk in Smart Home Ecosystems'. In: *Living in the Internet of Things: Cybersecurity of the IoT Conference*. IET, 2018.

[60] Daniel D McCracken and Edwin D Reilly. 'Backus-Naur Form (bnf)'. In: *Encyclopedia of Computer Science* (2003), pp. 129–131.

[61] Samsung. *SmartThings*. 2023. URL: `https://developer.smartthings.com/docs/devices/capabilities/capabilities` (visited on 05/10/2023).

[62] Google. *Weave*. 5th Oct. 2023. URL: `https://nest.com/weave/`.

[63] Danny Dolev and Andrew Yao. 'On the Security of Public Key Protocols'. In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–208.

[64] Federica Paganelli and David Parlanti. 'A DHT-based Discovery Service for the Internet of Things'. In: *Journal of Computer Networks & Communications* (2012).

[65] Simone Cirani et al. 'A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things'. In: *IEEE Internet of Things Journal* 1.5 (2014), pp. 508–521.

[66] Yong Wang and Jason Nikolai. 'Key Management in CPSs'. In: *Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications* (2017), pp. 117–136.

[67] Jun Xu, A. Kumar and Xingxing Yu. 'On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks'. In: *IEEE Journal on Selected Areas in Communications* 22.1 (Jan. 2004), pp. 151–163.

[68] Wikipedia contributors. *Key size — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Key_size&oldid=834490386`. [Online; accessed 05-Oct-2023]. 2023.

[69] Farida Chowdhury and Md. Sadek Ferdous. 'Performance Analysis of R/Kademlia, Pastry and Bamboo Using Recursive Routing in Mobile Networks'. In: *International Journal of Computer Networks & Communications (IJCNC)* 9.5 (2017).

[70] Farida Chowdhury, Jamie Furness and Mario Kolberg. 'Performance Analysis of Structured Peer-to-Peer Overlays for Mobile Networks'. In: *International Journal of Parallel, Emergent and Distributed Systems* 32.5 (2017), pp. 522–548. eprint: `https://doi.org/10.1080/17445760.2016.1203917`. URL: `https://doi.org/10.1080/17445760.2016.1203917`.

[71] AWS Authors. *AWS Lambda Developer Guide*. Amazon. 2023. URL: `https://docs.aws.amazon.com/lambda/latest/dg/lambda-dg.pdf#welcome`.

[72] Google Authors. *Google Cloud Functions*. Google. 2023. URL: `https://cloud.google.com/functions/`.

[73] OpenFaaS Authors. *OpenFaaS - Serverless Functions Made Simple*. OpenFaaS. 2023. URL: `https://docs.openfaas.com`.

[74] Scott Hendrickson et al. 'Serverless Computation with OpenLambda'. In: *8th USENIX Workshop on Hot Topics in Cloud Computing*. HotCloud '16. Denver, CO: USENIX Association, 2016.

[75] Eric Jonas et al. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. 2019. arXiv: `1902.03383 [cs.OS]`.

[76] Amine El Malki and Uwe Zdun. 'Guiding Architectural Decision Making on Service Mesh Based Microservice Architectures'. In: *Software Architecture*. Cham: Springer International Publishing, 2019, pp. 3–19.

[77] Wubin Li et al. 'Service Mesh: Challenges, State of the Art, and Future Research Opportunities'. In: *International Conference on Service-Oriented System Engineering*. New York, NY, USA: IEEE, 2019, pp. 122–1225.

[78] The Kubernetes Authors. *What is Kubernetes?* Kubernetes. 2023. URL: `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`.

[79] Thomas Lin et al. 'PhysarumSM: P2P Service Discovery and Allocation in Dynamic Edge Networks'. In: *International Symposium on Integrated Network Management*. Laxenburg, Austria: IFIP/IEEE, 2021, pp. 304–312.

[80] Dalton A. Hahn, Drew Davidson and Alexandru G. Bardas. 'MisMesh: Security Issues and Challenges in Service Meshes'. In: *Security and Privacy in Communication Networks*. Cham: Springer International Publishing, 2020, pp. 140–151.

[81] Dimitris Karnikis et al. *Atlas: Automated Scale-out of Trust-Oblivious Systems to Trusted Execution Environments*. 2022. URL: `https://github.com/atlas-runtime`.

[82] Fabrice Bellard. *QuickJS Javascript Engine*. Accessed: 2023-10-05. 2019. URL: `https://bellard.org/quickjs/` (visited on 05/10/2023).

[83] Daniel J Bernstein et al. 'TweetNaCl: A Crypto Library in 100 Tweets'. In: *International Conference on Cryptology and Information Security in Latin America*. Cham: Springer International Publishing, 2015, pp. 64–83.

[84] Sameer Ajmani et al. 'ConChord: Cooperative SDSI Certificate Storage and Name Resolution'. In: *First International Workshop on Peer-to-Peer Systems (IPTPS)*. Lecture Notes in Computer Science 2429. Mar. 2002, pp. 141–154.

[85] Giuseppe Pirrò, Domenico Talia and Paolo Trunfio. 'A DHT-based Semantic Overlay Network for Service Discovery'. In: *Future Generation Computer Systems* 28.4 (2012), pp. 689–707. URL: `https://www.sciencedirect.com/science/article/pii/S0167739X11002305`.

[86] T Labbai and S Jothi Prasanna. 'T2WSN: Titivated Two-Tired Chord Overlay Aiding Robustness and Delivery Ratio for Wireless Sensor Networks'. In: *Journal of Theoretical & Applied Information Technology* 91.1 (2016).

[87] Russ Cox, Athicha Muthitacharoen and Robert T. Morris. 'Serving DNS Using a Peer-to-Peer Lookup Service'. In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 155–165.

[88] 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. Stevenson, Washington, USA: Association for Computing Machinery, 2005, pp. 63–68. URL: https://doi.org/10.1145/1065983.1065999.

[89] Edward Eaton, Sajin Sasy and Ian Goldberg. 'Improving the Privacy of Tor Onion Services'. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2022, pp. 273–292.

[90] 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*. Vol. 10. ACSAC '04. USA: IEEE Computer Society, 2004, pp. 252–261.

[91] Emil Sit and Robert Morris. 'Security Considerations for Peer-to-Peer Distributed Hash Tables'. In: *Peer-to-Peer Systems*. Berlin, Heidelberg: Springer, 2002, pp. 261–269.

[92] 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. New York, NY, USA: ACM, 2002, pp. 193–206.

[93] 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. Chicago, Illinois, USA: ACM, 2009, pp. 161–172.

[94] Kevin R.B. Butler et al. 'Leveraging Identity-Based Cryptography for Node ID Assignment in Structured P2P Systems'. In: *IEEE Transactions on Parallel and Distributed Systems* 20.12 (2009), pp. 1803–1815.

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

[96] Bernd Prünster et al. 'A Holistic Approach Towards Peer-to-Peer Security and Why Proof of Work Won't Do'. In: *International Conference on Security and Privacy in Communication Systems*. Cham: Springer, 2018, pp. 122–138.

[97] Juan Benet. 'IPFS - Content Addressed, Versioned, P2P File System'. In: (2014). arXiv: 1407.3561 [cs.NI].

[98] Storj Lab. *Storj: A Decentralized Cloud Storage Network Framework*. Whitepaper. 2018. URL: https://static.storj.io/storjv3.pdf.

[99] 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. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 17–26.

[100] 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. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 53–57.

[101] Benjamin Fabian, Tatiana Ermakova and Cristian Muller. 'SHARDIS: A Privacy-Enhanced Discovery Service for RFID-Based Product Information'. In: *IEEE Transactions on Industrial Informatics* 8.3 (2012), pp. 707–718.

[102] Esther Palomar et al. 'A P2P Content Authentication Protocol Based on Byzantine Agreement'. In: *Emerging Trends in Information and Communication Security*. Berlin, Heidelberg: Springer, 2006, pp. 60–72.

[103] Miguel Castro et al. 'Secure Routing for Structured Peer-to-Peer Overlay Networks'. In: *SIGOPS Oper. Syst. Rev.* 36.SI (Dec. 2003), pp. 299–314.

[104] Tor Project. *Tor Metrics - Onion Services*. `https://metrics.torproject.org/hidserv-dir-v3-onions-seen.html`. Accessed: 2024-10-08. 2024.

[105] K. Gummadi et al. '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. Karlsruhe, Germany: Association for Computing Machinery, 2003, pp. 381–394. URL: `https://doi.org/10.1145/863955.863998`.

[106] Maxwell Ogden et al. *Dat-Distributed Dataset Synchronization and Versioning*. Jan. 2018. URL: `https://github.com/dat-ecosystem-archive/whitepaper/blob/master/dat-paper.pdf`.

[107] Nguyen Phong Hoang et al. 'An Empirical Study of the I2P Anonymity Network and its Censorship Resistance'. In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18. Boston, MA, USA: Association for Computing Machinery, 2018, pp. 379–392. URL: `https://doi.org/10.1145/3278532.3278565`.

[108] Nazanin Zahed Benisi, Mehdi Aminian and Bahman Javadi. 'Blockchain-based Decentralized Storage Networks: A survey'. In: *Journal of Network and Computer Applications* 162 (2020), p. 102656.

[109] Michal Krol et al. 'DISC-NG: Robust Service Discovery in the Ethereum Global Network'. In: *IEEE 9th European Symposium on Security and Privacy*. 2024, pp. 193–215.

[110] Swarm Team. *Swarm: Storage and Communication Infrastructure for a Self-Sovereign Digital Society*. 2024. URL: `https://www.ethswarm.org/swarm-whitepaper.pdf`.

[111] Yiannis Psaras and David Dias. 'The InterPlanetary File System and the Filecoin Network'. In: *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. 2020, pp. 80–80.

[112] Juan Pablo Timpanaro et al. 'Bittorrent's Mainline DHT Security Assessment'. In: *2011 4th IFIP International Conference on New Technologies, Mobility and Security*. IEEE. 2011, pp. 1–5.

[113] Roxana Geambasu et al. 'Vanish: Increasing Data Privacy with Self-Destructing Data.' In: *USENIX Security Symposium*. Vol. 316. 2009, pp. 10–5555.

[114] Inc. Storj Labs. 'Storj: A Decentralized Cloud Storage Network Framework'. In: *Storj White Paper* (2018).

[115] Ingmar Baumgart and Sebastian Mies. 'S/kademlia: A Practicable Approach Towards Secure Key-Based Routing'. In: *International Conference on Parallel and Distributed Systems*. New York, NY, USA: IEEE, 2007, pp. 1–8.

[116] Miguel Castro et al. 'Secure Routing for Structured Peer-to-Peer Overlay Networks'. In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 299–314.

[117] Nitesh Saxena, Gene Tsudik and Jeong Hyun Yi. 'Threshold Cryptography in P2P and MANETs: The Case of Access Control'. In: *Computer Networks* 51.12 (2007), pp. 3632–3649.

[118] Vivek Pathak and Liviu Iftode. 'Byzantine Fault Tolerant Public Key Authentication in Peer-to-Peer Systems'. In: *Computer Networks* 50.4 (2006), pp. 579–596.

[119] Esther Palomar et al. 'A P2P Content Authentication Protocol Based on Byzantine Agreement'. In: *International Conference on Emerging Trends in Information and Communication Security*. Cham: Springer, 2006, pp. 60–72.

[120] Rohit Gupta and Arun K. Somani. 'Reputation Management Framework and Its Use as Currency in Large-Scale Peer-to-Peer Networks'. In: *Fourth International Conference on Peer-to-Peer Computing*. New York, NY, USA: IEEE, 2004, pp. 124–132.

[121] George Danezis et al. 'Sybil-Resistant DHT Routing'. In: *Computer Security – ESORICS 2005*. Ed. by Sabrina de Capitani di Vimercati, Paul Syverson and Dieter Gollmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 305–318.

[122] 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.

[123] Baruch Awerbuch and Christian Scheideler. 'Towards a Scalable and Robust DHT'. In: *Theory of Computing Systems* 45.2 (2009), pp. 234–260.

[124] Kirsten Hildrum and John Kubiatowicz. 'Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks'. In: *International Symposium on Distributed Computing*. Springer. 2003, pp. 321–336.

[125] Tyson Condie et al. 'Induced Churn as Shelter from Routing-Table Poisoning.' In: *NDSS*. 2006.

[126] Atul Singh et al. 'Eclipse Attacks on Overlay Networks: Threats and Defenses'. In: *In IEEE INFOCOM*. Citeseer. 2006.

[127] Cyrus Harvesf and Douglas M Blough. 'The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables'. In: *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*. IEEE. 2006, pp. 57–6.

[128] Ricardo Villanueva, Maria del Pilar Villamil and Mile Arnedo. 'Secure Routing Strategies in DHT-based Systems'. In: *International Conference on Data Management in Grid and P2P Systems*. Springer. 2010, pp. 62–74.

[129] Keith Needels and Minseok Kwon. 'Secure Routing in Peer-to-Peer Distributed Hash Tables'. In: *Proceedings of the 2009 ACM Symposium on Applied Computing*. ACM. 2009, pp. 54–58.

[130] Peng Wang et al. 'Myrmic: Secure and Robust DHT Routing'. In: *U. of Minnesota, Tech. Rep* (2006).

[131] Marc Sànchez-Artigas, Pedro Garcia-López and Antonio G Skarmeta. 'Secure Forwarding in DHTs-is Redundancy the Key to Robustness?' In: *European Conference on Parallel Processing*. Springer. 2008, pp. 611–621.

[132] Sergio Marti, Prasanna Ganesan and Hector Garcia-Molina. 'DHT Routing Using Social Links'. In: *International Workshop on Peer-to-Peer Systems*. Springer. 2004, pp. 100–111.

[133] Dennis Trautwein et al. 'Design and Evaluation of IPFS: A Storage Layer for the Decentralized Web'. In: *Proceedings of the ACM SIGCOMM 2022 Conference*. 2022, pp. 739–752.

[134] Wei Wang et al. 'A Comprehensive Ontology for Knowledge Representation in the Internet of Things'. In: *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE. 2012, pp. 1793–1798.

[135] Matú Tomlein and Kaj Grønbæk. 'Semantic Model of Variability and Capabilities of IoT Applications for Embedded Software Ecosystems'. In: *Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE. 2016, pp. 247–252.

[136] Garvita Bajaj et al. '4W1H in IoT Semantics'. In: *IEEE Access* (2017).

[137] David J Wu et al. 'Privacy, Discovery, and Authentication for the Internet of Things'. In: *European Symposium on Research in Computer Security*. Springer. 2016, pp. 301–319.

[138] Dimitris N Kalofonos et al. 'Mynet: A platform for Secure P2P Personal and Social Networking Services'. In: *International Conference on Pervasive Computing and Communications (PerCom)*. IEEE. 2008, pp. 135–146.

[139] Joao Alveirinho et al. 'Flexible and Efficient Resource Location in Large-Scale Systems'. In: *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*. ACM. 2010, pp. 55–60.

[140] Istio Authors. *The Istio service mesh*. Istio. 2021. URL: `https://istio.io/latest/about/service-mesh/`.

[141] NGINX Authors. *NGINX Architecture*. F5. 2023. URL: `https://docs.nginx.com/nginx-service-mesh/about/architecture/`.

[142] Linkerd Authors. *Linkerd Architecture*. Linkerd. 2023. URL: `https://linkerd.io/2.11/reference/architecture/#`.

[143] AWS Authors. *AWS App Mesh User Guide*. Amazon. 2023. URL: `https://docs.aws.amazon.com/app-mesh/latest/userguide/app-mesh-ug.pdf`.

[144] Open Service Mesh Authors. *Open Service Mesh Docs*. Microsoft. 2023. URL: `https://docs.openservicemesh.io`.

[145] Jeff Escalante and Zachary Shilton. *Consul Architecture*. HashiCorp. 2023. URL: `https://www.consul.io/docs/architecture`.

[146] Dan S. Wallach. 'A Survey of Peer-to-Peer Security Issues'. In: *International Symposium on Software Security*. Cham: Springer, 2002, pp. 42–57.

[147] Leucio Antonio Cutillo, Refik Molva and Thorsten Strufe. 'Safebook: A Privacy-Preserving Online Social Network Leveraging on Real-Life Trust'. In: *IEEE Communications Magazine* 47.12 (2009), pp. 94–101.

[148] Guor-Huar Lu and Zhi-Li Zhang. 'Wheel of Trust: A Secure Framework for Overlay-Based Services'. In: *International Conference on Communications*. New York, NY, USA: IEEE, 2007, pp. 1148–1153.

[149] Kevin R.B. Butler et al. 'Leveraging Identity-Based Cryptography for Node ID Assignment in Structured P2P Systems'. In: *IEEE Transactions on Parallel and Distributed Systems* 20.12 (2008), pp. 1803–1815.

[150] Nirmala N. Jagadale and Thaksen J. Parvat. 'A Secured Key Issuing Protocol for Peer-to-Peer Network'. In: *2014 IEEE Global Conference on Wireless Computing & Networking*. New York, NY, USA: IEEE, 2014, pp. 213–218.

[151] Tanaka Hiroyuki, Saito Shoichi and Matsuo Hiroshi. 'Node Management without Directory Servers in DHT-Based Anonymous Communication Systems Using ID-Based Encryption'. In: *International Journal for Information Security Research* 1.3 (2011), pp. 154–163.

[152] Agapios Avramidis, Panayiotis Kotzanikolaou and Christos Douligeris. 'Chord-PKI: Embedding a Public Key Infrastructure into the Chord Overlay Network'. In: *Proceedings of the 4th European Conference on Public Key Infrastructure: Theory and Practice*. EuroPKI'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 354–361.

[153] Neander L. Brisola et al. 'A Public Keys Based Architecture for P2P Identification, Content Authenticity and Reputation'. In: *International Conference on Advanced Information Networking and Applications Workshops*. New York, NY, USA: IEEE, 2009, pp. 159–164.

[154] David Mazieres and M Frans Kaashoek. 'Escaping the Evils of Centralized Control with Self-Certifying Pathnames'. In: *Proceedings of the 8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications*. New York, NY, USA: ACM, 1998, pp. 118–125.

[155] Hugo Krawczyk, Kenneth G. Paterson and Hoeteck Wee. 'On the Security of the TLS Protocol: A Systematic Analysis'. In: *Annual Cryptology Conference*. Cham: Springer, 2013, pp. 429–448.

[156] Paul Wouters et al. *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. 2014.

[157] Yusef Napora. *Noise Specification*. libp2p. Oct. 2023. URL: `https://github.com/libp2p/specs/tree/master/noise`.

[158] Nadim Kobeissi, Georgio Nicolas and Karthikeyan Bhargavan. 'Noise Explorer: Fully Automated Modeling and Verification for Arbitrary Noise Protocols'. In: *European Symposium on Security and Privacy*. EuroS&P '19. New York, NY, USA: IEEE, 2019, pp. 356–370.

[159] Bartlomiej Polot and Christian Grothoff. 'Cadet: Confidential Ad-Hoc Decentralized End-to-End Transport'. In: *2014 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*. IEEE. 2014, pp. 71–78.

[160] GNUnet Authors. *CORE — GNUnet link layer*. 2024. URL: https://docs.gnunet.org/v0.20.x/developers/core/core.html.

[161] Juan Benet, Bigs and Yusef Napora. *Secio Specification*. libp2p. Oct. 2023. URL: https://github.com/libp2p/specs/tree/master/secio%5C#shared-secret-generation.

[162] Carmela Troncoso et al. 'Systematizing Decentralization and Privacy: Lessons from 15 Years of Research and Deployments'. In: *Proceedings on Privacy Enhancing Technologies* 4 (2017), pp. 404–426.

[163] Isabel Wagner and David Eckhoff. 'Technical Privacy Metrics: A Systematic Survey'. In: *ACM Comput. Surv.* 51.3 (June 2018). URL: https://doi.org/10.1145/3168389.

[164] 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*. Aug. 2004, pp. 28–35.

[165] Nikita Borisov and Jason Waddle. *Anonymity in Structured Peer-to-Peer Networks*. Tech. rep. Computer Science Division, University of California, 2005.

[166] Cynthia Dwork. 'Differential Privacy'. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2006, pp. 1–12.

[167] Miguel E Andrés et al. 'Geo-indistinguishability: Differential Privacy for Location-Based Systems'. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. 2013, pp. 901–914.

[168] 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. Sept. 2007, pp. 27–36.

[169] Alan Mislove et al. 'AP3: Cooperative, Decentralized Anonymous Communication'. In: *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*. EW '04. Leuven, Belgium: Association for Computing Machinery, 2004, 30–es.

[170] Apu Kapadia and Nikos Triandopoulos. 'Halo: High-Assurance Locate for Distributed Hash Tables.' In: *Proceedings of the 16th Network and Distributed System Security Symposium*. Vol. 8. NDSS '08. Citeseer. 2008, p. 142.

[171] Peng Wang et al. *Myrmic: Secure and Robust DHT Routing*. Tech. rep. University of Minnesota, 2006.

[172] Qiyan Wang and Nikita Borisov. 'Octopus: A Secure and Anonymous DHT Lookup'. In: *2012 IEEE 32nd International Conference on Distributed Computing Systems*. June 2012, pp. 325–334.

[173] Giuseppe Ciaccio. 'Improving Sender Anonymity in a Structured Overlay with Imprecise Routing'. In: *Privacy Enhancing Technologies*. Ed. by George Danezis and Philippe Golle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 190–207.

[174] 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. Washington, DC: Association for Computing Machinery, 2002, pp. 91–102.

[175] 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. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 141–150.

[176] Jon McLachlan et al. 'Scalable Onion Routing with Torsk'. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 590–599.

[177] 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. 2021, pp. 170–177.

[178] Michael Backes et al. 'Adding Query Privacy to Robust DHTs'. In: *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. AsiaCCS '12. Seoul, Korea: Association for Computing Machinery, 2012, pp. 30–31.

[179] Maxwell Young et al. 'Practical Robust Communication in DHTs Tolerating a Byzantine Adversary'. In: *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems*. June 2010, pp. 263–272.

[180] Miti Mazmudar, Shannon Veitch and Rasoul Akhavan Mahdavi. *Peer2PIR: Private Queries for IPFS*. 2024. eprint: `arXiv:2405.17307`.

[181] Guillaume Michel. *IPIP-373: Double Hash DHT Spec.* Tech. rep. IPFS, 2023. URL: `https://github.com/ipfs/specs/pull/373/files`.