ABSTRACT

Title of dissertation: LOOKUP PROTOCOLS AND TECHNIQUES FOR ANONYMITY

Ruggero Morselli
Doctor of Philosophy, 2006

Dissertation directed by: Professor Jonathan Katz
Department of Computer Science

This dissertation covers two topics of interest for network applications: lookup protocols, a basic building block for distributed systems, and ring signatures, a powerful primitive for anonymous communication. In the first part of this work, we review lookup protocols, distributed algorithms that allow users to publish a document as well as to look up a published document that matches a given name. Our first major contribution is to design Local Minima Search (LMS), a new efficient lookup protocol for a model in which a node is physically connected to a few other nodes and may only communicate directly with them. Our second major contribution is the formulation of a new model in which we allow an arbitrary number of misbehaving nodes, but we assume a restriction on their network addresses. We then design a new lookup protocol for this setting.

In the second part of this dissertation, we present our work on ring signatures, a variant of digital signatures, which enables a user to sign a message so that a set of possible signers is identified, without revealing which member of that set actually

generated the signature. Our first contribution on this topic is new definitions of security which address attacks not taken into account by previous work. As our second contribution, we design the first provably secure ring signature schemes in the standard model.

LOOKUP PROTOCOLS AND TECHNIQUES FOR ANONYMITY

by

Ruggero Morselli

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:
Professor Jonathan Katz, Chair/Advisor
Professor Bobby Bhattacharjee, Co-Advisor
Professor Pete Keleher
Professor Aravind Srinivasan
Professor Lawrence Washington

© Copyright by
Ruggero Morselli
2006

ACKNOWLEDGEMENTS

I want to thank all the people that supported me during my Ph.D. program. First of all, I want to thank my advisors, Bobby Bhattacharjee and Jonathan Katz, not only for the guidance that they provided me, but also for their enormous patience. I want to thank my labmates Vijay Gopalakrishnan, Dave Levin, Cristian Lumezanu, and Rob Sherwood, for providing me with such a friendly and intellectually stimulating environment, in which to develop my dissertation. I thank all the other people that have contributed to parts of this work, including Randy Baden, Adam Bender, Matt Mah, Michael Marsh, Neil Spring, and Aravind Srinivasan.

A special thank you goes to Ritchie Hudson, for making the last two years of my work as a graduate student so much happier, and for his moral support during my writing of this dissertation.

I thank Francesco Ferioli and Leonid Nikolayev for their sincere friendship, which has been integral part of my life in College Park. I finally thank Lorenzo Alvisi, Michele Colajanni, Leana Golubchik, and Paolo Tartarini, for their precious help and advice in the shaping of my career.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation deals with two important topics in the area of distributed applications. The first is lookup protocols, which are a useful building block for more complex distributed systems. The second is ring signatures, a powerful cryptographic primitive for anonymous communication.

Distributed applications are becoming increasingly important in our information economy. More and more personal, corporate and government transactions are made through the Internet, including online banking, electronic commerce, customer account management, access to government services, just to make a few examples. Additionally, large companies and organizations rely on complex computer systems, which may be distributed on tenths, hundreds or thousands of servers, in order to increase reliability and scale to a large number of accesses.

One of the tasks of computer science is to develop new tools and primitives to build distributed applications with better performance and functionality. One direction of research along these lines is concerned with designing efficient algorithms for basic tasks, such as information storage and retrieval in a distributed setting, that can be used as a building block for more complex applications. A different, but not independent, line of research is the design of security mechanisms for distributed systems. Security protects the system from threats carried out by a malicious entity

(the attacker). Such threats depends on the specific application and may include disruption of service, corruption of data, unauthorized access to restricted services and information or violation of privacy, just as a few examples.

A contribution to these two lines of research, this dissertation is divided into two parts: the topic of the first part is lookup protocols, a class of basic distributed algorithms, while the second part discusses ring signatures, a powerful primitive for anonymous communication.

1.1 Part One: Lookup Protocols

In the first part of this dissertation, we focus on lookup protocols, a specific building block for distributed computation and we address both efficiency and security aspects of their design. A lookup protocol is a distributed algorithm which allows any participant node to *publish* a data item (document) as well as to *look up* a published data item that matches a given item identifier (name). We review lookup protocols designed in previous work (in particular, Distributed Hash Tables), we discuss the assumptions made by such protocols with respect to *communication* and *security* and we pose the problem of designing new algorithms that allow for a relaxation of such assumptions.

With respect to communication, all *efficient* lookup protocols in previous work assume that any node may send a message directly to any other node. It is interesting, though, to study how to design an algorithm for a system in which a node is physically connected to a few assigned *neighbor* nodes and may only communicate

directly with them; we call this setting the *Given-Topology Model* . This models an ad-hoc network without a routing infrastructure. It also models a distributed application relying on trust relations between nodes. These trust relations define a graph which, when traversed along its links, provides a known level of assurance for operations.

The first major contribution of this work is to design Local Minima Search (LMS), a new efficient lookup protocol for the Given-Topology Model. Lookup operations in LMS are much more efficient than existing protocols (e.g. Gnutella [Com01], Gia [CRB+03]) that can be applied to the Given-Topology Model. We proved analytic bounds for the worst-case performance of LMS and, through detailed simulations, we show that the actual performance on realistic topologies is significantly better. As part of our analysis, we also derived a powerful theorem on statistical distributions, which is of independent interest. LMS revealed to be the essential ingredient we needed to build KeyChains, a completely decentralized public-key infrastructure, which we present elsewhere [MBKM06].

With respect to security, most previous work on lookup protocols assumes a cooperative environment, i.e. all participants in the distributed application behave according to the specified protocol. In practice, however, it is necessary to protect distributed applications from malicious behavior. We, thus, consider the problem of designing a lookup protocol that operates correctly even in the presence of arbitrarily misbehaving nodes. Without loss of generality, we can think of the misbehaving (or corrupted) nodes as controlled by a single attacker (or adversary), although in practice there may actually be controlled by multiple independent attackers or their

misbehavior could be caused by a software, hardware or network fault.

There are lookup protocols in the literature that solve this problem (e.g. [FSY05, CDG+02]) under the assumption that up to a small fraction of the nodes in the system may be misbehaving. While this is a reasonable assumption in many settings, we believe that it is not very realistic when applied to an Internet application with open node membership, such as the Gnutella network [Com01]. This is because even an attacker with limited resources may simulate a large number of nodes and make them join the application. On the other hand, it is our intuition that no practical lookup protocol may withstand an attacker that can corrupt an arbitrary number of nodes with no restriction.

The second major contribution of this dissertation is to consider a middle ground between these two extreme assumptions: bounded fraction of corrupted nodes on one hand and no restriction on the other. We formulate a new model in which we allow an arbitrary number of misbehaving nodes, but we assume that they are subject to a more realistic restriction on their network addresses; we call this model the *Autonomous System (AS) Model*. More specifically, the model dictates that the attacker can only control hosts in a small number of stub ASes (independently managed portions of the Internet), as we will clarify in Chapter 4.

As part of this contribution, we then proceed to design Secure Distributed Hash Table (SDHT), a new lookup protocol designed to be resilient in this AS model. We conjecture that SDHT provides acceptably resilient behavior, under some reasonable circumstances and we provide an informal proof of this property under some restricted conditions. We provide a prototype implementation of SDHT

and we show through experiments that the algorithm is practical for sizes up to 300 nodes, although with high overhead.

1.2 Part Two: Ring Signatures

In the second part of the dissertation, we turn our attention to security tools for anonymity. We believe that there is a wide class of distributed applications for which preserving user anonymity is desirable or even essential. In the example of a system that allows access to a database, a user may not want anyone to track what information she is retrieving and match that information with her identity. As a more critical example, an application that allows for anonymous reporting of mismanagement or violations of the law within a government agency should guarantee that the identity of the whistle-blower cannot be revealed.

There are many challenges in designing truly anonymous distributed systems. One example is preventing an entity that may eavesdrop on a large number of links in the network from tracing the communication between the user and the application; see the work by Sherwood et al. [SBS05] for a discussion of this issue and a possible solution. A detailed treatment of how to design anonymous distributed systems is beyond the scope of this work.

Part Two of this dissertation focuses on *ring signatures*, a cryptographic primitive that addresses one specific challenge: providing some form of authentication, while still preserving anonymity. Consider the above-mentioned whistle-blower application: it is desirable to authenticate the reported information, to ensure that

it has actually been leaked by an agency employee of a certain rank, without revealing which employee. Similarly, in the example of the database, the system may want to restrict access to a set of authorized users. In such a case, the user must prove that he belongs to the authorized group, without revealing her identity to the application.

Ring signatures are a variant of digital signatures introduced by Rivest, Shamir, and Tauman [RST], which can be used for such anonymous authentication. A ring signature enables a user to sign a message so that a set (or *ring*) of possible signers is identified, without revealing exactly *which member* of that ring actually generated the signature. In Part Two of this dissertation, we examine previous definitions of security for ring signature schemes and suggest that most of these prior definitions do not take into account certain realistic attacks. Our first set of contributions on this part is new definitions of security which address these threats, and separation results proving that our new notions are strictly stronger than previous ones. As our second contribution, we design the first constructions of ring signature schemes, which are provably secure (under widely-accepted computational assumptions), without using the so-called *random oracle* heuristic [BR93].[1] One scheme is based on generic assumptions and satisfies our strongest definitions of security. Two

[1]Informally, proofs that use the random oracle heuristic assume that a cryptographic hash function (such as SHA-1) produces independently random output for every possible input. In practice all hash functions are deterministic, so the random oracle assumption is false; however, cryptographers consider a proof in the random oracle model to be a strong argument in favor of an algorithm security, although not a real proof.

additional schemes are more efficient, but achieve weaker security guarantees and more limited functionality.

1.3 Roadmap

Part One of this dissertation begins with a review of lookup protocols and related literature in Chapter 2. In the same chapter, we also distinguish between two communication models for lookup protocols: the Given-Routing Model and the Given-Topology Model. In Chapter 3, we present our design of LMS, an efficient lookup protocol in the Given-Topology Model. In Chapter 4, we review security vulnerabilities of existing lookup protocols and some previous work on that addresses those vulnerabilities, then we argue that the security assumptions made in such work are very unrealistic in a system with open membership. In the same chapter we also present the AS Model: a novel sets of assumptions on what an attacker with limited resources can or cannot do when attacking an Internet application. In Chapter 5, we finally present SDHT, a lookup protocol designed to be secure in the AS Model. In Part Two of the dissertation, Chapter 6 presents and motivates our work on ring signatures. Finally, in Chapter 7 we conclude.

Part I

Lookup Protocols

Chapter 2

Lookup Protocols

In this section, we informally introduce and motivate the notion of a *lookup protocol*.

Informally, a lookup protocol is a distributed algorithm that allows nodes (or processors[1]) to share data items, which consist of an *identifier* (ID) and a *content*: a node can *publish* an item (make it available for others); a node can also *look up* an item (find a copy and retrieve it), given the item identifier. More specifically, a user can ask a node to perform one of two operations:

- Publish(id, content), which causes the node to publish the data item consisting of the identifier id and the value content; and
- Lookup(id), which causes the node to retrieve all data items that have the given identifier id, if any.

A lookup protocol is interesting as a building block for distributed applications, because it provides a basic form of shared storage. For example, given an efficient lookup protocol, it is known how to construct applications such as distributed file systems, multicasting, keyword-based searching, and more. We will discuss these applications in Section 2.3.

The rest of this chapter is organized as follows. In Section 2.1, we will introduce

[1]In this work, we will use the terms "nodes" and "processors" interchangeably.

some terminology that we will use throughout this document. In Section 2.2, we will discuss some examples of lookup protocols. In Section 2.3, we will discuss some applications of lookup protocols. In Section 2.4, we identify two alternative set of assumptions, with respect to node communication, and their impact on the design of lookup protocols.

2.1 Preliminaries on Distributed Algorithms

Informally a distributed algorithm involves a set of *nodes* or *processors*, which interact by running local computations and by exchanging messages with each other.

Each processor in the system has some state information that can change over time and may respond to a set of events. In response to an event, a node may update its state, send messages and generate local output. Events include reception of messages from other processors, timeouts and *activations*, which model stimuli from a user or another applications running on the same machine.

In an efficient distributed algorithms involving a large number of nodes, a node will not know the identity of every other node in the system and will not send messages to every other node in the system. We say that a node u has a distinct set of *neighbors*: u only stores information on the identity of its neighbors and sends messages directly only to its neighbors.

2.2 Examples in the Literature

Within the category of *Peer-to-Peer (P2P)* systems, there is a wide variety of related work on lookup protocols. The term P2P is widely used in the network research community to refer to any practical distributed system where participating nodes "are all peers" and each node can both perform as a client (i.e. an entity receiving a service from the system) and as a server (i.e. an entity providing a service).

P2P systems are divided into two categories: structured and unstructured. In the next sections, we will discuss each category in turn.

2.2.1 Structured and Unstructured P2P Systems

In a structured P2P system, a node's neighbors are chosen by the system according to certain properties of the nodes. For example, in Chord [SMK+01] each node is given an identifier, which is a random bit string of length 128 bits. A node with ID x has as neighbors the nodes with ID $x_1, \ldots, x_{128}$, where x_i is the (ID of the) node in the system that has the smallest ID between all the nodes with ID at least $x + 2^{i-1}$. The graph of such a P2P system has a particular structure that provides some useful properties. For example, in Chord, a node can send a message to another node with ID y (not her neighbor), and the message will reach y, traveling through at most approximately $\log N$ hops in the system (where N is the total number of nodes).

Unstructured P2P systems, on the other hand, place little restrictions on what

the neighbors of a node are. Usually, when a new node joins the system, it is given a list of the IP addresses of other peers already in the system and the new node will simply make those nodes as their neighbors. Therefore the system can usually assume very little on the structure of the graph. Examples of unstructured P2P systems are Gnutella [Com01] and GIA [CRB+03].

We will discuss more examples of structured and unstructured P2P systems in the following sections.

2.2.2 Distributed Hash Tables: CAN, Chord, Pastry, Tapestry

In this section, we describe in greater detail a class of lookup P2P systems called Distributed Hash Tables (DHTs). Those are considered the classical examples of structured P2P systems (Section 2.2.1) and include Pastry [RD01], Chord [SMK+01], CAN [RFH+01], Tapestry [ZKJ01] and many others. In what follows, we discuss the meaning of the acronym DHT and the various components of such a system: the routing protocol, the join and leave protocols and the replication techniques. In the section that follows, we will see a specific example of DHT.

The principle behind a DHT is the concept of hash table. A hash table is a data structure consisting of a series of storage spaces called buckets. When an item is inserted in the hash table, a *hash function* is applied to the identifier to output a random number. That random number is used to choose one of the buckets at random and the item is stored into that bucket. Given an item name, the hash table can quickly find an item with that name, if one exists: the item name is hashed to

determine the appropriate bucket, then that bucket is searched for the item.

A DHT builds a sort of hash table in a distributed way. Each data item is given an identifier, which is a random bit string of some fixed length m, typically 128 or 160 bits. The item ID might be obtained, for example, by hashing an application-dependent item name. Each node is also given a random ID of length m, which may be independently generated at random or obtained by hashing some other information about the node (like the IP address, port pair). Each item is then assigned to a *home node* through a deterministic rule based only on the item ID and the IDs of the peers in the system. Typically, the rule assigns an item with ID x to the node whose identifier is the numerically closest to x between all node IDs (Pastry) or all node IDs greater than x (Chord). In a basic DHT implementation, for each published item only one copy is stored and it is stored into the home node. There exist analytical results that show that the number of items a node is responsible for (as a home node) is not too different from node to node and this guarantees that the amount of data that peers have to stored is well-balanced through the system.

Like in any other P2P system, in a DHT, each node knows the IP address and the node ID of a set of other nodes in the system (the node's neighbors, see Section 2.1). DHTs are *structured* P2P systems, which means that the set of neighbors of a node is constrained by some rigid rules based on the node IDs.

In order to publish or look up an item, a node must be able to quickly locate the home node of an item, given the item ID x. This is the task of the *routing protocol* of the DHT, which allows any node in the network to address a message to

any identifier x and delivers the message to the home node of x. Given a message addressed to item ID x, a node u, uses an appropriate rule established by the routing protocol to choose a neighbor v and then forwards the message to v. The procedure repeats recursively until the home node of x receives the message.

Each node typically stores her neighbor set in a specific data structure, which depends on the protocol. Such data structure is sometimes referred to as the routing table. In some protocols, like Pastry, the routing table is only a portion of that data structure.

A DHT also needs a *join protocol* and a *leave protocol*, which define the operations that the peers in the system have to perform when a new peer wants to join the system or when an existing peer leaves the system (either by choice or by a failure).

In the join protocol, typically the new node u is given out of band the IP address of a peer v already in the system (v is sometimes called the bootstrap node for u). After u sends v a join request, the join protocol specifies which actions the various peer in the system have to perform such that u's routing table can be appropriately constructed and such that other peers in the system can update their routing table.

The leave protocol specifies how the system detects that one of the peers left and how it repairs the routing tables, in order to maintain the required constraint.

We finally discuss replication. We already said that, in a basic DHT implementation, the system stores the only copy of an item in the home node of that item. More advanced implementations keep multiple copies of each item: each copy

is called a replica and this technique is called replication. There are three main reasons to do replications: to make sure an item is not lost when the node that is storing it fails, to reduce the number of requests to retrieve the item the node has to process and to make the lookup operation faster.

One possible replication technique is that the number of replicas r is the same for all items and is fixed (static replication) and the replicas are stored at a set of r *replica nodes*. Such set is determined by some rule given the item ID and the node ids, which generalizes the rule for the home node. For example, in a variant of Chord, the replica nodes of item with ID x are the r nodes with the smallest node ID greater than x. In a variant of Pastry, the replica nodes are the r nodes (with ID) closest to x in the identifier space.

Other systems use *dynamic replication* techniques, which determine the number of replicas of an item at run time. For example, a system might want to create more replicas for a popular item (i.e. an item that is being retrieved by many users) and fewer replicas for a less popular item (see, for example, [RS04]).

In order to make our description of DHTs more concrete, in the following section we give some more details about Pastry, a specific DHT.

2.2.2.1 Pastry

In this section, we briefly discuss Pastry, an example of DHT.

In Pastry [RD01], each node or item ID is a string of m bits, which is also seen as a sequence of digits, each consisting of b bits (m and b are system parameters,

typically set to 160 and 4). Each digit can be considered a value between 0 and $2^b - 1$.

A peer has three data structures that store information about other peers (the peer neighbors): the *neighbor set*, the *leaf set* and the *routing table*.

The leaf set of a node with ID y contains (the ID and the IP address of) the l peers with the smallest IDs greater than y and the l peers with the largest IDs smaller than y, where l is a system parameter. The elements of the leaf sets are called the *leaves* of the node.

The routing table of the node is a table with m rows and 2^b columns. The entry in row i and column j of the table contains the (ID and the IP address of) a peer whose ID has the first $i - 1$ digits in common with y, but the i-th digit equal to j. An entry is allowed to be empty, if such a node does not exist.

The neighbor set of the node contains a list of other peers that are close to the node in network distance, i.e. the network latency between the node and the elements of her neighbor set is relatively small, compare to the other node. The neighbor set is used during updates to the routing table, with the goal to ensure that every table entry contains the node in the system that is the closest in network distance, between all nodes that qualify for that table entry. We refer the reader to [RD01] for more details.

Routing in Pastry proceeds as follows. When a node u receives or generates a message addressed to the home node of item ID x (a publish or a lookup request), first u determines whether she or any of her leaves is the home node for x; if yes, the message is delivered to the final destination. Otherwise, u determines the number d

of leading digits that the ID of u has in common with x and forwards the message to the entry in the routing table in row $d + 1$ and in the column corresponding to the $d + 1$-th digit of x.

The routing protocol in Pastry guarantees that a message addressed to an item ID will reach the corresponding home node from any node within about $(1/b) \log n$ hops with high probability, where n is the total number of nodes in the system.

2.2.3 Unstructured P2P Systems

Unstructured P2P protocols differs from structured P2P protocols, like DHTs, in that the set of neighbors of a node (that is, the set of peer nodes that the node knows and send messages to) is not constrained by some rigid rules based on the node IDs. There are several examples of unstructured P2P protocols, which include Gnutella [Com01], Gia [CRB+03] and Kazaa [Net], just to name a few.

These protocols typically implement the *search functionality*, which is more general then the lookup functionality. This means that, instead of the Lookup primitive, they provide a more general Search primitive, which takes a predicate as an argument and will retrieve some or all data items in the system that satisfy the predicate. The Publish operation is local, that is, publishing an item involves simply storing it locally in a list of shared items and does not involve network communication. Searching, in such systems, is inefficient in the general case, but becomes efficient when a large fraction of the peers in the networks have published an item that matches the search predicate. We find illuminating an observation by *Chawathe*

et al. [CRB+03], that we here summarize: unstructured P2P protocols like Gnutella are designed to search for hay, not for needles.

While Gnutella is maybe the first open-specification example of unstructured P2P protocol [Gnu], there has been a lot of work how to improve its (relatively poor) performance. An important example is the paper by *Lv et al.* [LCC+02], which compares different techniques for searching in this model. Such techniques include different flavors of *flooding* and *random walks*. In flooding (Gnutella technique), the node searching sends the request to all its neighbors, which, in turn, forward the request to all their neighbors and so on, until a copy of the request has reached all (or a large fraction of) the nodes in the network. The other technique involves sending the request to a random neighbor, which in turn forwards it to a random neighbor and so on, up to a predetermined number of hops; multiple requests can be sent out in parallel or in sequence.

Gia [CRB+03] is a more sophisticated member of the family. It improves on Gnutella through several techniques, which take into account node heterogeneity (i.e. the fact that some node have more bandwidth and processing capacity) and node congestion (the fact that a node performance degrades when it is required to process too many protocol messages). These techniques include *biased random walks* (towards nodes that have more resources), *token-based flow control* (a node cannot forward a request to another node, unless the other node has previously consented), *topology adaptation* (a node may add or drop neighbors) and allow Gia to beat Gnutella in performance by several orders of magnitude.

2.3 Applications of Lookup Protocols

Lookup protocols can be used as a building block for complex distributed applications. One such application is P2P file systems, such as Ivy [MMGC02], Pastis [BPS05] or OceanStore [KBC+00]. Such systems have the same interface as a distributed file system such NFS, but without requiring dedicated servers. These P2P file systems are built on top of efficient lookup protocols, namely DHTs. For example, Pastis employs Pastry [RD01] in the following way. Similarly to a Unix file system, each file has an associated i-node, which is stored as a data item into the DHT; additionally, file data is divided into fixed-sized blocks, each of which is also stored as a data item into Pastry. The i-node maintains the file meta-data and the identifiers of the data blocks.

Another example of application of lookup protocols is P2P multicast systems, which allows nodes to join named *multicast groups*, as well as efficiently sending a message to all members of a multicast group. For example, Scribe [RKCD01] uses Pastry to construct a communication tree consisting of a superset of the nodes in a multicast group. In order to multicast, the node sends the message to the node that is at the root of the tree. The message is then forwarded down the tree, reaching all targets.

A third example is P2P systems that enable keyword-based search [RV03, TD04, Gna02, LHH+04]. Such systems generalize lookup protocols in that they allow a node to locate all data items (documents) that match a query consisting of a list of keywords. The basic technique is to build an *inverted index* for each keyword

appearing in any document, which is a list of all data item identifiers containing a certain keyword, and to store each index as a data item using the corresponding keyword as its identifier. In [TD04], the authors present an alternative solution, which requires an appropriate modification of Chord [SMK+01].

2.4 Communication Models

So far, we have simply assumed that any node in the network can send a message directly to any other node in the network. This is not necessarily true. In this section, we consider the issue of communication in more details. We will define two *communication models*, which describe how nodes in a distributed algorithm are allowed to communicate to each other. The first attempts to portray a routed network such as an IP network: a node may send a message directly to any other node, as long as the sender knows the recipient's address. The second model portrays an ad-hoc network with no routing infrastructure, in which a node can only send a message directly to a few other nodes that are assigned to it by the environment. We call the first model the Given-Routing Model and the second the Given-Topology Model.

Designing a distributed algorithm with the same goals in the two communication models yields two different and interesting problems. The Given-Routing Model represents the case of an application running on the Internet or, more in general, on an IP network. Most previous work on lookup protocols and P2P protocols implicitly assumes the Given-Routing Model. For example, in a DHT (Section 2.2.2),

although a node only communicates with a small number of *neighbors*, the node chooses *who* its neighbors are (in this case, on the basis of the random identifiers).

While DHTs are very efficient, they might not be usable in some circumstances. Consider the case where the nodes are connected by a wireless ad-hoc network. In this case, a node u can send a message to a node v only if the two nodes are within transmission range. A lookup protocol in the Given-Topology Model is required for this application.

The Given-Topology Model also applies when the lookup protocol is executed as a component of a more complex P2P application and such application: a) establishes a neighborhood relation between nodes that is not under control of the lookup protocol; b) requires that messages only be sent to nodes that are neighbors according to such relation. This is especially the case when the links in the relation have particular significance, such as expressing statements of trust or belief between nodes. An example of this is discussed in another work of ours [MBKM06].

Note the fundamental difference between real-world systems, such as the Internet, which we model as given-routing, and one, such as an ad-hoc network, which corresponds to the Given-Topology Model. Although the underlying structure of the Internet is a sparsely-connected graph and the Given-Routing Model is only its abstraction, the nodes of the distributed algorithms that we are interested in are only end hosts on the Internet graph. Messages between nodes are forwarded by *routers*, which are not part of the distributed algorithm and are not under the control of the algorithm designer. In the ad-hoc network, instead, the same nodes that participate in the distributed algorithm also route messages for other nodes.

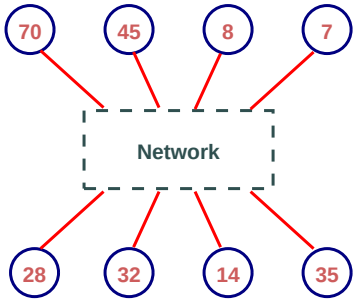


Figure 2.1: Representation of the Given-Routing Model: every node has a unique address and is connected to the network.

2.4.1 The Given-Routing Model

In this section, we introduce a communication model, called the Given-Routing Model. In this model, the network is treated as a black box that delivers messages, based on their destination address (Figure 2.1). All processors are given a unique *network address*. Any user can send a message, addressed to a certain *network address*, and the network will deliver it correctly, if any such destination user exists.

This communication model intends to model the behavior of any network that supports IP routing. The model is somewhat simplified and abstracts away the entire routing infrastructure. However, the model captures the essential aspects that are necessary to understand the design of virtually all P2P protocols available in the literature. In particular, the Given-Routing Model captures the fact that any node can send a message directly to any other node, given the recipient's address, but a node does not know *a priori* the number, the addresses or the identity of all other nodes in the protocols.

2.4.2 The Given-Topology Model

In this section, we discuss the *Given-Topology Model* . In this model, the network is modeled as a graph G (which may change over time), where vertices correspond to nodes. A node can only communicate with its *physical neighbors* (Figure 2.2), that is with the nodes that correspond to its neighbor vertices in G. We assume that all the nodes in the graph participate in the P2P protocol and that each node knows the identity of its neighbors.

Designing a lookup protocol in the Given-Topology Model is more challenging, because communication is heavily restricted. One possibility is to use unstructured search protocols (Section 2.2.3) such as Gnutella [Com01] or Gia [CRB+03], although such protocols have very inefficient *worst-case* performance (see the previously cited discussion of hay vs. needles in [CRB+03]). Note we would need to modify unstructured P2P protocols for them to work in the Given-Topology Model, since such protocols have been designed for the Internet, which is a Given-Routing setting. For example, Gia should be adapted by disabling its topology adaptation features, since the Given-Topology Model does not allow a node to change its set of neighbors

One of the major contributions of this dissertation is the design of an efficient lookup protocol in the Given-Topology Model. We will present such contribution in Chapter 3.

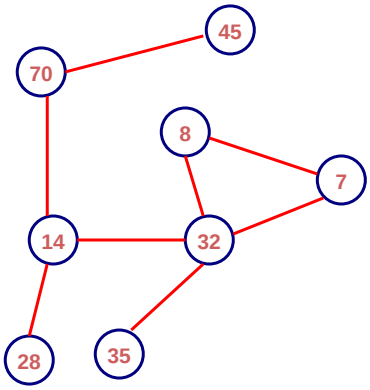


Figure 2.2: Given-Topology Model: every node is physically connected to some assigned neighbors. In this example, node 14 may send a message directly to the nodes 28, 32 and 70, but no others.

Chapter 3

Efficient Lookup in the Given-Topology Model

In this chapter,[1] we present LMS, an efficient lookup protocol for the Given-Topology Model (Section 2.4). It is more efficient than existing lookup protocols for unstructured networks, and thus is an attractive alternative for applications in which the topology cannot be structured as a Distributed Hash Table (DHT).

We present analytic bounds for the worst-case performance of LMS. Through detailed simulations (with up to 100,000 nodes), we show that the actual performance on realistic topologies is significantly better. We also show in both simulations and a complete implementation (which includes over five hundred nodes) that our protocol is inherently robust against multiple node failures and can adapt its replication strategy to optimize searches according to a specific heuristic. Moreover, the simulation demonstrates the resilience of LMS to high node turnover rates, and that it can easily adapt to orders of magnitude changes in network size. The overhead incurred by LMS is small, and its performance approaches that of DHTs on networks of similar size.

[1] The work presented in this chapter was published as [MBMS05, MBMS07].

3.1 Introduction

In this chapter, we present the Local Minima Search (LMS) protocol for the Given-Topology Model (Section 2.4.2). LMS extends the notion of random walks, which to date have proved the most promising approach to improving search performance on unstructured networks [LCC+02, GMS04, CRB+03]. In addition, LMS borrows the ideas of *namespace virtualization* employed by constrained-topology protocols such as distributed hash tables (DHTs) [RD01, ZKJ01, SMK+01, MNR02]. Namespace virtualization maps both peers and objects to identifiers in a single large space.

In LMS, the owner of each object places replicas of the object on several nodes. Like in a DHT, LMS places replicas onto nodes which have IDs "close" to the object. Unlike in a DHT, however, in an unstructured topology there is no mechanism to route to the node which is the closest to an item. Instead, we introduce the notion of a *local minimum*: a node u is a local minimum for an object if and only if the ID of u is the closest to the item's ID in u's *neighborhood* (those nodes within h hops of u in the network, where h is a parameter of the protocol, typically 1 or 2). In general, for any object there are many local minima in a graph, and replicas are placed onto a subset of these. During a search, random walks are used to locate minima for a given object, and a search succeeds when a minimum holding a replica is located.

While DHTs typically provide a worst-case bound of $O(\log n)$ messages sent for lookup operations in a network of n nodes, by storing only one replica of the data

item, LMS provides a worst-case bound of $O(\sqrt{n/d_h} \cdot (\log n)/g)$, with $O(\sqrt{n/d_h})$ replicas. In this expression, d_h denotes the minimum size of the h-hop neighborhood and g denotes the eigenvalue gap of a random walk in the graph (see Section 3.4). The eigenvalue gap g is a constant for a wide range of randomly-grown topologies (e.g. random and random-regular graphs). LMS is thus notably worse than DHTs, but is a considerable improvement over other (essentially linear-time) lookup techniques in networks that cannot support a structured protocol, and a vast improvement over flooding-based searches. Furthermore, as we shall see, LMS provides a high degree of fault-tolerance.

Note that the LMS protocol provides a *probabilistic* guarantee of success. Depending on the number of replicas placed and the number of search probes, an object may not be found even if it exists in the network, though the probability of failure can be made arbitrarily small. In Section 3.4, we derive expressions for the number of necessary replicas and probes for specific probabilities of success, for arbitrary graphs G. In particular, we deal with the following problem. Let D be an arbitrary distribution on a set of size k. Suppose we sample $r+s$ times independently from D; what is a good upper bound on the probability that the first r samples are disjoint from the last s samples? We present an upper-bound of $\exp(-\Omega(s \cdot \min\{r/k, 1\}))$, assuming without loss of generality that $s \leq r$. At first sight, this may appear related to the "birthday paradox", and D being the uniform distribution may appear to be the worst case. This turns out to not be true — this problem is more complex, as is illustrated in Section 3.7.

The object placement component of LMS distributes replicas randomly through-

out the network. This means that even if the LMS lookup is not used (e.g. for multi-attribute searches when the object ID-based lookup is not applicable) flooding will succeed with high probability even with relatively small bounded propagation. LMS can also be augmented with index-based searches in the same manner as DHTs. This flexibility suggests that LMS could become the underlying platform of choice for building wide-area applications.

We start with a description of related work in Section 3.2. The primary contribution of this chapter is the base LMS protocol described in Section 3.3, for which we derive expected and worst-case performance bounds in Section 3.4. We also analyze, in Section 3.5, the performance of LMS over realistic topologies via a set of comprehensive simulations on networks with 10^5 peers and over a testbed with 512 peers.

3.2 Related work

Gnutella [Com01] is probably the most-studied unstructured peer-to-peer network; much work has been done to understand Gnutella dynamics [RF02, SGG03, CLL02]. The original Gnutella search protocol was based on naive flooding. An improved flooding algorithm is discussed in [JGZ03]; it reduces the number of messages per search while still reaching the entire network. More sophisticated search techniques based on random walks are described in [LCC+02]. Their analysis yields a search cost of $\frac{n}{r}$, where n is the size of the network and r is the number of replicas, although it is based on the assumption that replicas are placed on uniformly

random nodes, while the paper does not describe any placement protocol. As a comparison, LMS achieves a search cost of $O(\frac{n}{rd_h}\log n)$, which is significantly better for the large class of topologies for which d_h is much larger than $\log n$. In Section 3.5 we directly compare LMS to the best protocol described in [LCC+02]. Yang and Garcia-Molina [YGM02] present more techniques and comparisons along the same lines.

The Gia system [CRB+03] improves on Gnutella using: topology adaptation, flow control and biased random walks. We do not compare LMS and Gia, because our model does not allow topology adaptation and, by design, the LMS substrate is not responsible for node heterogeneity and node congestion, without which flow control and biased random walks are of no use. Note that one can implement the latter two techniques on top of LMS. Also related is PlanetP [CAPMN03], a P2P system which enables searching and ranking documents on the basis of their content.

Highly-efficient DHT lookup algorithms (e.g., [SMK+01, RD01, ZKJ01, RFH+01, MNR02]) impose structure on the topology in order to achieve their performance bounds. Beehive [RS04] is a replication framework that can be built on top of a DHT. These systems employ the same namespace virtualization of which we make use. A virtualized namespace has previously been used on an unstructured topology in Freenet [CMH+02], though for a different reason.

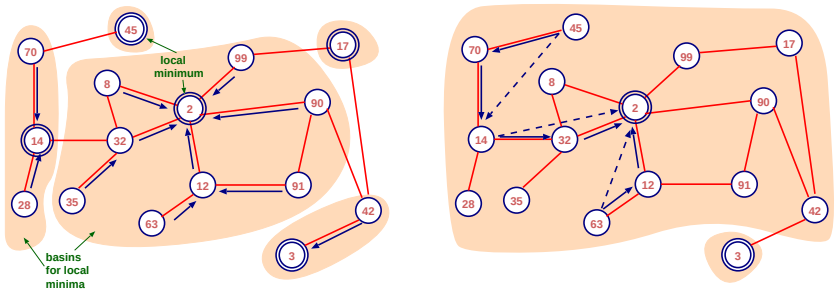
In [AM03] the authors build a P2P system where publishing involves storing on s random nodes in the graph and searching involves querying s independently chosen random nodes. Unlike LMS, [AM03] requires that the nodes may modify the topology of the neighborhood graph. The authors prove a result that is similar to our

Theorem 3.1 for the case $s = r$ in this chapter, although with a different technique. Other differences are: (a) LMS restricts replica placement to local minima; (b) our Theorem 3.1 is generalized beyond the special case in [AM03]; and (c) our proof technique for Theorem 3.1 also helps us show that LMS is robust under the loss of some replicas. (The result applies to [AM03] as well.)

YAPPERS [GSGM03] is a P2P system that, like LMS, assumes a "given topology" model and combines structured and unstructured designs. In YAPPERS, nodes publish an item by storing it at a node of the 2-hop neighborhood, which is structured as a small DHT. Search is achieved by flooding all and only the nodes in the network that *might* have the item, and requires nodes to keep state for an extended local neighborhood (within 5 hops). The authors do not present a formal analysis.

Related to the problem of lookup in a given topology is the work on name-independent routing (NIR) in a graph [ACL+03, AGM+04, AM05]. NIR allows a node to send a message to any other node, in a given topology setting. Existing work on NIR assumes that the topology graph never changes and requires an initial setup phase performed by a centralized algorithms that knows the entire graph.

Bloom filters [Blo70, Mit01], compact digests of sets of elements, have previously been considered for structured topologies [RK02] as a way to provide fast lookup. In this chapter, we present an application of Bloom filters on a completely unstructured topology. Theoretical properties of random walks have been exploited by [GMS04] to improve search in unstructured networks.



(a) 1-hop Neighborhoods (b) 2-hop Neighborhoods

Figure 3.1: Local minima, basins and deterministic forwarding. Nodes are labelled with their distances from the key. Concentric circles denote local minima, and shaded regions their basins. Arrows indicate the path from a node towards its local minimum. (a) All forwarding arrows for $h = 1$. (b) Forwarding arrows for two nodes with $h = 2$. A dashed arrow indicates the target towards which a node forwards a probe when different than the next hop.

3.3 LMS protocol description

We assume a system of principals (*nodes* or *peers*) structured as an overlay network on top of a communications infrastructure such as the Internet. The topology of this overlay network is dictated by external requirements, and its maintenance is beyond the scope of our protocol. We assume that nodes can communicate with one another if and only if they are neighbors in the overlay topology.[2] We model the topology as an undirected graph.

Nodes in the system have unique identifiers generated uniformly at random

[2]This assumption can be weakened and communications made possible over larger distances in the overlay. This does not substantially alter the protocol, other than to allow optimizations that reduce the number of messages exchanged.

from an *ID space* of all bit strings of some length λ. λ must be chosen large enough to guarantee uniqueness with high probability (for example, 160 bits). How this identifier is constructed is in general application-specific.[3]

We define a node v's *neighborhood* as the set of nodes at most h hops away from v in the overlay. For each of these nodes, v knows its unique identifier, how many hops away it is, and v's next hop towards it. We do not present a protocol for propagating this information, but it is easily implemented by repeated exchanges of a node's known $h-1$-hop neighborhood. Note that nodes have no knowledge of the topology beyond this neighborhood.

3.3.1 Protocol overview

LMS enables nodes to publish data items (i.e. files, documents) and to retrieve data items published by others. Items published by the system are given *key identifiers* (or simply *keys*) in the same ID space as the nodes, for example by hashing the name or contents of the item. The protocol then attempts to store an item at nodes that are close to its key in the (circular) ID space. In terms of the distance between a node and key, we do not find the global minimum (as in a distributed hash table), but rather a *local* minimum.

Publishing an item involves storing *replicas* of the item at a number of randomly selected local minima; retrieving an item involves querying randomly selected local minima until one is found that holds a replica. Crucially, the distribution of these random selections is typically not the uniform distribution; it is a function of

[3]For instance, the SHA-1 hash of a node's public key.

the underlying topology, and is naturally generated by the distributed algorithms that we employ. Intuitively, the more replicas are placed, the easier it will be to find one of them. Random minimum selection is accomplished by performing a random walk through the overlay before actively looking for a local minimum.

3.3.2 The basic protocol

For clarity of the exposition, we first present a slightly simplified version of the LMS protocol. In this version, placing and locating items are essentially identical. We will then refine this to account for the differences between the two operations and to add optimizations.

To select a random local minimum, a node generates a *probe* message, which has the general form ⟨`probe`, *initiator, key, walk_length, path*⟩. A probe moves through the network with a *random walk* followed by a *deterministic walk*. The *walk_length* parameter is initialized to some positive value. A node that receives the probe (including the initiator) first examines this parameter. If it is greater than zero, the probe is in the random walk phase, and the node forwards it to a randomly selected neighbor[4] and decrements the value. If *walk_length* is zero, the probe is forwarded according to the deterministic walk, which is described below. In either case, a node appends itself to *path* before forwarding the probe so that a

[4]The distribution for this selection is uniform in our case, but nonuniform choice is also possible. For instance, if the topology is dictated by trust relations, more highly trusted nodes might be given greater weight in selection. Another possibility, introduced in [CRB+03], is to weight the distribution according to resource availability.

local minimum can direct its response back to the initiator.[5]

Deterministic walk Deterministic forwarding (Figure 3.1) is done using a greedy algorithm. A node v receiving a probe computes the distance between *key* and every node in its neighborhood including itself. We define the distance between a key x and a node w as

$$d(x, w) = \min\{x - w \bmod 2^\lambda, w - x \bmod 2^\lambda\}.$$

Let v' be the node that minimizes $d(key, \cdot)$ over the neighborhood of v. If $v = v'$, then v is the local minimum; it then stores or returns a replica, depending on the type of probe. Otherwise, v determines the next-hop node towards v' (from its local knowledge of the topology) and forwards the probe to this node. (Recall that we assume v cannot communicate directly with v' if it is more than one hop away.) It is also possible that a message "intended" for a node v' will be redirected towards a better minimum v'' at an intermediate hop, as shown in Figure 3.2.

For a local minimum of a key x, v_x, we define the set of nodes that deterministically forward to this minimum as its *basin of attraction* (or *basin*). Note that while it is possible for a local minimum to "attract" nodes from outside of its neighborhood, it is also possible for the basin to be only the minimum itself. This is shown in Figure 3.1.

[5]If messages can be exchanged by nodes that are not direct neighbors, then *path* is not needed, since the local minimum can communicate directly with the initiator.

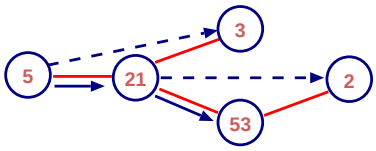


Figure 3.2: An illustration of "unexpected" message forwarding. Node 5 sees 3 in its 2-hop neighborhood, and forwards the probe to 21, the next hop towards 3. Node 21 sees a better match in 2, and so forwards the probe to 53 rather than 3.

3.3.3 Protocol details

Replica placement We now distinguish between probes for placing replicas and those for locating replicas. A `REPLICA-PLACE` probe is initiated by the owner of an item, and includes an additional field *item*. When a local minimum receives a `REPLICA-PLACE` probe, it checks if it already holds a replica, and if not whether it has the resources to store one. If the minimum is able to store a replica of the item, it informs the initiator.

When a local minimum cannot store a new replica, it performs *duplication avoidance*: it doubles the initial random walk length and restarts the probe's random walk. Because the probe is starting from a new location, it is less likely to again reach a duplicate minimum than if it were started with the same random walk length from the initiator.

We must limit the number of times that duplication avoidance is invoked, because the owner of an item might attempt to place more replicas than there are local minima for the item's key. This leads to `REPLICA-PLACE` probes that circulate through the network indefinitely, with progressively longer random walks. Consequently, we include a parameter initialized with the maximum number of failures permitted. Duplication avoidance decrements this parameter, discarding the

probe when it becomes negative. The final form of a `REPLICA-PLACE` probe is then ⟨`REPLICA-PLACE`, *initiator, key, walk_length, item, initial_walk_length, failure_count, path*⟩.

Replica lookup A `SEARCH-PROBE` performs a lookup operation on an item key. A local minimum that receives a `SEARCH-PROBE` checks whether it holds a replica of the item and returns either the replica or notice of failure to the initiator. The form of a `SEARCH-PROBE` is simply ⟨`SEARCH-PROBE`, *initiator, key, walk_length, path*⟩.

Note that LMS is a probabilistic algorithm. The probability of a single `SEARCH-PROBE` locating a replica is fairly low, so a node will have to initiate a number of probes when looking for an item. This can be done in serial or in parallel. The former increases the expected time until the node receives a successful response, while the latter increases the load on the system.

3.3.4 Variant: Bloom filters

In this section, we describe a variant of the protocol, called the *Bloom filter variant*, which improves the efficiency of item lookup at the expense of additional communications overhead.

Each node periodically constructs a Bloom filter [Blo70] of the keys for which it holds a replica and provides a copy of this Bloom filter to all the nodes in its h_B-neighborhood, where h_B is a global parameter. It is possible for h_B to be different than h, the depth of the neighborhood for the definition of local minimum (see Section 3.3).

Use of the filters within LMS is relatively straightforward: upon receiving a `SEARCH-PROBE` message, a node v can check all of the Bloom filters it holds for each of its h_B-hop neighbors. If there is a match of the key being searched for in the Bloom filter of some neighbor u, v can short-circuit the usual search protocol and forward the `SEARCH-PROBE` message directly to u.

Note that it is possible to use an *attenuated* Bloom filter [RK02] to combine incoming filters and transmit only a single filter for each distance. Thus, from each neighbor v a node receives a single filter for the items replicated at v, a single filter for all of v's one hop neighbors, a single filter for v's two hop neighbors, and so on. It is possible to further reduce the filter overhead by using arithmetic encoding to compress the filters (as described in [Mit01]).

In general, it is not necessary to keep (attenuated) Bloom filters for the entire neighborhood for which a node keeps ID information (i.e. in general one can choose h_B to be smaller than h). Often it is sufficient to choose h_B to be 1 or 2 to get significant benefit without incurring the (computation and bandwidth) overhead of constructing and disseminating large filters from multiple hops away.

In the special case that the Bloom filter and identifier neighborhood are identical (i.e., $h_B = h$), the search phase of the protocol can dispense with the deterministic walk (Section 3.3.2). More specifically, if $h_B = h$, the Bloom filter variant of LMS mandates that search probes only perform a random walk. The search probe fails if *walk_length* reaches 0 without finding a Bloom filter match at any node on the path.

3.3.5 The adaptive protocol

Having only local information, a node has no *a priori* way to know how many replicas of its items it must place in order for them to be easily found by other nodes. Increasing the number of replicas reduces the average number of search probes needed to find a replica, and vice-versa. Search performance thus provides a feedback mechanism to determine if an item has been replicated sufficiently.

The *adaptive protocol* is the component of LMS that determines and dynamically adjusts the number of replicas to be placed. It is run periodically for each item by its owner and ensures that a sufficient number r of replicas are available at any time. Let s be the average number of probes needed to find the item at a given time. The adaptive protocol ensures that $s \approx f(r)$, where f is an arbitrary function, chosen by the owner.[6] For simplicity of the exposition we present the protocol for the case $f(r) = r$ (see Section 3.4). The overhead of this protocol is minimal, aside from any additional replica placements, as it leverages existing actions.

A search from a random node in the system takes on average s probes to find a replica of an item I owned by node v. Node v periodically learns an estimate of s, as explained below; from the current number of reachable replicas r, it computes the number of replicas r' that it adaptively determines are needed as $r' = \lceil \alpha r + (1 - \alpha)s \rceil$. Here α is a hysteresis parameter between 0 and 1 (we use 0.9) that controls how sensitive the algorithm is to fluctuations in s. We define $\delta = r' - r$ as the needed change in number of replicas, and a non-zero value results in either

[6] For example, this could be used to take the popularity of an item into account by storing many more replicas so that most searches require very few probes.

placing or deleting replicas.

Since searches are a normal and frequent part of the system's operation, it is a simple matter for a node v_i to keep track of the number of probes s_i it needed to send before finding a replica of I.[7] If s_i is included in a `SEARCH-PROBE` message, the replica holder can store it and forward a batch of probe results to v periodically. These results provide a representative sample from which v can estimate the average s. Note that nodes on the threshold of being able to find replicas of I will tend to drive r up, allowing new nodes to find replicas and improving the sampling over the network.

3.4 Analysis of LMS

We now give a rigorous analysis of the protocol; we make no assumptions about the topology layer, other than the necessary condition that the topology, which is an undirected graph G, is connected. The three main results derived are: (a) tight bounds on the probability that a search in LMS fails; (b) bounds on the expected walk-length (i.e. the number of nodes visited by a search probe, Section 3.3.2) conditional on successful search; and (c) the asymptotic performance of LMS.

3.4.1 Mixing time and eigenvalue gap

We first define the *mixing time* of a graph and the *eigenvalue gap* of a random walk in that graph. We will use these concepts to analyze the performance of LMS.

[7] For an unpopular item, the owner of an item might select random nodes and request that they perform searches for the item to make up for the lack of search feedback.

We also refer the reader to [MR97, Chapter 6] for a more thorough discussion of random walks.

Let G be a connected undirected graph and let n denote the number of nodes in G. Let $\mathcal{RW}(u,t)$ be the distribution of the vertex visited by a random walk on graph G after t steps, starting at vertex u. Only for the purpose of this analysis, we define the random walk in a way that is slightly different than what we did in the protocol description (Section 3.3.2): at each step, the walk remains at the current vertex with probability 1/2 and otherwise moves to a uniformly random neighbor. Let P be the probability transition matrix of the random walk. Namely, $P[i,j]$ $(1 \leq i,j \leq n)$ is the probability that the random walk will move from node i to node j, at any step: for all edges (i,j) of G, it is $P[i,j] = 1/(2d_i)$, where d_i is the degree of node i; for all nodes i, $P[i,i] = 1/2$; $P[i,j] = 0$ otherwise. We denote by $1 = \mu_1 > \mu_2 \geq \cdots \geq \mu_n \geq 0$ the eigenvalues of P. We define the *eigenvalue gap* of the random walk as $g(G) = 1 - \mu_2$. [8]

It is a known fact in graph theory that $\mathcal{RW}(u,t)$ approaches a unique *stationary distribution* $\mathcal{SRW}$, as t approaches infinity, *no matter what the starting vertex u is.* $\mathcal{SRW}$ places probability $d(v)/(2m)$ on each node v, where $d(v)$ is the degree —number of neighbors— of v, and m is the total number of edges in G. We want to formalize this notion.

[8] We are aware that the *eigenvalue gap of a random walk in the graph* is a non-standard notion. Some authors define the *eigenvalue gap of the graph* as the difference between the two largest (in absolute value) eigenvalues of the adjacency matrix or of the normalized adjacency matrix. Those notions of eigenvalue gap differ from ours.

We can define the ϵ-mixing time of G, denoted by $T(G, \epsilon)$, as the smallest integer t such that for any vertex u:

$$\mathsf{SD}(\mathcal{RW}(\mathsf{u}, \mathsf{t}), \mathcal{SRW}) \leq \epsilon/2$$

where $\mathsf{SD}(\mathsf{D}, \mathsf{D}') = \frac{1}{2} \sum_{\mathsf{i}} |\Pr_{\mathsf{D}}[\mathsf{i}] - \Pr_{\mathsf{D}'}[\mathsf{i}]|$ denotes the statistical difference between the distributions D and D'. It follows immediately that, if $t \geq T(G, \epsilon)$, then the statistical difference between $\mathcal{RW}(u, t)$ and $\mathcal{RW}(v, t)$ is at most ϵ for every two vertices u and v.

It is known how to express the ϵ-mixing time as a function of the *eigenvalue gap*. To see this, we can apply a technique analogous to the proof of Theorem 6.21 in [MR97]: the statistical difference between $\mathcal{RW}(u, t)$ and $\mathcal{SRW}$ is at most $\sqrt{n}(\mu_2)^t$. From this, we obtain the following expression for the ϵ-mixing time:

$$T(G, \epsilon) \leq \frac{1}{g} \left[\frac{1}{2} \ln n + \ln(1/\epsilon) \right]. \tag{3.1}$$

For most natural models of randomly chosen/evolving graphs (such as random graphs and random regular graphs), it is known that the eigenvalue gap is lower bounded by a positive constant, independent of G and n; see Section 3.8 for details and references. For such classes of graphs, $T(G, \epsilon) = O(\log n + \log(1/\epsilon))$ suffices. There exist, however, pathological graphs in which $T(G, \epsilon) = \Omega(n)$. This happens, for example, if G is a path.

In LMS, ideally we would take our parameter `INITIAL-WALK-LENGTH` to be at least $T(G, \epsilon)$, for a sufficiently small ϵ, although this is not always possible because the topology G is unknown. In our simulations, choosing `INITIAL-WALK-LENGTH` to

be 3 and doubling its value each time a duplicate local minimum is found is sufficient to obtain very good results.

3.4.2 Probability of successful search

Consider an item x. Let r denote the number of replicas of x that the owner of x places, and let s be the number of search probes that another node conducts to search for x. We derive an upper bound on the probability that the search fails. If G is regular, we expect the r replicas of a key x to be placed uniformly at random at the k local minima for x; similarly for the s locations at which the s searches end. In such a case, the probability of "failure" (unsuccessful search) is essentially of the form $(1 - \frac{r}{k})^s \leq \exp(-\frac{rs}{k})$, where $\exp(a)$ denotes e^a. However, we desire protocols that work for arbitrary topologies G where the distribution may be quite non-uniform. Our first main result is that the failure probability is always bounded by a function of the form $\exp(-\Omega(rs/k))$, in the case of interest where r and s are bounded by $O(k)$. (Indeed, LMS is not an interesting algorithm if either r or s is $\Omega(k)$, because it becomes a trivial linear-time algorithm.)

Theorem 3.1. *Let G be any connected undirected graph, on which we run LMS with random walks of length at least $T(G, \epsilon)$. Let u and v be two arbitrary nodes in G. Let x be an arbitrary identifier. Let $K(x)$ be the random variable representing the number of local minima for x in G, with respect to a random ID assignment to the nodes of G. Suppose node u publishes an item with ID x, using r replicas, and suppose that, later, node v looks up an item with ID x using s search probes. Let*

p_f be the probability that v fails to find the item. Then, conditioned on the event $K(x) = k$, the following bound on the failure probability holds:

$$p_f \leq \min\{r, s\}\epsilon + \exp(-\Omega(\frac{rs}{k})).$$

The proof of this theorem relies on Theorem 3.5, a result on probability distributions that we present in Section 3.7. We consider Theorem 3.5 a contribution of independent interest.

Proof. We first present the main ideas of the proof. For simplicity, let's focus on the case that ϵ is so small that can be neglected. Since all the random walks are chosen to be of length $t \geq T(G, \epsilon)$, we can pretend that the distributions $\mathcal{RW}(u, t)$ of the random walk during publishing and $\mathcal{RW}(v, t)$ of the random walk during lookup are both equal to the stationary distribution $\mathcal{SRW}$.

Our goal is to show that even when r and s are only moderately large, a search for x will succeed with high probability, no matter what G is. We proceed as follows. Let D be the probability distribution of choosing a vertex w (which is a local minimum for x) as follows: choose a vertex v using distribution $\mathcal{SRW}$, and then deterministically go to a local minimum w starting at v, as described in Section 3.3. Then, replica-placement is as follows: choose a multiset R of r local minima by sampling r times independently from D, and place the replicas at the locations in R. Search for x is as follows: choose a multiset S of s local minima by sampling s times independently from D, and search is successful if and only if S intersects R.

When can we show that $\Pr[R \cap S \neq \emptyset]$ is high? The heuristic argument a

few paragraphs above shows that if D is the uniform distribution on K, then, the probability of unsuccessful search is of the form $\exp(-\frac{rs}{k})$. Theorem 3.5 (Section 3.7) shows that this bound indeed holds for any distribution D if $r, s \leq k$.

We are now ready for the full proof. Without loss of generality, we assume $s \leq r$. Let t be the length of the random walks used. For any vertex y, let D_y be the probability distribution of choosing a vertex w (which is a local minimum for x) as follows: choose a vertex w' using distribution $\mathcal{RW}(y, t)$, and then deterministically go to a local minimum w starting at w', as described in Section 3.3. Then, replica-placement is as follows: choose a multiset R of r local minima by sampling r times independently from D_u, remove duplicates from R, and place the replicas at the locations in R. Search for x is as follows: choose a multiset S of s local minima by sampling s times independently from D_v, and search is successful if and only if S intersects R.

We introduce a hybrid experiment, where we choose a set S^* of s random samples according to the distribution D_u. Theorem 3.5 implies that:

$$\Pr[R \cap S^* = \emptyset] \leq B(r, s, k),$$

where $B(r, s, k)$ is as defined in Theorem 3.5. Under the assumption $r, s \leq k$ of this theorem, $B(r, s, k) = \exp(-\Omega(\frac{rs}{k}))$.

We now have to relate $p_f = \Pr[R \cap S = \emptyset]$ with $\Pr[R \cap S^* = \emptyset]$. Since $t \geq T(G, \epsilon)$, this implies that the statistical difference between D_u and D_v is at most ϵ. Let D_u^s (resp. D_v^s) be the distribution consisting of s independent samples

from D_u (resp. D_v). A straightforward hybrid argument shows that

$$\mathsf{SD}(\mathsf{D}_\mathsf{u}^\mathsf{s}, \mathsf{D}_\mathsf{v}^\mathsf{s}) \leq \mathsf{s}\epsilon$$

which implies that the statistical difference between the random variables S and S^* is also at most $s\epsilon$.

We now need the following claim:

Claim 3.2. *Let X_1 and X_2 be two random variable with statistical difference at most ϵ. Let $E(X, Y)$ be an event that depends only on the underlying random variable X, Y (i.e. E is a predicate of two variables). Then, for any distribution of random variable Y:*

$$|\Pr[E(X_1, Y)] - \Pr[E(X_2, Y)]| \leq \epsilon.$$

Proof. Fix an arbitrary value y. We first show that the statement holds when conditioning on the event $Y = y$. Let:

$$A^+ = \{x : E(x, y) \text{ and } \Pr[X_1 = x] > \Pr[X_2 = x]\}$$

$$A^- = \{x : E(x, y) \text{ and } \Pr[X_1 = x] < \Pr[X_2 = x]\}$$

We get:

$$|\Pr[E(X_1, Y)|Y = y] - \Pr[E(X_2, Y)|Y = y]| =$$

$$|\sum_{x \in A^+} \Pr[X_1 = x] + \sum_{x \in A^-} \Pr[X_1 = x] +$$

$$\sum_{x \in A^+} \Pr[X_2 = x] + \sum_{x \in A^-} \Pr[X_2 = x]| =$$

$$\left| \sum_{x \in A^+} (\Pr[X_1 = x] - \Pr[X_2 = x]) - \sum_{x \in A^-} (\Pr[X_2 = x] - \Pr[X_1 = x]) \right|.$$

This quantity is maximized when the first summation is maximum and the second is minimum. Specifically, for any fixed choice of the distributions of X_1 and X_2, the quantity is maximized by choosing E such that $A^+ = \{x : \Pr[X_1 = x] > \Pr[X_2 = x]\}$ and A^- is empty. Therefore:

$$|\Pr[E(X_1, Y)|Y = y] - \Pr[E(X_2, Y)|Y = y]| \leq$$

$$\sum_{x:\Pr[X_1=x]>\Pr[X_2=x]} (\Pr[X_1 = x] - \Pr[X_2 = x])$$

and simple algebra shows that the right hand side of the last inequality is equal to $\mathsf{SD}(\mathsf{X}_1, \mathsf{X}_2)$, which is at most ϵ. The claim follows by unconditioning. □

Using the Claim, we can write:

$$|\Pr[R \cap S = \emptyset] - \Pr[R \cap S^* = \emptyset]| \leq s\epsilon$$

from which the theorem follows. □

In practice, we choose the parameters of LMS to obtain a small constant probability of failure. Therefore, Theorem 3.1 tells us that we need random walks of length $T(G, O(1/s)) = O((\log n)/g)$; as mentioned above, this is $O(\log n)$ for many realistic networks. We also need to choose r and s such that $rs = \Omega(k)$. We expand on this issue below.

3.4.3 Expected walk-length

A second quantity of interest is the number N of nodes visited by any single search probe. This, multiplied by the number s of probes, yields the total cost of a search query (similarly, we get a multiplier of r in the replica-placement). We can

write $N = T + l$, where T is the length of the random walk and l is the length of the deterministic walk. As we discussed above, $T = O((\log n)/g)$, where g is constant for practical networks. As far as l is concerned, we present the following result which shows that $l = O(\log n)$ with high probability:

Theorem 3.3. *For any graph G of n nodes and for any positive constant c, the number l of steps of any deterministic walk to a local minimum is at most $(c+1)\log n$ with high probability (at least $1 - 2/n^c$) over the choices of the node identifiers.*

Proof. Fix an arbitrary object x. We aim to show that the "deterministic walk to a local minimum for x" takes at most $O(\log n)$ steps in expectation, and also with high probability.

Recall that we currently hash to a very large universe of (say, 160-bit) IDs, and also recall our notion of distance between two hash values. Since the ID-space has size large enough (2^{160}), the situation is essentially equivalent to the following. Let C be a circle of unit circumference. We then hash each entity v (whether it is an object or a node) to a random point $h(v)$ on the circumference of C. The distance between two points p and q that lie on C, denoted $\Delta(p, q)$, is the length of the shorter of the two arcs that connect them; thus, $\Delta(p, q) \leq 1/2$. Note that this is the same notion of distance that we currently employ; thus, our notion of local minima etc. carries over here exactly. (That is, we always aim to find a neighbor that has the smallest distance $\Delta(\cdot, \cdot)$ to the value $h(x)$.) The utility of mapping on to the circle is that the analysis becomes smoother (e.g., via integrals).

Suppose we start at a node u_0, and are routing to a local minimum for x.

We assume without loss of generality that the hash value $h(x)$ of x, is the lowest point P on C. Whenever we say "distance", we will mean the distance function $\Delta(\cdot,\cdot)$. We now introduce some random variables. Let $u_1, u_2, \ldots$ be the successive nodes visited; if u_i is the local minimum found, then u_{i+1}, u_{i+2} etc. equal u_i. Let $M_i = \Delta(h(u_i), P)$ denote the distance "between u_i and x"; note that the sequence $M_0, M_1, \ldots$ decreases until we hit a local minimum. Our key idea is to show that this sequence decreases fast enough. For $t \geq 0$, let A_t be the indicator random variable for the event that the walk has **not yet** stopped after t steps (i.e., after visiting $u_0, u_1, \ldots, u_t$).

Our key lemma is the following:

Lemma 3.4. *For any $t \geq 1$ and any $z \in [0, 1/2]$, $\mathrm{E}[M_{t+1}A_t \mid M_tA_{t-1} = z] \leq z/2$.*

We will prove Lemma 3.4 below; let us now see why the lemma yields the desired $O(\log n)$ bound. Note that $\mathrm{E}[M_1A_0] \leq \mathrm{E}[M_1] \leq 1/2$; thus, Lemma 3.4 and an induction on t yield that $\mathrm{E}[M_tA_{t-1}] \leq 2^{-t}$. In particular, letting $T = (c+1)\log n$ where c is some suitable constant, we get

$$\mathrm{E}[M_TA_{T-1}] \leq n^{-(c+1)}. \tag{3.2}$$

Now, suppose $u_T = u$ and that $M_TA_{T-1} = z$. Node u has at most n unexplored neighbors; for each of these neighbors v, $\Pr[h(v) < z] = 2z$. (The factor of two comes from the fact that $h(v)$ can fall on either side of point P on the circle.) Thus, the probability that u is **not** a local minimum is at most $2nz$; more formally,

$$\forall z,\ \Pr[A_T = 1 \mid M_TA_{T-1} = z] \leq 2nz.$$

So,

$$\Pr[A_T = 1] \leq 2n \cdot \mathrm{E}[M_T A_{T-1}] \leq 2/n^c$$

by (3.2), which is negligible if, say, $c \geq 2$. Thus, the routing takes at most $O(\log n)$ steps in expectation, and also with high probability. □

We now prove Lemma 3.4:

Proof. First of all, we can assume that $z \neq 0$; since $M_{t+1} \leq M_t$ and $A_t \leq A_{t-1}$, the lemma is trivial if $z = 0$. Therefore, we have $A_{t-1} = 1$ and $M_t = z$; i.e., the walk has not stopped after visiting node u_{t-1}, and also $\Delta(h(u_t), P) = z > 0$. Let Y be the random variable denoting the number of "unexplored" neighbors of u_t; i.e., the number of neighbors of u_t that do not belong to the set $\{u_0, u_1, \ldots, u_{t-1}\}$. If $Y = 0$, then u_t is a local minimum, and hence $M_{t+1}A_t = 0$. We will now prove that for all $d \geq 1$,

$$\mathrm{E}[M_{t+1}A_t \mid ((M_t A_{t-1} = z) \wedge (Y = d))] \leq z/2. \tag{3.3}$$

If we can do so, we will be done, since if the lemma holds conditional on all positive values of d, it also holds unconditionally. For notational simplicity, we will from now on refer to the l.h.s. of (3.3) as Φ.

Fix some $d \geq 1$; in all arguments below, we are conditioning on the event "$(M_t A_{t-1} = z) \wedge (Y = d)$". Let $v_1, v_2, \ldots, v_d$ denote the d unexplored neighbors of u_t. If $h(v_i) > h(u_t)$ for all i, then u_t is a local minimum, and hence $M_{t+1}A_t = 0$. Therefore, conditioning on the value $y = \min_i d(h(v_i), P) \leq z$, and also considering

the d possible values of i that achieve this minimum, we get

$$\Phi = 2d \cdot \int_{y=0}^{z} y(1-2y)^{d-1} dy.$$

Again, the factor of two up-front comes from the fact that the "minimizing neighbor" v_i can fall on either side of point P on the circle. A simple computation yields

$$\Phi = \frac{1-(1-2z)^d \cdot (1+2zd)}{2(d+1)}. \tag{3.4}$$

We need to show that the r.h.s. of (3.4) is at most $z/2$. Changing variables to $y := 2z$ and rearranging, we need to show that

$$f(y) \doteq (d+1)y/2 + (1-y)^d(1+yd) \geq 1,$$

where $0 \leq y \leq 1$ and $d \geq 1$ is an integer. This inequality is easily seen to hold if $d = 1$, so we may assume $d \geq 2$.

It can be verified that $f'(y)$ equals $(d+1)/2 + d(1-y)^d - d(1+yd)(1-y)^{d-1}$. Also, $f'(1/d)$ equals

$$(d+1) \cdot (1/2 - (1-1/d)^{d-1}), \tag{3.5}$$

and $f''(y)$ equals

$$d(1-y)^{d-2}(d+1) \cdot (dy-1). \tag{3.6}$$

We see from (3.6) that f' has a unique minimum (in our domain $0 \leq y \leq 1$) at $y = 1/d$. Also, for integer $d \geq 2$, the function $(1-1/d)^{d-1}$ has value $1/2$ when $d = 2$, and decreases as d takes on higher integral values. Thus, (3.5) shows that $f'(1/d) \geq 0$. Since this minimum value is non-negative, it follows that $f'(y) \geq 0$ for $0 \leq y \leq 1$. So, since $f(0) = 1$, we get $f(y) \geq 1$ for all $y \in [0,1]$, as required. □

	Search cost	State
general case	$O(\frac{n}{rd_h}(\log n)/g)$	$O(d_h)$
$r = d_h = \Theta(n^{1/3})$	$O(n^{1/3}(\log n)/g)$	$O(n^{1/3})$

Table 3.1: Asymptotic performance of LMS. n is the number of nodes, d_h is the minimum size of a h-hop neighborhood and r is the number of replicas.

3.4.4 Asymptotic performance of LMS

Theorem 3.1 allows us to estimate the performance of LMS. For any given k, we can choose $rs = \Theta(k)$ to make the failure probability an arbitrary small constant. For example, if $r = s = \lceil 2\sqrt{k} \rceil$, the probability of unsuccessful search is essentially at most $e^{-4} \sim 0.018$. (In fact, this upper bound only holds for relatively regular graphs; as G gets more irregular, the failure probability becomes smaller.)

Note that the number of local minima is, *in expectation*, $k = O(n/d_h)$, where d_h is the minimum h-hop neighborhood size in the graph. This means that, in any class of graphs where $d_h = d_h(n)$, if LMS places $r = r(n)$ replicas, then a search requires $s = O(\frac{n}{rd_h})$ probes. This translates into a search cost of $O(\frac{n}{rd_h}(\log n)/g)$ by virtue of Theorem 3.3 and Eq. (3.1). As far as the *routing state* kept by a node, this is proportional to the size of its h-hop neighborhood. Since this can be much larger than d_h, we assume that a node has a cap $\Delta = O(d_h)$ on the number of nodes in its neighborhood it is willing to keep state for.

Table 3.1 summarizes the asymptotic performance of the protocol. The table also shows, for concreteness, the interesting special case, where $d_h = \Omega(n^{\frac{1}{3}})$ (i.e., the given topology has h-hop neighborhoods that are not too small) and where we choose r such that r and s are of the same order of magnitude.

3.5 Experimental results

In this section we present an experimental study of LMS using a simulation (described in Section 3.5.1) both to demonstrate large-scale behavior of the algorithm and to compare its performance on various topologies and with different extensions to the base protocol. In addition, we present results from a complete implementation of the base protocol.

3.5.1 Simulation methodology

The message-level protocol simulator (written in Perl, about 1600 lines of code) chooses a node, uniformly at random, and simulates replica placement performed by that node. A search is performed from another similarly chosen node, which sends out search probes one at a time and records how many probes are necessary to find the item. This placement-search simulation is repeated for 10,000 trials with the same graph and key.

We present results from three kinds of graphs: *random*, *power-law*, and *Gnutella*. The random graphs have uniform edge probability between any two nodes. (In order to generate very large random sparse graphs, we generate an even larger sparse graph and take the giant component). We consider a variety of random graphs with sizes ranging from 10,000 to 100,000 nodes with different average degrees. In a power law graph of n nodes, node i ($i = 1, \ldots, n$) has degree $d_i = \omega/i^{\alpha}$ for some positive constants ω and α. The properties of these types of graphs have recently been extensively studied in the context of the Internet [MMB00] and of peer-to-

peer networks [RF02]. Our power-law graphs are generated using the `inet` (version 3.0) topology generator [WJ02], which is intended to model AS-level topology of the Internet. Specifically, we consider 10,000 node `inet` graphs. For this topology size, `inet` yields graphs with average degree 4.11. The third type of graph we use represents a snapshot generated by crawling the Gnutella peer-to-peer network. The Gnutella topology has 61,274 nodes with average degree 4.70, and also exhibits some power-law type properties [RF02]. Nominally, for all topologies, we assume that each peer keeps identity information for 2-hop neighbors.

3.5.2 Lookup performance

The results of the first set of experiments are shown in Table 3.2. The Gnutella graph represents a crawl of the actual deployed system [Rip, RF02], while the others are generated. One randomly generated item is replicated in each trial, and the number of replicas is determined by an iterative procedure so that the average number of lookup probes needed to find one replica is approximately equal to the number of replicas. The "Visited" column shows the cost of the search either using the basic LMS protocol (the "LM" column) or the Bloom filter variant with $h_B = 2$ (the "LM+BF" column, see Section 3.3.3). The costs listed are the total number of nodes visited by the lookup probes until a replica is located (including all unsuccessful probes, executed serially).

The 10K graphs were chosen for comparison with the results of [LCC+02]. The most successful protocol in [LCC+02], *check*, visits approximately 150 nodes (slightly

Graph Type	Size	Avg. deg.	# Repl.	Visited	
				LM	LM+BF
power-law	10K	4.11	3	17.7	4.4
random	10K	4.11	22	131.1	21.8
Gnutella	61K	4.7	16	83.9	15.7
random	61K	4.7	45	282.8	43.8
random	100K	17	14	55.9	14.0
random	100K	12	19	87.1	19.0
random	100K	7	34	185.4	34.0

Table 3.2: Lookup performance for different graphs. The average number of lookup probes is approximately equal to the number of replicas. Measured quantities are averaged over 10,000 trials for each of 60 generated graphs, excluding the Gnutella graph.

more than our 130), but requires over 90 replicas in contrast to our 22. As the graphs grow larger, LMS continues to perform reasonably. On the Gnutella graph, in particular, LMS requires relatively few replicas and lookup probes; we expect this graph to be typical of many applications with unstructured topologies. Note that the Bloom-filter-based lookup performs considerably better than the standard LMS lookup, but this comes at the cost of maintaining and propagating Bloom filters.

In this experiment, we assume homogeneous nodes we do not take into account the effects of node congestion. Further our model does not allow a node to add neighbors. Therefore we do not discuss a comparison with Gia here (see Section 3.2).

Random graphs show the worst-case performance of LMS. In order to demonstrate the lower-bound behavior of the protocol, we restrict our experiments to these graphs hereafter.

3.5.3 Number of replicas vs. lookup overhead

In this section, we further consider details of LMS performance. We introduce the notion of an *iso-success curve*. An *iso-success* curve, for a given graph G and a given probability p, is the set of all pairs (r, s), such that if the owner of an item places r replicas and a node v uses *up to* s search probes to search the item, then the probability that v finds the item is p.

Iso-success curves precisely capture the tradeoff between number of replicas and lookup overhead. In Figure 3.3, we present the iso-success curves for a single random graph of size 100K nodes (average degree 17). The figure shows the *worst-case* number of probes required for each number of replicas to achieve the given level of success probability (without using Bloom filters). For example, a little more than 20 probes were required with 10 replicas for 70% probability of finding an item. The curve was obtained by measuring the distribution of the number of probes for each number of replicas, using 10,000 trials.

For each success probability, the average number of search probes for each number of replicas was fit to the function $f(r) = \frac{a}{r} + b$, using the standard deviation of the numbers of probes as the uncertainty in their average. The fits demonstrate that $rs = Constant$ for a fixed probability is a very good approximation (b is small and negative in all fits).

A global fit to all of the data reveals that approximately half of the local minima dominate the network, greatly reducing the expected number of replicas needed for reliable lookup ($k/2$ rather than k in the results of Section 3.4).

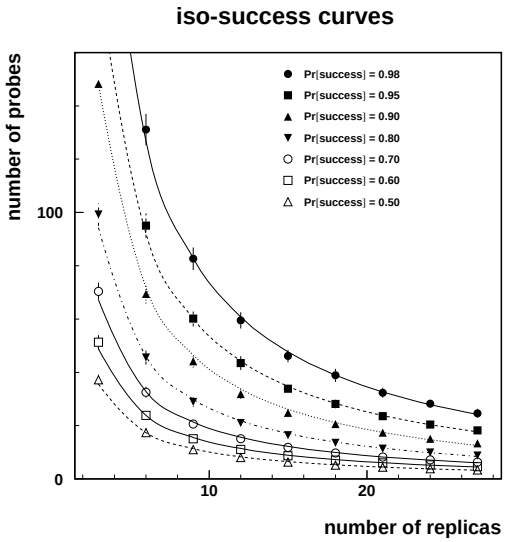


Figure 3.3: Number of probes as a function of the number of replicas. Random graph with 10^5 nodes, of average degree 17. The numbers labeling the curves are the particular success probabilities. The curves show fits to the functional form $f(r) = \frac{a}{r} + b$.

Along with the tradeoff measure, the iso-success curves also provide us with a snapshot of the distribution of the expected number of probes required for lookups on a given graph. For example, with 15 replicas, 50% of the lookups require less than 8 probes (in the worst case), and only 5% of the lookups required more than 40 probes. This information is useful in implementation since it provides a strategy for fixing the number of probes that should be launched in parallel in order to reduce lookup latency.

3.5.4 Failure analysis

LMS is extremely robust, and in this section, we analyze its resilience under the failure model described in Section 3.4. Specifically, we consider random failures in which a given fraction of nodes in the networks fail. In these results, we consider

Failure Probability	# Replicas	Average Visited	Analytic Bound
0	36	188	36
0.1	38	200	37.9
0.2	41	213	40.2
0.3	45	231	43.0
0.4	48	262	46.5
0.5	53	289	50.9

Table 3.3: Performance under random failures: random graph with 10^5 nodes, average degree of 7. The analytic bound predicts the number of replicas which in this case is equal to the number of probes.

how many replicas should be provisioned to handle specific failures (and validate our analysis). In these experiments, the adaptation is via static provisioning, i.e. the application/system designer places extra replicas. In later simulation (Section 3.5.5) and implementation results (Section 3.5.6), we consider how the system can adapt at run-time as it becomes aware of new failures.

In Table 3.3, we consider a random graph with 100,000 nodes (average degree 7). We consider that each replica, after being placed, can independently fail (the node discards the data) with the probability specified in the first column. In each case, we present an average over 10,000 runs of the number of replicas that needed to be placed *before the failures* such that the probability of a successful search *after the failures* is at least 99%. We present the specific placement that equalizes the number of search probes and the number of replicas.

If f is the failure probability, the number of replicas that equalizes the expected number of probes is $r(f) = \frac{r(0)}{\sqrt{1-f}}$. This analytic prediction is the last column of the table. It is clear that the analysis is extremely accurate in this case, and that LMS scales extremely well with failures. The most important point to note here is that

the number of replicas only has to scale with the square root of the fraction of failures, which is an extremely positive outcome.

3.5.5 Performance under high churn

We now examine the behavior of LMS in a network with a high rate of nodes leaving and joining. The topology is a random graph of 10K nodes of average degree 7. A departing node is immediately replaced by a new node with the same number of neighbors; this maintains both the size of the graph and the degree distribution. Bloom filters are not used in this experiment. The neighborhood distance h is 2.

At the start of the simulation, a node u creates one replica of its item. Every minute (in simulation time) a random node searches for the item (over 99% of all searches succeed). The simulator also chooses q nodes at random (excluding u) to remove, where $q = 10000/T$ for an average node lifetime of T minutes. Every three minutes, the item's owner u selects a node by initiating a random walk (of length 6) and has that node perform a search for the item. Based on the result of the search, u applies the adaptive protocol (Section 3.3.5); placing or removing replicas as needed. The number of replicas r needed is based on a *cost ratio*, an input parameter defined as r/s, where s is the expected number of probes to find the item[9]. The adaptive protocol then attempts to equalize the replication cost and the total cost of all (expected) searches.

The results of the experiment are shown in Table 3.4, which averages results

[9]In the notation of Section 3.3.5, this means running the adaptive protocol with $f(r) = r/(\text{cost ratio})$.

Cost Ratio	Node Lifetime	Search Cost	Adaptive Overhead	Average Replicas
2	15	44.2	69.8	16.4
	30	35.9	63.4	20.3
	60	37.2	53.2	18.6
5	15	29.6	116.6	27.4
	30	35.7	70.7	23.8
	60	32	62.1	25.8

Table 3.4: Cost of searching, publishing and maintaining the overlay in basic LMS, in presence of high node churn. Lifetimes are measured in minutes.

over seven initial random graphs. The search cost is collected from the one-minute lookups. The adaptive overhead includes the cost of the search initiated by u and the cost of placing additional replicas. The search cost and number of replicas are fairly stable over all scenarios. The adaptive overhead is also relatively stable, with the exception of 15-minute lifetimes for a cost ratio of 5. Note that we do not include the overhead of maintaining the network, as it is external to the LMS protocol.

3.5.6 Implementation

We have implemented the complete LMS protocol (without Bloom filters). Multiple nodes are run on a single host, enabling us to test fairly large networks. The topology is maintained by a separate topology server that creates edges between nodes at random while enforcing a minimum degree for each node. All nodes are fail-stop; as nodes leave the network their neighbors obtain replacements as needed to maintain the minimum degree. The available bandwidth for each host and the capacity of the topology server define the limits of the network size that we were able to construct. The deployment was limited to locally available hosts.

Two experiments were performed, one with 512 nodes and one with 1024, both focusing on the behavior of the adaptive protocol, and both with a minimum node degree of 3 (average 3.9 ± 1.1) and $h = 2$. For the smaller network, we investigate a system in which searches and adaptation are frequent. The purpose of this experiment is to demonstrate that our protocols are functional in a real deployment under high (for the limited resources available) load. The behavior of the adaptive protocol is shown in Figure 3.4. Each node replicates one item[10] and performs a search for a random item every 10 seconds (staggered by random offsets at start-up), so that a new search is started roughly every 20 milliseconds. The adaptive protocol is run every 20 seconds (again staggered), and is based on the results of these searches. The average number of replicas tends to converge on a value dependent on the specific topology. The standard-deviation error band around the average number of replicas is dominated by the variation in the number of search probes, which we expect to be close to 1, as observed.

LMS probes increase in size as they propagate, but almost all are 400 bytes or smaller (excluding item size), and all are less than 1000 bytes. A successful lookup sends an "adaptive hint" back to the item's owner, the average size of which is 28.5 ± 0.5 bytes; most of this is the key identifier and can be batched by the replica holder or otherwise amortized. Replicas are soft state, so the owner of an item must send a

[10]Note that the number of items per node is not important when considering network load. What matters is the number of probes circulating through the network, or essentially the number of searches initiated per second. Our experiments are fundamentally limited by having eleven I/O-intensive processes on a single host, forcing us to restrict the rate at which searches are initiated.

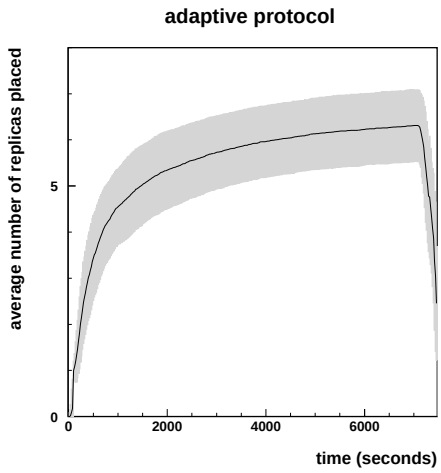


Figure 3.4: Effect of the adaptive protocol. The x axis shows elapsed time, with 20s between subsequent runs of the adaptive protocol. The y axis shows the average number of replicas placed for all items in the system. The shaded region indicates the one standard deviation region. The steep drop-off at the right is due to the system shutting down.

refresh message (66.9 ± 0.8 bytes on average) to replica holders, which is done every 20 seconds in this experiment. Replicas that are not refreshed are garbage-collected by their holder. Neighborhood propagation consumes considerable bandwidth, but we have constructed a much more efficient protocol that only propagates changes.

The experiment with 1024 nodes was used to verify that the adaptive protocol can cope with the sudden loss of half the network. Only one item is replicated, and its owner initiates periodic searches for it. Halfway through the experiment 512 of the nodes depart. We do not present specific results of this, as providing no special insight into the system, but mention that it performed very well under extremely adverse conditions.

3.6 Conclusions

In this chapter, we designed LMS, an efficient lookup P2P protocol for the Given-Topology Model. Unlike a DHT [SMK+01, RD01, RFH+01], LMS is designed for a model in which the topology (i.e. the graph where a vertex denotes a peer and an edge denotes that two peers know of each other and can communicate directly) is determined by an external entity, therefore peers are not allowed to choose their neighbors. We demonstrated through analysis and simulation that LMS is an efficient protocol with stronger performance guarantees than other unstructured protocols that work in the same model. We also presented a prototype implementation, which shows the practicality of the design.

LMS is of practical interest for distributed applications relying on trust relations between nodes. These trust relations define a graph which, when traversed along its links, provides a known level of assurance for operations. Specifically, we developed a decentralized public-key infrastructure based on a web-of-trust. Traversing links in the trust graph implicitly generates certificate chains assuring a node searching for a public key of that key's correctness. More details on this application will be available in [MBKM06].

3.7 A Useful Result on Distributions

In this section, we present our result on probability distributions, which we use to prove Theorem 3.1 and is also of independent interest.

Theorem 3.5. *Let r, s, k be three integers. Let D be an arbitrary distribution over*

the set $K = \{1, \ldots, k\}$. *Let* R *be a set obtained by taking* r *independent samples from* D *and let* S *be a set obtained by taking* s *additional independent samples from* D. *Then:* $\Pr[R \cap S = \emptyset] \leq B(r, s, k)$ *where:*

$$B(r, s, k) = \begin{cases} \exp(-\Omega(s \cdot \min\{r/k, 1\})) & \text{if } s \leq r \\ \exp(-\Omega(r \cdot \min\{s/k, 1\})) & \text{if } s \geq r. \end{cases}$$

Before proving this theorem, we make an observation. It seems reasonable to conjecture that D being the uniform distribution is the worst case (that is the case in which $\Pr[R \cap S = \emptyset]$ is highest), since otherwise elements of K that have high probability could appear in both R and S with increased likelihood. This intuition turns out be false: consider the case where $k = 2$, $s \ll r$, and D places probabilities $s/(r+s)$ and $r/(r+s)$ on the two elements of K. In such case, $\Pr[R \cap S = \emptyset]$ is approximately $(s/r)^s \cdot \exp(-s)$ which is much larger than the value $2^{-(r+s-1)}$ obtained in the uniform case. In general, the case where r and s are quite different, needs care.

The proof of this theorem relies on the notion of *negative association* and a result on balls and bins, both due to Dubhashi and Ranjan [DR98], which we now recall.

Definition 3.6. *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is* non decreasing *(resp.* non-increasing*) if, for any* $x, y \in \mathbb{R}^n$, *if* $x_i \leq y_i$ *for all* $i = 1, \ldots, n$, *then* $f(x) \leq f(y)$ *(resp.* $f(x) \geq f(y)$*).*

Definition 3.7. *[DR98, Definition 3] Random variables* $X_1, X_2, \ldots, X_n$ *are* nega-

tively associated *if, for every two disjoint index sets* $I, J \subseteq [n]$*,*

$$\mathrm{E}[f(X_I)g(X_J)] \leq \mathrm{E}[f(X_I)]\,\mathrm{E}[g(X_J)] \tag{3.7}$$

(where $X_I = (X_i)_{i \in I}$*) for all functions* $f : \mathbb{R}^{|I|} \to \mathbb{R}$ *and* $g : \mathbb{R}^{|J|} \to \mathbb{R}$ that are both non-increasing or both non-decreasing.

Lemma 3.8. *[DR98, Proposition 12] Consider an experiment where we throw* m *balls into* n *bins independently at random and such that the probability that ball* j *goes into bin* i *is* $p_{i,j}$*. The* $p_{i,j}$ *are arbitrary, except that, for each* j*,* $\sum_i p_{i,j} = 1$*. Let* $B_{i,j}$ *be the random variable indicating whether ball* j *falls into bin* i*. Then, the random variables* $B_{i,j}, i \in [n], j \in [m]$ *are negatively associated.*

We are now ready to prove Theorem 3.5.

Proof. Before diving into the proof details, we first discuss a simple approach to the problem and we show why it does *not* work. One would be tempted to proceed as follows: denote the s samples from D that form the set S by the random variables $e_1, \ldots, e_s$ and try to express $\Pr[R \cap S = \emptyset]$ as a function of $y = \Pr[e_i \notin R]$. We immediately note a stumbling block: given that one element of S did not lie in R (over the random choices of both the e_i's and R), the conditional probability of this happening for another element of S can go up! Indeed $e_i \notin R$ is a sign that the set R is "small" and this fact increases the probability that e_j does not fall in R either. Thus we have an undesirable positive correlation among these events: in particular, if y is the probability that an arbitrary single element of S does not lie in R, then the probability that R and S are disjoint is *at least as large as* y^s.

We take a different approach, wherein the correlations actually help us. Recall that $K = \{1, 2, \ldots, k\}$, and let X_i be the event that i lies in both R and S. As usual, $\overline{X_i}$ denotes the complement of X_i; letting p_i be the probability that distribution D places on i and setting $q_i = 1 - p_i$, we have $\Pr[\overline{X_i}] = (q_i^r + q_i^s - q_i^{r+s})$.

The probability that R and S are disjoint is $\Pr[\bigwedge_{i=1}^{k} \overline{X_i}]$. Our first key idea is that the events $\overline{X_i}$, $i = 1, 2, \ldots, k$, are negatively correlated! Intuitively, this seems clear: given that none of $X_1, X_2, \ldots, X_{i-1}$ held, this informally "leaves more slots free" for X_i to occur. We prove this by using the above-mentioned result by Dubhashi and Ranjan. For $i = 1, \ldots, k$ and for $j = 1, \ldots, r + s$, let $B_{i,j}$ be the indicator of the event $\{\sigma_j = i\}$, where $\sigma_1, \ldots, \sigma_r$ are the samples from D that constitute R and $\sigma_{r+1}, \ldots, \sigma_{r+s}$ are the samples that constitute S. Lemma 3.8 tells us that the $B_{i,j}$ are *negatively associated*. For every i, we can write the event $\overline{X_i}$ as $f(B_{i,1}, \ldots, B_{i,r+s})$ and the event "$\bigwedge_{l=1}^{i-1} \overline{X_l}$" as $g((B_{l,j})_{l=1,\ldots,i-1;j=1,\ldots,r+s})$, where f and g are appropriate non-increasing functions. Applying the definition of negative association, we get that $\overline{X_i}$ as "$\bigwedge_{l=1}^{i-1} \overline{X_l}$" are negatively *correlated*, as needed. This negative correlation leads to the bound

$$\Pr[\bigwedge_{i=1}^{k} \overline{X_i}] \leq \prod_{i=1}^{k} \Pr[\overline{X_i}] \tag{3.8}$$

$$= \prod_{i=1}^{k} (q_i^r + q_i^s - q_i^{r+s}). \tag{3.9}$$

W.l.o.g., we assume $s \leq r$. The basic idea to complete the proof is as follows.

Partition K into three sets:

$$\begin{aligned} C_1 &= \{i : p_i \leq 1/r\}; \\ C_2 &= \{i : 1/r < p_i \leq 1/s\}, \text{ and} \\ C_3 &= \{i : p_i > 1/s\}. \end{aligned}$$

The product in (3.9) now splits into three products, which we will analyze separately and then combine. For $j = 1, 2, 3$, let $v_j = \sum_{i \in C_j} p_i$ and note that the sum $v_1 + v_2 + v_3 = 1$.

Let us first consider any $i \in C_1$. Since $0 \leq p_i \leq 1/r$, it is easy to show here that

$$q_i^r + q_i^s - q_i^{r+s} = 1 - \Theta(rsp_i^2).$$

Next, basic convexity arguments show that for any constant $\alpha > 0$, and any given values for $|C_1|$ and v_1, the quantity $\prod_{i \in C_1}(1 - \alpha rsp_i^2)$ is maximized (as a function of the variables p_i) when $p_i = v_1/|C_1|$ for each $i \in C_1$. Further simplification yields

$$\prod_{i \in C_1}(q_i^r + q_i^s - q_i^{r+s}) \leq \exp(-\Omega(rsv_1^2/|C_1|)). \tag{3.10}$$

Next consider C_2. In this case,

$$q_i^r + q_i^s - q_i^{r+s} = 1 - \Theta(sp_i).$$

A simple argument yields

$$\prod_{i \in C_2}(q_i^r + q_i^s - q_i^{r+s}) \leq \exp(-\Omega(s \cdot v_2)). \tag{3.11}$$

The set C_3 is where the function $q_i^r + q_i^s - q_i^{r+s}$ exhibits complex behavior. Fortunately, we can deal with it as follows. For any $i \in C_3$, $q_i^r + q_i^s - q_i^{r+s}$ is at most

$$q_i^r + q_i^s \leq 2 \cdot q_i^s \leq 2 \cdot \exp(-sp_i) \leq \exp(-\Omega(sp_i)),$$

where the final inequality follows from the fact that $p_i \geq 1/s$. Therefore,

$$\prod_{i \in C_3} (q_i^r + q_i^s - q_i^{r+s}) \leq \exp(-\Omega(s \cdot v_3)). \tag{3.12}$$

Putting (3.10), (3.11) and (3.12) together with (3.9), we get that for some constant $\gamma > 0$, $\Pr[\bigwedge_{i=1}^{k} \overline{X_i}]$ is upper-bounded by

$$\exp(-\gamma \cdot s \cdot (rv_1^2/|C_1| + v_2 + v_3)),$$

which equals $\exp(-\gamma \cdot s \cdot (rv_1^2/|C_1| + 1 - v_1))$; this, in turn, is at most $\exp(-\gamma \cdot s \cdot (rv_1^2/k + 1 - v_1))$.

We now apply elementary calculus to determine the maximum of this last expression as a function of v_1 (where v_1 takes values in $[0, 1]$). There are two cases. If $r \leq k/2$, then the function is maximum for $v_1 = 1$, in which case the bound becomes $\exp(-\gamma rs/k)$. If $r > k/2$, the maximum occurs for $v_1 = k/(2r)$, in which case the bound becomes $\exp(-\gamma s(1 - k/(4r))$ which is at most $\exp(-(\gamma/2)s)$. In the special case $k/2 < r \leq k$ this can be further upper bounded by $\exp(-(\gamma/2)rs/k)$.

In summary, we have shown that, for $s \leq r$:

$$\Pr[R \cap S = \emptyset] \leq \begin{cases} \exp(-(\gamma/2)rs/k) & \text{if } r \leq k \\ \exp(-(\gamma/2)s) & \text{if } r > k \end{cases}$$

which is what we wanted to prove. We conclude by pointing out γ can be taken to be 0.13, as it can be shown by a more careful analysis. □

3.8 Mixing time in certain classes of graphs

In this section, we justify our claim that random and random regular graphs have constant eigenvalue gap, and therefore logarithmic mixing time, under reason-

able assumptions.

3.8.1 Mixing time in random graphs

A random $G(n,p)$ graph has a constant eigenvalue gap, i.e. $g = \Omega(1)$, with high probability, if $p = \Omega((\ln n)/n)$. Although we believe this fact was previously known, we could not find a reference. Therefore, we provide a simple proof sketch.

Let Φ be the expansion factor of a graph. If $p \geq 4(\ln n)/n$, then the random graph $G(n,p)$ has an expansion factor of at least $.066np$, with high probability (at least $1-1/n$). We prove this, by showing that for all $t = 1, \ldots, n/2$, and for any set S of t nodes in the graph, the probability that S has too few outgoing edges (i.e. less than $(1-\delta)$ times the expected number of outgoing edges) is at most $\frac{1}{n\frac{n}{2}\binom{n}{t}}$. The proof is a standard application of the Chernoff bound.

Let P be the probability transition matrix of a random walk on a graph G, as defined in Section 3.4.1, and let $1 = \mu_1 > \mu_2 \geq \cdots \geq \mu_n \geq 0$ be its eigenvalues.

Frieze [Fri00, Eq.(12,13)] gives the following bound on the eigenvalue gap $g = 1 - \mu_2$:

$$g \geq \frac{\Phi^2 \delta^2}{8\Delta^4},$$

where Φ is the expansion factor of G and δ and Δ are the minimum and maximum degree of G.

In case of $G(n,p)$, if $p \geq 8(\ln n)/n$, then $\delta \geq .13p(n-1)$ and $\Delta \leq 1.87p(n-1)$, with high probability (application of Chernoff bound). This means that $\delta^2/\Delta^4 \geq .00131/(p(n-1))$. Therefore $g \geq .39$ with high probability.

Note that the condition $p \geq 8(\ln n)/n$ is not really restrictive, because a similar condition $p \geq (1+\epsilon)(\ln n)/n$ is required for the graph to be connected with high probability.

3.8.2 Mixing time in random regular graphs

Friedman [Fri03] shows that the second eigenvalue of the *adjacency matrix* of a random d-regular graph is at most $2\sqrt{d-1}+\epsilon$, with high probability, for any fix value of d. This implies our claim that the graph has constant eigenvalue gap, as explain below.

If we call A the adjacency matrix of a random d-regular graph and we call P the probability matrix of the random walk, it is easy to see that $P = \frac{1}{2}(I + A/d)$. Let λ_i (resp. μ_i) be the eigenvalues of A (resp. P). It follows that $\mu_i = \frac{1}{2}(1 + \lambda_i/d)$.

The second eigenvalue is therefore $\mu_2 \leq 1/2 + \sqrt{d-1}/d + \epsilon/2d$ which is at most $0.98 + \epsilon/6$, in the worst case $d = 3$. This implies that $g = 1 - \mu_2$ is a positive constant, as needed.

Chapter 4

Lookup Protocols and Security

In the previous chapter, we defined lookup protocols and we presented several examples. So far, our discussion of those algorithms assumes that all processes in the systems behave according to the prescribed protocol, except for the fact that processes may exhibit stopping failures. In this chapter, we concern ourselves with the possibility that some processes may arbitrarily misbehave. We will say that a lookup protocol is *secure*, if it can operate correctly in such setting. In the course of the chapter, we clarify the notion of security further and we present previous work on the topic.

Next, we present a new model of security for lookup protocols, which we will call the *Autonomous System (AS) Model*, and we argue that it is more realistic than the security models considered in previous work, at least for some applications. The definition of the AS Model is one of the main contributions of this dissertation.

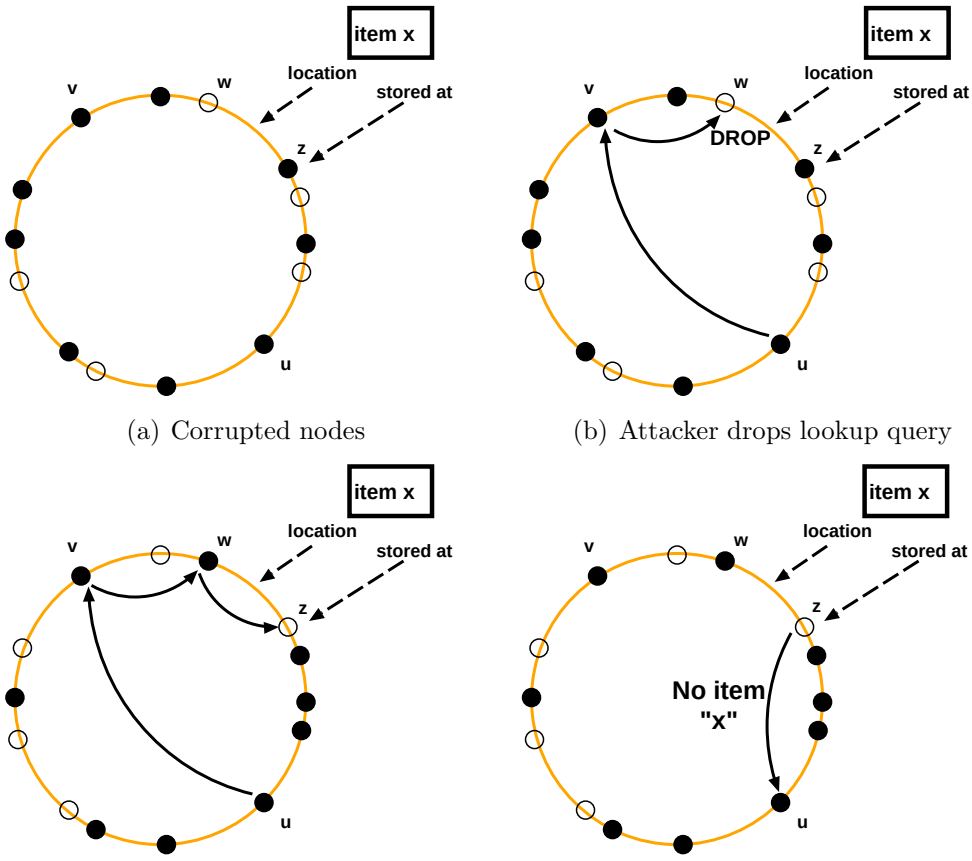
4.1 Attacks on Existing Lookup Protocols

An important aspect of distributed applications is their ability to operate correctly even in the presence of malicious participants. With respect to lookup protocols, an attacker could try to tamper with the system behavior in order to achieve several goals:

- *denial of service*, i.e. preventing most or all nodes from retrieving any published data item;
- *selective denial of service*, i.e. preventing certain nodes from retrieving certain data items;
- *data corruption*, i.e. causing nodes to retrieve data items that have never been published.

In general, an attacker may interfere with the system in several ways, for example, by controlling some of the nodes in the P2P system, by intercepting, modifying or preventing delivery of messages sent between nodes. In this dissertation, we focus on attackers that control some of the nodes that are part of the P2P system, while we assume that the attacker cannot interfere with messages between nodes that she does not control. We believe that this problem is the most interesting. Informally, we will say that a lookup protocol is *secure*, if it is resilient to a wide range of attacks. Following previous work, we refer to a node controlled by the attacker as a *corrupted* or *malicious* node and to other nodes as *correct* or *honest* nodes.

While DHTs such as Chord [SMK+01] or Pastry [RD01] are very efficient and resilient in a cooperative setting, they are relatively easy to manipulate by a few corrupted nodes. We illustrate this fact with the example in Figure 4.1. In this example, a data item with identifier x has been previous published and stored at the home node and now node z. Another node u is performing a lookup for x. In a first scenario (Figs. 4.1(a) and 4.1(b)), the lookup request message generated by u is recursively forwarded to the correct node v and then to the corrupted node w. Then



(a) Corrupted nodes

(b) Attacker drops lookup query

(c) Attacker controls home node ...

(d) ... and gives incorrect information

Figure 4.1: Two different attacks against Chord. White nodes are under control of the adversary. A data item with identifier x has been previous published and now node u is performing a lookup for x. In the first attack (Figs. 4.1(a) and 4.1(b)), the adversary blocks the lookup request, because it controls one of the nodes in the path of the request. In the second (Figs. 4.1(c) and 4.1(d)), the attacker controls the home node corresponding to x. This allows the attacker to incorrectly report to u that no item with that identifier was published.

the attacker simply discards the request message, thus preventing the lookup from completing. In a different scenario (Figs. 4.1(c) and 4.1(d)), the attacker controls the home node z corresponding to x. This allows the attacker to incorrectly report to u that no item with that identifier was published. Alternatively, z could report a modified copy of the item.

Some previous work has designed lookup protocols that provide some form of security against node corruption. We will survey this work in the next section.

4.2 Related Work

In this section, we survey some of the related work on the design of secure lookup protocols.

4.2.1 Castro et al.'s Work on Secure Routing

A very interesting piece of work on this subject is by *Castro et al.* [CDG+02]. That work shows how to make a DHT, in particular Pastry, resilient to a powerful adversary that can control up to a fraction $f < 1/2$ of the nodes in the system. The security model assumes that:

- there is a centralized trusted authority that issues membership certificates binding the IP address, the identifier (which the authority generates uniformly at random) and the signing public key of every node that wants to enter the system;
- the number of certificates that the adversary can obtain from the authority is bounded;
- all peers know the central authority's public key;
- data items are self-certifying[1], that is, given a data item it is possible to verify

[1] We will provide a more detailed discussion of self-certifying data items in Section 4.3.

whether the data item is original or if it has been altered.

The modified DHT protocol in [CDG+02] uses several techniques. Every node maintains a *constrained routing table*, in addition to the regular routing table. The constrained routing table has the property that, for any entry in the table, there is exactly one node in the system that can occupy the entry. As a contrast, in Pastry, for any entry of the (regular) routing table, there are many possible nodes that can occupy the entry and the protocol chooses the one that appears to have the smallest ping time. It is much harder for the adversary to use malicious protocol messages to "pollute" the constrained routing table with corrupted entries than it is to pollute the regular routing table. The protocol also requires that a new node joining the system contact multiple bootstrap nodes, instead of one, in the hope that at least one of them will not be malicious. The specifics of the protocol guarantee that the new node will construct a correct routing table, in such a favorable event.

The second technique used is the *routing failure test*. Assume that a node u wants to publish or look up an item with ID x. We require that u's request be routed to a node that can determine the set of the replica nodes for x (in Pastry, the r live nodes whose ID is numerically closest to x) and such a set be returned to u in the response. In [CDG+02], a simple computation (the routing failure test) is described that allows u to determine whether the returned replica set is correct. The test is based on the comparison of the density of the node in the replica set and the expected density of nodes in the ID space. The test is only probabilistic and it is subject to both false positives (the routing appears to have failed, but it did not)

and false negatives (the routing succeeded, but the test reports that it failed).

Finally, the third security component is *redundant routing*, which is used by a node u that wants to retrieve the replica set corresponding to an ID x, after the regular routing operation has produced a result that fails the routing failure test. The redundant routing carefully attempts to route the message from u to the replica set through multiple independent paths and then appropriately reconciles possibly incompatible results, returned by each of the paths.

4.2.2 S-Chord

A different approach was taken by *Fiat et al.* [FSY05]. In their work, the authors consider a different model, which they call the *Byzantine Join Model*. In this model it is assumed that, during the period of time where malicious nodes are present in the system: "1) there are always at least z total peers in the network for some integer z; 2) there are never more than $(1/4 - \epsilon)z$ Byzantine peers in the network for a fixed $\epsilon > 0$; and 3) the number of peer insertion and deletion events is no more than z^k for some tunable parameter k". No centralized authority is assumed and the adversary that coordinates the malicious nodes can arbitrarily choose IP addresses of nodes and is computationally unbounded.

Under these assumptions, *Fiat et al.* present *S-Chord*, a variant of Chord [SMK+01] that can withstand the attack with high probability. S-Chord is not much more inefficient than Chord, at least asymptotically speaking: the cost of a lookup or publish operation is $O(\log^2 n)$ messages (with $O(\log n)$ latency) and each node is only re-

quired to store $O(\log^2 n)$ neighbors. As a comparison, Chord achieve $O(\log n)$ for such performance measures, in absence of corrupted nodes, but does not work in the Byzantine Join Model.

The main idea behind S-Chord is the concept of a *swarm*. For any ID x (of an item or of a node), the swarm $S(x)$, is the set of peers whose IDs are located within a clockwise distance of $(C \ln n)/n$ of the point x. S-Chord maintains the invariant that less than one-fourth of the members of *every* swarm is controlled by the adversary. This, essentially, allows each swarm to simulate the behavior of a honest peer, even though some of its members are dishonest.

When a new node p joins the S-Chord ring, it contacts an already-in-the-system node q, which must be honest. Then q alerts its swarm $S(q)$, which runs a multi-party coin-tossing protocol to generate a random ID, that will become p's node identifier. Then $S(q)$ will locate $S(p)$ (the swarm that p will become part of) and introduces p to $S(p)$. The introduction protocol is designed in such a way that no node will be allowed to join a swarm, unless it has been properly introduced by a majority of the members of another swarm.

4.3 Data Corruption vs. Denial of Service

In Section 4.1, we mentioned that attacks on a lookup protocol may have as goals either data corruption or (selective) denial of service. In this section we discuss the solution proposed by Castro et al. ([CDG+02], see Section 4.2.1) of using self-certifying data items. Such solution effectively solves the problem of data

corruption for most applications. This observation means that, in order to obtain a secure lookup protocol, it is sufficient to design one that is resilient to denial of service, and then requiring self-certifying data items.

4.3.1 Self-Certifying Data Items

A data item $(\mathsf{id}, \mathsf{content})$ is self certifying if there exists a known algorithm to verify its authenticity, given only id and $\mathsf{content}$. Two techniques for self-certifying data items are presented in [CMH+02]. The first technique is to use data items in which the identifier is a *content-hash key*, i.e. a hash of the content of the document, obtained through a hash function, which is conjectured to be collision-resistant [Dam87], such as SHA-1 [oSN02]. To verify the authenticity of the data item, it is sufficient to verify that the identifier equals the hash of the content. By definition of collision resistance, it is unfeasible to construct two distinct data items with the same identifier, which both verify. This prevents an attacker from changing the content of a published data item without being detected.

The other technique is to use *signed-subspace keys* as identifiers. Each system user that wants to publish documents with a signed-subspace key, must generate a public-private key pair for a digital signature scheme. The public key will act as a pseudonym under which the user will publish the documents. The user, for each data item of content c she wants to publish, chooses a descriptive string s and sets the identifier $\mathsf{id} = (pk, s)$ to be the concatenation of her public key pk and the string s. The user then signs (pk, s, c) with her private key to produce a signature σ and

sets $\mathsf{content} = (c, \sigma)$. A user that retrieves $(\mathsf{id}, \mathsf{content})$ can verify the correctness of the data item, by parsing id as (pk, s), parsing $\mathsf{content}$ as (c, σ), and then checking that σ is a valid signature on (pk, s, c). The unforgeability of the digital signature guarantees that an attacker cannot forge a valid data item with the same identifier (pk, s) as the data item published by the honest user. It actually provides a stronger property: an item identifier uniquely identifies the user authorized to publish a value under that identifier and the scheme securely enforces this provision.

The two techniques presented here are not the only possible solutions to certify the authenticity of a data item. Depending on the application that employs the lookup substrate, it is possible to design other authentication and access-control mechanisms.

4.4 Towards a More Realistic Attack Model

As discussed in Section 4.2, previously designed lookup protocols that claim security assume that the attacker is subject to certain constraints, namely that the number of nodes controlled by the attacker is a small fraction of the total number of nodes. In this dissertation, we are similarly interested in designing a practical and efficient solution to the lookup problem, in the presence of attackers with limited resources. Indeed, we conjecture that no efficient lookup protocol can be secure against an unrestricted adversary. With this in mind, we will compare different types of restrictions.

The assumption that the number of corrupted nodes is a small fraction of the

total (e.g. [CDG+02, FSY05]) is reasonable in applications for which there exists a central authority that limits node membership (such as the one that is explicitly required by [CDG+02]). However, there are circumstances where assuming that the adversary can only control a small number of nodes in unrealistic. For example, consider an open P2P system running on the Internet, such as the Gnutella network [Com01], where any user can create a node and make it join, without any form of access control. It is relatively easy for anyone to generate a large number of nodes, *even with limited resources*, by simply running a large number of processes on a small number of machines. The problem is that it is not easy for an external observer to tell whether two remote nodes are controlled by the same or by different real-world identities.

If we look at the issue from a system perspective, we may suggest using IP addresses as identities. For example, the distributed algorithm could enforce that no two nodes have the same IP address. We quickly realize that such algorithm would not work. The main problem is that it is relatively easy, in many networks, to simulate a large number of machines with a single machine, by spoofing IP addresses; we expand on this point in Section 4.5.1, for the interested reader. Another issue is that we may want to allow two different users to both run a node from the same machine. Finally, we want to allow users on different machines that access the Internet through the same network address translation (NAT) box.

So, we are faced with the question: what is it that an attacker with limited resources *cannot* do, if she can control a large fraction of the nodes? What restriction is it reasonable to assume on the adversary?

We conjecture that it is hard for a weak attacker to control a *diverse* set of IP addresses. This is due to the fact that it is hard for the attacker to control hosts in many different physical networks, thus the addresses that she owns or can effectively spoof are limited to the IP addresses assigned to a few networks. Pursuant to our conjecture, in Section 4.6 we describe the Autonomous System (AS) attack model. In Chapter 5, we will present a lookup protocol that is secure in the AS Model.

4.5 Autonomous Systems and IP Spoofing

The Internet may be described as an interconnection of Autonomous Systems (ASes), which are independently owned and managed IP networks [PD00, Section 4.3]. Each AS administrator manages a set of IP addresses that she can assign to hosts and routers within her network.

ASes are traditionally divided into *stub*, *transit*, and *backbone*, according to a non-strict hierarchical structure, as shown in Figure 4.2. A stub AS only routes packets originating from or addressed to a node within itself, while transit ASes also carry through traffic. Backbone ASes are a special case of transit ASes and they are so called, because they correspond to the root of the hierarchy, under a simplified tree approximation of the Internet.

In Section 4.6, we apply the notion of ASes in the definition of our new attack model.

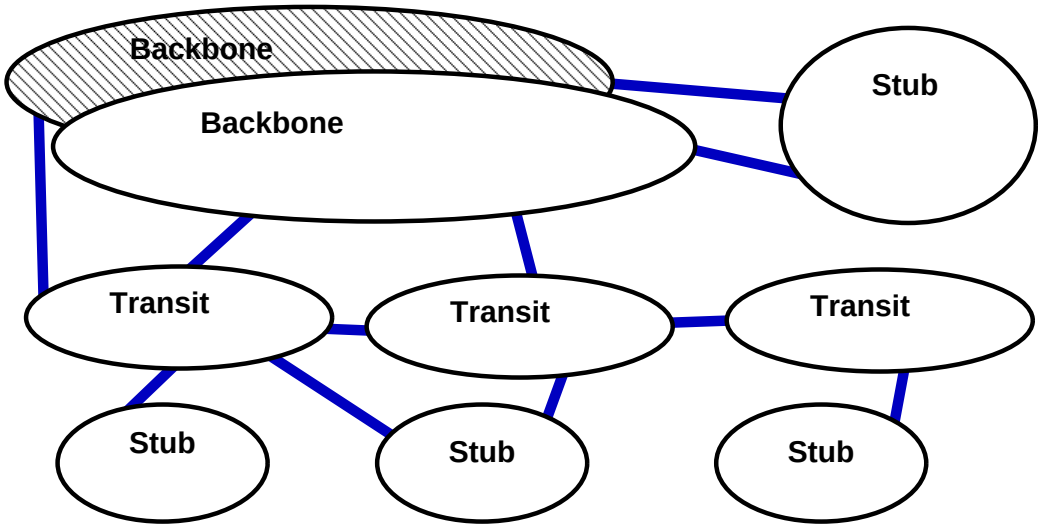


Figure 4.2: Stub, transit and backbone ASes. Stub ASes do not carry through traffic, while transit ASes do. Backbones are a special case of transit ASes.

4.5.1 IP Spoofing

IP spoofing is the practice of an host to send IP packets carrying a source IP address that does not correspond to one of the IP addresses that the administrator assigned to the sender. Spoofing is sometimes used by an attacker to hide her identity or location. It can also be used to simulate multiple identities, in systems that associate identities to IP addresses.

When an attacker uses IP spoofing to create fake identities, she is faced with the problem that a reply message sent to the spoofed address may never reached the attacker. Attacks against certain protocols are not hindered by this problem, since the attacker does not need to receive the reply.

In some networks, the attacker may choose to spoof an IP address a in such a way that it will receive packets addressed to a. For example, this happens if the attacker controls a host with an address a' in a subnet s, the spoofed address

$a \neq a'$ logically belongs to the subnet and the link layer allows the attacker's host to overhear traffic direct to address a. Another example occurs when the attacker acquires control of some routers within the AS responsible for address a and can, therefore, intercept traffic directed to spoofed address a (which may or may not be assigned to a host in the AS).

In some instances, AS administrators can limit IP spoofing by employing appropriate measures, such that *egress filtering*: instructing routers that handle packets leaving a region of the AS to discard packets with a source address that cannot correspond to any legitimate host within the region. However, egress filtering is not widely used and may be challenging to implement in some networks.

4.6 The AS Model

In this section, we present an attack model, which we name the *AS Model* that we believe realistically describes an attacker with limited resources. Informally, in the AS Model we assume that the adversary can only control an arbitrary number of nodes, but all of those corrupted nodes are located in a few ASes. This assumption is shown in Figure 4.3.

We simplistically portray the Internet as a collection of stub ASes connected to an Internet *core* or backbone. We assume the existence of an adversary, which may control nodes in up to f ASes, which we call *corrupt ASes*; f is a parameter of the model. This assumption models our intuition that it is hard for an entity with limited resources to obtain access to too many independently-managed net-

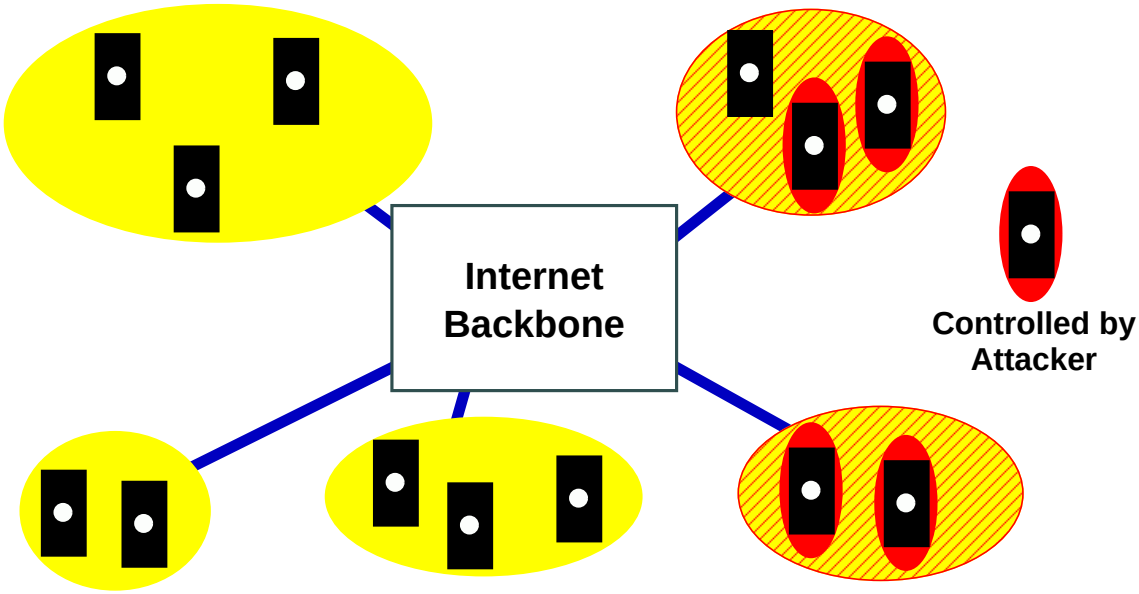


Figure 4.3: Simplified representation of the Internet assumed in the AS Model. The attacker may control hosts in at most f stub ASes ($f = 2$ in the example), but cannot affect routing through the core of the Internet.

works; we expand on this justification in Section 4.7. Within each corrupt AS, we conservatively assume that the adversary may create as many corrupt nodes as she wants, which she fully controls. We assume that the attacker cannot affect routing of messages through the core of the Internet.

Another important assumption that we make is that the mapping between IP addresses and ASes is public. This means that a node can easily determine the AS number of any other node, given the other node's IP address.

In order to make our model concrete, we also assume that the system is synchronous, that delivery of messages is guaranteed, although one could consider a more general version of the AS Model that does not. Similarly to the work by Castro et al. [CDG+02], we also assume the existence of a public-key infrastructure; we justify this decision in Section 4.7. We chose this particular set of assumptions,

because we believe that they can be implemented in practice for some applications and because we were able to derive a positive result (which we describe in the next chapter) under these conditions.

We are now ready to provide a detailed description of the AS Model.

1. The network address space is partitioned into sets, called ASes.

2. There exists a known efficient algorithm $\mathsf{AS}()$ that, on input a network address a, outputs the corresponding AS number $\mathsf{AS}(a)$.

3. We assume the existence of an adversary that can generate an arbitrary number of corrupted nodes and assign them a network address. Network addresses of corrupted nodes must belong to one of f ASes (the corrupt ASes).

4. We assume the existence of a global clock and that each node has access to this global clock.[2]

5. We assume that every message that is sent will be delivered to the destination. This means that we ignore packet loss. We also ignore packet duplication.

6. We assume that there is a bound δ on the delivery time of messages.

7. Honest nodes (i.e. nodes that are not corrupted nodes) may join the system at any time. Additionally, for any time interval T_0, a honest node has a probability p_0 of crashing. Node crashes are independent.

[2]This assumption may be relaxed by assuming that all nodes have local clocks with bounded drift. However, in this work, we make the assumption of a global clock for simplicity of presentation.

8. Each node in the system generates a public-private key pair for a signature scheme.

9. There exists a public-key infrastructure, i.e. there exists a public trusted service that allows a node to obtain the public key of another node, given the other node's network address.

1 and 2 summarize the portion of the assumptions related to the subdivision in ASes that is really necessary for the design of our secure lookup protocol (Chapter 5). Assumptions 4, 5, 6 ensure a synchronous model with guaranteed message delivery, while Assumption 7 provides for crash failures of correct nodes. Finally, the last two assumptions are related to the public-key infrastructure.

In the next section, we will discuss how realistic and interesting the AS Model is. In the next chapter, we will design a lookup protocol that is secure in this model.

4.7 Discussion on the AS Model

. We argue that the formulation of the AS Model is an interesting step forward towards establishing reasonable assumptions on what a limited attacker may do in a practical Internet application. Although we acknowledge that our model makes some simplifying assumptions, which are not fully realistic, we remark that all previous work has been based on a much less realistic assumption that the fraction of adversarial nodes may be bounded by a small constant. The AS Model is therefore a reasonable middle ground between assuming an authority limiting the number of identities that the attacker can assume and assuming an unrestricted adversary, for

which no efficient lookup protocol can be likely designed.

With this in mind, we expand on the limitations of the AS Model. First, the public mapping assumption is not completely realistic. It is now known how to map an IP address to an AS number, in general. However, we note that the IP address to AS mapping needs not to be perfect, but it can be a conservative approximation. A conservative approximation may report two addresses that belong to different real world ASes as belonging to the same AS, but never the vice versa. Note that an attacker that may corrupt up to f ASes in the real world would also corrupt up to f ASes in the conservative approximation. Obviously, the approximation should not be as conservative as to reduce the number of distinct ASes too much (this will be clearer when we present the SDHT protocol in Chapter 5).

The second limitation of the AS Model is that it does not defend against large scale compromises of hosts across many ASes, as it can be achieved through the employment of worms and viruses. In particular, an attacker that acquires control of a *botnet* [McC03], a large army of compromised hosts that is configured for remote control by the compromiser, would control hosts on a very large number of domains and would thus be more powerful than what the security model allows. This fact can be a significant shortcoming, especially taking into account that botnets are illegally rented by compromisers to malicious users that are willing to pay for them. However, the AS Model remains interesting, because it provides defense against an attacker that does not have the connections or the money to acquire a botnet. It also defends against attackers that do not want to expose themselves to the legal liability associated with the use of compromised machines. On the flipside, note that

an attack carried out by a botnet with tens of thousands of hosts from hundreds of domains is extremely debilitating. It is unlikely that any system with open membership can be both reasonably efficient and resilient against such an attack.

The third limitation of the AS Model is our assumption of a public-key infrastructure (Assumption 9 in Section 4.6). Such assumption requires, in practice, an (offline) trusted authority that certifies public keys. Nodes that join the system need to register their keys with the certification authority. At first glance, it could appear that this requirement defeats the purpose of the AS Model, which is to avoid a central membership authority in the first place! However, a public-key certification authority is a much weaker assumption than a membership authority, such as the one required by Castro et al. [CDG+02] or implicitly assumed by other work that relies on the fraction of corrupted nodes to be small (e.g. [FSY05]). This is because the only thing that the former type of authority has to do is to certify a mapping between public keys and network addresses and does not need any knowledge of the real-world identities behind such addresses. A certification authority does not stop a single user from registering keys for an overwhelming number of network addresses. Instead, a real membership authority has to be able to limit the number of nodes that an attacker controls, which requires either an out-of-band verification of the real-world identity of a node's administrator or some payment mechanism; see [CDG+02] for a discussion.

All in all, we believe that the AS Model is interesting on its own and is the necessary first step towards considering models that are neither too unrealistic, by neglecting most malicious behavior, nor too harsh for the designer, by assuming

that attackers are all powerful and therefore ruling out the feasibility of efficient and secure peer-to-peer applications.

Chapter 5

Secure Distributed Hash Table

In this chapter,[1] we present the design of Secure Distributed Hash Table (SDHT), a lookup protocol designed to be secure in the AS Model.

In Chapter 4, we motivated the need for lookup protocols that are secure, i.e. operate correctly in the presence of an attacker that may control a fraction of the nodes. We also introduced the AS Model, which defines a class of attackers with limited resources but that can still control a large fraction of the nodes in the system. We remind the reader that the AS Model is defined by the set of assumptions listed in Section 4.6. As part of our dissertation, we developed SDHT, a protocol that we believe to be secure against such class of attackers.

We provide an informal analysis of SDHT in which we show that the protocol can still provide service even in the presence of a computationally-bounded attacker. Although informal, our analysis is very thorough; the informal arguments that we provide for our claims fall short to be actual proofs only because they do not handle a long list of easy and boring details. We also describe a prototype implementation with which we show that the protocol is practical, although with high overhead, for a small number of nodes.

The rest of this chapter is structured as follows. In Section 5.1, we present

[1] This is joint work with Bobby Bhattacharjee, Jonathan Katz, Neil Spring and Rob Sherwood of the Department of Computer Science, University of Maryland. See also Footnote 6.

the basic ideas behind our algorithm. In Section 5.2, we review Byzantine agreement algorithms, which is a primitive that we use as part of the design of SDHT. In Section 5.3, we present the SDHT algorithm in the details. In Section 5.4, we present an informal argument demonstrating that SDHT is actually secure. Finally, in Section 5.5, we present our implementation of the SDHT protocol and the corresponding experimental results.

5.1 Main Protocol Ideas

Although developed independently, the basic ideas behind the protocol are similar to those of S-Chord [FSY05], in that our protocol is based on Chord [SMK+01] and nodes in the system are partitioned into *neighborhoods* (which corresponds, roughly, to swarms in S-Chord). Each neighborhood is a set of nodes whose identifiers belong to an interval of the Chord ring.

The nodes in a neighborhood operate collectively to act like a single correct Chord node, *even if some of the members are under control of the attacker.* In order to enforce this condition, we need to make sure that at least some nodes in the neighborhood are correct in the first place. We achieve this fact by enforcing *diversity* in the neighborhood: we require that a neighborhood contains nodes from at least $2f+1$ different ASes. Following the notation in Section 4.6, f is a parameter of the protocol that represents an upper bound on the number of ASes where the adversary can control nodes. This is shown in Figure 5.1, while a summary of the parameters of the protocol appears in Table 5.1.

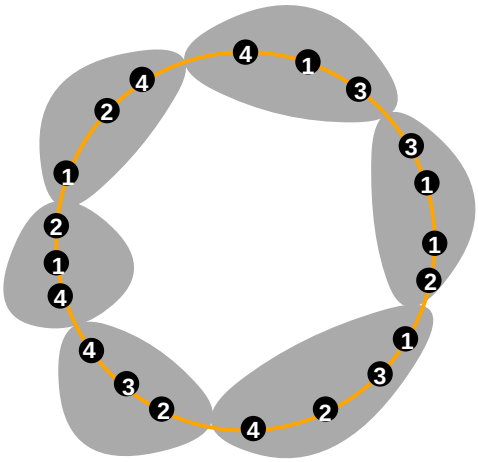


Figure 5.1: Neighborhoods in SDHT. The picture shows the nodes currently in the system as points on the ring-shaped identifier space (the interval $[0, 2^m)$) and their subdivision into neighborhoods (gray areas). The number inside the node indicates its AS. In this example, f is assumed to be equal to 1. Note that each neighborhood contains at least $2f + 1 = 3$ nodes from different ASes.

A node in the protocol knows the composition only of a few of the neighborhoods. Nodes learn dynamically the composition of other neighborhoods by querying other nodes. It is important, therefore, that an attacker cannot lie about the real membership of a neighborhood, without being detected. We achieved this requirement by having nodes in a neighborhood create a *neighborhood certificate*, which lists all the nodes in the neighborhood, plus other protocol information, and is signed by enough nodes to guarantee its validity. In particular, a certificate is valid if it is signed by at least $f + 1$ nodes from different ASes (this ensures that at least one of the signer is not corrupt). This is shown, for the case $f = 1$, in Figure 5.2.

Neighborhoods naturally define a partition of the Chord ring into identifier intervals. A certificate for a neighborhood N also carries the corresponding interval

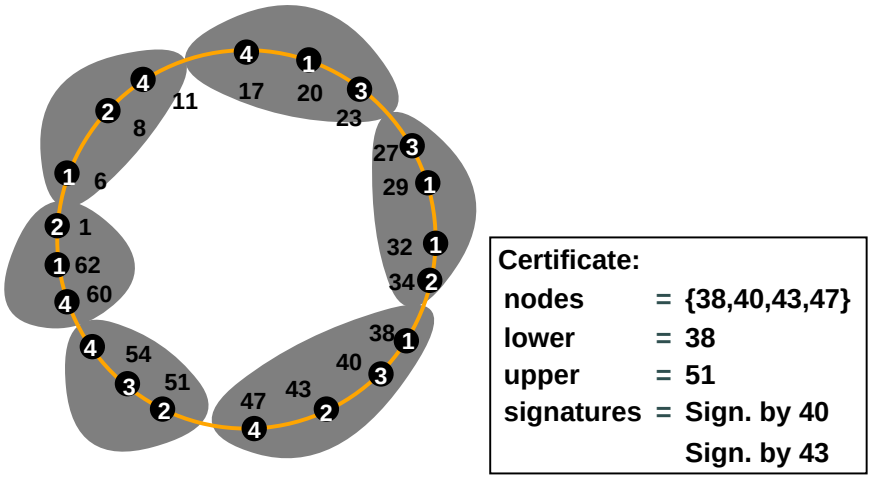
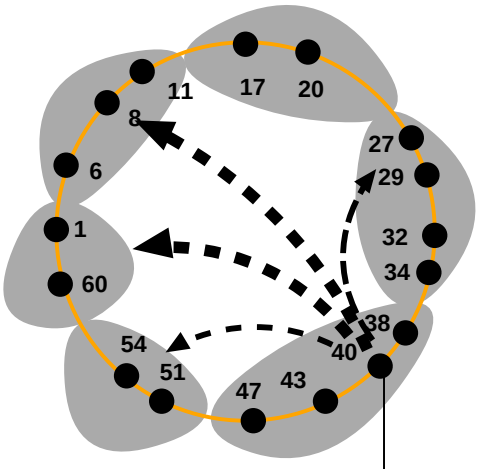


Figure 5.2: The figure shows a possible neighborhood certificate for the neighborhood $\{38, 40, 43, 47\}$, with $f = 1$. The notation is similar to Figure 5.1, except nodes are now tagged with their Chord identifier (in black, next to the node). Note that the certificate contains signatures by $f + 1 = 2$ nodes from different ASes.

$[N.l, N.u)$. All the nodes in a neighborhood must have identifiers that fall in that interval.

In addition to a certificate for its own neighborhood, a node stores a successor pointer, a predecessor pointer and a finger table. These are similar to Chord, except that each of these pointers is a neighborhood certificate, instead of the address of a single node. In particular, for a node u with identifier x in neighborhood N, u's successor pointer is a certificate for the neighborhood N_s such that $N_s.l = N.u$. Similarly, u's predecessor pointer is a certificate for N_p such that $N_p.u = N.l$. The finger table of u contains entries $\mathsf{finger}[1], \ldots, \mathsf{finger}[d]$ where the i-th finger $\mathsf{finger}[i]$ contains a certificate for the neighborhood encompassing $x + 2^{m-i}$, where m is the length of the node identifier in bits. (A neighborhood N is said to encompass id if $N.l \leq \mathsf{id} < N.u$.) This is shown in Figure 5.3.

In Section 5.1.1, we explain the basics of the lookup and publish operation; in



Self:	**{38,40,43,47}**
Predecessor:	**{27,29,32,34}**
Successor:	**{51,54}**
Finger[1]:	**{6,8,11}**
Finger[2]:	**{60,1}**

Figure 5.3: State stored at node with ID 40 includes successor pointer, predecessor pointer and finger entries.

Section 5.1.2, we cover the evolution of the neighborhoods over time.

5.1.1 Performing Lookup

Let's see how neighborhoods and neighborhood certificates allow the performance of secure lookup. Similarly to S-Chord, SDHT stores a copy of a published data item (id, content) at each of the nodes in the neighborhood encompassing id.

Both a publish and lookup operations for an item with identifier id involve obtaining a certificate of the neighborhood encompassing id. Publishing is then achieved by sending a copy of the item to each member of the neighborhood, while lookup is achieved by querying each member. Since the authenticity of the neighborhood certificate can be verified and since a neighborhood is guaranteed to have some correct node, this procedure will guarantee that published items will be correctly retrieved. Therefore the design of both the lookup and publish operations reduces to the design of a NEIGHBORHOODLOOKUP(id) operation that returns the certificate of the neighborhood encompassing id.

The NEIGHBORHOODLOOKUP operation in SDHT is shown in Figure 5.4. Let u be a node performing a NEIGHBORHOODLOOKUP for identifier x. Node u uses the certificates it possesses (as successor, predecessor and fingers) to choose the neighborhood N that it knows about that is clockwise-closest to x (Figure 5.4(a)). In the example u chooses the neighborhood $N = \{17, 20\}$. Next, u selects a node v at random in N; we will see in Section 5.3.3 how the random choice is made. Then u sends a lookup query message to v requesting ID x (Figure 5.4(b)). In response v

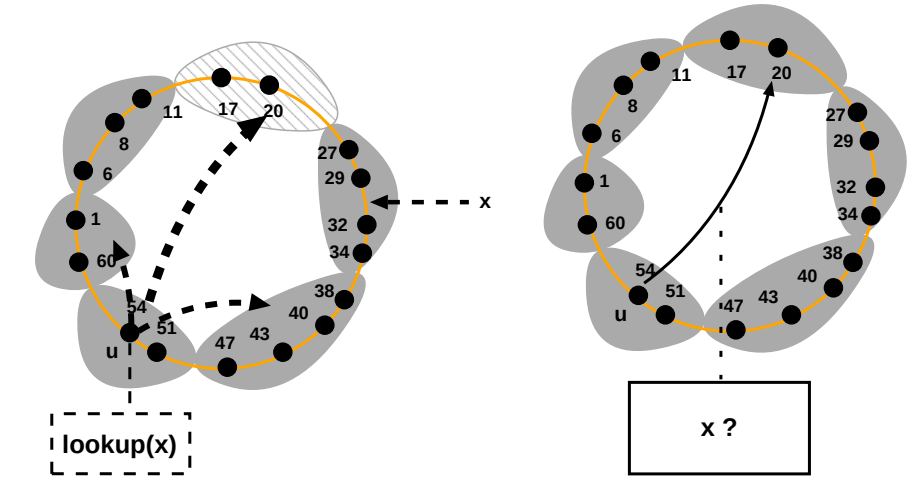
chooses the neighborhood N' it knows about that is closest to x (Figure 5.4(c)) and returns its certificate to u (Figure 5.4(d)). Node u repeats the procedure iteratively until it learns the certificate of the neighborhood encompassing x.

While in Chord [SMK+01] lookups could be performed both recursively and iteratively, in SDHT lookups are iterative only. (In a recursive lookup, v would forward u's request message herself to a random node in N', instead of sending N' back to u.) Iterative lookups require slightly more communications but ensure that the node initiating the operation remains in control of the progress at all time.

5.1.2 Dealing with Change

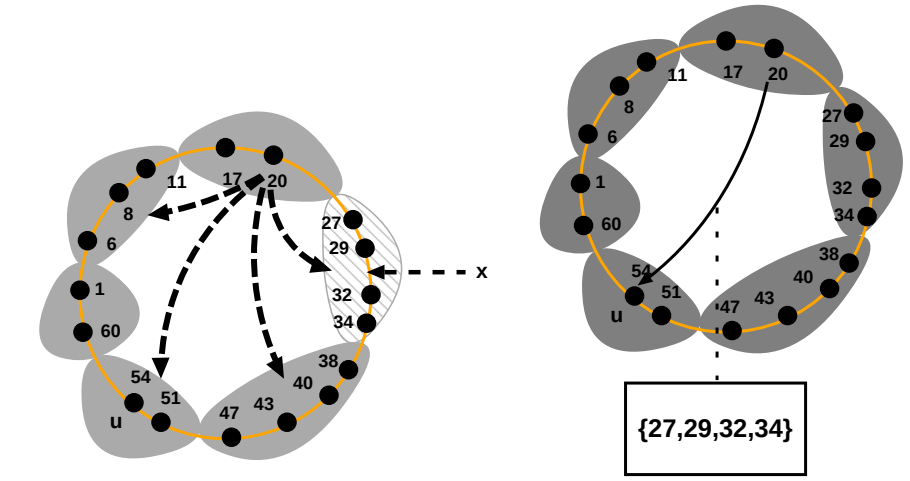
As nodes join and leave the system, neighborhoods change over time, so it is important to make sure that a neighborhood certificate is up to date. This is achieved by incorporating a timestamp in a neighborhood certificate and in verifying that the timestamp is not "too old", when a certificate is used. In order to simplify this process, time will be divided in *epochs*, each consisting of several rounds. The protocol will generate a new neighborhood certificate for each neighborhood and for each epoch.

More specifically, at the beginning of each epoch, the nodes within each neighborhood will execute a sub-protocol called *epoch update*. During an epoch update, the nodes will recompute the membership of their neighborhood, by removing nodes that have crashed and by adding nodes that have joined the system. Then the node will generate a new certificate for the neighborhood.



(a) 54 chooses known neighborhood closest to target.

(b) 54 queries random node (20) in neighborhood.



(c) 20 chooses known neighborhood closest to target.

(d) 20 sends neighborhood certificate to 54

Figure 5.4: Example of NEIGHBORHOODLOOKUP operation performed by node u with identifier 54 for identifier $x = 30$. We only show a simple example, in which u learns the desired neighborhood in one iteration, to avoid cluttering the figure with too many neighborhoods.

A property that we need to provide during an epoch update is *consistency*. All (honest) nodes in a neighborhood should have a uniform view of the neighborhood memberships. In order to achieve this, our protocol will execute a *Byzantine agreement* [PSL80, Fis83] subprotocol at every epoch. Our intuition is that, in our model, Byzantine agreement is indeed necessary to achieve this consistency property (although it might not be necessary to implement lookup). We will review Byzantine agreement in Section 5.2.

SDHT also needs to ensures that there are enough nodes from different ASes in the neighborhood to produce a valid certificate and not too many nodes to make the overhead too large. We achieve this through several mechanisms:

- For each neighborhood, we introduce a quantity called *neighborhood weakness*, which is a real value between 0 and 1 that measures the diversity of the neighborhood. A weakness of 1 or close to 1 means that the diversity is insufficient and that the neighborhood may fail to produce a valid certificate. Occasionally, a neighborhood will fail to produce a valid certificate during an epoch; we will call this undesirable event a *neighborhood failure*. The weakness of the neighborhood upper bounds the probability of such event during an epoch. We will show in Section 5.3.2 how to calculate the neighborhood weakness.

- We allow a neighborhood to split into several neighborhoods, whenever such subdivision generates new neighborhoods that are sufficiently strong (i.e. the weakness of the new neighborhoods is smaller than a threshold).

- We establish a bound a on the number of nodes from the same AS that may

join the same neighborhood. This bound, coupled with the ability of a neighborhood to split, effectively limits the size of a neighborhood.

- We allow adjacent neighborhoods to merge if some of them are too weak. This prevent neighborhoods for growing too small or not sufficiently diverse.

There are important consequences of our design decisions that are important to stress.

- A node that requests to join has to wait until the following epoch update before joining. Therefore the epoch length determines a tradeoff: a shorter epoch length allows for shorter waiting time of joining nodes at the expense of higher overhead, due to the more frequent invocations of the epoch update sub-protocol.

- Occasionally nodes that attempt to join the SDHT instance will be rejected and will not be able to join. This occurs if the neighborhood that the node is supposed to join already contains at least a nodes from the same AS as the joining nodes. The node will be allowed to join again later, if the size and the diversity of the system increase enough so that the assigned neighborhood will have room for the joining node. We believe that this limitation of a protocol in the AS Model is inevitable, because the system needs to maintain diversity. A higher value of a reduces the likelihood a node is rejected but also increases the overhead caused by large neighborhoods.

In the following section, we present more details on the protocol.

5.2 Byzantine Agreement

A primitive that we employ in our design of SDHT is *Byzantine agreement* [PSL80, Fis83] and the related notion of *broadcast*[2]. A broadcast protocol is a distributed algorithm that allows a distinguished node (the *sender*) to send its input x to a set of other nodes, even when a fraction of the nodes may arbitrarily misbehave. The critical property that broadcast provides is that all correct nodes agree on the value of x at the end of the execution, even if the sender is corrupt. In a Byzantine agreement protocol, as defined by Pease, Shostak and Lamport [PSL80, LSP82], each node broadcasts its own individual input to every other node, so that, at the end, each correct node will have a consistent view of the inputs of everyone else. Note that the two notions are equivalent: given a broadcast algorithm, a Byzantine agreement is immediately constructed by running in parallel one broadcast for each node; similarly, broadcast can be achieved by running Byzantine agreement and giving a dummy input to all nodes other than the sender.[3]

In particular, we say that a distributed algorithm Π is a (t, n)-*broadcast protocol*, if it satisfies the following properties, when executed by a set of n nodes, in which there is a distinguished *sender* node and of which at most t are corrupted

[2]Fischer [Fis83] calls broadcast the *generals problem*.

[3]Some authors use the term Byzantine agreement to refer to *consensus*, which is a slightly different notion. In consensus, correct nodes must also agree on an output, but with the requirement that, if all correct nodes' inputs are the same, then they have to output that value. Consensus only makes sense if a majority of the nodes is correct, in which case it is equivalent to broadcast and Byzantine agreement.

nodes controlled by an adversary and may behave arbitrarily:

- **Termination**: all nodes eventually terminate.
- **Agreement**: all correct nodes have the same output.
- **Validity**: if the sender is correct, then all the correct nodes output the sender's input.

Some of these requirements may be relaxed to hold with high probability. We refer the reader to [Lyn96, Section 6.3] for a more formal definition.

There are several known positive and negative results on the implementation of broadcast, depending on the bound t on the number of corrupted nodes, on whether the system is assumed to be synchronous or asynchronous, on whether the algorithm must be deterministic or may be randomized. Additionally, the results vary widely depending on whether messages are *authenticated*. Informally, we say that the system provides authenticated messages if, for any nodes u, v, w and for any message m sent from u to v, v may convincingly prove to w that m was sent by u; w also acquires the ability to prove the authenticity of m to a third party. Authenticated messages can be implemented in practice with digital signatures and a public-key infrastructure, as follows.

1. Each node generates beforehand a public-private key pair for a signature scheme.
2. Each node acquires the public key of every other nodes through a trusted mechanism (the public key infrastructure). For example, there could be a

trusted authority, which always acts correctly, that acquires all public keys and then provides them to all nodes.

3. Each node signs every message it generates and appends the signature to the message.

4. Each node discards any incoming message with an invalid signature with respect to the public key of the sender.

5. Any node may verify the authenticity of a message m, which claims to have been generated by another node u, by verifying that it contains a valid signature with respect to the public key of u.

Under the assumptions that the signature scheme is secure and that corrupted nodes are computationally bounded, the mechanism provides authenticated messages. In particular, it is infeasible for an adversary to deceive a correct node into believing that a certain message was sent by another correct node, when it was not.

In the following sections, we will summarize the known results for Byzantine agreement (which imply a corresponding result for broadcast).

5.2.1 Known Results for Non-Authenticated Byzantine Agreement

If we do not assume authenticated messages, Pease, Shostak and Lamport (PSL) [PSL80] have shown that no Byzantine agreement protocol exist for $t \geq n/3$. The authors also show how to construct a deterministic Byzantine agreement protocol for $t < n/3$, assuming a synchronous system.

For an asynchronous system, Fischer et al. [FLP85] proved that no deterministic algorithm solves Byzantine agreement, except in the trivial case $t = 0$. However, it is possible to solve the problem with a randomized algorithm, for any $t < n/3$ [Bra84].

5.2.2 Known Results for Authenticated Byzantine Agreement

Assuming authenticated messages, Pease, Shostak and Lamport (PSL) constructed a deterministic synchronous algorithm that provides Byzantine agreement for any value of t, n [PSL80]. The algorithm requires $t+1$ rounds of communication, which is optimal for deterministic algorithms [Lyn96].

Without the synchronous assumption, Toueg proved that no Byzantine agreement algorithm exists for $t \geq n/3$, even if authenticated messages are assumed and the algorithm can be randomized [Tou84]. Obviously Bracha's protocol [Bra84] as well as subsequent improved algorithms for the non-authenticated case can be applied to the authenticated case as well.

We review the PSL Byzantine agreement algorithm in Figure 5.5, with an optimization described by Lynch [Lyn96, *OptEIGStop*, Sections 6.2.3 and 6.2.4]. This is the algorithm that we employ in SDHT. For simplicity of exposition, the figure actually shows the corresponding broadcast protocol pslBroadcast and, for concreteness, it also shows how authenticated messages are implemented by using digital signatures. The algorithm can be described in the following manner. In the first round, the sender sends its input to all other nodes. In each following

round, when a node v receives a message, it verifies that the message is of the form $< x, u_1, \sigma_1, u_2, \sigma_2, \ldots, u_{r-1}, \sigma_{r-1} >$, meaning "$u_{r-1}$ says that u_{r-2} said that $\ldots u_2$ said that the sender u_1 said that its input is x". If the message is in such form and passes a series of validity tests (Figure 5.5), the node accepts x as a value broadcast by the sender. Node v then informs the other nodes, by sending $< x, u_1, \sigma_1, u_2, \sigma_2, \ldots, u_{r-1}, \sigma_{r-1}, u_r = v, \sigma_r >$ to everyone, where σ_r is v's signature on the rest of the message. At the end of the execution, if (as expected) the node has accepted only one value, the node will output that value. Otherwise, the node assumes that the sender is faulty and outputs the default value $\perp$ instead.

It is easy to estimate the cost of the PSL algorithm, in terms of the number of messages sent, by simple inspection. Each node will send a message either *once* or *twice* to every other node, therefore the total number of messages sent is at least n^2 and at most $2n^2$. The size of the messages is dominated by the digital signatures, which are usually fairly large (e.g. 128 bytes each for 1024bit RSA signatures). However, note that it is possible to use known techniques, such as aggregate signatures [LMRS04, BGLS03], to reduce the total size of the sequence $(\sigma_1, u_2, \sigma_2, \ldots, u_{r-1}, \sigma_r)$ to that of a single signature.

5.2.3 Byzantine Agreement in SDHT

In SDHT, we employ a Byzantine agreement algorithm to ensure that all the nodes in a neighborhood have the same view of the neighborhood membership. We chose the PSL algorithm for this purpose, because in our model (Section 4.6), we can

```
pslBroadcast(nodes, sender, t, input)
  if this = sender
    then σ_1 ← sign(< input, sender >, this.secretkey)
         for each v in set
             send(v, < input, sender, σ_1 >)
  values ← {}
  for r ← 2 to t + 1
      waitFor( beginning of round r )
      for each message m received during round r − 1
          if m of type < x, u_1, σ_1, u_2, σ_2, ..., u_{r−1}, σ_{r−1} >
             and u_1 = sender and validSignatures(m)
             and u_1, ..., u_{r−1} are all distinct
             and x ∉ values and |values| < 2
            then values ← values ∪ {x}
                 σ_r ← sign(< x, u_1, σ_1, u_2, σ_2, ..., u_{r−1}, σ_{r−1}, u_r >, this.secretkey)
                 for each v in set
                     send(v, < x, u_1, σ_1, u_2, σ_2, ..., u_{r−1}, σ_{r−1}, u_r, σ_r >)
  if |values| = 1
    then output ← x : values = {x}
    else output ← ⊥
  return output

validSignatures(< x, u_1, σ_1, u_2, σ_2, ..., u_{r−1}, σ_{r−1} >)
  allvalid ← true
  for i ← 1 to r − 1
      valid ← verifySignature(m, < x, u_1, σ_1, ..., u_{i−1}, σ_{i−1}, u_i >, publicKey(u_i))
      if not valid
        then allvalid ← false
  return allvalid
```

Figure 5.5: PSL algorithm (pslBroadcast) for broadcast in a synchronous system with digital signatures. Each node receives as input the set of nodes executing the algorithm (`nodes`), the bound t on the number of corrupted nodes and the identity of the sender (`sender`). The sender node also receives the value to be broadcast via the argument `input` (this is ignored by non-sender nodes). At the end, pslBroadcast returns the broadcast value, as perceived by the node; this may be the default value $\perp$. Implicit parameter of the procedure is **this**, the node itself.

assume synchronous communication and a public-key infrastructure. Additionally, the PSL algorithm not only is correct under these assumptions, but it is relatively easy to implement in practice.

Note that, in SDHT, all nodes in a neighborhood may potentially exhibit a form of failure, because any correct node may crash. Even if we assumed that correct nodes cannot crash, the number of misbehaving nodes alone could be large. Therefore it is impossible to apply our solution if we relax either the synchronous or public-key infrastructure restrictions, because $t \geq n/3$.[4]

In SDHT, we will actually use the procedure BYZANTINE, which is the Byzantine agreement version of the PSL algorithm. BYZANTINE is implemented by running in parallel $|\texttt{nodes}|$ separate instances of PSLBROADCAST. The number of messages sent by the BYZANTINE procedure is thus $\Theta(|\texttt{nodes}|^3)$.

5.3 SDHT Protocol Details

In this section, we present the SDHT protocol in the details.

[4]One could modify the protocol to ensure that misbehaving nodes account for less than one third of the nodes in the neighborhood, by requiring that the neighborhood contains at least $(2a+1)f+1$ ASes, where a is the maximum number of nodes allowed from the same AS (Table 5.1, Section 5.1.2). Such change would allow the use of other Byzantine protocols that require fewer assumptions, but at the price of a much higher diversity requirement and a much higher neighborhood size.

Table 5.1: Parameters of SDHT protocol.

Symbol	Explanation
a	Tentative maximum number of nodes from the same AS in the same neighborhood.
E	Length of an epoch.
f	Maximum number of ASes in which the adversary controls nodes.
m	Length in bits of the Chord identifier.
p_0	Maximum probability of failure of an honest node during a period T_0.
T_0	See p_0.
p	Maximum probability of failure of an honest node during a two epoch period.
`SDHT_WEAK`	High threshold for neighborhood weakness.
`SDHT_STRONG`	Low threshold for neighborhood weakness.

Table 5.2: List of other symbols.

Symbol	Explanation
α	An AS number.
$d(N, x)$	Distance between neighborhood N and ID x.
e	An epoch.
n	Total number of nodes currently in the system.
N	A neighborhood.
x	A point in the identifier space.
`weak`	High threshold for neighborhood weakness for this neighborhood.
`strong`	Low threshold for neighborhood weakness for this neighborhood.

5.3.1 Initialization

SDHT requires the system to be in a valid state at all times. This means that the protocol execution must begin with the presence of (at least) one neighborhood. Therefore, in order to start an instance of SDHT, it is necessary to identify and configure a distinguished set of *initial nodes*. The initial nodes will then execute a component of the SDHT protocol called *initialization* that will result in the generation of a protocol instance consisting of one neighborhood. The neighborhood will consist of the initial nodes or, if failures occur, a subset of the initial nodes.

The initialization phase requires some manual setup and coordination. However, this operation only needs to be executed once. After that, the system becomes self-sustaining and further addition or removal of nodes from the protocol instance is dealt automatically by the protocol in a decentralized manner.

It is a requirement that the set of initial nodes be sufficiently diverse. At a minimum, it should contain $2f + 1$ nodes from different ASes. Better yet the *weakness* of the initial set should be low. We have already introduced the notion of weakness and we will discuss it more in detail in Section 5.3.2.

Before initialization begins, each initial node is given as input the list of all the initial nodes. The initialization phase begins with the execution of a Byzantine agreement sub-protocol (Section 5.2). This step ensures that all nodes output the same set of initial nodes and also removes from the list initial nodes that may have crashed. Next, each node constructs a certificate describing the only neighborhood that has just been created and sends a signature on that certificate to all other

```
INITIALIZE(initialnodes, starttime)
    set ← SETBYZANTINE(initialnodes, ∅, starttime)
    certtext ← NEWCERTTEXT(set, [0, 1), currentepoch)
    certsig ← SIGN(certtext, this.secretkey)
    for each v in set
        SEND(v, < CERT, certtext, certsig >)
    WAITFOR( f + 1 valid CERT messages with signatures certsig[1], ..., certsig[f + 1] )
    this.neighborhood ← NEWCERTIFICATE(certtext, certsig[1], ..., certsig[f + 1])

SETBYZANTINE(nodes, joining, starttime)
    input ← {this} ∪ joining
    output ← BYZANTINE(nodes, input, starttime)
    return ⋃_{i∈nodes} output[i]
```

Figure 5.6: Initialization in SDHT. The procedure SETBYZANTINE is also used in other parts of the protocol. The procedure BYZANTINE is discussed in Section 5.2.

nodes. The final result is that each node will be able to output a certificate that contains at least $f + 1$ signatures from different ASes as required. The pseudocode for the initialization is shown in Figure 5.6, procedure INITIALIZE. In an attempt to improve clarity, we use an object-oriented notation: we assume that the node that is invoking the algorithm is an implicit parameter and is accessible as **this**.

5.3.2 Neighborhood Weakness

In Section 5.1.2, we introduced the notion of weakness of a neighborhood. In this section, we explain this concept in more detail.

In order for a neighborhood to function correctly, it is necessary that it can generate a neighborhood certificate in every epoch. Such certificate generation occurs in the epoch update (Section 5.3.5). Recall that a neighborhood certificate is valid if it contains $f + 1$ signatures from nodes of different ASes. The neighbor-

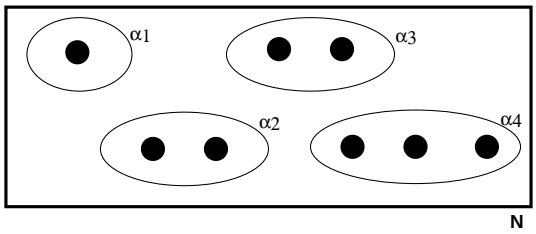
hood weakness is an upper bound to the probability that a neighborhood fails to produce a valid certificate during an epoch (in Section 5.1.2, we called this event a *neighborhood failure*).

In order to estimate this upper bound, we need to realize what can go wrong. Recall that in our model (Section 4.6), an attacker may control up to f ASes. In particular, nodes controlled by the attacker may refuse to cooperate in the certificate generation. Therefore, if a neighborhood contains nodes from less than $2f+1$ ASes, we define the weakness to be 1, because in the worst case the nodes from f of the ASes will not produce any signature.

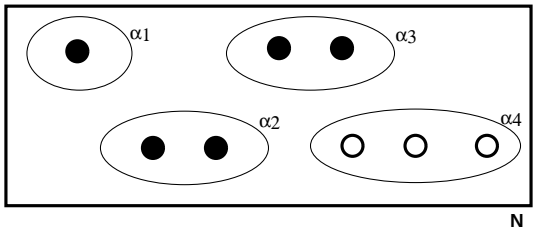
If the number of ASes represented in a neighborhood is at least $2f+1$, that would be sufficient to guarantee successful certificate generation, if there were no honest node failures. Recall that, in our model, we assume that for every time period of duration T_0 and for every honest node u there is a probability p_0 that u will crash.

We introduce two new important parameters: E and p. E represents the duration of an epoch, while p is the probability that a node will crash during a two-epoch period. We can use the union bound to estimate $p = p_0 \cdot \lceil 2E/T_0 \rceil$. Under this model which involves both stopping and Byzantine failures, we can compute the weakness as follows. Assume the set of nodes in a neighborhood N at a point in time during an epoch e is S. At some point during epoch $e+1$, the neighborhood will need to generate a new certificate, as it will be made clearer in Section 5.3.5.

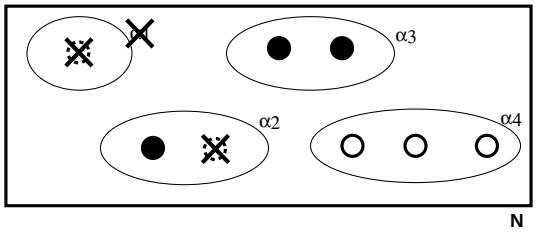
Let $\alpha_1, \ldots, \alpha_q$ be the set of ASes represented in S and, for $i = 1, \ldots, q$, let c_i be the number of nodes in S that belong to AS α_i. Without loss of generality, $c_i \leq c_{i+1}$;



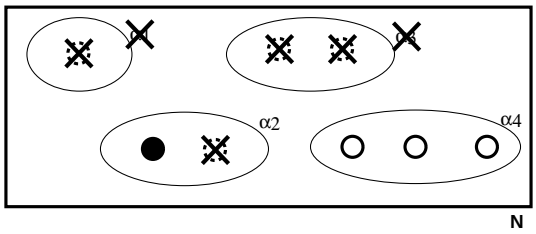
(a) Nodes in neighborhood grouped by AS



(b) Assume largest f ASes are corrupt



(c) Good outcome: at least $f + 1$ correct ASes survive



(d) Bad outcome: less than $f + 1$ correct ASes survive

Figure 5.7: Example of definition of weakness with $f = 1$ for a neighborhood with $q = 4$ ASes, each containing $c_1 = 1$, $c_2 = 2$, $c_3 = 2$ and $c_4 = 3$ nodes respectively. A full circle represents a correct alive node, an empty circle represents a node controlled by the attacker and a crossed out circle represents a crashed correct node. Figure 5.7(a) shows the nodes in the neighborhood N, grouped by AS. First, we conservatively assume that the largest AS (α_4) is controlled by the attacker (Figure 5.7(b)). We define weakness as the probability that less than $f + 1 = 2$ ASes remain with a correct node, when each correct node crashes with independent probability p. Figure 5.7(c) represents an example of "good" outcome, while Figure 5.7(d) shows a "bad" outcome.

Figure 5.8: Neighborhood weakness.

```
WEAKNESS(nodeset)
    AS_set ← {AS(u) : u ∈ nodeset}
    i ← 1
    for each α in AS_set
        ascount[i] ← | {u ∈ nodeset : AS(u) = α} |
        i ← i + 1
    q ← |AS_set|
    SORT(ascount)
    μ ← Σ_{i=1}^{q−f} p^{ascount[i]}
    δ ← (q − (2f + 1))/q − 1
    return F+(μ, δ)
```

Figure 5.9: Computing the weakness of a set of nodes. F^+ is defined in [MR97, Section 4.1].

see Figure 5.7(a) for an example. We conservatively assume that the f ASes with the largest representation in S (namely $\alpha_{q-f+1}, \ldots, \alpha_q$) are completely under control of the adversary, while the remaining ASes contain nodes that behave correctly, but may randomly crash. Let S' denote the random variable representing the subset of nodes in S, belonging to one of $\alpha_1, \ldots, \alpha_{q-f}$, which did not crash during the epochs e or $e+1$. A certificate will be generated if S' contains representatives from at least $f+1$ ASes.

Therefore, we define:

$$\mathsf{weakness}(N) = \Pr_{S'}[S' \text{ contains representatives from at least } f+1 \text{ ASes}]$$

The neighborhood weakness can be estimated using the Chernoff's bound [MR97, Section 4.1]. The pseudocode is given in Figure 5.9.

We introduce two threshold values for the weakness denoted by `weak` and `strong`, where `weak` > `strong`. A neighborhood with a weakness higher than `weak`

will attempt to merge with nearby neighborhoods, to avoid the risk of a neighborhood failure. Vice versa, a neighborhood will split into two or more neighborhoods if each of the resulting neighborhoods have weakness smaller than `strong`. This two-threshold approach prevents neighborhoods from oscillating between merging and splitting.

We choose to give each neighborhood its own value of `weak`, in order to ensure that the probability that a neighborhood failure occurs in *some* neighborhood is at most `SDHT_WEAK`, where `SDHT_WEAK` is a global parameter. We achieve this by employing the following algorithm. During initialization, when the system consists of only one neighborhood, the property `weak` is set to `SDHT_WEAK`. Every time a neighborhood N splits into sub-neighborhoods $N_1, \ldots, N_q$, each N_i sets its value of `weak` to w/q, where w is the value of the property `weak` of the "parent" neighborhood N. Every time neighborhoods $N_1, \ldots, N_q$ merge into a neighborhood N, the resulting neighborhood sets its `weak` property to the sum of the values in the N_i. This ensures that at all times the sum across all neighborhoods of the `weak` properties is at most `SDHT_WEAK`. We similarly handle the `strong` property of the neighborhoods, by introducing the global parameter `SDHT_STRONG`.

5.3.3 NEIGHBORHOODLOOKUP()

A node invokes the procedure NEIGHBORHOODLOOKUP() (Figure 5.10) to discover the certificate of the neighborhood encompassing a given identifier x. This procedure is used in lookup and publish operations, as well as in other components of

```
NEIGHBORHOODLOOKUP(x)
    found ← false
    nhd ← LOOKUPSTEP(x)
    repeat
            if nhd.l ≤ x < nhd.u
              then found ← true
              else
                    stepdone ← false
                    repeat
                            node ← CHOOSERANDOMNODE(nhd.nodeset)
                            SEND(u, < RLOOKUP_REQUEST, x >)
                            RECEIVE( message m of type < RLOOKUP_REPLY, nhd1 > )
                            if  nhd1 valid and nhd1 closer to x than nhd
                              then stepdone ← true
                                   nhd ← nhd1
                      until stepdone

      until found

LOOKUPSTEP(x)
    S ← {this.nhd, this.pred, this.succ} ∪ {this.fing[i] : i ≥ 1}
    return argmin_{N∈S} d(N, x)

CHOOSERANDOMNODE(nodeset)
    AS_set ← {AS(u) : u ∈ nodeset}
    as ← U(AS_set)
    v ← U({w ∈ nhd : AS(w) = as})
```

Figure 5.10: NEIGHBORHOODLOOKUP() operation in SDHT.

the protocol, such as joining (Section 5.3.4) and finger table updates (Section 5.3.5).

The procedure is simple and has already been described in Section 5.1.1. There are, however, a few details that we should clarify.

The algorithm uses a notion of *counterclockwise distance* between a neighborhood N and an identifier x. This is defined as:

$$d(N, x) = \begin{cases} 0 & \text{if } N.l \leq x < N.u \\ x - N.u & \text{otherwise.} \end{cases}$$

The procedure LOOKUPSTEP returns the known neighborhood that minimizes the counterclockwise distance to the target point x.

One detail we should explain is the subroutine CHOOSERANDOMNODE. Instead of choosing a node uniformly at random in the neighborhood, the procedure chooses an AS uniformly at random (within the set of ASes represented in the neighborhood) and then a node uniformly at random between the nodes in the neighborhood from that AS. This ensure that an honest node will be chosen with probability larger than 1/2, regardless of the distribution of nodes between the ASes.

5.3.4 Joining and Leaving

Like in every other lookup protocol, in SDHT nodes can dynamically join and leave the system.

In general, nodes may leave the system either gracefully, by executing a specific leave operation specified by the protocol, or ungracefully, by crashing. For simplicity of design, SDHT does not prescribe a graceful leave operation, so both types of events are identical: the node simply cease to participate in the protocol with no warning.

```
JOIN(bootstrap)
   SEND(bootstrap, < RLOOKUP_REQUEST, this.id >)
   RECEIVE( message m of type < RLOOKUP_REPLY, cert > )
   VERIFY( cert valid neighborhood certificate )
   VERIFY( nhd encompasses this.id )
   for each v in set
       SEND(v, < JOIN2_REQUEST, this >)
   success ← false
   repeat
           receive( message m of type < JOIN2_REPLY, nhd, pred, succ > )
           if pred, nhd, succ valid consecutive neighborhood certificates
             then if nhd contains this
                       then success ← true
                            this.nhd ← nhd
                            this.pred ← pred
                            this.succ ← succ
     until success
   EPOCHUPDATE2()
```

Figure 5.11: Join operation in SDHT.

Nodes that have left or crashed are removed from their neighborhood by the next invocation of the epoch update sub-protocol Section 5.3.5.

Nodes that decide to join an SDHT instance (other than during initialization) execute the procedure depicted in Figure 5.11. The argument bootstrap must be an honest node which has successfully joined. First, the joining node u asks the bootstrap node for the current neighborhood containing its identifier (**this**.id); let *cert* be such certificate. Next u sends a `JOIN2_REQUEST` to all nodes in the neighborhood defined by *cert*. Finally, u waits to receive a `JOIN2_REPLY` message from any of the nodes in that neighborhood, notifying u that it has successfully joined. The reply carries the new certificates of the neighborhood containing u (`nhd`), its successor (`succ`) and its predecessor (`pred`).

We omitted some details from this description. When a node v receives a

JOIN2_REQUEST message from u, v simply adds u to a list of nodes that have requested a join (joiningNodes), after verifying that u's Chord identifier falls in v's neighborhood. Node v will send u the JOIN2_REPLY during the following epoch update, as we explain in Section 5.3.5.

5.3.5 Epoch Update

The epoch update sub-protocol is invoked periodically to recompute the membership of the neighborhoods. A node invokes the epoch update when the current time is a multiple of the epoch length. We assume that times are synchronized, therefore all nodes in a neighborhood will enter the epoch update phase at the same time. The pseudocode for the epoch update appears in Figure 5.12.

The epoch update phase is divided into several phases:

1. In epochUpdateBA (Figure 5.13), the node discovers the set of nodes previously in its neighborhood are still alive (stayset) and the set of nodes that have requested to join through any of the nodes in the neighborhood (joinset). Each node adjusted the resulting joinset to remove nodes that should not enter the neighborhood as well as nodes that are in excess of the limit a of nodes from the same AS. This phase uses Byzantine agreement as a tool to ensure that all nodes obtain the same values of stayset and joinset. The function also returns the provisional membership newset of the new neighborhood, which is the union of stayset and joinset. The membership is provisional, because it may change during the merging and splitting

phases (below).

2. In PROTOCERTGEN (Figure 5.13), the nodes in a neighborhood create a certificate corresponding to the provisional neighborhood membership `newset`. We call such a certificate *protocertificate* (`proto`) to insist on its provisional nature. Also, for every pair of consecutive neighborhoods N and N' on the Chord ring, N and N' exchange their protocertificates, so that, at the end of PROTOCERTGEN phase, a node will also have a current protocertificate of its `success` or (`protosucc`) and predecessor (`protopred`).

3. In NOTIFYJOINED, a node sends a `JOIN2_REPLY` message to all nodes that have successfully joined its neighborhood (i.e. the nodes in `joinset`). The message carries the protocertificates of the neighborhood, successor and predecessor. As discussed in Section 5.3.4, the notified joining nodes now take part in the remaining portion of the epoch update, which is specified by EPOCHUPDATE2.

4. In MERGE, nodes determine whether their neighborhood should merge with other adjacent neighborhoods. Merging occurs only if some of the neighborhoods involved are not sufficiently diverse, i.e. if their weakness is above a `weak` threshold. The details of the procedure are shown in Figure 5.14. The procedure returns the new neighborhood `nhd` of the node, as well as (a subset of) the successor (`succnhd`) and predecessor (`prednhd`) neighborhoods.

5. In SPLIT, a neighborhood determines if it can split into two or more adjacent neighborhoods. The splitting only occurs if each of the sub-neighborhoods

after the split is sufficiently diverse, i.e. if its weakness is smaller than a `strong` threshold.

6. Phase certGen (not shown) is analogous to protocertGen: nodes in the same neighborhood exchange the signatures necessary to construct their neighborhood certificate, which this time correspond to the final neighborhood for the epoch (as opposed to the provisional neighborhood). Then nodes in consecutive neighborhoods exchange certificates, so that they all learn the new up-to-date successors and predecessors.

7. In phase itemRedistribute (Figure 5.16, a node u forwards copies of data items that it currently stores (Section 5.3.6) to the new nodes of its neighborhood, if the item identifier still falls in u's neighborhood, or republishes the item, otherwise.

8. In phase fingerUpdate (Figure 5.15), a node updates its finger table by performing one neighborhoodLookup for all the entries in the table.

5.3.6 Publish and Lookup

We finally discuss how SDHT handles data items. The procedures for publishing and lookup are shown in Figure 5.16 and are relatively straightforward. When a node u publishes a data item, u locates the neighborhood that encompasses its identifier and asks each of its members to store a copy of the item. When a node u looks up an identifier, u similarly locates the neighborhood of responsibility and

```
epochUpdate()
    this.oldnhd ← this.nhd
    (newset, joinset, stayset) ← epochUpdateBA()
    (proto, protosucc, protopred) ← protocertGen(newset, joinset, stayset)
    notifyJoined(joinset, proto, protosucc, protopred)
    epochUpdate2(proto, protosucc, protopred)

epochUpdate2(proto, protosucc, protopred)
    (prednhd, nhd, succnhd) ← merge(proto, protosucc, protopred)
    (prednhd, nhd, succnhd) ← split(prednhd, nhd, succnhd)
    certGen(prednhd, nhd, succnhd)
    itemRedistribute()
    fingerUpdate()
```

Figure 5.12: Epoch Update in SDHT. Function epochUpdate2 is factored out because it is shared by the join procedure (Figure 5.11).

queries all nodes in the neighborhood. Then u outputs all the data items that are stored at any member of the neighborhood.[5]

5.3.7 Neighborhood Failures and System Failures

As already mentioned, a node may fail to compute a certificate during an epoch update: we call this event a *neighborhood failure*. A node that reports a neighborhood failure simply leaves the system.

This event is considered a catastrophic event that only occurs in one of two circumstances:

[5]If the application uses self-certifying data items (Section 4.3), the node will also filter out any item that does not pass verification. If, in addition to the above, the application guarantees that no two data items may exist with the same identifier, publish may return after retrieving the first valid item. We do not discuss such details further, because they are application-specific and are not relevant to our discussion.

```
epochUpdateBA()
  S ← this.nhd.nodeset
  result ← setByzantine(S, this.joiningNodes, now)
  stayset ← result ∩ S
  joinset ← result − S
  l = this.nhd.l
  u = this.nhd.u
  for each v in joinset
      if v.id < l or v.id ≥ u
        then joinset ← joinset − {v}
  for each α : ∃v ∈ joinset and AS(v) = α
      n_joining ← |{v ∈ joinset : AS(v) = α}|
      n_staying ← |{v ∈ stayset : AS(v) = α}|
      n_allowed ← max{a − n_staying, 0}
      if n_allowed > n_joining
        then remove the n_joining - n_allowed nodes from joinset with lowest ids
  newset ← joinset ∪ stayset
  return (newset, joinset, stayset)

protocertGen(newset, joinset, stayset)
  nhd ← newCertText(newset, [this.nhd.l, this.nhd.u), currentepoch)
  certsig ← sign(nhd, this.secretkey)
  for each v in stayset
      send(v, < CERT, nhd, certsig >)
  waitFor( f + 1 valid CERT messages with signatures certsig[1], ..., certsig[f + 1] )
  proto ← newCertificate(nhd, certsig[1], ..., certsig[f + 1])
  for each v in this.pred ∪ this.succ
      send(v, < CERT2, proto >)
  receive( message m of type < CERT2, nhd1 > )
  if nhd1 valid and nhd1.l= this.nhd.u
    then protosucc ← nhd1
  receive( message m of type < CERT2, nhd1 > )
  if nhd1 valid and nhd1.u= this.nhd.l
    then protopred ← nhd1
  return (proto, protosucc, protopred)
```

Figure 5.13: Phases of the epoch update sub-protocol.

```
notifyJoined(joinset, proto, protosucc, protopred)
  for each v in joinset
      send(v, < JOIN2_REPLY, proto, protosucc, protopred >)

merge(proto, protosucc, protopred)
  (neighb[−1], neighb[0], neighb[1]) ← (protopred, proto, protosucc)
  for r ← 1 to MERGE_ROUNDS
      for each v in neighb[−r]
          send(v, < MERGE, r, neighb[−1] >)
      for each v in neighb[r]
          send(v, < MERGE, r, neighb[1] >)
      receive( message m of type < MERGE, r, nhd1 > )
      if nhd1.l = neighb[r].u
        then neighb[r + 1] = nhd1
      receive( message m of type < MERGE, r, nhd1 > )
      if nhd1.u = neighb[−r].l
        then neighb[−(r + 1)] = nhd1
  (left, right) ← findSeparators(neighb)
  nhd ← ⋃_{i=left+1}^{right} neighb[i]
  (prednhd, succnhd) ← (neighb[left], neighb[right + 1])
  return (prednhd, nhd, succnhd)

split(prednhd, nhd, succnhd)
  subneighb ← findSubneighborhoods(nhd)
  // subneighb is an array of consecutive neighborhoods
  i ← index s.t. subneighb[i] contains this
  nhd ← subneighb[i]
  if i > 1
    then prednhd ← subneighb[i − 1]
  if i < subneighb.length
    then prednhd ← subneighb[i + 1]
  return (prednhd, nhd, succnhd)
```

Figure 5.14: More phases of the epoch update sub-protocol. In MERGE, the union of two consecutive neighborhoods is defined in the natural way (i.e. the set of nodes of the union is the union of the sets and the interval of the union is the union of the intervals). In SPLIT, the procedure FINDSUBNEIGHBORHOODS (not shown) returns a partition of the given neighborhood into consecutive sub-neighborhoods, such as each sub-neighborhoods has weakness at most `strong`. In MERGE, the procedure FINDSEPARATORS (not shown) determines whether merging is necessary and, if yes, computes the interval of consecutive neighborhoods to be merged with this node's neighborhood.

```
fingerUpdate()
  i ← 0
  while this.id + 2^(m−i) ∈ [this.nhd.u, this.id)
  do nhd ← neighborhoodLookup(this.id + 2^(m−i))
     this.fing[i] ← nhd
```

Figure 5.15: Phase of the epoch update that updates the finger table list.

```
itemRedistribute()
  newnodes ← this.nhd.nodeset − this.oldnhd.nodeset
  for each d in this.dataItems
      if d.id ∈ [this.nhd.l, this.nhd.u)
        then for each v in newnodes
                  send(v, < ITEM_PUT, d >)
        else publish(d)

publish(d)
  nhd ← neighborhoodLookup(d.id)
  for each v in nhd.nodeset
      send(v, < ITEM_PUT, d >)

lookup(x)
  nhd ← neighborhoodLookup(x)
  retrieved ← ∅
  for each v in nhd.nodeset
      send(v, < ITEM_GET, x >)
      receive( message m of type < ITEM_GET_REPLY, items > )
      for each d in items
          if d.id = x
            then retrieved ← retrieved ∪ d
  return retrieved
```

Figure 5.16: Procedures in SDHT that deal with data items. This includes the procedure for publishing, for looking up and for the itemRedistribute phase of the epoch update.

- the number and diversity of the set of nodes in the system is insufficient to sustain even a single neighborhood; or

- the system as a whole is sufficiently diverse, but, because of an unlikely chain of event, too many correct nodes in the same neighborhood crash during the same two epoch period.

A neighborhood failure of the second type occurs with very low probability. In particular, such probability is bounded by the neighborhood weakness (Section 5.3.2), which in turns is kept by the protocol between the two configurable values `strong` and `weak`.

On the other hand, in the worst case, there is no way to bound the probability of a neighborhood failure of the first type. We conjecture that this is an intrinsic problem with the model: in a scenario in which the alive nodes in the system do not represent a sufficiently large number of ASes, it is not possible to guarantee any correct behavior.

When a neighborhood failure of the second type occurs in a neighborhood N, the nodes in the predecessor N' of N will detect this fact, by not receiving any certificate from N during that epoch. The nodes in N' will then execute a *recovery* procedure, in which they look up the value of $N.u$ and change the interval of their neighborhood to $[N'.l, N.u)$ to cover the interval of the failed neighborhood.

A particularly catastrophic event is the *system failure*. A system failure occurs when all the nodes in the system terminate because of a neighborhood failure event. This occurs when a neighborhood failure of the first type. It may also occur when

too many neighborhood failures of the second type happen in a short time interval, such that recovery is not possible.

5.4 Security Analysis

In this section, we analyze the security of SDHT. We were unable to provide a full proof for the claims we present in this section, because of the extreme complexity of the protocol and we present informal arguments instead. We believe that our arguments are very strong, as well as intuitive, and fall short to be proofs only because of a large number of minor technicalities. We leave the development of a formal proof of these claims as an open problem.

Our main result is that, under the assumptions of the AS Model (Section 4.6), SDHT behaves correctly with a reasonably high probability, in the presence of an arbitrary computationally-bounded adversary, during any period of time in which "not too many nodes fail".

Before giving a precise statement of our result, we define the notions of *sane* and *safe* state for a running instance of the SDHT protocol. In what follows, if u is a node, we denote the variable `var` of node u as u.`var`. We similarly denote the property `prop` of a neighborhood N with N.`prop`. Intuitively, the system is sane if the state of all nodes in the system is consistent with what the protocol description assumes it to be.

Definition 5.1. *We say that, at a given time, an SDHT protocol instance is in a* sane *state, if the following properties are satisfied:*

1. *If node v belongs to the neighborhood u.*`nhd`*, then v.*`nhd` $= u$.`nhd`*.*

2. *In each neighborhood N there are at least $f+1$ alive correct nodes from different ASes.*

3. *The successor and predecessor variables of each node contain a valid certificate for appropriate neighborhoods (Section 5.1).*

4. *For every point x in the identifier space, there exists one and only one neighborhood N, such that $N.l \leq x < N.u$ and such that there exists a correct alive node u for which u.*`nhd` *is a valid certificate corresponding to N. We call N the* neighborhood encompassing *x and will denote it by* `nhd`(x, e)*, where e is the current epoch.*

Additionally, we say that the system is in a safe *state, if it is in sane state and every neighborhood N has weakness at most N.*`weak`*.*

Note that the protocol actively seeks to maintain a sane state, by updating all node information at each epoch. The protocol also tries to maintain a safe state by merging neighborhoods that are too weak. We can now state our main result:

Informal Claim 5.2. *Let p and* `SDHT_WEAK` *be the two parameters of the SDHT, as defined in Table 5.1. Assume that the SDHT is in a* safe *state at a time t_1. Assume further that at a time t_f during the interval $[t_1, t_2]$ some of the correct nodes may crash: specifically, every correct node has an independent probability of at most p of crashing, where p is the appropriate parameter in the protocol (Table 5.1). Assume that at some point during $[t_1, t_2]$ a node u publishes a data item I with identifier x*

and at some later time, within the same time interval, a node v looks up identifier x. Then, node v will correctly retrieve item I, except with probability at most SDHT_WEAK

.

What this result is saying is that, as long as the system is in a safe state, the lookup protocol behaves correctly, *regardless of the action of the corrupted nodes.*

However, this result, on its own, is not sufficient to demonstrate the usefulness of the algorithm. We need to show instead that, under many reasonable circumstances, that the algorithm *remains* in a safe state. Another issue to take into account is that joining nodes may be rejected by the system. Therefore, we also need to show that, under reasonable circumstances, most joining nodes are accepted, as opposed to rejected. We will demonstrate these facts through our set of experiments in Section 5.5.

The remainder of this section will be used to show Informal Claim 5.2. In Section 5.4.1, we show that no neighborhood failure occurs except with probability SDHT_WEAK; this leaves to argue that the claim holds with probability 1 conditioned on the fact that no neighborhood failure occurs. In Section 5.4.2, we show that, if no neighborhood failure occurs, the system remains at least in a *sane* state over time. In Section 5.4.3, we show that the lookup operation is correct, under the assumptions of Informal Claim 5.2. Finally, in Section 5.4.4, we "prove" our main result.

5.4.1 Dealing with Neighborhood Failures

Towards a justification of our main result (Informal Claim 5.2), we argue that the probability that some neighborhood failure occurs is at most `SDHT_WEAK`. This leaves us the task to "prove" that the claim holds when neighborhood failures do not occur.

Informal Claim 5.3. *In the setting of Informal Claim 5.2, let e be the epoch that contains t_f and let $e+1$ be the epoch that immediately follows. For each x, the probability that a correct node in* $\texttt{nhd}(x, e)$ *reports a neighborhood failure during either e or $e+1$ is at most* $\texttt{nhd}(x, e).\texttt{weak}$.

Informal Argument A node in $\texttt{nhd}(x, e)$ reports a neighborhood failure during epoch e or $e+1$ if, after the crashes at time t_f, the number of correct ASes with alive nodes in the neighborhood becomes less than $f+1$.

The claim follows from the definition of weakness of a neighborhood, given in Section 5.3.2 and from the fact that the weakness of $\texttt{nhd}(x, e)$ is at most $\texttt{nhd}(x, e).\texttt{weak}$, by assumption of safety of the system. □

Informal Claim 5.4. *In the setting of Informal Claim 5.2, the probability that any correct node reports a neighborhood failure during $[t_1, t_2]$ is at most* `SDHT_WEAK`.

Informal Argument Let e as in Informal Claim 5.3. No neighborhood failure can occur except during epochs e and $e+1$. The probability that any neighborhood fails during e or $e+1$ is at most $\sum_N N.\texttt{weak}$, by Informal Claim 5.3. Such summation is at most `SDHT_WEAK` by construction. □

5.4.2 Sanity is Preserved

The next step towards the justification of Informal Claim 5.2 involves showing that the system remains in a sane state over time, if no neighborhood failure occurs.

Informal Claim 5.5. *If an instance of the SDHT protocol is in a sane state at the beginning of an epoch and no neighborhood failure occurs, then the SDHT is also in a sane state at the beginning of the next epoch.*

Our informal proof of this result follows from Informal Claim 5.11 through Informal Claim 5.14.

Informal Claim 5.6. *Assume that the properties of Definition 5.1 at the beginning of* epochUpdateBA*. Then, at the end of the procedure all correct processes in a neighborhood output the same sets* `newset`*,* `joinset`*,* `stayset`*.* `joinset` *contains all correct nodes that have executed a join operation in the previous neighborhood, while* `stayset` *contains all correct nodes in the neighborhood that have survived.* `joinset` *may also contain arbitrary corrupt nodes with an identifier in the neighborhood, while* `stayset` *may also contain correct nodes in the neighborhood that have failed.*

Informal Argument This follows from the property of the Byzantine agreement protocol and by the post-processing stage of epochUpdateBA. □

Informal Claim 5.7. *Unless a neighborhood failure occurs, at the end of the procedure* protocertGen*, the return value* `proto` *contains a valid certificate for the neighborhood defined by* `newset`*.*

Informal Argument This follows easily by inspection of PROTOCERTGEN, by the fact that all alive correct nodes in `newset` will enter the procedure with the same value of the `newset` argument (Informal Claim 5.6) and by the fact that, if a neighborhood failure does not occur, then the node will be able to obtain at least $f + 1$ signatures from nodes in `newset` belonging to different ASes. □

Informal Claim 5.8. *Let u be a node executing* EPOCHUPDATE2 *during epoch e. Let v be any node in the neighborhood in the* `proto` *argument for node u. Then the arguments* `proto`*,* `protosucc` *and* `protopred` *represents respectively the same neighborhoods for the invocation of* EPOCHUPDATE2 *of u and v.*

Informal Argument This is an immediate consequence of the Informal Claim 5.7. □

Informal Claim 5.9. *Let u be a correct node and let N'_u be the return value* `nhd` *of the invocation of* MERGE *made by u during epoch e. Let v be any correct node in the neighborhood N'_u and let N'_v be the return value* `nhd` *of the* MERGE *invocation by v. Then $N'_u = N'_v$.*

Informal Argument Let η, l, r (resp. η', l', r') be the content of the variables `neighb`, `left`, `right` computed by u (resp. v) inside MERGE. Since v belongs to N'_u by assumption, it must be the case that there exists a k such that v is in the neighborhood $\eta[k]$ and $l + 1 \leq k \leq r$ (by inspection of MERGE).

The first fact we need is that η and η' represent *consistent* views of the Chord ring, i.e.:

$$\eta'[i] = \eta[i + k] \tag{5.1}$$

for all i for which i and $i+k$ are within the bounds of the arrays η and η'. This follows easily from the structure of the MERGE procedure and from Informal Claim 5.8.

We also need to ensure that the procedure FINDSEPARATORS satisfies the following properties:

1. it is deterministic;

2. it always returns a pair $(\texttt{left}, \texttt{right})$ such that $\texttt{left} + 1 \leq 0 \leq \texttt{right}$;

3. if $\eta'[i] = \eta[i+k]$, for some k and for all i, then either $l' = l + k$ and $r' = r + k$ or the the two intervals $[l+1, r]$ and $[l'+k+1, r'+k]$ do not intersect.

It is relatively easy to construct a procedure FINDSEPARATORS that satisfies such property and also causes neighborhoods to merge when appropriate.

Eq. (5.1) combined with Property 3 of FINDSEPARATORS ensures that $l' = l+k$ and $r' = r + k$, since $[l+1, r]$ and $[l'+k+1, r'+k]$ intersect in this case. (Property 2 ensures $l' + 1 \leq 0 \leq r'$, which implies $l' + k + 1 \leq k \leq r' + k$, while we argued earlier that $l + 1 \leq k \leq r$.)

By inspection of MERGE we can then conclude that u and v return the same value of `nhd` as needed. □

Informal Claim 5.10. *Let u be a node and let* `nhd` *be the return value of the* SPLIT *executed by u during epoch e. Let v be any node in the neighborhood* `nhd`*. Then u and v obtain the same return values from the invocation of* SPLIT *during epoch e.*

Informal Argument By inspection of SPLIT and its invocation within `epochUpdate2` (Figure 5.12), it is easy to see that v also belongs to the return value `nhd` of the

invocation of MERGE executed by u. By virtue of Informal Claim 5.9, u and v enter the procedure SPLIT with the same values of `nhd`. Once within the SPLIT procedure, nodes u and v will compute the same SUBNEIGHB array, since it is obtained via a deterministic computation on `nhd`. By assumption, it must be that u and v compute the same value of i and, therefore, return the same value of `nhd` from the procedure.

□

Informal Claim 5.11. *Property 1 of Definition 5.1 holds at the end of the epoch update phase.*

Informal Argument Let u be a node and let N be the value of the variable `nhd` as returned by the SPLIT procedure executed by u during epoch e. Let v be any node in the neighborhood N. By virtue of Informal Claim 5.10, u and v have the same return values from SPLIT.

Since no neighborhood failure occurs, then u and v will be able to construct a neighborhood certificate and therefore will have the same neighborhoods in the variable **this**.`nhd`.

□

Informal Claim 5.12. *Property 2 of Definition 5.1 holds at the end of the epoch update phase.*

Informal Argument This follows directly from Informal Claim 5.4. □

Informal Claim 5.13. *Property 3 of Definition 5.1 holds at the end of the epoch update phase.*

Informal Claim 5.14. *Property 4 of Definition 5.1 holds at the end of the epoch update phase.*

Both Informal Claim 5.13 and Informal Claim 5.14 can be "proven" similarly to Informal Claim 5.11, by tracing the behavior of the protocol through the different phases of the algorithm. We omit those easy, but tedious details.

5.4.3 Correctness of Publish and Lookup Operations

The correctness of publish and lookup operations follow from two results: the first (Informal Claim 5.15) says that the NEIGHBORHOODLOOKUP operation always return the desired certificate and the second (Informal Claim 5.16) says that, once a data item is published on a neighborhood, the item remains correctly replicated on that neighborhood.

Informal Claim 5.15. *Under the assumptions of Informal Claim 5.2, the* NEIGHBORHOODLOOKUP *subroutine, on input an ID x, returns a current valid neighborhood certificate for a neighborhood encompassing x, conditioned on the fact that no neighborhood failure occurs.*

Informal Argument Conditioned on the fact that a neighborhood failure does not occur, at some point in time after the NEIGHBORHOODLOOKUP is requested, the system is guaranteed to be sane, by virtue of Informal Claim 5.5. At such time, the neighborhood lookup operation can be performed correctly, as we now argue.

Recall the definition of d given in Section 5.3.3 and let N_i, for $i = 0, 1, \ldots,$ be the value of the variable `nhd` at the beginning of the $(i+1)$-th iteration of the

external **repeat** loop in Figure 5.10.

First, we want to show that $d(N_{i+1}, x) < d(N_i, x)$, for all $i \leq r - 2$, where r is the total number of iterations. x does not lie in N_i, otherwise there would be no $i+2$ iteration of the loop. Informal Claim 5.5 tells us that, for all correct nodes u in N_i, u has a successor pointer pointing to a neighborhood N' such that $N'.l = N_i.u$ (Property 3 of Definition 5.1). That claim also tells us that N_i has an alive correct node. This means that the internal **repeat** loop of the NEIGHBORHOODLOOKUP function may have one of two outcomes: either a corrupt node is contacted, which returns a neighborhood N_{i+1} such that $d(N_{i+1}, x) < d(N_i, x)$; or an honest node is contacted, which will return the neighborhood N_{i+1} for which it holds a certificate and such that $d(N_{i+1}, x)$ is minimized. In the latter case, note that $d(N_{i+1}, x) \leq d(N', x) < d(N_i, x)$, because N' belongs to the set of neighborhoods for which the contacted node holds a certificate. In both cases, our claim is true.

We have shown that $d(N_{i+1}, x) < d(N_i, x)$ for all i, unless x lies in N_i and that the external **repeat** loop terminates only at the iteration r, such that x falls in the N_{r-1}. Since the number of neighborhoods is at most the number of nodes and no neighborhood is visited twice, the loop will actually terminate.

□

Informal Claim 5.16. *If at any time a node u stores a copy of a published data item with ID x, then at all subsequent times all correct nodes of* $\texttt{nhd}(x, e)$ *store that item, unless a neighborhood failure occurs at* $\texttt{nhd}(x, e)$.

Informal Argument This follows immediately from inspection of the ITEMREDISTRIBUTE procedure. □

5.4.4 Informal Proof of Informal Claim 5.2

We are now ready to argue Informal Claim 5.2.

Informal Argument We know from Informal Claim 5.4 that there is no neighborhood failure, except with probability $SDHT\_WEAK$. Given that a neighborhood failure did *not* occur, we show that v correctly retrieves the data item I, when performing a lookup for identifier x.

Informal Claim 5.15 guarantees that u correctly retrieves $\texttt{nhd}(x, e)$ at the epoch e in which u performs the publish operation. It also guarantees that v correctly retrieves $\texttt{nhd}(x, e')$, where e' is the epoch at the time of the lookup. $\texttt{nhd}(x, e)$ contains at least one alive honest node, therefore such honest node stores a copy of the data item during the Publish operation. Because of Informal Claim 5.16, any data item stored in a correct node on $\texttt{nhd}(x, e)$ will also be stored on any correct node in $\texttt{nhd}(x, e')$. $\texttt{nhd}(x, e')$ contains at least one correct node and therefore will provide the data item to u during the Lookup operation. This completes the argument. □

5.5 Experimental Results

In this section, we describe our implementation of SDHT and a set of experimental results obtained from such implementation.

5.5.1 Implementation of SDHT

We have implemented a prototype of SDHT, which provides the main functionalities of the protocol, for the purpose of evaluating the algorithm correctness and performance.

As part of the development of the implementation, we have also developed a novel application that we call Simulation Implementation Engine (SIMPL Engine), which allows both the simulation and the implementation of an arbitrary protocol.[6] In order to use SIMPL Engine, the user has to provide a shared library (the *agent*) which contains the code of the protocol of interest and which implements a given API. The agent communicates with the outside world exclusively via the abstract API, which allows the agent to send and receive messages as well as to perform other basic operations such as obtaining the current time and scheduling timeouts. When in *simulation mode*, a single instance of SIMPL Engine creates a number of instances of the user-specified agent, each instance corresponding to a node; both time and communication are simulated through the event-driven engine. When in *implementation mode*, a single instance of SIMPL Engine becomes a single node, by creating one instance of the agent; messages are sent and received to other nodes via TCP connections.

SIMPL Engine is written in C, approximately 7100 lines of code. The agent that implements the SDHT node is also written in C and consists of approximately

[6]SIMPL Engine is joint work with Randy Baden, Adam Bender, Matt Mah and Rob Sherwood, of the Department of Computer Science at University of Maryland. The SDHT agent is also joint work with Baden, Bender and Mah.

Table 5.3: Approximate message sizes (bytes) in the SDHT prototype, as estimated by code inspection. Message size depends on several factor, especially the value of f and the number of nodes in a neighborhood. In this table, sizes are shown as the sum of three components: the first is fixed, the second is proportional to the neighborhood size and the third is proportional to $(f + 1)$. For concreteness, sizes are also shown for an example where $f = 5$ and neighborhoods have size 50. The "other" category of messages has been divided into messages that contain one certificate (such as `CERT2`) and that do not contain certificates (such as `JOIN2_REQUEST`, `RLOOKUP_REQUEST`).

Type	Fixed Component	Per Node in Neighborhood	Times $f + 1$	Example Total
pslBroadcast (worst case)	40	144	0	7240
pslBroadcast (no failures)	328	0	0	328
`CERT`	164	16	0	964
`JOIN2_REPLY`	76	48	384	4780
other (with cert.)	36	16	128	1604
other (without cert.)	16	0	0	16

11500 lines of code.

The implementation has the following limitations, due to its prototype nature:

- Digital signatures are not implemented. More precisely, in the current implementation all signatures are of length zero and always verify.

- The recovery phase is not currently implemented, which means that a neighborhood failure always causes a system failure (Section 5.3.7).

- The code is neither optimized nor extremely robust, which limits the size of the experiments that can be performed to approximately 100–200 nodes, depending on the setup.

Table 5.3 shows the size of protocol messages in the SDHT implementation, as determined by inspecting the code and assuming that digital signatures are of length 128 bytes (e.g. 1024-bit RSA). The messages used by pslBroadcast are the

most important, because their number dominates the number of all other messages. Although those messages can be very large in the worst case (because they may contain a chain of signatures that is as long as the number of nodes in a neighborhood), they are of a reasonable size (328 bytes) for executions in which no failures occur (including no misbehavior and no correct node crashes), because they contain at most two signatures. Other messages of large size are the ones that carry certificates, especially the `JOIN2_REPLY` message type, because it carries three certificates. For the latter category, the dominant component of the message is the vector of $f + 1$ signatures.

5.5.2 Experiment Setup

Our experiments with the SDHT implementation are performed on a 30-host cluster. Each machine is a single-processor Pentium III 650 MHz with 768M of memory and running a 2.4 Linux kernel. The machines are connected with a 100Mbit Ethernet switch.

The mapping between a node address to the corresponding AS number is simulated, in order to allow all nodes to run within the same real-world AS. The simulated mapping is implemented by generating a file that maps IP address–port pairs to AS numbers; every node is given access to a copy of such file.

We employ a *bootstrap agent* to aid our experiment setup. SDHT nodes contact first the bootstrap agent to obtain the experiment parameters and, for nodes that are not part of the initialization phase (Section 5.3.1), the addresses of the twenty

Table 5.4: Parameters for the lookup validation experiment. See Table 5.1 and Table 5.2 for more information.

	Value
Max nodes from 1 AS (a)	10
Epoch length (E)	150 sec
Max bad ASes (f)	1
Assumed prob. of crash (p)	0.04
`SDHT_WEAK`	0.01
`SDHT_STRONG`	0.001
Number of nodes (n)	100
AS distribution	10 ASes of 10 nodes each

last nodes that have successfully joined the system.

5.5.3 Lookup Correctness

In this first experiment, we validate the implementation, by verifying it correctly performs the publish and lookup operation and by measuring the corresponding cost.

On a 100-node instance of SDHT, we publish 3281[7] data items with random identifiers and we verify that a lookup operation on each of those successfully locates the item. We also look up 4042 data items with random identifiers that have not been published and we verify that all such operations successfully complete with a negative result. Table 5.4 summarizes the parameters of the experiment. For such settings, the SDHT instance consists of three neighborhoods of sizes 30, 32 and 38.

Figure 5.17 shows the distribution of the time to complete each of the three types of operation. We observe that an overwhelming majority of the operations

[7]In the setup we determine the duration of the experiment in seconds, which indirectly determines the number of publish and lookup operations.

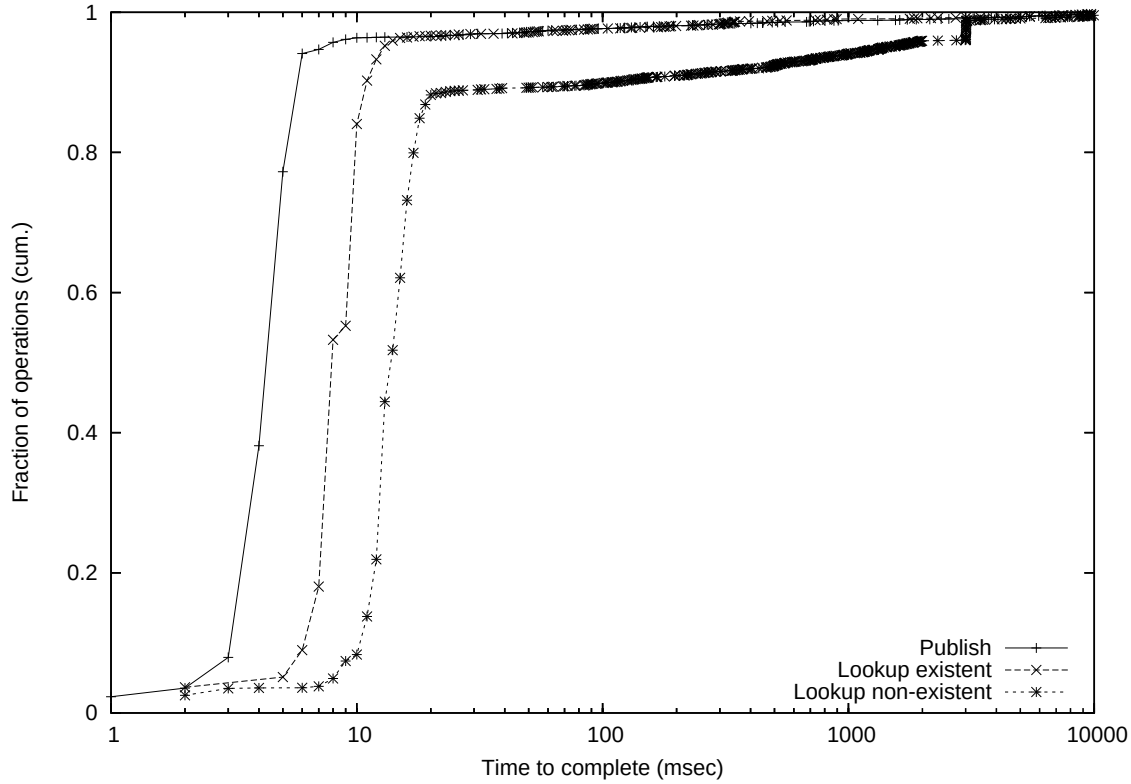


Figure 5.17: Lookup validation experiment. For each of the three types of operation (publish, lookup of published item, lookup of non-existent item), the plot shows the cumulative distribution of the time required for the operations of that type to complete. The scale is semilogarithmic.

complete in a relatively small time, as expected: reading the "knee" of the three curves, we note that over 94% of the publish operations complete in 6 msec or less, while 95% of the lookups for the published items complete in 13 msec or less. The lookup operations for non-existing items are slower (88% complete in 20 msec or less; 95% complete in 1.43 sec or less), as expected, since all the nodes of the neighborhood encompassing the target identifier need to be contacted.

We also observe, that a small fraction of the operations require a significantly large amount of time, of the order of several seconds. This is due to the fact that, in the prototype implementation, nodes cannot process NEIGHBORHOODLOOKUP operations during a portion of the epoch update, which typically lasts 1 second, but is observed to be as long as 14 sec in some instances during this experiment. The operations are paused during such critical section of the code and this fact is responsible for the occasional long delays. This problem can be avoided with an optimized implementation.

5.5.4 Lookup Under Attack

In this experiment, we verify that the SDHT implementation is resilient to a very basic attack. The attacker controls one AS and the nodes from this AS always report that no data item exists in response to a `ITEM_GET` message, but otherwise follow the protocol correctly. All other nodes behave correctly and never fail. The setup of the experiment is otherwise identical to that of Section 5.5.3. We verified that all lookup operations return the outcome we expect (positive if the item has

Table 5.5: Parameters for the maintenance cost experiment.

	Value
Max nodes from 1 AS (a)	10
Epoch length (E)	150 sec
Max bad ASes (f)	1
Assumed prob. of crash (p)	0.04
`SDHT_WEAK`	0.01
`SDHT_STRONG`	0.001
Join rate	100 nodes/hour
Node lifetime	exp. distr. with mean 1 hour
AS distribution	10 equally-sized ASes

been published and negative otherwise).

5.5.5 Maintenance Cost

In this experiment, we measure the overhead that SDHT incurs during the epoch update phase. We demonstrate that the overhead of SDHT is practical for a relatively-static system with long epochs.

Table 5.5 summarizes the parameters of the experiment. The SDHT instance begins with 100 nodes and a batch of 100 additional nodes join every hour. Each node has an exponentially distributed lifetime with average 1 hour, after which the node silently crashes. The epoch length is set to 150 seconds.

Figure 5.18 shows the number of nodes in the system, as well as the protocol overhead, measured as the number of messages sent over the entire system, as a function of time. Note that, as expected, the number of nodes increases by about 100 at the beginning of every hour and declines slowly over time because of the random node crashes. The experiment terminates after 3 hours, when a random sequence of crash events causes a total system failure.

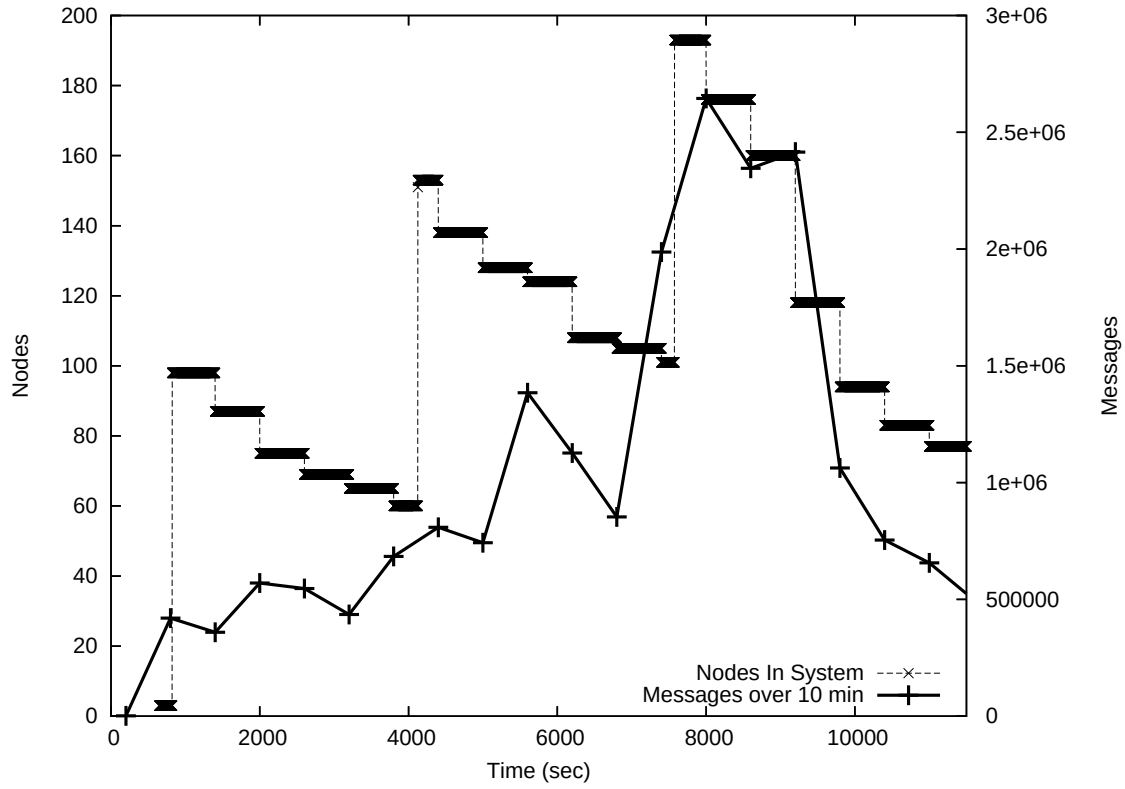


Figure 5.18: Cost of maintenance experiment. The plot shows the number of nodes that are part of the SDHT instance as a function of time, as well as the number of messages sent during every 10 minute period.

We observe that the cost of maintenance of SDHT is somewhere between 12,500 and 50,000 messages *per node per epoch* and grows linearly with the number of nodes in the system. An inspection of the experiment raw data shows that the component of the overhead that dominates this cost is the Byzantine agreement phase at the beginning of the epoch update (Section 5.1.2, Section 5.3.5). As mentioned in Section 5.2.3, the number of messages sent by the Byzantine agreement protocol *for one neighborhood* is cubic in the size of the neighborhood. If all the neighborhoods contain approximately the same number of nodes, the total cost is thus linear in the number of nodes and quadratic in the size of a neighborhood. This overhead is very high: in the same setting Chord [SMK+01] would require less than 75 messages per node per minute for its `stabilize` and `fix_fingers` periodic procedures,[8] plus a negligible number of messages to handle joining nodes.[9] However, as already discussed in Chapter 4, Chord does not provide any protection against malicious behavior.

It is important to understand that the overhead of SDHT is proportional to the frequency of the epoch updates and that this experiment shows only the extreme case in which an epoch lasts only 2 1/2 minutes. The overhead can be dramatically lowered, by choosing a much longer epoch length, for example 1 hour or even 1

[8] We assume that `stabilize` and `fix_fingers` are executed every 30 seconds, as in the setup of [SMK+01, Section 6.5], that running `stabilize` requires 4 messages, that fixing a finger also requires 4 messages (because a strong hint is available during lookup) and that all $\log n$ fingers are fixed at every update.

[9] 100 nodes join every hour, each join requiring at most $\log^2 n \leq \log^2 300 \leq 81$ messages. This means at most 81 messages per node per hour.

Table 5.6: Parameters for the effect of parameter f experiment.

	Value
Max nodes from 1 AS (a)	10
Epoch length (E)	150 sec
Max bad ASes (f)	1, 5, 10
Assumed prob. of crash (p)	0.04
`SDHT_WEAK`	0.01
`SDHT_STRONG`	0.001
Join rate	100 nodes/hour
Node lifetime	until end of experiment
AS distribution	all nodes are from distinct ASes

day. The two conditions that are necessary to choose longer epochs are: (1) the rate at which nodes crash is sufficiently low, so that the probability that a node crashes over an epoch is small (e.g. 4%–8%); and (2) it is acceptable that a node may have to wait for up to an entire epoch to be able to join or gracefully leave the system. While the second condition is obvious, the first condition is necessary to avoid excessively large neighborhoods that would be required by a large value of p (Section 5.3.2).

5.5.6 Effect of Parameter f

In this experiment, we study the effect of the parameter f on the protocol. As we will show, as f increases, the performance of SDHT degrades, because neighborhoods grow larger, thus increasing the overhead of the epoch update.

We run instances of SDHT with 100 initial nodes and, at every hour, we allow 100 new nodes to join, up to a maximum of 3 hours and 300 nodes. The parameters of the experiment (Table 5.6) are similar to those in the previous experiment, except that all nodes live until the end of the experiment and that we vary f to assume

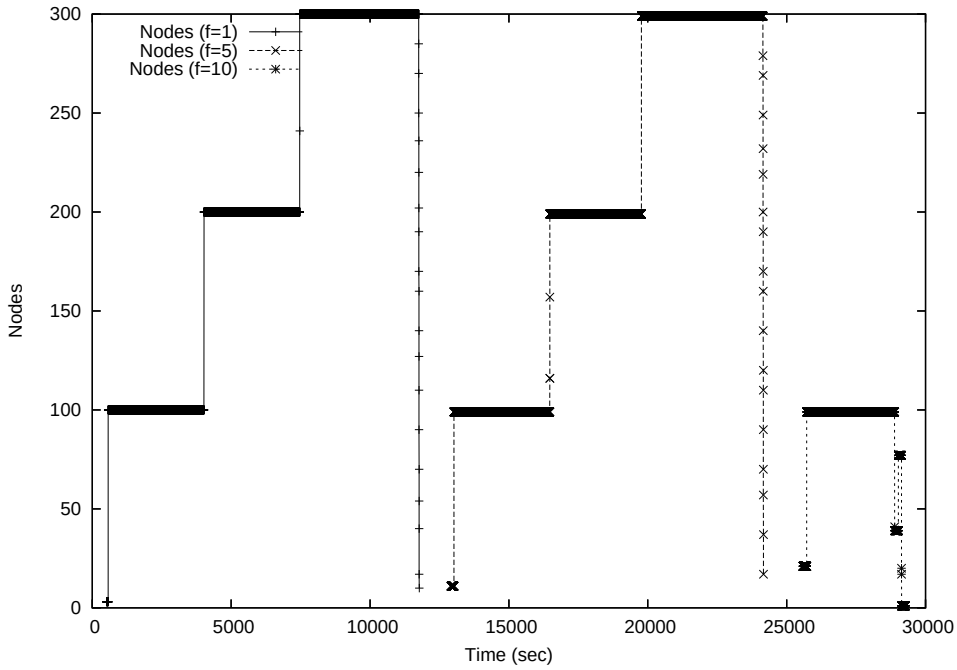


Figure 5.19: Effect of varying f. The plot shows the number of nodes in the system as a function of time. The three experiments are shown sequentially, for better clarity.

values 1, 5 and 10 respectively, in each instance. For a more meaningful comparison between different values of f, all nodes will originate from distinct ASes.

Figure 5.19 shows the number of nodes in the system as a function of time, during the sequence of the three experiments. The experiment with $f = 1$ is performed between time 0 and (approximately) time 12,000 sec; the experiment with $f = 5$ occurs between time 13,000 sec and time 25,000; the last experiment ($f = 10$) is performed afterwards.

We observe how the experiment completes successfully for the cases of $f = 1$ and $f = 5$, while a system failure occurs in the case $f = 10$, right at the time when the second batch of 100 nodes joins the system. This phenomenon is due to the fact that neighborhoods become larger and larger as f increases, thus increasing the

Table 5.7: Neighborhood size for different values of f, at the time the SDHT instance contains 100, 200 and 300 nodes respectively. For $f = 10$, the system never reaches the two larger sizes.

Max bad ASes (f)	Number of nodes (n)	Neighborhood sizes
1	100	18, 18, 18, 18, 28
1	200	21, 31, 33, 29, 30, 21, 35
1	300	30, 42, 22, 23, 21, 24, 21, 21, 21, 25, 28, 22
5	100	27, 27, 46
5	200	29, 39, 44, 29, 29, 30
5	300	54, 30, 32, 50, 49, 42, 43
10	100	39, 60

number of messages that need to be sent during the epoch update (Table 5.7). As discussed in Section 5.5.5, the communication required during an epoch update is quadratic in the size of the neighborhoods, for a fixed number of nodes.

With respect to Table 5.7, we notice the high variability in the size of a neighborhood, for a given system size and set of parameters. For example, for $f = 5$ and $n = 300$, we see both a 30 node neighborhood and a 54 node neighborhood. This behavior is due to the fact that a neighborhood may only split when it reaches a critical size (say 56 nodes). As nodes randomly join there will be some neighborhoods that barely reach the splitting threshold (e.g. they become 60 nodes) and then split into neighborhoods of smaller size (e.g. 30 each). Some other neighborhoods, instead, narrowly miss the threshold (56 in the example) and are stuck with, say, 54 nodes. This explanation is an oversimplification because what determines whether a neighborhood may split is the weakness of the sub-neighborhoods (Section 5.3.2) and not the size of the neighborhood. This explains why for $f = 1$ and $n = 300$ we observe both a neighborhood of size 21 and one of size 42. However, the size of a neighborhood is intuitively a good predictor of its weakness, for a given AS

Table 5.8: Parameters for the rejection rate and neighborhood size high-level simulation.

	Value
Max nodes from 1 AS (a)	10, 20, 30
Max bad ASes (f)	1
Assumed prob. of crash (p)	0.04
`SDHT_WEAK`	0.01
`SDHT_STRONG`	0.001
join rate	0.05, 0.1, 0.2, 0.5
max SDHT size	2000
AS distribution	Uniform from a set of 10 ASes

distribution.

5.5.7 Rejection Rate and Neighborhood Size

As discussed in Section 5.1.2, nodes that attempt to join a running instance of SDHT may be rejected because a nodes from the same AS are already in the neighborhood (Section 5.1.2). It is possible to reduce the rejection rate by increasing the value of a, but at the price of larger neighborhoods. In this experiment, we study how the value of a and the rate at which nodes join affect the rejection rate (e.g. the fraction of joining nodes that are rejected), as well as the average neighborhood size. We perform this study with a high-level simulator (Ruby, approximately 1000 l.o.c.), because we do not need the full power of an implementation and our Ruby simulator enables more complex experiments in a shorter amount of time.

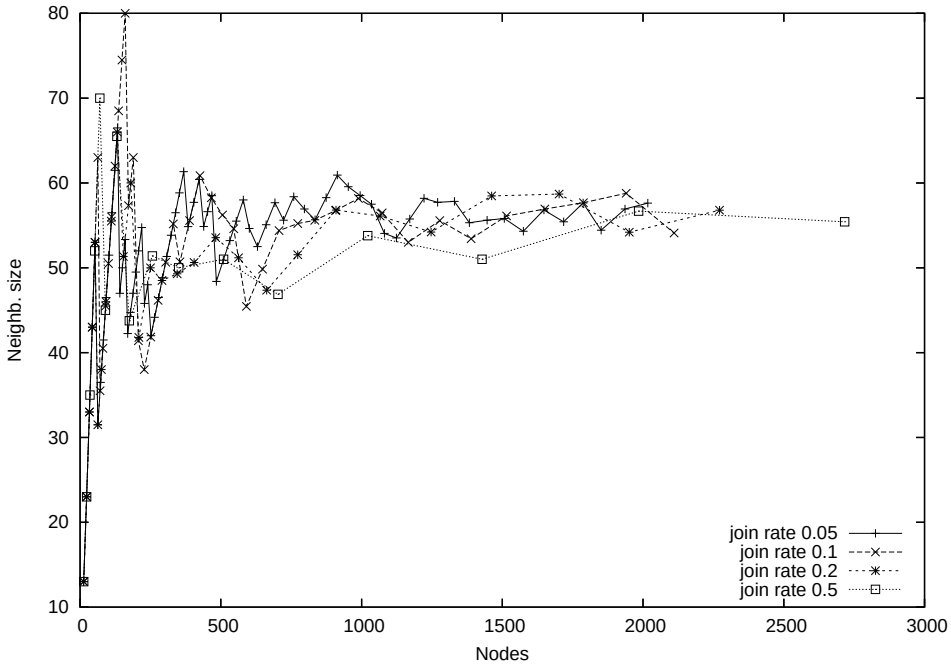
A basic simulation run creates a SDHT with $2f + 1$ nodes from different ASes. At every epoch e, a new set S_e of nodes attempts to join the system. We obtain the number $|S_e|$ of joining nodes by multiplying the number of nodes currently in the SDHT by a given *join rate* (Table 5.8); we establish that a minimum of 10 nodes

join at every epoch. Each joining node is associated with a uniformly random AS in a set of 10 ASes. No nodes misbehave or crash, so the number of nodes increases exponentially over time. The run stops when the number of nodes in the SDHT exceeds 2000.

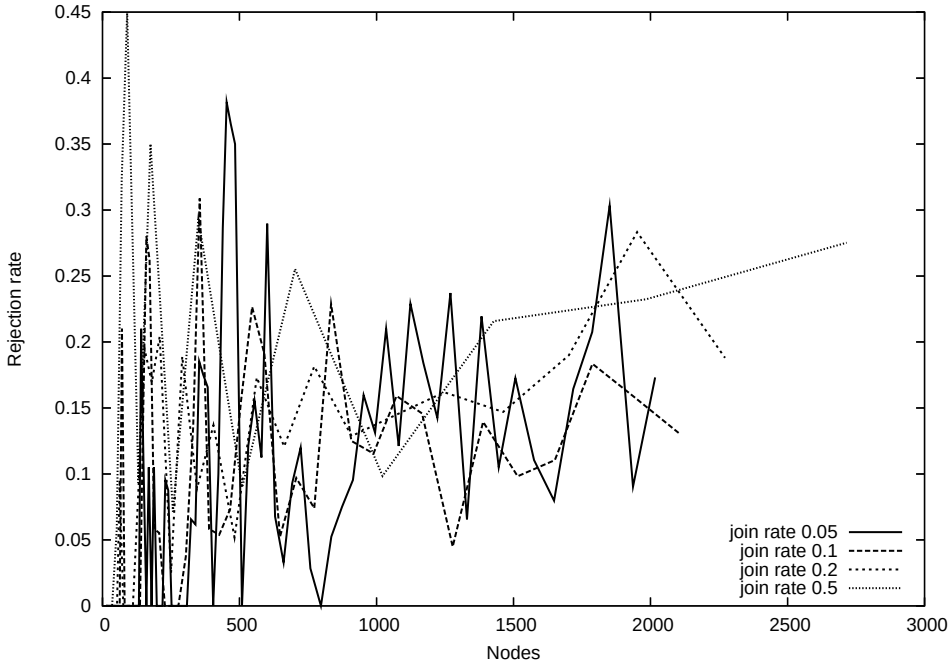
Figure 5.20 shows how the rejection rate and the neighborhood size vary within a single run, for the case $f = 1$ and $a = 10$. Four runs are shown, for different values of the join rate. Figure 5.21 shows the rejection rate and neighborhood size, averaged across an entire run, as a function of the join rate. From both figures we see that the neighborhood size is not significantly influenced by the join rate. This phenomenon is due to the fact that the merge-split process of SDHT maintains the neighborhoods around the desired size, regardless of how fast nodes join.

For the case $a = 10$, the rejection rate increases from an average of 10% when the join rate is 0.05 or 0.1 to an average close to 20%, when the join rate is 0.5. Indeed, when nodes join faster, neighborhoods do not have the time to split as often, since splitting occurs only once per epoch. This fact leads to more neighborhoods in which the quota of a nodes from the same AS is filled up. When a is 20 or more, we notice that the rejection rate is negligible for the given parameters, although it still increases with the join rate.

More surprising is the fact that the neighborhood size does not seem to change with the value of a. This is due to the fact that, in this experiment, the number of joining nodes from each AS is about the same, therefore the splitting process successfully compensates for the occasional larger number of nodes from the same AS that may land on the same neighborhood. However, it is important not to be



(a) Average neighborhood size



(b) Rejection rate

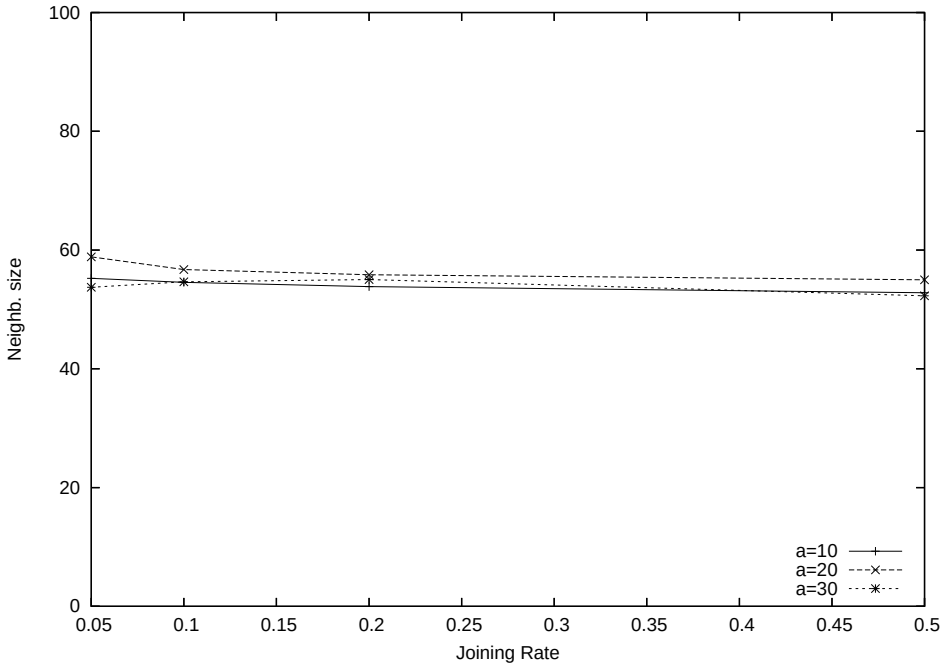
Figure 5.20: Neighborhood size and rejection rate, within a simulation run with $f = 1$ and $a = 10$, as a function of the number of nodes in the SDHT. The figure shows four different settings of the join rate.

tricked into believing that a larger value of a can be chosen for free. Increasing a allows an attacker to increase the size of the neighborhoods significantly. The attacker could do so by generating a very large number of nodes from the ASes that she controls, therefore increasing the number of nodes in every neighborhood to the maximum allowed. In the example of Figure 5.21, each neighborhood contains an average of about 5 nodes from each AS (approximately 55 nodes divided equally between 10 ASes). In the case $a = 30$ and $f = 1$ the attacker could raise the number of nodes from her AS from 5 to 30, thus bringing the size of the neighborhood to about 80.[10] However, if $a = 10$, the attacker can raise the neighborhood size only to about 60. Obviously, this penalty in increasing a grows linearly with the value of f. The same penalty also applies to a setting in which there is no attacker, but a few of the ASes contribute to a much larger fraction of the nodes; we can use the same analysis, since we can think of such more prominent ASes as under control of an attacker.

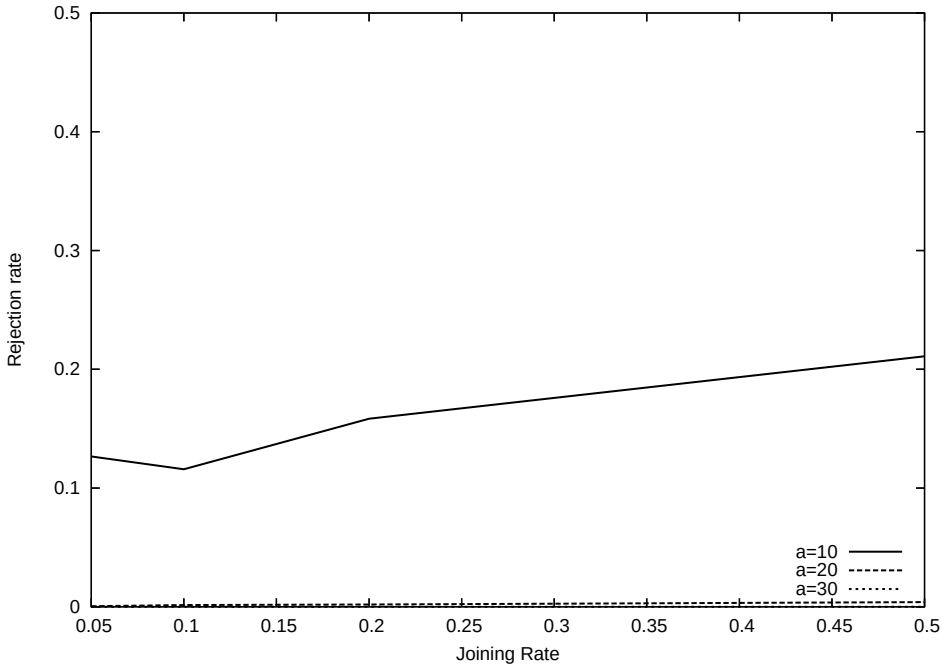
5.5.7.1 Increasing the Diversity of Joining Nodes

We repeat the simulation with a more diverse set of joining nodes. We associate each node with a uniformly random AS from a set of 25 (up from 10) ASes. Other parameters are as in Table 5.8. The results are shown in Figure 5.22. We notice how the neighborhood size decreases from approximately 55 (in Figure 5.21) to approximately 35. This is due to the fact that a smaller number of nodes is necessary

[10]Remember that the weakness of a neighborhood does not decrease when the size of the f largest ASes in a neighborhood increases.



(a) Neighborhood size

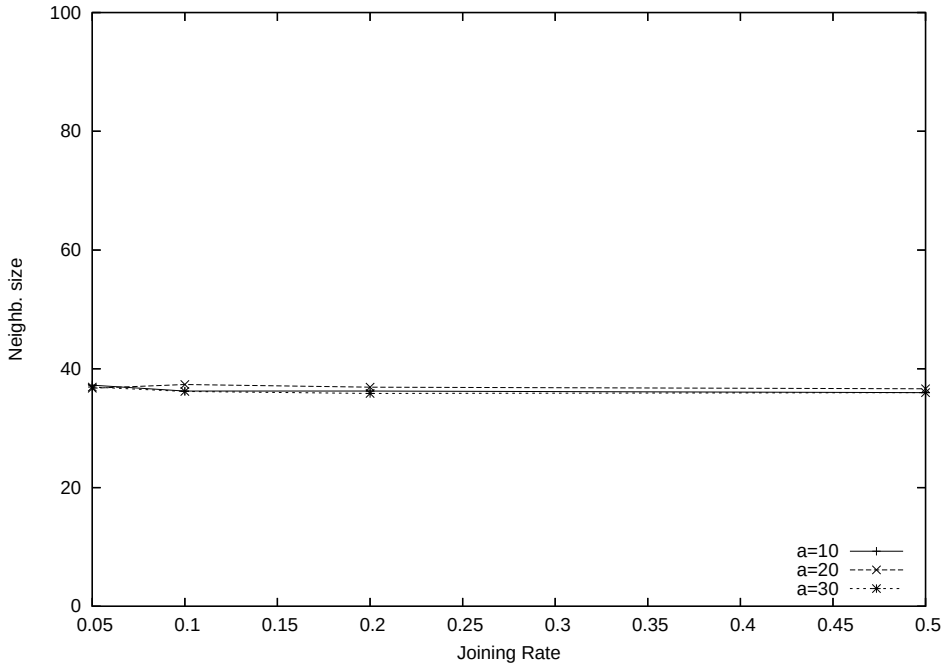


(b) Rejection rate

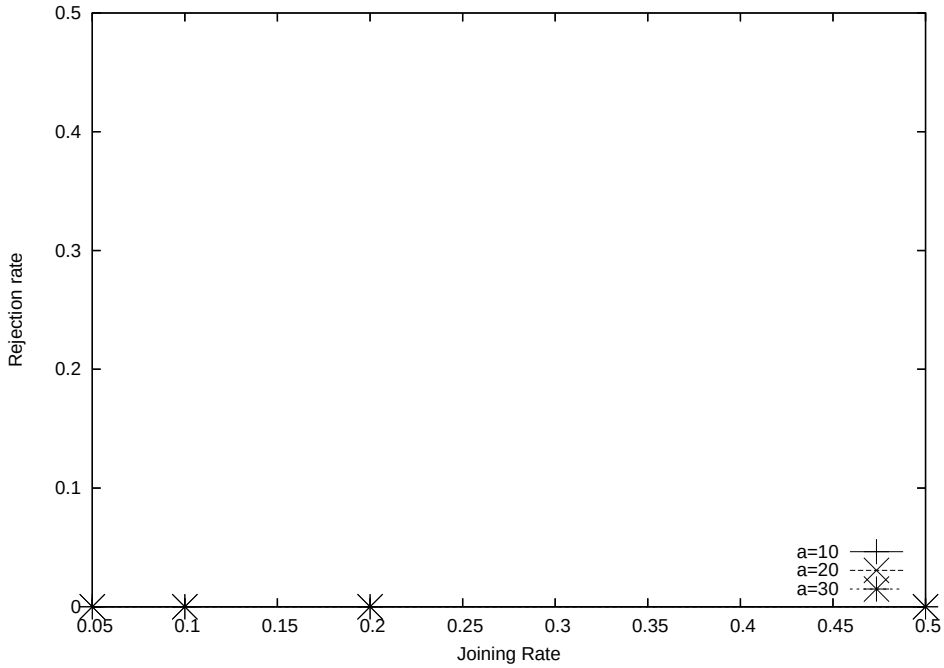
Figure 5.21: Neighborhood size and rejection rate as a function of the join rate, for three different values of a and $f = 1$. Each node is associated with a uniformly random AS in a set of 10 ASes.

to obtain the same value of the neighborhood weakness, if the nodes are more diverse (Section 5.3.2). Since neighborhoods are both smaller and more diverse, incoming nodes are never rejected, because they never find a nodes from their AS already in the neighborhood.

Finally, we repeat the last experiment with $f = 5$ (Figure 5.23). As expected, the neighborhood size increases significantly to about 80. We also notice a small rejection rate (below 5%) for the case $a = 10$. We insist that the neighborhood sizes shown here do not take into account the effect of an attacker that generates a much larger number of nodes within her controlled ASes. In the example of $f = 5$ and $a = 30$, the neighborhood size could grow to more than 210, because, for each of 5 ASes, the attacker would be able to place 30 nodes per neighborhood, up from the average of $80/25 = 3.2$ nodes per AS observed in Figure 5.23.

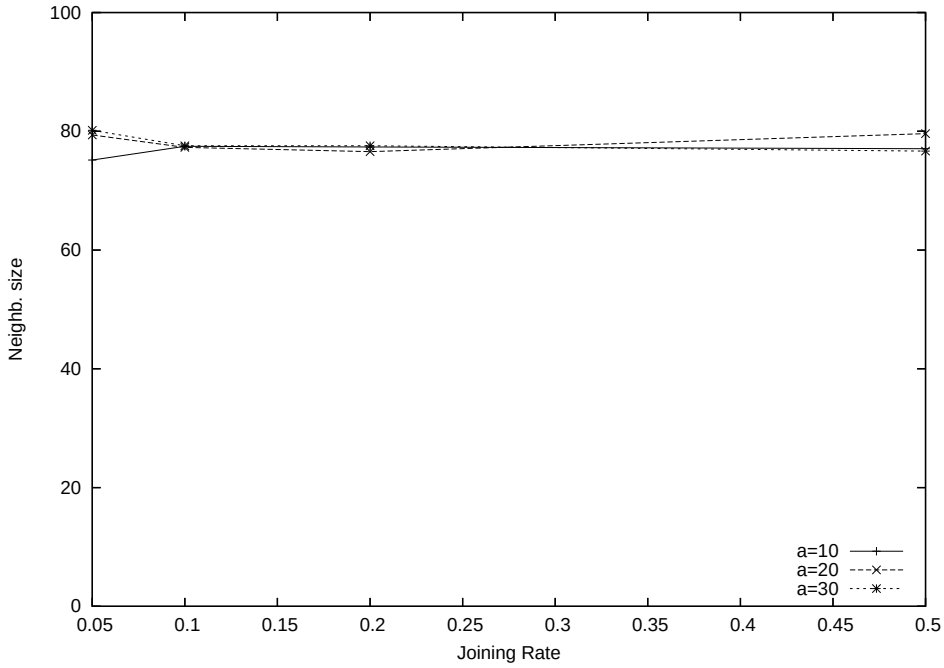


(a) Neighborhood size

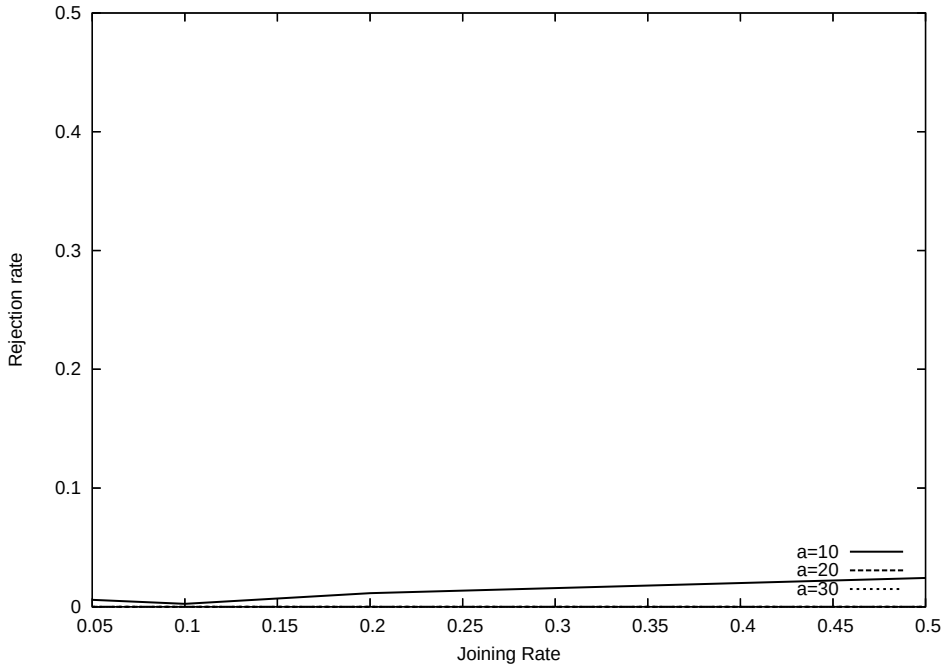


(b) Rejection rate

Figure 5.22: Neighborhood size and rejection rate as a function of the join rate, for three different values of a and $f = 1$. Each node is associated with a uniformly random AS in a set of 25 ASes.



(a) Neighborhood size



(b) Rejection rate

Figure 5.23: Neighborhood size and rejection rate as a function of the join rate, for three different values of a and $f = 5$. Each node is associated with a uniformly random AS in a set of 25 ASes.

Part II

Techniques for Anonymity

Chapter 6

New Results on Ring Signatures

Ring signatures,[1] first introduced by Rivest, Shamir, and Tauman, enable a user to sign a message so that a *ring* of possible signers (of which the user is a member) is identified, without revealing exactly *which member* of that ring actually generated the signature. In contrast to group signatures, ring signatures are completely "ad-hoc" and do not require any central authority or coordination among the various users (indeed, users do not even need to be aware of each other); furthermore, ring signature schemes grant users fine-grained control over the level of anonymity associated with any particular signature.

This chapter has two main areas of focus. First, we examine previous definitions of security for ring signature schemes and suggest that most of these prior definitions are too weak, in the sense that they do not take into account certain realistic attacks. We propose new definitions of anonymity and unforgeability which address these threats, and give separation results proving that our new notions are strictly stronger than previous ones. Second, we show the first constructions of ring signature schemes in the standard model. One scheme is based on generic assumptions and satisfies our strongest definitions of security. Two additional schemes are more efficient, but achieve weaker security guarantees and more limited functionality.

[1] The work presented in this chapter was published as [BKM06].

6.1 Introduction

Ring signatures enable a user to sign a message so that a "ring" of possible signers (of which the user is a member) is identified, without revealing exactly which member of that ring actually generated the signature. This notion was first formally introduced by Rivest, Shamir, and Tauman [RST], and ring signatures — along with the related notion of ring/ad-hoc identification schemes — have been studied extensively since then [BSS, Nao, AOS, ZK, BGLS, HS, DKNS, XZF04, LWW, AHR05]. Ring signatures are related, but incomparable, to the notion of group signatures [CvH]. On the one hand, group signatures have the additional feature that the anonymity of a signer can be revoked (i.e., the signer can be traced) by a designated group manager. On the other hand, ring signatures allow greater flexibility: no centralized group manager or coordination among the various users is required (indeed, users may be unaware of each other at the time they generate their public keys), rings may be formed completely "on-the-fly" and in an ad-hoc manner, and users are given fine-grained control over the level of anonymity associated with any particular signature (via selection of an appropriate ring).

Ring signatures naturally lend themselves to a variety of applications which have been suggested already in previous work (see especially [RST, Nao, DKNS, AHR05]). The original motivation was to allow secrets to be leaked anonymously. Here, for example, a high-ranking government official can sign information with respect to the ring of *all* similarly high-ranking officials; the information can then be verified as coming from *someone* reputable without exposing the actual signer. Ring

signatures can also be used to provide a member of a certain class of users access to a particular resource without explicitly identifying this member; note that there may be cases when third-party verifiability is required (e.g., to prove that the resource has been accessed) and so ring signatures, rather than ad-hoc identification schemes, are needed. Finally, we mention the application to designated-verifier signatures [JSI] especially in the context of e-mail. Here, ring signatures enable the sender of an e-mail to sign the message with respect to the ring containing the sender and the receiver; the receiver is then assured that the e-mail originated from the sender but cannot prove this to any third party. We remark that for this latter application it is sufficient to use a ring signature scheme which supports only rings of size two. See also [CKP] for another proposed application of ring signatures which support only rings of size two.

6.1.1 Our Contributions in Relation to Previous Work

This chapter focuses on both definitions and constructions. We summarize our results in each of these areas, and relate them to prior work.

Definitions of security. Prior work on ring signature/identification schemes provides definitions of security that are either rather informal or seem (to us) unnaturally weak, in that they do not address what seem to be valid security concerns. One example is the failure to consider the possibility of *adversarially-chosen* public keys. Specifically, both the anonymity and unforgeability definitions in most prior work assume that honest users always sign with respect to rings consisting entirely

of *honestly-generated* public keys; no security is provided if users sign with respect to a ring containing even one adversarially-generated public key. Clearly, however, a scheme which is not secure in the latter case is of limited use; this is especially true since rings are constructed in an ad-hoc fashion using keys of (possibly unknown) users which are not validated as being correctly constructed by any central authority. We formalize security against such attacks (as well as others), and show separation results proving that our definitions are strictly stronger than those considered in previous work. In addition to the new, strong definitions we present, the *hierarchy* of definitions we give is useful for characterizing the security of ring signature constructions.

Constructions. We present three ring signature schemes which are provably secure in the standard model. We stress that these are the *first* such constructions, as all previous constructions of which we are aware rely on random oracles/ideal ciphers.[2] It is worth remarking that ring identification schemes are somewhat easier to construct (using, e.g., techniques from [CDS]); ring signatures can then easily be derived from such schemes using the Fiat-Shamir methodology in the random oracle model [FS]. This approach, however, is no longer viable (at least, based on our current understanding) when working in the standard model.

[2] Although Xu, Zhang, and Feng [XZF04] claim a ring signature scheme in the standard model based on specific assumptions, their proof was later found to be flawed (personal communication from J. Xu, March 2005). Concurrently to our work, Chow, Liu and Yuen [CLY05] show a ring signature scheme that they prove secure in the standard model (for rings of *constant* size) based on a new number-theoretic assumption.

Our first construction is based on generic assumptions, and satisfies the strongest definitions of anonymity and unforgeability considered here. This construction is inspired by the generic construction of group signatures due to Bellare, et al. [BMW] and, indeed, the constructions share some similarities at a high level. However, a number of subtleties arise in our context that do not arise in the context of group signatures, and the construction given in [BMW] does not immediately lend itself to a ring signature scheme. Two issues in particular that we need to deal with are the fact that we have no central group manager to issue "certificates" as in [BMW], and that we additionally need to take into account the possibility of adversarially-generated public keys as discussed earlier (this is not a concern in [BMW] where there is only a single group public key published by a (semi-)trusted group manager).

Our other two constructions are more efficient than the first, but rely on specific number-theoretic assumptions. Furthermore, they provide more limited functionality and security guarantees than our first construction; most limiting is that they only support rings of size two. Such schemes are still useful for certain applications (as discussed earlier); furthermore, constructing an efficient 2-user ring signature scheme without random oracles is still difficult, as we do not have the Fiat-Shamir methodology available in our toolbox. These two schemes are based, respectively, on the recent (standard) signature schemes of Waters [Wat] and Camenisch and Lysyanskaya [CL].

6.2 Preliminaries

We use the standard definitions of public-key encryption schemes and semantic security, signature schemes and existential unforgeability under adaptive chosen-message attacks, and computational indistinguishability. In this chapter we will assume public-key encryption schemes for which, with all but negligible probability over (pk, sk) generated at random using the specified key generation algorithm, $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(M)) = M$ holds with probability 1.

We will also use the notion of a *ZAP*, which is a 2-round, public-coin, witness-indistinguishable proof system for any language in $\mathcal{NP}$ (the formal definition is given in Section 6.7). ZAPs were introduced by Dwork and Naor [DN00], who show that ZAPs can be constructed based on any non-interactive zero-knowledge proof system; the latter, in turn, can be constructed based on trapdoor permutations [FLS99]. For notational purposes, we represent a ZAP by a triple $(\ell, \mathcal{P}, \mathcal{V})$ such that (1) the initial message r from the verifier has length $\ell(k)$ (where k is the security parameter); (2) the prover $\mathcal{P}$, on input the verifier-message r, statement x, and witness w, outputs $\pi \leftarrow \mathcal{P}_r(x, w)$; finally, (3) $\mathcal{V}_r(x, \pi)$ outputs 1 or 0, indicating acceptance or rejection of the proof.

6.3 Definitions

We begin by presenting the functional definition of a ring signature scheme. We refer to an ordered list $R = (PK_1, \ldots, PK_n)$ of public keys as a *ring*, and let $R[i] = PK_i$. We will also freely use set notation, and say, e.g., that $PK \in R$ if

there exists an index i such that $R[i] = PK$. We will always assume, without loss of generality, that the keys in a ring are ordered lexicographically.

Definition 6.1 (**Ring signature**). *A* ring signature scheme *is a triple of* PPT *algorithms* (Gen, Sign, Vrfy) *that, respectively, generate keys for a user, sign a message, and verify the signature of a message. Formally:*

- $\mathsf{Gen}(1^k)$*, where* k *is a security parameter, outputs a public key* PK *and secret key* SK*.*
- $\mathsf{Sign}_{s,SK}(M, R)$ *outputs a signature* σ *on the message* M *with respect to the ring* $R = (PK_1, \ldots, PK_n)$*. We assume the following: (1)* $(R[s], SK)$ *is a valid key-pair output by* Gen*; (2)* $|R| \geq 2$ *(since a ring signature scheme is not intended*[3] *to serve as a standard signature scheme); and (3) each*[4] *public key in the ring is distinct.*
- $\mathsf{Vrfy}_R(M, \sigma)$ *verifies a purported signature* σ *on a message* M *with respect to the ring of public keys* R*.*

We require the following completeness condition to hold: for any integer k*, any* $\{(PK_i, SK_i)\}_{i=1}^n$ *output by* $\mathsf{Gen}(1^k)$*, any* $s \in [n]$*, and any* M*, we have* $\mathsf{Vrfy}_R(M, \mathsf{Sign}_{s,SK_s}(M, R)) = 1$ *where* $R = (PK_1, \ldots, PK_n)$*.*

[3]Furthermore, it is easy to modify any ring signature scheme to allow signatures with $|R| = 1$ by including a special key for just that purpose.

[4]This is without loss of generality, since the signer/verifier can simply take the sub-ring of distinct keys in R and correctness is unchanged.

A c-user ring signature scheme *is a variant of the above that only supports rings of fixed size* c *(i.e., the* Sign *and* Vrfy *algorithms only take as input rings* R *for which* $|R| = c$*, and correctness is only required to hold for such rings).*

To improve readability, we will generally omit the input "s" to the signing algorithm (and simply write $\sigma \leftarrow \mathsf{Sign}_{SK}(M, R)$), with the understanding that the signer can determine an index s for which SK is the secret key corresponding to public key $R[s]$. Strictly speaking, there may not be a unique such s when R contains incorrectly-generated keys; in real-world usage of a ring signature scheme, though, a signer will certainly be able to identify their public key.

A ring signature scheme is used as follows: At various times, some collection of users runs the key generation algorithm Gen to generate public and secret keys. We stress that no coordination among these users is assumed or required. When a user with secret key SK wishes to generate an anonymous signature on a message M, he chooses a ring R of public keys which includes his own, computes $\sigma \leftarrow \mathsf{Sign}_{SK}(M, R)$ and outputs (σ, R). (In such a case, we will refer to the holder of SK as the *signer* of the message and to the holders of the other public keys in R as the *non-signers*.) Anyone can now verify that this signature was generated by *someone* holding a key in R by running $\mathsf{Vrfy}_R(M, \sigma)$.

We remark that although our functional definition of a ring signature scheme (cf. Def. 6.1) requires users to generate keys specifically for that purpose (in contrast to the requirements of [AOS, AHR05]), our first construction can be easily modified to work with any ring of users as long as they each have a public key for both

encryption and signing (see Section 6.5).

As discussed in the Introduction, ring signatures must satisfy two independent notions of security: anonymity and unforgeability. There are various ways each of these notions can be defined (and various ways these notions have been defined in the literature); we present our definitions in Sections 6.3.1 and 6.3.2, and compare them in Section 6.4.

6.3.1 Definitions of Anonymity

The anonymity condition requires, informally, that an adversary not be able to tell which member of a ring generated a particular signature.[5] We begin with a basic definition of anonymity which is already stronger than that considered in most previous work in that we give the adversary access to a signing oracle (this results in a stronger definition even in the case of unconditional anonymity).

Definition 6.2 (Basic anonymity). *Given a ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$*, a polynomial* $n(\cdot)$*, and a* PPT *adversary* $\mathcal{A}$*, consider the following game:*

1. *Key pairs* $\{(PK_i, SK_i)\}_{i=1}^{n(k)}$ *are generated using* $\mathsf{Gen}(1^k)$*, and the set of public keys* $S \stackrel{\text{def}}{=} \{PK_i\}_{i=1}^{n(k)}$ *is given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *is given access (throughout the entire game) to an oracle* $\mathsf{OSign}(\cdot,\cdot,\cdot)$ *such that* $\mathsf{OSign}(s, M, R)$ *returns* $\mathsf{Sign}_{SK_s}(M, R)$*, where we require* $R \subseteq S$ *and*

[5] All the anonymity definitions that follow can be phrased in either a *computational* or an *unconditional* sense (where, informally, in the former case anonymity holds for polynomial-time adversaries while in the latter case anonymity holds even for all-powerful adversaries). For simplicity, we only present the computational versions.

$PK_s \in R$.

3. *$\mathcal{A}$ outputs a message M, distinct indices i_0, i_1, and a ring $R \subseteq S$ for which $PK_{i_0}, PK_{i_1} \in R$. A random bit b is chosen, and $\mathcal{A}$ is given the signature $\sigma \leftarrow \mathsf{Sign}_{SK_{i_b}}(M, R)$.*

4. *The adversary outputs a bit b', and succeeds if $b' = b$.*

$(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *achieves* basic anonymity *if, for any* PPT *$\mathcal{A}$ and any polynomial $n(\cdot)$, the success probability of $\mathcal{A}$ in the above game is negligibly close to* $1/2$.

(Some previous papers consider a variant of the above in which the adversary is given a signature computed by a randomly-chosen member of R, and should be unable to guess the actual signer with probability better than $1/|R| + \mathsf{negl}(k)$. A hybrid argument shows that such a variant is equivalent to the above.)

Unfortunately, the above definition of basic anonymity leaves open the possibility of the following attack: (1) an adversary generates public keys in some arbitrary manner (which may possibly depend on the public keys of the honest users), and then (2) a legitimate signer generates a signature with respect to a ring containing some of these adversarially-generated public keys. The definition above offers no protection in this case! This attack, considered also in [Nao] (in a slightly different context) is quite realistic since, by their very nature, ring signatures are intended to be used in settings where there is not necessarily any central authority checking validity of public keys. This motivates the following, stronger definition:

Definition 6.3 (Anonymity w.r.t. adversarially-chosen keys). *Given a ring*

signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$*, a polynomial* $n(\cdot)$*, and a* PPT *adversary* $\mathcal{A}$*, consider the following game:*

1. *As in Definition 6.2.*
2. *As in Definition 6.2, except that we no longer require* $R \subseteq S$*.*
3. *As in Definition 6.2, except that we no longer require* $R \subseteq S$*.*
4. *The adversary outputs a bit* b'*, and succeeds if* $b' = b$*.*

$(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *achieves* anonymity w.r.t. adversarially-chosen keys *if for any* PPT $\mathcal{A}$ *and polynomial* $n(\cdot)$*, the success probability of* $\mathcal{A}$ *in the above game is negligibly close to* $1/2$*.*

The above definition only guarantees anonymity of a particular signature as long as there are at least two honest users in the ring. In some sense this is inherent, since if an honest signer U chooses a ring in which all *other* public keys (i.e., except for the public key of U) are owned by an adversary, then that adversary "knows" that U must be the signer (since the adversary did not generate the signature itself).

A weaker requirement one might consider when the signer U is the only honest user in the ring is that the other members of the ring should be unable to *prove* to a third party that U generated the signature (we call this an *attribution attack*). Preventing such an attack in general seems to require the involvement of a trusted party (or at least a common random string), something we would like to avoid. We instead define a slightly weaker notion which, informally, can be viewed as offering honest user U some protection against attribution attacks as long as at least one

other user U' in the ring was honest *at the time U' generated his public key.* However, we allow this user U', as well as all other honest users in the ring (except for U), to later collude with an adversary by revealing their secret keys in an attempt to attribute the signature to U.[6] (Actually, we even allow these users to reveal the *randomness*[7] used to generate their secret keys.) Note that security in such a setting also ensures some measure of security in case secret keys are exposed or stolen.

In addition to the above, we consider also the stronger variant in which the secret keys of *all* honest users in the ring (i.e., including U) are exposed. This parallels (in fact, is stronger than) the anonymity definition given by Bellare, et al. in the context of group signatures [BMW]. For simplicity, we also protect against adversarially-chosen keys, although one could consider the weaker definition which does not.

Definition 6.4 (**Anonymity against attribution attacks/full key exposure).** *Given* (Gen, Sign, Vrfy), $n(\cdot)$, *and* $\mathcal{A}$ *as in Definition 6.3, consider the following game:*

1. *For* $i = 1$ *to* $n(k)$, *generate* $(PK_i, SK_i) \leftarrow \mathsf{Gen}(1^k; \omega_i)$ *for randomly-chosen* ω_i. *Give to* $\mathcal{A}$ *the set of public keys* $\{PK_i\}_{i=1}^{n(k)}$.

2. *The adversary* $\mathcal{A}$ *is given access to a signing oracle as in Definition 6.3.* $\mathcal{A}$ *is*

[6]The idea is that everyone in the ring is trying to "frame" U, but U is (naturally) refusing to divulge her secret key. Although this itself might arouse suspicion, the point is that it still cannot be proved — in court, say — that U was the signer.

[7]This ensures security when erasure cannot be guaranteed, or when it cannot be guaranteed that all users will comply with the directive to erase their random coins.

also given access to a $\mathsf{Corrupt}(\cdot)$ *oracle that, on input* i*, returns* ω_i*.*

3. $\mathcal{A}$ *outputs a message* M*, distinct indices* i_0, i_1*, and a ring* R *for which* $PK_{i_0}, PK_{i_1} \in R$*. A random bit* b *is chosen and* $\mathcal{A}$ *is given* $\sigma \leftarrow \mathsf{Sign}_{SK_{i_b}}(M, R)$*.*

4. *The adversary outputs a bit* b'*, and succeeds if* $b' = b$ *and* $i_0 \notin C$*, where* C *is the set of queries to the corruption oracle.*

$(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *achieves* anonymity against attribution attacks *if, for any* PPT $\mathcal{A}$ *and polynomial* $n(\cdot)$*, the success probability of* $\mathcal{A}$ *in the above game is at most* $1/2 + \mathsf{negl}(k)$*. If we drop the requirement* $i_0 \notin C$*, then we say* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *achieves* anonymity against full key exposure.

Linkability. Another desideratum of a ring signature scheme is that it be *unlinkable*; that is, it should be infeasible to determine whether two signatures (possibly generated with respect to different rings) were generated by the same signer. As in [BMW], all our definitions imply (appropriate variants of) unlinkability.

6.3.2 Definitions of Unforgeability

The intuitive notion of unforgeability is, as usual, that an adversary should be unable to output (R, M, σ) such that $\mathsf{Vrfy}_R(M, \sigma) = 1$ unless either (1) one of the public keys in R was chosen by the adversary, or (2) a user whose public key is in R explicitly signed M previously (with respect to the same ring R). Some subtleties arise, however, when defining a chosen-message attack on the scheme. Many previous works (e.g., [RST]), assume a definition like the following:

Definition 6.5 (**Unforgeability against fixed-ring attacks).** *A ring signature scheme* (Gen, Sign, Vrfy) *is* unforgeable against fixed-ring attacks *if for any* PPT *adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$*, the probability that* $\mathcal{A}$ *succeeds in the following game is negligible:*

1. *Key pairs* $\{(PK_i, SK_i)\}_{i=1}^{n(k)}$ *are generated using* $\mathsf{Gen}(1^k)$*, and the set of public keys* $R \stackrel{\text{def}}{=} \{PK_i\}_{i=1}^{n(k)}$ *is given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *is given access to a* signing oracle $\mathsf{OSign}(\cdot,\cdot)$*, where* $\mathsf{OSign}(s, M)$ *outputs* $\mathsf{Sign}_{SK_s}(M, R)$*.*

3. $\mathcal{A}$ *outputs* (M^*, σ^*)*, and succeeds if* $\mathsf{Vrfy}_R(M^*, \sigma^*) = 1$ *and also* $\mathcal{A}$ *never made a query of the form* $\mathsf{OSign}(\star, M^*)$*.*

Note that not only is $\mathcal{A}$ restricted to making signing queries with respect to the *full* ring R, but its forgery is required to verify with respect to R as well. The following stronger, and more natural, definition was used in, e.g., [AOS]:

Definition 6.6 (**Unforgeability against chosen-subring attacks).** *A ring signature scheme* (Gen, Sign, Vrfy) *is* unforgeable against chosen-subring attacks *if for any* PPT *adversary* $\mathcal{A}$ *and for any polynomial* $n(\cdot)$*, the probability that* $\mathcal{A}$ *succeeds in the following game is negligible:*

1. *Key pairs* $\{(PK_i, SK_i)\}_{i=1}^{n(k)}$ *are generated using* $\mathsf{Gen}(1^k)$*, and the set of public keys* $S \stackrel{\text{def}}{=} \{PK_i\}_{i=1}^{n(k)}$ *is given to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *is given access to a* signing oracle $\mathsf{OSign}(\cdot,\cdot,\cdot)$*, where* $\mathsf{OSign}(s, M, R)$ *outputs* $\mathsf{Sign}_{SK_s}(M, R)$ *and we require that* $R \subseteq S$ *and* $PK_s \in R$*.*

3. *$\mathcal{A}$ outputs (R^*, M^*, σ^*), and succeeds if $R^* \subseteq S$, $\mathsf{Vrfy}_{R^*}(M^*, \sigma^*) = 1$, and $\mathcal{A}$ never queried $(\star, M^*, R^*)$ to its signing oracle.*

While the above definition is an improvement, it still leaves open the possibility of an attack whereby honest users are "tricked" into generating signatures using rings containing adversarially-generated public keys. (Such an attack was also previously suggested by [Nao, LWW].) The following definition takes this into account as well as (for completeness) an adversary who adaptively corrupts honest participants and obtains their secret keys. Since either of these attacks may be viewed as the outcome of corrupting an "insider," we use this terminology.[8]

Definition 6.7 (Unforgeability w.r.t. insider corruption). *A ring signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* unforgeable w.r.t. insider corruption *if for any* PPT *adversary $\mathcal{A}$ and for any polynomial $n(\cdot)$, the probability that $\mathcal{A}$ succeeds in the following game is negligible:*

1. *Key pairs $\{(PK_i, SK_i)\}_{i=1}^{n(k)}$ are generated using $\mathsf{Gen}(1^k)$, and the set of public keys $S \stackrel{\text{def}}{=} \{PK_i\}_{i=1}^{n(k)}$ is given to $\mathcal{A}$.*

2. *$\mathcal{A}$ is given access to a* signing oracle $\mathsf{OSign}(\cdot, \cdot, \cdot)$, *where $\mathsf{OSign}(s, M, R)$ outputs $\mathsf{Sign}_{SK_s}(M, R)$ and we require that $PK_s \in R$.*

3. *$\mathcal{A}$ is also given access to a* corrupt oracle $\mathsf{Corrupt}(\cdot)$, *where $\mathsf{Corrupt}(i)$ outputs SK_i.*

[8] We are aware that, technically speaking, there are not really any "insiders" in the context of ring signatures.

4. $\mathcal{A}$ outputs (R^*, M^*, σ^*), and succeeds if $\mathsf{Vrfy}_{R^*}(M^*, \sigma^*) = 1$, $\mathcal{A}$ never queried $(\star, M^*, R^*)$, and $R^* \subseteq S \setminus C$, where C is the set of corrupted users.

We remark that Herranz [Her05] considers, albeit informally, a definition intermediate between our Definitions 6.6 and 6.7 in which corruptions of honest players are allowed but adversarially-chosen public keys are not explicitly mentioned.

6.4 Separations Between the Security Definitions

In the previous section, we presented various definitions of anonymity and unforgeability. Here, we show that these definitions are in fact distinct, in the sense that there exist (under certain assumptions) schemes satisfying a weaker definition but not a stronger one. First, we show separations for the definitions of anonymity, considering in each case a scheme simultaneously satisfying the strongest definition of unforgeability. (Proofs for the claims presented in this section are given in Section 6.8.1.)

Claim 6.8. *If there exists a scheme which achieves* **basic** *anonymity and is unforgeable w.r.t. insider corruption, then there exists a scheme which achieves these same properties but which is* **not** *anonymous w.r.t.* **adversarially-chosen keys**.

Claim 6.9. *If there exists a scheme which is anonymous w.r.t.* **adversarially-chosen keys** *and is unforgeable w.r.t. insider corruption, then there exists a scheme which achieves these same properties but which is* **not** *anonymous against* **attribution attacks**.

We also show separations for the definitions of unforgeability, considering now schemes which simultaneously achieve the strongest definition of anonymity:

Claim 6.10. *If there exists a scheme which is anonymous against full key exposure and unforgeable w.r.t. insider corruption, then there exists a scheme which is anonymous against full key exposure and unforgeable against* **fixed-ring attacks**, *but* **not** *unforgeable against* **chosen-subring attacks**.

In contrast to the rest of the claims, the assumption in the above claim is not minimal. We remark that the scheme of [HS] serves as a *natural* example of a scheme that is unforgeable against fixed-ring attacks, but which is **not** unforgeable against chosen-subring attacks (in the random oracle model); this was subsequently fixed in [Her05]. See Section 6.8.2.

Claim 6.11. *If there exists a scheme which is anonymous against full key exposure and unforgeable against* **chosen-subring attacks**, *then there exists a scheme achieving these same properties which is* **not** *unforgeable w.r.t.* **insider corruption**.

6.5 Ring Signatures Based on General Assumptions

We now describe our construction of a ring signature scheme that satisfies the strongest of our proposed definitions, and is based on general assumptions. In what follows, we let $(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec})$ be a semantically-secure public-key encryption scheme, let $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ be a (standard) signature scheme, and let $(\ell, \mathcal{P}, \mathcal{V})$ be a ZAP (for an $\mathcal{NP}$-language that will become clear once we describe the scheme). We denote by $C^* \leftarrow \mathsf{Enc}^*_{R_E}(m)$ the probabilistic algorithm that takes as input a set

of encryption public keys $R_E = \{pk_{E,1}, \ldots, pk_{E,n}\}$ and a message m, and does the following: it first chooses random $s_1, \ldots, s_{n-1} \in \{0,1\}^{|m|}$ and then outputs:

$$C^* = \left(\mathsf{Enc}_{pk_{E,1}}(s_1), \mathsf{Enc}_{pk_{E,2}}(s_2), \cdots, \mathsf{Enc}_{pk_{E,n-1}}(s_{n-1}), \mathsf{Enc}_{pk_{E,n}}(m \oplus \bigoplus_{j=1}^{n-1} s_j) \right).$$

Note that, informally, encryption using Enc^* is semantically secure as long as at least one of the corresponding secret keys is unknown.

The idea of our construction is the following. Each user has an encryption key pair (pk_E, sk_E) and a standard signature key pair (pk_S, sk_S). To generate a ring signature with respect to a ring R of n users, the signer produces a standard signature σ' with her signing key. Next, the signer produces two ciphertexts C_0^*, C_1^* using the Enc^* algorithm and the set R_E of all the *encryption* public keys in the ring; one of these ciphertexts will be an encryption of σ'. Finally, the signer produces a proof π, using the ZAP, that one of the ciphertexts is an encryption of a valid signature on the message with respect to the signature public key of one of the ring members.

Toward a formal description, let L denote the $\mathcal{NP}$ language:

$$\left\{ (pk_S, M, R_E, C^*) : \exists \sigma, \omega \text{ s.t. } C^* = \mathsf{Enc}^*_{R_E}(\sigma; \omega) \bigwedge \mathsf{Vrfy}'_{pk_S}(M, \sigma) = 1 \right\};$$

i.e., $(pk_S, M, R_E, C^*) \in L$ iff C^* is an encryption (using $\mathsf{Enc}^*_{R_E}$) of a valid signature of M with respect to the verification key pk_S. We now give the details of our construction, which is specified by the key-generation algorithm Gen, the ring signing algorithm Sign, and the ring verification algorithm Vrfy:[9]

[9] In our conference paper [BKM06], we presented a less efficient version of the scheme, in which a ring signature contains n ciphertexts, as opposed to two.

$\underline{\mathsf{Gen}(1^k)}$:

1. Generate signing key pair $(pk_S, sk_S) \leftarrow \mathsf{Gen}'(1^k)$.
2. Generate encryption key pair $(pk_E, sk_E) \leftarrow \mathsf{Gen}(1^k)$ and erase sk_E.
3. Choose an initial ZAP message $r \leftarrow \{0,1\}^{\ell(k)}$.
4. Output the public key $PK = (pk_S, pk_E, r)$, and the secret key $SK = sk_S$.

$\underline{\mathsf{Sign}_{i^*, SK_{i^*}}(M, (PK_1, \ldots, PK_n))}$:

1. Parse each PK_i as $(pk_{S,i}, pk_{E,i}, r_i)$, and parse SK_{i^*} as sk_{S,i^*}. Set $R_E := \{pk_{E,1}, \ldots, pk_{E,n}\}$.
2. Set $M^* := M \,|\, PK_1 \,|\, \cdots \,|\, PK_n$, where "|" denotes concatenation. Compute the signature $\sigma'_{i^*} \leftarrow \mathsf{Sign}'_{sk_{S,i^*}}(M^*)$.
3. Choose random coins ω_0, ω_1, choose a random bit β and: (1) compute $C^*_\beta = \mathsf{Enc}^*_{R_E}(\sigma'_{i^*}; \omega_\beta)$ and (2) compute[10] $C^*_{1-\beta} = \mathsf{Enc}^*_{R_E}(0^{|\sigma'_{i^*}|}; \omega_{1-\beta})$.
4. For $i \in [n]$ and $j \in \{0,1\}$, let $x_{i,j}$ denote the statement: "$(pk_{S,i}, M^*, R_E, C^*_j) \in L$", and let $x := \bigvee_{i=1}^{n} \bigvee_{j=0}^{1} x_{i,j}$. Compute the proof $\pi \leftarrow \mathcal{P}_{r_1}(x, (\sigma'_{i^*}, \omega_\beta))$.
5. The signature is $\sigma = (C^*_0, C^*_1, \pi)$.

$\underline{\mathsf{Vrfy}_{PK_1,\ldots,PK_n}(M, \sigma)}$

1. Parse each PK_i as $(pk_{S,i}, pk_{E,i}, r_i)$. Set $M^* := M \,|\, PK_1 \,|\, \cdots \,|\, PK_n$ and $R_E := \{pk_{E,1}, \ldots, pk_{E,n}\}$. Parse σ as (C^*_0, C^*_1, π).

[10] We assume for simplicity that valid signatures w.r.t. the public keys $\{pk_{S,i}\}_{i \neq i^*}$ always have the same length as valid signatures w.r.t. pk_{S,i^*}. The construction can be adapted when this is not the case.

2. For $i \in [n]$ and $j \in \{0,1\}$, let $x_{i,j}$ denote the statement: "$(pk_{S,i}, M^*, R_E, C^*_j) \in L$", and let $x := \bigvee_{i=1}^{n} \bigvee_{j=0}^{1} x_{i,j}$.

3. Output $\mathcal{V}_{r_1}(x, \pi)$.

It is easy to see that the scheme above satisfies the functional definition of a ring signature scheme (recall that the $\{PK_i\}$ in a ring are always ordered lexicographically). We now prove that the scheme satisfies strong notions of anonymity and unforgeability:

Theorem 6.12. *If encryption scheme* $(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure, signature scheme* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ *is existentially unforgeable under adaptive chosen-message attacks, and* $(\ell, \mathcal{P}, \mathcal{V})$ *is a ZAP for the language* $L' = \{(x_1, \ldots, x_n) : \exists i : x_i \in L\}$*, then the above ring signature scheme is (computationally) anonymous against attribution attacks, and unforgeable w.r.t. insider corruption.*

The proof is given in Section 6.9.1.

Extension. The scheme above can also be used (with a few easy modifications) in a situation where some users in the ring have not generated a key pair according to Gen, as long as (1) every ring member has a public key both for encryption and for signing (these keys may be associated with different schemes), and (2) at least one of the members has included a sufficiently-long random string in his public key. Thus, a *single* user who establishes a public key for a ring signature scheme suffices to provide anonymity for everyone. This also provides a way to include "oblivious" users in the signing ring [AOS, AHR05].

Achieving a stronger anonymity guarantee. The above scheme is not secure against full key exposure, and essential to our proof of anonymity is that the adversary not be given the random coins used to generate *all* (honest) ring signature keys.[11] (If the adversary gets all sets of random coins, it can decrypt ciphertexts encrypted using $\mathsf{Enc}^*_{R_E}$ for any ring of honest users R and thereby determine the true signer of a message.) It is possible to achieve anonymity against full key exposure using an enhanced form of encryption for which, informally, there exists an "oblivious" way to generate a public key without generating a corresponding secret key. This notion, introduced by Damgård and Nielsen [DN], can be viewed as a generalization of *dense cryptosystems* in which the public key is required to be a uniformly distributed string (in particular, dense cryptosystems satisfy the definition below). We review the formal definition here.

Definition 6.13. *An* oblivious key generator *for the public-key encryption scheme* $(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a pair of* PPT *algorithms* $(\mathsf{OblEGen}, \mathsf{OblRand})$ *such that:*

- $\mathsf{OblEGen}$*, on input* 1^k *and random coins* $\omega \in \{0,1\}^{n(k)}$*, outputs a key* pk*;*
- $\mathsf{OblRand}$*, on input a key* pk*, outputs a string* ω*;*

and the following distribution ensembles are computationally indistinguishable:

$$\left\{\omega \leftarrow \{0,1\}^{n(k)} : (\omega, \mathsf{OblEGen}(1^k; \omega))\right\}$$

[11] We remark that anonymity still holds if the adversary is given all *secret keys* (but not the randomness used to generate all secret keys). This is because the decryption key sk_E is erased, and not included in SK.

and

$$\left\{(pk, sk) \leftarrow \mathsf{EGen}(1^k); \omega \leftarrow \mathsf{OblRand}(pk) : (\omega, pk)\right\} .$$

Note that if $(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec})$ is semantically secure, then (informally speaking) it is also semantically secure to encrypt messages using a public key pk generated by $\mathsf{OblEGen}$, even if the adversary has the random coins used by $\mathsf{OblEGen}$ in generating pk. We remark for completeness that the El Gamal encryption scheme (over the group of quadratic residues modulo a prime) is an example of a scheme having an oblivious key generator.

Given the above, we adapt our construction in the natural way: specifically, the Gen algorithm is changed so that instead of generating pk_E using EGen (and then erasing the secret key sk_E and the random coins used), we now generate pk_E using $\mathsf{OblEGen}$. Adapting the proof of Theorem 6.12, we can easily show:

Theorem 6.14. *Under the assumptions of Theorem 6.12 and assuming* (EGen, Enc, Dec) *has an oblivious key generator, the modified ring signature scheme described above is (computationally) anonymous against full key exposure, and unforgeable w.r.t. insider corruption.*

The proof is given in Section 6.9.2.

6.6 Efficient Two-User Ring Signature Schemes

In this section, we present more efficient constructions of two-user ring signature schemes based on specific assumptions. Our first scheme is based on the

(standard) signature scheme constructed by Waters [Wat], whereas the second is based on the Camenisch-Lysyanskaya signature scheme [CL].

6.6.1 The Waters Scheme

We briefly review the Waters signature scheme. Let $\mathbb{G}, \mathbb{G}_1$ be groups of prime order q such that there exists an efficiently computable bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$. We assume that $q, \mathbb{G}, \mathbb{G}_1, \hat{e}$, and a generator $g \in \mathbb{G}$ are publicly known. The Waters signature scheme for messages of length n is defined as follows:

Key Generation. Choose $\alpha \leftarrow \mathbb{Z}_q$ and set $g_1 = g^\alpha$. Additionally choose random elements $h, u', u_1, \ldots, u_n \leftarrow \mathbb{G}$. The public key is $(g_1, h, u', u_1, \ldots, u_n)$ and the secret key is h^α.

Signing. To sign the n-bit message M, first compute $w = u' \cdot \prod_{i:M_i=1} u_i$. Then choose random $r \leftarrow \mathbb{Z}_q$ and output the signature $\sigma = (h^\alpha \cdot w^r,\ g^r)$.

Verification. To verify the signature (A, B) on message M with respect to public key $(g_1,\ h,\ u',\ u_1,\ \ldots,\ u_n)$, compute $w = u' \cdot \prod_{i:M_i=1} u_i$ and then check whether $\hat{e}(g_1, h) \cdot \hat{e}(B, w) \stackrel{?}{=} \hat{e}(A, g)$.

6.6.2 A 2-User Ring Signature Scheme

The main observation we make with regard to the above scheme is the following: element h is arbitrary, and only knowledge of h^α is needed to sign. So, we can dispense with including h in the public key altogether; instead, a user U with secret α and the value $g_1 = g^\alpha$ in his public key will use as his "h-value" the value

$\bar{g}_1$ contained in the public key of a *second* user $\bar{U}$. This provides anonymity since $\bar{U}$ could *also* have computed the same value $(\bar{g}_1)^\alpha$ using the secret value $\bar{\alpha} = \log_g \bar{g}_1$ known to him (because $\bar{g}_1^\alpha = g_1^{\bar{\alpha}}$). We now proceed with the details.

Key Generation. Choose $\alpha \leftarrow \mathbb{Z}_q$ and set $g_1 = g^\alpha$. Additionally choose random elements $u', u_1, \ldots, u_n \leftarrow \mathbb{G}$. The public key is $(g_1,\ u',\ u_1,\ \ldots,\ u_n)$ and the secret key is α. (We again assume that $q, \mathbb{G}, \mathbb{G}_1, \hat{e}$, and g are system-wide parameters.)

Ring Signing. To sign message $M \in \{0,1\}^n$ with respect to the ring $R = \{PK, \overline{PK}\}$ using secret key α (where we assume without loss of generality that α is the secret corresponding to PK), proceed as follows: parse PK as $(g_1,\ u',\ u_1,\ \ldots,\ u_n)$ and $\overline{PK}$ as $(\bar{g}_1,\ \bar{u}',\ \bar{u}_1,\ \ldots,\ \bar{u}_n)$, and compute $w = u' \cdot \prod_{i:M_i=1} u_i$ and $\bar{w} = \bar{u}' \cdot \prod_{i:M_i=1} \bar{u}_i$. Then choose random $r \leftarrow \mathbb{Z}_q$ and output the signature

$$\sigma = (\bar{g}_1^\alpha \cdot (w\bar{w})^r,\ g^r)\ .$$

Ring Verification. To verify the signature (A, B) on message M with respect to the ring $R = \{PK, \overline{PK}\}$ (parsed as above), compute $w = u' \cdot \prod_{i:M_i=1} u_i$ and $\bar{w} = \bar{u}' \cdot \prod_{i:M_i=1} \bar{u}_i$ and then check whether $\hat{e}(g_1, \bar{g}_1) \cdot \hat{e}(B, (w\bar{w})) \stackrel{?}{=} \hat{e}(A, g)$.

It is not hard to see that correctness holds. We prove the following regarding the above scheme:

Theorem 6.15. *Assume the Waters signature scheme is existentially unforgeable*[12] *under adaptive chosen message attack. Then the 2-user ring signature scheme described above is unconditionally anonymous against full key exposure, and unforgeable against chosen-subring attacks.*

[12] This holds [Wat] under the computational Diffie-Hellman assumption in $\mathbb{G}$.

Proof. Unconditional anonymity against full key exposure follows easily from the observation made earlier: namely, that only the value $\bar{g}_1^{\alpha} = g_1^{\bar{\alpha}}$ (where $\bar{\alpha} \stackrel{\text{def}}{=} \log_g \bar{g}_1$) is needed to sign, and either of the two (honest) parties can compute this value.

We now prove that the scheme satisfies Definition 6.6. We do this by showing how an adversary $\mathcal{A}$ that forges a signature with respect to the ring signature scheme with non-negligible probability can be used to construct an adversary $\hat{\mathcal{A}}$ that forges a signature with respect to the Waters signature scheme (in the standard sense) with the same probability. For simplicity in the proof, we assume that $\mathcal{A}$ only ever sees the public keys of two users, requests all signatures to be signed with respect to the ring R containing these two users, and forges a signature with respect to that same ring R. By a hybrid argument, it can be shown that (for this scheme) this is equivalent to the more general case when $\mathcal{A}$ may see multiple public keys, request signatures with respect to various (different) 2-user subsets, and then output a forgery with respect to any 2-user subset of its choice.

Construct $\hat{\mathcal{A}}$ as follows: $\hat{\mathcal{A}}$ is given the public key ($\hat{g}_1$, $\hat{h}$, $\hat{u}'$, $\hat{u}_1$, …, $\hat{u}_n$) of an instance of the Waters scheme. $\hat{\mathcal{A}}$ constructs two user public keys as follows: first, it sets $g_1 = \hat{g}_1$ and $\bar{g}_1 = \hat{h}$. Then, it chooses random $u', u_1, \ldots, u_n \leftarrow \mathbb{G}$ and sets $\bar{u}' = \hat{u}'/u'$ and $\bar{u}_i = \hat{u}_i/u_i$ for all i. It gives to $\mathcal{A}$ the public keys (g_1, u', u_1, …, u_n) and ($\bar{g}_1$, $\bar{u}'$, $\bar{u}_1$, …, $\bar{u}_n$). Note that both public keys have the appropriate distribution. When $\mathcal{A}$ requests a ring signature on a message M with respect to the ring R containing these two public keys, $\hat{\mathcal{A}}$ requests a signature on M from its signing oracle, obtains in return a signature (A, B), and gives this signature to $\mathcal{A}$.

Note that this is indeed a perfect simulation, since

$$\left(\hat{h}^{\log_g \hat{g}_1} \cdot \left(\hat{u}' \prod_{i:M_i=1} \hat{u}_i\right)^r,\ g^r\right) \ = \ \left(\bar{g}_1^{\log_g g_1} \cdot \left(u'\bar{u}' \prod_{i:M_i=1} u_i\bar{u}_i\right)^r,\ g^r\right),$$

which is an appropriately-distributed ring signature with respect to the public keys given to $\mathcal{A}$.

When $\mathcal{A}$ outputs a forgery (A^*, B^*) on a message M^*, this same forgery is output by $\hat{\mathcal{A}}$. Note that $\hat{\mathcal{A}}$ outputs a valid forgery whenever $\mathcal{A}$ does, since

$$\hat{e}(g_1, \bar{g}_1) \cdot \hat{e}\left(B^*, (u'\bar{u}' \textstyle\prod_{i:M_i^*=1} u_i\bar{u}_i)\right) = \hat{e}(A^*, g)$$

implies

$$\hat{e}(\hat{g}_1, \hat{h}) \cdot \hat{e}\left(B^*, (\hat{u}' \textstyle\prod_{i:M_i^*=1} \hat{u}_i)\right) = \hat{e}(A^*, g)\,.$$

We conclude that $\hat{\mathcal{A}}$ outputs a forgery with the same probability as $\mathcal{A}$. Since, by assumption, the Waters scheme is secure, this completes the proof. □

We remark that the security reduction to the Waters scheme is tight.

An efficiency improvement. A (slightly) more efficient variant of the above scheme is also possible. Key generation is the same as before, except that an additional, random *identifier* $I \in \{0,1\}^k$ is also chosen and included in the public key. Let $<_{\text{lex}}$ denote lexicographic order. To sign message $M \in \{0,1\}^n$ with respect to the ring $R = \{PK, \overline{PK}\}$, first parse PK as $(I,\ g_1,\ u',\ u_1,\ \ldots,\ u_n)$ and $\overline{PK}$ as $(\bar{I},\ \bar{g}_1,\ \bar{u}',\ \bar{u}_1,\ \ldots,\ \bar{u}_n)$. Choose random $r \leftarrow \mathbb{Z}_q$. If $I \leq_{\text{lex}} \bar{I}$, compute $w = u' \cdot \prod_{i:M_i=1} u_i$ and the signature

$$\sigma = (s \cdot w^r,\ g^r)\ ;$$

if $\bar{I} <_{\text{lex}} I$, compute $\bar{w} = \bar{u}' \cdot \prod_{i:M_i=1} \bar{u}_i$ and the signature

$$\sigma = (s \cdot \bar{w}^r,\ g^r) \ ,$$

where, in each case, $s = \bar{g}_1^{\alpha} = g_1^{\bar{\alpha}}$ is computed using whichever secret key is known to the signer. Verification is changed in the obvious way. A proof similar to the above shows that this scheme satisfies the same security properties as in Theorem 6.15.

6.6.3 A Construction Based on the Camenisch-Lysyanskaya Scheme

A second ring signature scheme based on similar ideas can be derived from the signature scheme of Camenisch and Lysyanskaya (scheme A in [CL]), which we briefly review. Let $\mathbb{G}, \mathbb{G}_1, q, \hat{e}, g$ be as above (we again assume that these are publicly known). The Camenisch-Lysyanskaya signature scheme for messages in $\mathbb{Z}_q$ is defined as follows:

Key Generation. Choose $x, y \leftarrow \mathbb{Z}_q$ and set $X = g^x$ and $Y = g^y$. The public key is (X, Y) and the secret key is (x, y).

Signing. To sign the message $m \in \mathbb{Z}_q$, choose a random value $a \in \mathbb{G}$ and output the signature (a, a^y, a^{x+mxy}).

Verification. To verify the signature (a, b, c) on message m with respect to public key (X, Y), check that $\hat{e}(a, Y) \stackrel{?}{=} \hat{e}(g, b)$ and $\hat{e}(X, a) \cdot \hat{e}(X, b)^m \stackrel{?}{=} \hat{e}(g, c)$.

The reader is referred to [CL] for details regarding the assumption under which the above scheme can be proven secure. As for adapting the above to a two-user ring signature scheme, our key observation is that knowledge of *either* (x, Y) *or* (X, y) is sufficient to generate a signature. In more detail:

- Using (x, Y), a signature on m may be computed as follows: choose random $r \in \mathbb{Z}_q$ and set $a = g^r$. Then output the signature $(a, Y^r, a^x Y^{mxr})$.

- Using (X, y), a signature on m may be computed as follows: choose random $r \in \mathbb{Z}_q$ and set $a = g^r$. Then out put the signature (a, a^y, X^{r+mry}).

This suggests the following ring signature scheme: to generate a public key, choose $x \leftarrow \mathbb{Z}_q$ and a random identifier $I \in \{0,1\}^k$; the public key is $(I, X = g^x)$ and the secret key is x. To sign message m with respect to ring $\{(I, X), (\bar{I}, \bar{X})\}$, proceed as follows: if $I \leq_{\text{lex}} \bar{I}$, compute a Camenisch-Lysyanskaya signature (as described above) for the "public key" $(X, \bar{X})$; if $\bar{I} <_{\text{lex}} I$, compute a Camenisch-Lysyanskaya signature for the "public key" $(\bar{X}, X)$. Verification is done in the obvious way. Unconditional anonymity against full key exposure is immediate, and unforgeability against chosen-subring attacks (assuming security of the Camenisch-Lysyanskaya scheme) can be easily proven exactly as in Theorem 6.15.

6.7 ZAPs

Let L be an $\mathcal{NP}$ language with associated polynomial-time and polynomially-bounded *witness relation* $\mathcal{R}_L$ (i.e., such that $L \stackrel{\text{def}}{=} \{x \mid \exists w : (x, w) \in \mathcal{R}_L\}$). If $(x, w) \in \mathcal{R}_L$ we refer to x as the *statement* and w as the associated *witness* for x. We now recall the definition of a ZAP from [DN00]:

Definition 6.16 (ZAP). *A ZAP for an $\mathcal{NP}$ language L (with associated witness relation $\mathcal{R}_L$) is a triple $(\ell, \mathcal{P}, \mathcal{V})$, where $\ell(\cdot)$ is a polynomial, $\mathcal{P}$ is a* PPT *algorithm, and $\mathcal{V}$ is polynomial-time deterministic algorithm, and such that.*

Completeness *For*[13] *any* $(x, w) \in \mathcal{R}_L$ *and any* $r \in \{0,1\}^{\ell(k)}$*:*

$$\mathbf{Pr}\left[\pi \leftarrow \mathcal{P}_r(x, w) : \mathcal{V}_r(x, \pi) = 1\right] = 1 \ .$$

Adaptive soundness *There exists a negligible function* ε *such that*

$$\mathbf{Pr}\left[r \leftarrow \{0,1\}^{\ell(k)} : \exists(x, \pi) : \quad x \notin L \text{ and } \mathcal{V}_r(x, \pi) = 1\right] \leq \varepsilon(k)\,.$$

Witness indistinguishability *(Informal) For any* $x \in L$*, any pair of witnesses* w_0, w_1 *for* x*, and any* $r \in \{0,1\}^{\ell(k)}$*, the distributions* $\{\mathcal{P}_r(x, w_0)\}$ *and* $\{\mathcal{P}_r(x, w_1)\}$ *are computationally indistinguishable. (Note: more formally, we need to speak in terms of sequences* $\{r_k \in \{0,1\}^{\ell(k)}\}$*,* $\{x_k\}$*, and* $\{(w_{k,0}, w_{k,1})\}$ *but we avoid doing so for simplicity of exposition.)*

A ZAP is used in the following way: The verifier generates a random first message $r \leftarrow \{0,1\}^{\ell(k)}$ and sends it to the prover $\mathcal{P}$. The prover, given r, a statement x, and associated witness w, sends $\pi \leftarrow \mathcal{P}_r(x, w)$ to the verifier. The verifier then runs $\mathcal{V}_r(x, \pi)$ and accepts iff the output is 1.

6.8 Separation Results

6.8.1 Proofs of Claims 6.8–6.11

Proof of Claim 6.8: Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a ring signature scheme satisfying the conditions stated in the claim. Construct the following scheme Π':

[13] We remark that the definition in [DN00] allows for a negligible completeness error. However, their construction achieves perfect completeness when instantiated using the NIZK of [FLS99].

the key generation algorithm $\mathsf{Gen}'(1^k)$ computes $(PK, SK) \leftarrow \mathsf{Gen}(1^k)$ and outputs $PK' = 0|PK$ and $SK' = SK$. The signing algorithm $\mathsf{Sign}'_{s,SK_s}(M, R)$ checks whether all public keys in R begin with a "0": if so, it outputs $\sigma \leftarrow \mathsf{Sign}_{s,SK_s}(M, \bar{R})$ (where $\bar{R}$ contains the same keys as R, but with the leading bit of each key removed); otherwise, it outputs s. $\mathsf{Vrfy}'_R(M, \sigma)$ similarly checks whether all public keys in R begin with a "0": if so, it outputs $\mathsf{Vrfy}_{\bar{R}}(M, \sigma)$ (with $\bar{R}$ as above); otherwise, it outputs 0. (Recall that completeness is only required to hold for rings containing honestly-generated public keys.)

Clearly, the above scheme does not achieve anonymity w.r.t. adversarially-chosen keys. On the other hand, it clearly still achieves basic anonymity. It is also not difficult to see that it remains unforgeable w.r.t. insider corruption.

Proof of Claim 6.9: Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a ring signature scheme satisfying the conditions stated in the claim, and assume a symmetric-key encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ (which exists given the assumption of the claim). Construct scheme Π' as follows: the key generation algorithm $\mathsf{Gen}'(1^k)$ computes $(PK, SK) \leftarrow \mathsf{Gen}(1^k)$ but additionally chooses $\kappa \leftarrow \{0,1\}^k$; it outputs $PK' = PK$ and $SK' = SK|\kappa$. The signing algorithm $\mathsf{Sign}'_{s,SK_s|\kappa}(M, R)$ computes $\sigma \leftarrow \mathsf{Sign}_{s,SK_s}(M, R)$ and $C \leftarrow \mathsf{Enc}_\kappa(0^k)$; it outputs the signature (σ, C). Verification is changed in the obvious way, simply ignoring the ciphertext included as part of the signature.

The scheme does not achieve anonymity under attribution attacks since, given a signature computed by a user with secret key $SK|\kappa$ with respect to any ring, as long as the adversary has all-but-one of the secret keys of the members of the

ring (and, in particular, has the $\{\kappa_i\}$ values for all-but-one of the members), it can determine the correct signer with all but negligible probability. On the other hand, it is not hard to show that the scheme remains anonymous w.r.t. adversarially-chosen keys and also remains unforgeable w.r.t. insider corruption.

We remark that although the modified scheme, above, does not satisfy our formal definition of anonymity against attribution attacks, it does not quite allow an adversary to unambiguously prove to a third party that some user was the signer. (The issue is that the adversary can output whatever $\{\kappa_i\}$ it likes, and not the "actual" values it chose at the time of key generation.) This can be prevented, however, if we additionally require users to include a commitment to κ as part of their public key, and to include the corresponding decommitment as part of their secret key.

Proof of Claim 6.10: Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a ring signature scheme satisfying the conditions of the claim. Construct $\Pi' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ as follows. The key generation algorithm Gen' is the same as Gen. The signing algorithm $\mathsf{Sign}'_{s,SK_s}(M, R)$ sets $R' = R \cup \{M\}$ (where M is treated as a public key) and computes $\sigma_1 \leftarrow \mathsf{Sign}_{s,SK_s}(M, R)$ and $\sigma_2 \leftarrow \mathsf{Sign}_{s,SK_s}(0^k, R')$. The output is the signature (σ_1, σ_2). To verify a signature (σ_1, σ_2) (using Vrfy'), simply verify that signature σ_1 is correct (using Vrfy).

It is easy to see that the scheme is insecure against chosen-subring attacks. Specifically, consider the adversary $\mathcal{A}$ who receives the set of public keys (PK_1, PK_2, PK_3) and then requests a signature on the message $M = PK_3$ with respect to the ring

$R = (PK_1, PK_2)$. Let (σ_1, σ_2) be the response of the signing oracle. $\mathcal{A}$ outputs $(0^k, (\sigma_2, \sigma_2), (PK_1, PK_2, PK_3))$ and terminates. Note that (σ_2, σ_2) is a valid ring signature (with respect to the scheme Π') for the message 0^k with respect to the ring (PK_1, PK_2, PK_3). Also note that $\mathcal{A}$ never requested a signature for such a message/ring pair. We therefore conclude that $\mathcal{A}$ succeeds in producing a valid forgery with probability 1.

It is quite obvious that Π' remains anonymous against full key exposure. Π' is also unforgeable against fixed-ring attacks. We prove this by contradiction. Let $\mathcal{A}'$ be an adversary that breaks the unforgeability of Π' against fixed-ring attacks. We construct an adversary $\mathcal{A}$ that breaks the unforgeability of Π w.r.t. insider corruption. $\mathcal{A}$ takes as input a ring $R = (PK_1, \ldots, PK_n)$ and feeds it to $\mathcal{A}'$. When $\mathcal{A}'$ requests a signature (under Sign') on the message M with respect to the ring R, $\mathcal{A}$ uses its signing oracle to obtain the two components σ_1 and σ_2. Note that $\mathcal{A}$ can obtain σ_2, because it can request a signature on a ring that contains public keys of its choice (M in this case). When $\mathcal{A}'$ outputs a candidate forgery $(M, (\sigma_1, \sigma_2))$, then $\mathcal{A}$ outputs σ_1 as a candidate forgery for message M with respect to the ring R. Note that if the output of $\mathcal{A}'$ is a valid signature with respect to Π', then the output of $\mathcal{A}$ is a valid signature with respect to Π. Also, if $\mathcal{A}'$ never requested a signature on M, then $\mathcal{A}$ never requested a signature on M with respect to the ring R. We conclude that $\mathcal{A}$ outputs a valid forgery whenever $\mathcal{A}'$ does.

Proof of Claim 6.11: Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a scheme satisfying the conditions of the claim. We construct the scheme Π' as follows. The key generation algorithm

Gen' runs Gen to obtain (PK, SK), then outputs $PK' = 0|PK$ and $SK' = SK$. We will say that a public key is "good" if it begins with a zero and that it is "bad" if it begins with a one. Note that all public keys generated by Gen' are "good."

The signing algorithm $\mathsf{Sign}'_{s,SK_s}(M, R)$ proceeds as follows: let R' be the ring consisting of only the "good" public keys from R, with the initial bit stripped. Then compute $\sigma \leftarrow \mathsf{Sign}_{s,SK_s}(M, R')$ and output this as the signature. The verification algorithm is modified in the appropriate way.

Π' is not unforgeable w.r.t. insider corruption. To see this, consider the adversary $\mathcal{A}$ who receives public keys (PK'_1, PK'_2). Next, $\mathcal{A}$ generates an arbitrary "bad" public key $PK' = 1 \mid PK''$. The adversary then requests a signature on an arbitrary message M with respect to the ring (PK'_1, PK'_2, PK') on behalf of the signer holding PK'_1. The signing oracle returns a signature σ that is a valid signature for message M respect to the ring (PK'_1, PK'_2) (recall that PK' is ignored, since it is "bad"). But now $\mathcal{A}$ can output the forgery $(M, \sigma, (PK'_1, PK'_2))$ and succeed with probability 1.

It is not hard to see that Π' remains unforgeable against chosen-subring attacks (since, in such attacks, the adversary can only request signatures with respect to rings that consist only of "good" public keys). One can also easily show that Π' remains anonymous w.r.t. key exposures.

6.8.2 The Herranz-Sáez Ring Signature Scheme

In the proof of Claim 6.10 (above), we presented an "artificial" ring signature scheme that is unforgeable against fixed-ring attacks, but not against chosen-subring attacks. We now show a "natural" scheme, the Herranz-Sáez ring signature scheme [HS], that illustrates the same separation (albeit under less general assumptions).

We first review the Herranz-Sáez ring signature scheme. Let $\mathbb{G}$ be a group of prime order q, such that, given a bit string y it is possible to efficiently verify whether $y \in \mathbb{G}$. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ be a hash function modeled as a random oracle. We assume that H, q, $\mathbb{G}$, and a generator $g \in \mathbb{G}$ are publicly known. The scheme is defined as follows:

Key Generation. Choose $x \leftarrow \mathbb{Z}_q$ and set $y = g^x$. The public key is y and the secret key is x.

Ring Signing. To sign message M with respect to the ring $R = \{y_1, \ldots, y_n\}$ (where $y_i \in \mathbb{G}$ for all i) using secret key x_s, proceed as follows:

1. for $i = 1, \ldots, n$, $i \neq s$, choose random $a_i \leftarrow \mathbb{Z}_q$ and set $C_i = g^{a_i}$;
2. choose random $a_s \leftarrow \mathbb{Z}_q$;
3. compute C_s and b as follows:

$$\begin{aligned} C_s &= g^{a_s} \prod_{i \neq s} y_i^{-H(M, C_i)} \\ b &= a_s + \sum_{i \neq s} a_i + x_s H(M, C_s); \end{aligned}$$

4. in the unlikely event that the C_i are not all distinct, restart from the beginning;

5. output the signature $\sigma = (b, C_1, \ldots, C_n)$.

Ring Verification. To verify the signature $(b, C_1, \ldots, C_n)$ on message M with respect to the ring $R = \{y_1, \ldots, y_n\}$ (where $y_i, C_i \in \mathbb{G}$ for all i), check that the C_i are all distinct and that:

$$g^b \stackrel{?}{=} \prod_{i=1}^{n} C_i \cdot y_i^{H(M,R_i)}.$$

It is not hard to see that the scheme above is unconditionally anonymous against full key exposure, even in the standard model. This is because a ring signature on message M with respect to a ring R is a uniformly random sample from the set of the tuples $(b, C_1, \ldots, C_n)$ that satisfy the ring verification condition, and this distribution is independent of the index s of the signing key used. Additionally, Herranz and Sáez [HS] prove that this scheme is unforgeable against fixed-ring attacks[14] under the discrete logarithm assumption in the random oracle model.

However, the Herranz-Sáez scheme is *not* unforgeable against chosen-subring attacks. Consider an adversary that requests two signatures on the same arbitrary message M with respect to the *disjoint* rings $R = (y_1, \ldots, y_n)$ and $R' = (y'_1, \ldots, y'_m)$, obtaining signature $\sigma = (b, C_1, \ldots, C_n)$ in the first case and $\sigma' = (b', C'_1, \ldots, C'_n)$ in the second. The adversary then outputs the forged signature

$$\sigma^* = (b + b', C_1, \ldots, C_n, C'_1, \ldots, C'_m)$$

on M with respect to the ring $R \cup R' = (y_1, \ldots, y_n, y'_1, \ldots, y'_m)$. Applying the ring verification algorithm shows that this is indeed a valid forgery, except in the unlikely

[14]The authors do not formally define unforgeability, but an inspection of their proof of security reveals that their notion of unforgeability matches our Definition 6.5.

case that $C_i = C'_j$ for some i, j.

The above attack was, in fact, addressed in subsequent work of Herranz [Her05], where it is shown that a simple modification of the scheme (in which the ring R is included as an additional input to the hash function) is unforgeable against chosen-subring attacks.[15] (In fact, examination of the proof shows that the modified scheme is also secure with respect to our Definition 6.7, although adversarially-chosen keys are not explicitly addressed in [Her05].) Nevertheless, the attack on the original scheme that we have demonstrated shows that security against chosen-subring attacks is strictly stronger than security against fixed-ring attacks, and illustrates yet again the importance of rigorously formalizing desired notions of security.

6.9 Proofs of Theorems 6.12 and 6.14

6.9.1 Proof of Theorem 6.12

We restate Theorem 6.12 for convenience:

If encryption scheme $(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure, signature scheme* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ *is existentially unforgeable under adaptive chosen-message attacks, and* $(\ell, \mathcal{P}, \mathcal{V})$ *is a ZAP for* L *as described above, then the above ring signature scheme is (computationally) anonymous against attribution attacks, and unforgeable w.r.t. insider corruption.*

Proof. We prove each of the desired security properties in turn.

[15] We thank Javier Herranz for pointing this out to us.

Anonymity. For simplicity of exposition, we consider Definition 6.4 with $n = 2$; i.e., we assume only two users. By a straightforward hybrid argument, this implies the general case. Given a PPT adversary $\mathcal{A}$, we consider a sequence of experiments E_0, Hybrid_0, Hybrid_1, E_1 such that E_0 (resp., E_1) corresponds to the experiment of Definition 6.4 with $b = 0$ (resp., $b = 1$), and such that each experiment is computationally indistinguishable from the one before it. This implies that $\mathcal{A}$ has negligible advantage in distinguishing E_0 from E_1, as desired.

For convenience, we review experiment E_0. Here, two key pairs $(PK_0 = (pk_{S,0}, pk_{E,0}, r_0), SK_0)$ and $(PK_1 = (pk_{S,1}, pk_{E,1}, r_1), SK_1)$ are generated and $\mathcal{A}$ is given PK_0 and the randomness used to generate (PK_1, SK_1) (by hybrid argument, we can assume that $i_0 = 0$ and $i_1 = 1$). The adversary is also given access to a signing oracle (which can be used to obtain signatures computed using SK_0). $\mathcal{A}$ then outputs a message M along with a ring of public keys R containing both PK_0 and PK_1. Finally, $\mathcal{A}$ is given $\sigma \leftarrow \mathsf{Sign}_{SK_0}(M, R)$.

Experiment Hybrid_0 is the same as experiment E_0 except that we change how the signature σ is generated. In particular, step 3 of the ring signing algorithm is modified as follows: let R_E and M^* be as in the description of the ring signing algorithm given earlier. In step 3, instead of setting $C^*_{1-\beta}$ to be an encryption of all zeros, we now compute $\sigma'_1 \leftarrow \mathsf{Sign}_{sk_{S,1}}(M^*)$ and then set $C^*_{1-\beta} = \mathsf{Enc}^*_{R_E}(\sigma'_1; \omega_{1-\beta})$. We stress that, as in E_0, the ciphertext C^*_β is still set to be an encryption of the signature σ'_0.

It is not hard to see that experiment Hybrid_0 is computationally indistinguishable from experiment E_0, assuming semantic security of the encryption scheme

$(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec})$. This follows from the observations that (1) adversary $\mathcal{A}$ is *not* given the random coins used in generating PK_0 and so, in particular, it is not given the coins used to generate $pk_{E,0}$; (2) (informally) semantic security of encryption under $\mathsf{Enc}_{pk_{E,0}}$ implies semantic security of encryption using $\mathsf{Enc}^*_{R_E}$ as long as $pk_{E,0} \in R_E$ (a formal proof is straightforward); and, finally, (3) the coins $\omega_{1-\beta}$ used in generating $C^*_{1-\beta}$ are not used in the remainder of the ring signing algorithm.

Experiment Hybrid_1 is the same as Hybrid_0 except that we use a different witness when computing the proof π for the ZAP. In particular, instead of using witness $(\sigma'_0, \omega_\beta)$ we use the witness $(\sigma'_1, \omega_{1-\beta})$. The remainder of the signing algorithm is unchanged.

It is relatively immediate that Hybrid_1 is computationally indistinguishable from Hybrid_0, assuming witness indistinguishability of the ZAP. (We remark that the use of a ZAP, rather than non-interactive zero-knowledge, is essential here since the adversary may choose the "random string" component of all the adversarially-chosen public keys any way it likes.) In more detail, we can construct the following malicious verifier algorithm $\mathcal{V}^*$ using $\mathcal{A}$: verifier $\mathcal{V}^*$ generates (PK_0, SK_0) and (PK_1, SK_1) exactly as in experiments Hybrid_0 and Hybrid_1, and gives these keys and the appropriate associated random coins to $\mathcal{A}$. The signing queries of $\mathcal{A}$ can easily be answered by $\mathcal{V}^*$. When $\mathcal{A}$ makes its signing query, $\mathcal{V}^*$ computes the C^*_j exactly as in Hybrid_1 and then gives to the prover $\mathcal{P}$ the keys $\{pk_{S,i}\}_{i \in R}$, the message M^*, the set of keys R_E, and the ciphertexts (C^*_0, C^*_1); this defines the $\mathcal{NP}$-statement x exactly as in step 4 of the ring signing algorithm. In addition, $\mathcal{V}^*$ gives the two witnesses $w_0 = (\sigma'_0, \omega_\beta)$ and $w_1 = (\sigma'_1, \omega_{1-\beta})$ to $\mathcal{P}$. Finally, $\mathcal{V}^*$ sends as its first message

the "random string" component r of the lexicographically-first public key in R (this r is the random string that would be used to generate the proof π in step 4 of the ring signing algorithm). The prover responds with a proof $\pi \leftarrow \mathcal{P}_r(x, w_b)$ (for some $b \in \{0,1\}$), and then $\mathcal{V}^*$ outputs (C_0^*, C_1^*, π).

Note that if the prover uses the first witness provided to it by $\mathcal{V}^*$ then the output of $\mathcal{V}^*$ is distributed exactly according to Hybrid_0, while if the prover uses the second witness provided to it by $\mathcal{V}^*$ then the output of $\mathcal{V}^*$ is distributed exactly according to Hybrid_1. Witness indistinguishability of the ZAP thus implies computational indistinguishability of Hybrid_0 and Hybrid_1.

We may now notice that Hybrid_1 is computationally indistinguishable from E_1 by exactly the same argument used to show the indistinguishability of Hybrid_0 and E_0. This completes the proof.

Unforgeability. Assume there exists a PPT adversary $\mathcal{A}$ that breaks the above ring signature scheme (in the sense of Definition 6.7) with non-negligible probability. We construct an adversary $\mathcal{A}'$ that breaks the underlying signature scheme $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ (in the standard sense of existential unforgeability) with non-negligible probability.

$\mathcal{A}'$ receives as input a public key pk_S. Let $n = n(k)$ be a bound on the number of (honest user) public keys that $\mathcal{A}$ expects to be generated. $\mathcal{A}'$ runs $\mathcal{A}$ with input public keys $S = \{PK_1, \ldots, PK_n\}$, that $\mathcal{A}'$ generates as follows. $\mathcal{A}'$ chooses $i^* \leftarrow \{1, \ldots, n\}$ and sets $pk_{S,i^*} = pk_S$. The remainder of public key PK_{i^*} is generated exactly as prescribed by the Gen algorithm, with the exception that the

decryption key sk_{E,i^*} that is generated is *not* erased. Public keys PK_i for $i \neq i^*$ are also generated exactly as prescribed by the Gen algorithm, again with the exception that the decryption keys $\{sk_{E,i}\}$ are not erased.

$\mathcal{A}'$ then proceeds to simulate the oracle queries of $\mathcal{A}$ in the natural way:

1. When $\mathcal{A}$ requests a signature on message M, with respect to ring R (which may possibly contain some public keys generated in an arbitrary manner by $\mathcal{A}$), to be signed by user $i \neq i^*$, then $\mathcal{A}'$ can easily generate the response to this query by running the Sign algorithm completely honestly;

2. When $\mathcal{A}$ requests a signature on message M, with respect to ring R (which, again, may possibly contain some public keys generated in an arbitrary manner by $\mathcal{A}$) to be signed by user i^*, then $\mathcal{A}'$ cannot directly respond to this query since it does not have sk_{S,i^*}. Instead, $\mathcal{A}'$ sets M^* appropriately, submits M^* to its signing oracle, and obtains in return a signature σ'_{i^*}. It then computes the remainder of the ring signature by following the rest of the Sign algorithm; note, in particular, that sk_{S,i^*} is not needed for this;

3. Any corruption query made by $\mathcal{A}$ for a user $i \neq i^*$ can be faithfully answered by $\mathcal{A}'$. On the other hand, if $\mathcal{A}$ ever makes a corruption query for i^*, then $\mathcal{A}'$ simply aborts.

At some point, $\mathcal{A}$ outputs a forgery $\bar{\sigma} = (\bar{C}_0^*, \bar{C}_1^*, \bar{\pi})$ on a message $\bar{M}$ with respect to some ring of honest-user public keys $\bar{R} \subseteq S$. If $PK_{i^*} \notin \bar{R}$, then $\mathcal{A}'$ aborts. Otherwise, since $\mathcal{A}'$ knows all relevant decryption keys (recall that the ring $\bar{R}$ contains public keys of honest users only, and these keys were generated by $\mathcal{A}'$) it can

decrypt both $\bar{C}_0^*, \bar{C}_1^*$ and obtain a candidate signatures $\bar{\sigma}_{i^*,0}, \bar{\sigma}_{i^*,1}$ respectively. Finally, $\mathcal{A}'$ sets $\bar{M}^* = \bar{M} \,|\, \overline{PK}_1 \,|\, \cdots \,|\, \overline{PK}_{n'}$ (where $\bar{R} = \{\overline{PK}_i\}$) and verifies which of the $\sigma_{i^*,0}, \bar{\sigma}_{i^*,1}$ is a valid signature for M^*: if $\mathsf{Vrfy}'_{pk_S}(M^*, \sigma_{i^*,0}) = 1$, $\mathcal{A}'$ outputs $(\bar{M}^*, \bar{\sigma}_{i^*,0})$; else outputs $(\bar{M}^*, \bar{\sigma}_{i^*,1})$. Note that (by requirement) $\mathcal{A}$ never requested a signature on message $\bar{M}$ with respect to the ring $\bar{R}$, and so $\mathcal{A}'$ never requested a signature on message $\bar{M}^*$ from its own oracle.

We claim that if $\mathcal{A}$ forges a signature with non-negligible probability $\varepsilon = \varepsilon(k)$, then $\mathcal{A}'$ forges a signature with probability at least $\varepsilon' = \varepsilon/n - \mathsf{negl}(k)$. To see this, note first that if $\mathcal{A}$ outputs a valid forgery then with all but negligible probability (by soundness of the ZAP) it holds that $(\overline{pk}_{S,i}, \bar{M}^*, \bar{R}_E, \bar{C}_j^*) \in L$ for some i, j (where $\overline{pk}_{S,i}$ and $\bar{R}_E$ are defined in the natural way based on the ring $\bar{R}$ and the public keys it contains). Conditioned on this, with probability $1/n$ it is the case that (1) $\mathcal{A}'$ did not abort and furthermore (2) $(\overline{pk}_{S,i^*}, \bar{M}^*, \bar{R}_E, \bar{C}_j^*) \in L$. When this occurs, then with all but negligible probability $\mathcal{A}'$ will recover (by decrypting as described above) a valid signature $\bar{\sigma}_{i^*,j}$ on the message $\bar{M}^*$ with respect to the given public key $\overline{pk}_{S,i^*} = pk_S$ (relying here on the fact that with all but negligible probability over choice of encryption public keys, Enc^* has zero decryption error). Security of $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ thus implies that ε is negligible. □

6.9.2 Proof of Theorem 6.14

We restate Theorem 6.14 for convenience:

Under the assumptions of Theorem 6.12 and assuming $(\mathsf{EGen}, \mathsf{Enc}, \mathsf{Dec})$ *has an obliv-*

ious key generator, the modified ring signature scheme described above is (computationally) anonymous against full key exposure, and unforgeable w.r.t. insider corruption.

Proof. The proof of unforgeability follows immediately from Theorem 6.12 since, by Definition 6.13, the adversary cannot distinguish between the original scheme (in which the encryption key is generated using EGen) and the modified scheme (in which the encryption key is generated using $\mathsf{OblEGen}$).

We now argue that the modified scheme achieves anonymity against full key exposure. First we note that the anonymity against attribution attacks claimed in Theorem 6.12 holds even when the adversary is given all random coins used to generate (PK_0, SK_0) *except* for those coins used to generate $pk_{E,0}$ (using EGen). Now, if there exists a PPT adversary $\mathcal{A}$ that breaks anonymity of the modified scheme in the sense of full key exposure, we can use it to construct a PPT adversary $\mathcal{A}'$ that breaks anonymity of the original scheme against attribution attacks. $\mathcal{A}'$ receives PK_0, the random coins $\omega_{S,1}, \omega_{E,1}$ used to generate (PK_1, SK_1), and the random coins $\omega_{S,0}$ used to generate $pk_{S,0}$ (i.e., $\mathcal{A}$ is *not* given the coins used to generate $pk_{E,0}$). Next, $\mathcal{A}'$ runs $\omega'_{E,0} \leftarrow \mathsf{OblRand}(pk_{E,0})$ and $\omega'_{E,1} \leftarrow \mathsf{OblRand}(pk_{E,1})$ and gives to $\mathcal{A}$ the public key PK_0 it received as well as the random coins $\omega_{S,0}, \omega'_{E,0}, \omega_{S,1}, \omega'_{E,1}$. The remainder of $\mathcal{A}$'s execution is simulated in the natural way by $\mathcal{A}'$.

Now, Definition 6.13 implies that the advantage of $\mathcal{A}$ in the above is negligibly close to the advantage of $\mathcal{A}$ in attacking the modified scheme in the sense of full key exposure. But the advantage of $\mathcal{A}$ in the above is exactly the advantage of

$\mathcal{A}'$ in attacking the original scheme via key attribution attack. Since we have already proved that the original scheme is anonymous against attribution attacks (cf. Theorem 6.12), the modified scheme is anonymous against full key exposure. □

Chapter 7

Conclusions

In this dissertation, we presented four major contributions. Our first contribution is the design of LMS, an efficient lookup protocol for a setting, where each node is directly connected to a small set of neighbors and can send a message only to them. This models an ad-hoc network without a routing infrastructure and it also has applications in trust networks. As a demonstration of the power of LMS, in other work we used it to build a completely decentralized public-key distribution system [MBKM06]. We proved analytic bounds for the worst-case performance of LMS and, through detailed simulations, we show that the actual performance on realistic topologies is significantly better. As part of our analysis, we also derived a powerful theorem on statistical distributions, which is of independent interest.

Our second contribution is the definition of the AS attack model and the design of SDHT, a lookup protocol that is resilient within such model. In the AS Model, an attacker may control an arbitrary number of nodes, but with the restriction that all of those nodes belong to a small set of ASes. The AS Model represents a reasonable intermediate assumption between assuming that the attacker can control only a small fraction of the nodes in the system, which is unrealistic for Internet applications with open membership, and assuming that the attacker may control an arbitrary number of nodes with no restrictions, which makes it unlikely that any

lookup protocol exists that is simultaneously secure and efficient.

Our last two contributions are on the topic of ring signatures, a variant of digital signatures with an interesting anonymity property. First, we have devised new definitions of security for ring signatures that address realistic threats ignored by definitions provided in previous work. Finally, we have designed the first three constructions of ring signatures that are provably secure in the standard model, i.e. without the use of the random oracle heuristic.

Bibliography

[ACL+03] Marta Arias, Lenore J. Cowen, Kofi A. Laing, Rajmohan Rajaraman, and Orjeta Taka. Compact routing with name independence. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 184–192, New York, NY, USA, 2003. ACM Press.

[AGM+04] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 20–24, New York, NY, USA, 2004. ACM Press.

[AHR05] B. Adida, S. Hohenberger, and R.L. Rivest. Ad-hoc-group signatures from hijacked keypairs. Available at `http://theory.lcs.mit.edu/~srhohen/papers/AHR.pdf`, 2005.

[AM03] Ittai Abraham and Dahlia Malkhi. Probabilistic quorums for dynamic systems. In *International Symposium on Distributed Computing (DISC)*, pages 60–74, 2003.

[AM05] Ittai Abraham and Dahlia Malkhi. Name independent routing for growth bounded networks. In *SPAA'05: Proceedings of the 17th annual ACM symposium on Parallelism in algorithms and architectures*, pages 49–55, New York, NY, USA, 2005. ACM Press.

[AOS] M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n signatures from a variety of keys. In *Advances in Cryptology — Asiacrypt 2002*.

[BGLS] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology — Eurocrypt 2003*.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *IACR Theory of Cryptography Conference*, New York, NY, March 2006.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[BMW] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Advances in Cryptology — Eurocrypt 2003*.

[BPS05] Jean-Michel Busca, Fabio Picconi, and Pierre Sens. Pastis: A highly-scalable multi-user peer-to-peer file system. In José C. Cunha and Pedro D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 1173–1182. Springer, 2005.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, 1993.

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, New York, NY, USA, 1984. ACM Press.

[BSS] E. Bresson, J. Stern, and M. Szydlo. Threshold ring signatures and applications to ad-hoc groups. In *Advances in Cryptology — Crypto 2002*.

[CAPMN03] Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, June 2003.

[CDG+02] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[CDS] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology — Crypto '94*.

[CKP] L. Chen, C. Kudla, and K.G. Patterson. Concurrent signatures. In *Advances in Cryptology — Eurocrypt 2004*.

[CL] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology — Crypto 2004*.

[CLL02] Jacky Chu, Kevin Labonte, and Brian Neil Levine. Availability and locality measurements of peer-to-peer file systems. In *Proc. ITCom: Scalability and Traffic Control in IP Networks II Conferences*, volume 4868, July 2002.

[CLY05] S. S.M. Chow, J.K. Liu, and T. H. Yuen. Ring signature without random oracles. Cryptology ePrint Archive, 2005. http://eprint.iacr.org/2005/317.

[CMH+02] I. Clarke, S. G. Miller, T. W. Hong, O. S. Sandberg, and B. Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1), Jan./Feb. 2002.

[Com01] Open Source Community. Gnutella, 2001. See http://gnutella.wego.com.

[CRB+03] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM Press, 2003.

[CvH] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology — Eurocrypt '91.*

[Dam87] Ivan Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987.

[DKNS] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad-hoc groups. In *Advances in Cryptology — Eurocrypt 2002.*

[DN] I. Damgård and J.B. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Advances in Cryptology — Crypto 2000.*

[DN00] C. Dwork and M. Naor. Zaps and their applications. In *Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2000.

[DR98] Devdatt Dubhashi and Desh Ranjan. Balls and bins: a study in negative dependence. *Random Struct. Algorithms*, 13(2):99–124, 1998.

[Fis83] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In Marek Karpinski, editor, *FCT*, volume 158 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 1983.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[FLS99] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM J. Computing*, 29(1):1–28, 1999.

[Fri00] Alan M. Frieze. Edge-disjoint paths in expander graphs. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 717–725, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[Fri03] Joel Friedman. A proof of alon's second eigenvalue conjecture. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 720–724, New York, NY, USA, 2003. ACM Press.

[FS] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — Crypto '86*.

[FSY05] Amos Fiat, Jared Saia, and Maxwell Young. Making chord robust to byzantine faults. In *European Symposium on Algorithms (ESA 05)*, October 2005.

[GMS04] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *IEEE Infocom*, 2004.

[Gna02] Omprakash D Gnawali. A keyword set search system for peer-to-peer networks. Master's thesis, Massachusetts Institute of Technology, June 2002.

[Gnu] `http://rfc-gnutella.sourceforge.net/developer/index.html`.

[GSGM03] P. Ganesan, Q. Sun, and H. Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitrary topology. In *22nd Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[Her05] J. Herranz. *Some digital signature schemes with collective signers*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, April 2005. Available at `http://www.lix.polytechnique.fr/~herranz/thesis.htm`.

[HS] J. Herranz and G. Sáez. Forking lemmas for ring signature schemes. In *Progress in Cryptology — Indocrypt 2003*.

[JGZ03] Song Jiang, Lei Guo, and Xiaodong Zhang. LightFlood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. In *International Conference on Parallel Processing*, 2003.

[JSI] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *Advances in Cryptology — Eurocrypt '96*.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 190–201, New York, NY, USA, 2000. ACM Press.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM Press, 2002.

[LHH+04] Boon Thau Loo, Joseph M. Hellerstein, Ryan Huebsch, Scott Shenker, and Ion Stoica. Enhancing p2p file-sharing with an internet-scale query processor. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB '04)*, pages 432–443, Toronto, Canada, August 2004.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2004.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[LWW] J.K. Liu, V.K. Wei, and D.S. Wong. Linkable spontaneous anonymous group signatures for ad hoc groups. In *ACISP 2004*.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MBKM06] Ruggero Morselli, Bobby Bhattacharjee, Jonathan Katz, and Michael Marsh. Keychains: A decentralized public-key infrastructure. Technical Report CS-TR-4788, Department of Computer Science, University of Maryland, March 2006.

[MBMS05] Ruggero Morselli, Bobby Bhattacharjee, Michael A. Marsh, and Aravind Srinivasan. Efficient lookup on unstructured topologies. In *Symposium on Principles of Distributed Computing (PODC 2005)*, Las Vegas, Nevada, USA, July 2005. ACM SIGACT and SIGOPS.

[MBMS07] Ruggero Morselli, Bobby Bhattacharjee, Michael A. Marsh, and Aravind Srinivasan. Efficient lookup on unstructured topologies. *IEEE*

Journal on Selected Areas in Communications, 2007. Issue on Peer-to-Peer Communications and Applications.

[McC03] Bill McCarty. Botnets: Big and bigger. *IEEE Security & Privacy*, 1(4):87–90, 2003.

[Mit01] M. Mitzenmacher. Compressed bloom filters. In *20th Annual ACM Symposium on Principles of Distributed Computing*, pages 144–150, 2001.

[MMB00] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. *SIGCOMM Comput. Commun. Rev.*, 30(2):18–28, 2000.

[MMGC02] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: a read/write peer-to-peer file system. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 31–44, New York, NY, USA, 2002. ACM Press.

[MNR02] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192. ACM Press, 2002.

[MR97] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[Nao] M. Naor. Deniable ring authentication. In *Advances in Cryptology — Crypto 2002*.

[Net] Sharman Networks. Kazaa. `http://www.kazaa.com`.

[oSN02] National Insititute of Standards and Technology (NIST). Fips 180-2: Secure hash standard. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`, August 2002.

[PD00] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kauffman Publishers, second edition, 2000.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[RF02] M. Ripeanu and I. Foster. Mapping the Gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*, pages 161–172, San Diego, CA USA, August 27–31 2001. ACM.

[Rip] `http://people.cs.uchicago.edu/~matei /GnutellaGraphs/`.

[RK02] Sean C. Rhea and John Kubiatowicz. Probabilistic location and routing. In *Proceedings of INFOCOM 2002*, 2002.

[RKCD01] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In Jon Crowcroft and Markus Hofmann, editors, *Networked Group Communication*, volume 2233 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2001.

[RS04] Venugopalan Ramasubramanian and Emin Gun Sirer. Beehive: Exploiting power law query distributions for O(1) lookup performance in peer to peer overlays. In *Proceedings of NSDI*, 2004.

[RST] R.L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Asiacrypt 2001.* Full version available at `http://www.mit.edu/~tauman` and to appear in *Essays in Theoretical Computer Science: in Memory of Shimon Even.*

[RV03] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of IFIP/ACM Middleware 2003*, 2003.

[SBS05] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communications. *Journal of Computer Security*, 13(6):839–876, 2005.

[SGG03] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Syst.*, 9(2):170–184, 2003.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[TD04] Chunqiang Tang and Sandhya Dwarakadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of USENIX NSDI '04 Conference*, San Fransisco, CA, March 2004.

[Tou84] Sam Toueg. Randomized byzantine agreements. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 163–178, New York, NY, USA, 1984. ACM Press.

[Wat] B. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology — Eurocrypt 2005*.

[WJ02] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator. Technical Report UM-CSE-TR-456-02, University of Michigan, 2002.

[XZF04] J. Xu, Z. Zhang, and D. Feng. A ring signature scheme using bilinear pairings. In *Workshop on Information Security Applications (WISA)*, 2004.

[YGM02] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *ICDCS*, pages 5–14, 2002.

[ZK] F. Zhang and K. Kim. ID-based blind signature and ring signature from pairings. In *Advances in Cryptology — Asiacrypt 2002*.

[ZKJ01] B. Zhao, K. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley Technical Report, 2001.