Towards freedom of speech on the Internet: Censorship-resistant communication and storage

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Eugene Y. Vasserman

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

June, 2010

Acknowledgements

I would like to thank my advisors, Nicholas Hopper and Yongdae Kim, for their continuing support, patience, and encouragement. I also offer profuse thanks to my other committee members, Andrew Odlyzko and Zhi-Li Zhang, for helpful and insightful comments on earlier drafts.

Early membership-concealing network designs were conceived with the help of Jon McLachlan. Much of this work stems from earlier discussions with him, and I am extremely grateful for his help.

Dedication

For my family,
for their infinite patience,
never-ending support,
and their uncountable sacrifices.
Thank you.

Моей семье.
За их бесконечное терпение,
постоянную поддержку, и за всё,
чем они пожертвовали ради меня.
Спасибо.

Abstract

This work explores the problem space of censorship resistance with the explicit goal of protecting a censorship-resistant system and its users from powerful adversaries who control the network gateways. The result of this work is a document storage system which is highly available and robust to targeted censorship. It is designed to resist attacks from very powerful adversaries, who are willing to shut down large sections of the Internet in order to accomplish their censorship goals. Our design aims to be as easy to use, but far more robust than, some current centralized systems, so we use a completely distributed peer-to-peer infrastructure but still support human-readable keyword search. Network participants who contribute storage enjoy plausible deniability, in that they have no easy way to determine what content they are storing locally. We also explicitly support edited content, such that any information can be published, but only popular or editor-approved information will be kept.

A major building block of our system is *membership concealment* — the idea of a network that hides the real-world identities of participants. We formalize the concept of membership concealment, show that it is *required for censorship resistance*, discuss a number of attacks against existing systems, and present real-world attack results. Since membership concealment requires resisting hypothesis testing and brute-force scanning, we ensure that network members are not identifiable as such by unauthorized parties. To that end, we construct an authenticated transmission control protocol, adding steganographic authentication to TCP in a provably undetectable manner. Finally, we show through theoretical analysis and simulation that the complete system, while imposing a factor of 10 storage overhead, *can tolerate node failure rates up to* 70% while retaining the ability to route messages and retrieve *every stored file with probability* 99.99998666%, even when the volume of stored content is on the order of hundreds of exabytes.

Contents

List of Tables

List of Figures

"The Net interprets censorship as damage and routes around it."

John Gilmore, 1993 [1]

But what if the censor controls the routing infrastructure?

Chapter 1

Introduction

The focus of this dissertation is constructing censorship-resistant systems for communicating and storing information prohibited by certain regimes or within certain network domains. This topic of research is becoming increasingly important due to the rise of Internet censorship by private and state interests, who use a variety of social and technological means to limit expression or availability of information. The Open Net Initiative (ONI) [2], which catalogs world-wide censorship efforts, groups them into four different categories: (i) technical blocking (such as DNS filtering), IP blocking, URL filtering, and content inspection; (ii) search removal, i.e. suppression of web sites or terms from search engines; (iii) take-down, the use of legal or regulatory power to demand the removal of content; and (iv) induced self-censorship, through intimidation including surveillance or the perception of surveillance. In 2006, ONI reported strong evidence of filtering in 26 of 40 countries surveyed [3], with anecdotal evidence suggesting widespread use of social and legal means as well. This list includes Western democracies such as the U.S. and EU member nations [4]. Such prevalence suggests that censorship by governments, ISPs, and corporations represents a valid threat to freedom of speech on the Internet.

Due to differing libel laws [5, 6], private corporate control of large-scale Internet-accessible storage, and almost complete control of the core Internet routing fabric by governments and private organizations, publishing controversial information online and *keeping it available* is not easy. Whatever legal protections free speech and whistle-blowing theoretically enjoy, online content can nonetheless be removed in practice using mechanisms readily available to most large organizations, and available to anyone willing to pay sufficiently for the privilege. Furthermore, since legal channels may be under the control of the very same parties who are attempting to prevent disclosure of information, we need technological measures to overcome these censorship efforts and bring the technological possibilities of free speech in line with their theoretical ideals.

As more censorship-*enabling* systems are deployed, and as more legislative control is imposed on online content, we see increased usage of censorship-*resistance* technologies — tools designed to circumvent the technological filters and bypass or evade legislative blocking. This dissertation provides a technological solution to this problem. While numerous currently-deployed systems are already used for censorship resistance, such as Tor bridges [7], Freenet [8], `Anonymizer.com` [9], and WikiLeaks [10], they are all

vulnerable to a number of attacks. For instance, the use of several of these systems is problematic in environments where they are explicitly proscribed. Some censorship-resistant designs [11] have taken the all-or-nothing approach, assuming that an adversary would want to disable access to selected content, but not to the entire system. Since the most important requirement for censorship resistant networks is availability — an attack against availability is in itself an act of censorship — any design candidate must be robust to attempts at complete blocking. We advocate a powerful adversary model, where the censor is willing to prevent access to popular services, websites, or even sections of the IP address space in order to block some targeted content, regardless of collateral damage. Events such as [12, 13, 14] support our position. It is likely that such an adversary would be willing to persecute users of censorship circumvention technologies as well, so censorship-resistant systems *should also prevent the censor from identifying the system's participants*, protecting both content and users not only from technological, but also from social and judicial attacks. If neither content providers nor users can be identified, they cannot be coerced to stop accessing content, publishing new content, or otherwise participating in the network.

One example of an entity with similar goals is WikiLeaks, a website that publishes documents of social significance, which are generally private and are "leaked" by individuals with privileged access. WikiLeaks' stated mission is to "protect internal dissidents, whistleblowers, journalists and bloggers who face legal or other threats related to publishing" [10]. Their multi-jurisdictional management and hosting structure, combined with anonymity and cryptographic technologies, make it difficult to discover the source of leaked documents and/or restrict their online distribution. Unfortunately, this approach is ultimately limited by the human and technological resources of the organization: WikiLeaks has a finite (and presumably small) number of servers and editors, representing a relatively tractable attack surface for someone sufficiently motivated. A fully-distributed system, on the other hand, spreads the information distribution role over far more entities, and is thus more resistant to denial of service (DoS) and coersion attacks. Such a system has the potential to be strictly more censorship-resistant than WikiLeaks, since a distributed system will attract more individual users contributing bandwidth, computation, and storage, and can also use all the technological resources currently controlled by WikiLeaks.

Note that a censorship-resistant system *does not aim to provide data privacy.* The opposite is true — all stored content should be searchable and available as conveniently as possible. There are two exceptions to this rule: data on the wire must be encrypted to avoid keyword-based blocking by a censor who controls the network, and member nodes who contribute storage should have no easy way to determine what they are storing, ensuring plausible deniability.

1.1 Contribution of This Work

We design a robust, highly available, and censorship-resistant system for permanent online document storage that resists blocking attempts from very powerful adversaries who are willing to shut down large sections of the Internet in order to accomplish their censorship goals. While neither censorship resistance nor robust permanent storage are new ideas, no design considers as strong an adversary model as a nation-state that controls network entry and exit points. We show that with a storage overhead of a factor of 10 *our system can tolerate node failure rates up to* 70% while retaining the ability to route messages and retrieve *all* stored files with 99.99998666% probability, even when the volume of stored content is on the order of hundreds of exabytes. Our system allows anyone to publish and/or retrieve any content stored in the overlay using a simple and intuitive keyword search, while preventing internal or external adversaries from censoring or modifying content, or determining users' physical identities[1] . This system is constructed of three major building blocks, described below.

1.1.1 Membership-concealing Overlay Networks

If a censor who is both willing and capable of preventing access to a given network, application, protocol, or large section of the Internet, any system whose participants can be enumerated can be blocked (e.g. by listing participants' IP addresses and adding them to a blacklist at the network level), disrupting access to the service. Therefore, the first building block of a robust censorship-resistant system is a communication layer with a difficult-to-enumerate membership set. In Chapter 2 we initiate a systematic study of membership concealment as a security goal. We introduce "membership-concealing

[1] e.g. IP addresses

overlay networks" (MCONs), which are peer-to-peer (P2P) overlays whose membership set is hidden from both insiders and outsiders. Overlays and membership concealment may sound incompatible, since nodes must always rely on others for communication and connectivity, but it is possible to *minimize the number of other overlay nodes who know the identity of any given node*, to the point where a member only needs to disclose its identity to a small constant number of other nodes. Such systems need pseudonyms to allow for one-to-one communication. Pseudonyms should preserve unlinkability between MCON identities and real-world identities, whether for targeted individuals or for a non-trivial fraction of MCON members. To be useful, MCONs must support scalable and efficient routing and search. Finally, robust censorship resistance requires that MCONs remain available even during high member churn,[2] and when under attack from insider and/or outsider groups.

While membership concealment is not a new idea, and has been implicitly described in other work (sometimes referred to as "darknet"), it was not rigorously defined. This lack of formal definition caused membership concealment features to be implemented in an ad-hoc fashion, usually resulting in vulnerabilities. We propose three proof-of-concept membership-concealing designs: one that is more efficient, another that is more robust to membership churn, and yet another which is a hybrid of the first two. All schemes are robust to security problems present in prior approaches, protecting the system from both insider and outsider attackers. Our MCON can be bootstrapped from any social graph of offline face-to-face relationships. (Basing a network on a social network graph allows us to use Sybil attack [15] mitigation systems such as SybilLimit or SybilInfer [16, 17].) Membership is by invitation only, so our network is not "open" in the same sense as other P2P systems, which allow anyone who knows at least one member to become a member themselves. Finally, our designs use distributed hash tables (DHTs) to enable efficient search and ensure that both popular and rare files can be located within a predictable period of time.[3]

[2] Members can go offline without disrupting the network

[3] Files can be arbitrary named data, so "locating files" does not imply a traditional file-sharing system.

1.1.2 SilentKnock: Provably Covert Authentication

A major problem with past approaches to censorship resistance has been vulnerability to scanning and/or hypothesis testing — if an adversary can identify that a particular computer is running a given service (such as the MCON network software), the membership-hiding properties of that service are compromised. Therefore, MCON members must not respond to communication attempts from unauthorized parties. To that end, we use the previously ill-defined method of service hiding called "port knocking," define it rigorously, and describe a provably-secure design.

Port knockinghas historically been used to hide the services available on a given server by concealing open ports from attackers has been historically used as an extra layer of protection for web-facing servers. To use a server implementing port knocking, a client must transmit a special "knock" that authenticates it. Any attempt to connect that is not associated with the correct knock will be dropped; thus to an unauthorized user it should appear as if no network services are running on the server. A variety of knocking methods have been proposed, such as a sequence of dropped connection attempts to closed ports [18], cryptographic authenticators in the initial connection request packet [19], "funny-looking" DNS lookups [20], and IPsec tunneling [21]. However, many of these schemes have been accused of offering "security through obscurity," since it is trivially easy for an intelligent adversary to detect and steal knocks in non-cryptographic systems. By making the distinction between flawed *implementations*, which are only secure if the details of the system are unknown, and the *concept* of port knocking (such that *even given the details of a port knocking scheme* one cannot tell if it is being employed), we argue that the *concept* of port knocking is not fundamentally flawed. In fact, it is a good fit to conceal the presence of MCON software on a given system. This also means that the use of port knocking should itself be concealed, as it may cause an adversary to assume a protected server is running an MCON, and physically confiscate it to examine the software. All existing port knocking implementations fail to conceal their presence even under relatively weak attacker models.

In Chapter 3, we develop a formal security model which captures the notion of provably undetectable port knocking. Our notion of security implies that while a computationally bounded adversary may observe many authenticated sessions and arbitrarily inject, delete, and reorder messages between the client and server, he cannot

distinguish a port knocking client and server from a pair using ordinary TCP/IP plus some out-of-band authentication mechanism that prevents unauthorized clients from connecting. That is, our definition allows the adversary to observe authenticated sessions and necessarily allows the adversary to observe that *somehow* sessions are being authenticated, but ensures that no *additional* information about the authenticating mechanism is leaked. This leaves many plausible explanations for the behavior, such as dynamic firewall rules.[4] We also present SilentKnock, our implementation of secure port knocking including several necessary tricks for secure and reliable interaction with TCP/IP, such as replay attack protection, client/server synchronization, and indistinguishability. We then analyze a number of possible attacks on our implementation and show results demonstrating the performance of our system on real-world hardware.

1.1.3 Censorship-resistant Overlay Publishing System

Finally, Chapter 4 builds on our previous work and describes CROPS, a censorship-resistant overlay publishing system implemented as a storage layer on top of an existing MCON. CROPS allows anyone to publish and retrieve content stored in the overlay while preventing internal or external adversaries from censoring or modifying content, or determining the physical identities of participating entities. CROPS member nodes can assume combinations of the following roles: a *publisher*, who uploads content; a *storer*, who stores content; an *intermediary*, who helps route control and data messages through the network; and a *searcher*, who searches and downloads content. All honest nodes must serve at least as intermediaries, but not necessarily publishers, storers, or searchers. Furthermore, all nodes are considered peers, or equal participants in the network, (even if they do not contribute storage).[5] We may refer to members interchangeably by any one of their roles, or as *peers* or *clients*.

Our design supports keyword searches, so users are not required to know a cryptographic hash of the file (or similar hard-to-remember information) in order to retrieve content from the network. We do not discriminate based on file popularity, but instead allow anything to be stored in the network for a limited period of time, while content

[4] e.g. a service that is only available at given times, or a software firewall that allows the user to manually approve connection requests.

[5] An incentive scheme for storage contribution (similar to [22] and [23]) may be added later, if this can be achieved in a membership-concealing manner.

vetted by designated editors is retained forever. CROPS incorporates a self-cleaning mechanism, such that *non-vetted and unpopular files* are eventually removed from the network, freeing space for more important content. We show that with a storage overhead of a factor of 10 *we can tolerate node failure or maliciousness rates up to* 70% while retaining the ability to route packets and retrieve *all* stored files with at least 99.99998666% probability, even when the volume of stored content is on the order of hundreds of exabytes.

1.2 Justification

When people ask whether censorship resistance technology can be used for nefarious purposes, the answer is a resounding "yes." This technology, along with things like kitchen utensils, common soil bacteria, or power generation equipment can all be repurposed for use in destruction and/or crime. This does not make censorship resistance technologies any less valuable. On the contrary, as Noam Chomsky once said, "If you believe in freedom of speech, you believe in freedom of speech for views you don't like" [24]. Since one certainly believes one's own opinions to be true, in order for one to embrace freedom of speech for others, one must defend their ability to express opinions one does not share, and may even consider utterly false or downright offensive. Freedom of speech is certainly a double-edged sword.

However, freedom of speech has received unambiguous recognition as a necessity. The First Amendment to the United States' Constitution [25] guarantees that no federal law will be passed "prohibiting ... or abridging the freedom of speech" [26]. This amendment has been historically interpreted even more broadly to guarantee freedom of *expression*, not just speech, allowing opinions to be expressed freely in forms other than oral communication. Freedom of expression is also considered a "universal human right," as stated in Article 19 of the Universal Declaration of Human Rights [27], adopted by the United Nations in 1948. While the ability to express one's opinions freely does not imply free access to mass communication technologies to do so, that same document acknowledges the importance of distribution channels, stating that freedom of expression "includes ... to seek, receive and impart information and ideas through any media and regardless of frontiers."

Censorship resistance is a critical foundation for *effective* freedom of speech, versus technical freedom of speech. We define the latter as the ability to express opinions without fear of punishment, independent of the content of those opinions. However, if such expression is blocked from distribution — if one cannot make one's opinions available to the intended audience, freedom of speech is fundamentally limited. Any distribution channel that can be censored based on the content of the information it carries, is not an effective means to express opinions. A censorship-resistant communication channel is therefore required for freedom of speech to be meaningful.

Recall the previously-mentioned censorship resistance group WikiLeaks. Beyond resisting censorship, their success also relies on protecting individuals submitting the information that the website makes available. "Whistleblowers," as these individuals are sometimes called, face prosecution for their behavior, and thus the recipients of leaked documents must protect the identity of their source. If individuals submitting documents were frequently or even occasionally exposed, it would have a chilling effect on sites like WikiLeaks. This is the reason for whistleblower protection laws [28, 29] and the general recognition that journalists have the right (and perhaps obligation) to protect the identities of their anonymous sources [30].

A major difference between technological solutions that completely shield the source from identification and protections received from a single entity is that disclosure of an anonymous source may be compelled when malicious behavior can be identified and proven. Technological solutions such as ours prevent such recourse. However, since technological solutions are clearly needed, we posit that enabling censorship resistant communication is equivalent to preserving freedom of speech, however that freedom may be exercised. We ask the reader to keep this in mind while considering the whole of the following work.

Chapter 2

Membership-concealing overlay networks

In this chapter we present the fundamental building block of censorship-resistant systems — membership-concealing overlay networks, or MCONs. We show three candidate designs — one that is efficient in terms of communication, another is less efficient but is extremely robust, and a third is a hybrid of the two, increasing efficiency over the second scheme but at the cost of somewhat decreased robustness. We start by defining terms relationships between anonymity concepts in Section 2.1. In Section 2.2 we describe the requirements for a system to be considered an MCON, and discuss related work in Section 2.3. In Section 2.4 we show show proof-of-concept attacks against Freenet [8] and Tor bridges [7], two systems that implement membership-concealment-like features in an ad-hoc manner. Our design is described in Section 2.5, and we present a theoretical analysis in Section 2.6, and evaluate our system using simulation in Section 2.7.

2.1 Relationships Between Concepts

The concept of membership concealment is not new: organized crime and terrorist networks routinely use compartmentalization to hide the identities of cell members from people outside a given cell (a network is composed of many cells, which mostly act independently). Such networks are not foreign to the computer science community either: overlays with some membership concealment properties have been used for covert activity, such as sharing classified, censored, or copyrighted content. Generally called "darknets," these networks are built to be difficult to join or detect, but most do not protect from malicious insiders. One typically becomes a member through social means: an existing member "vouches" for the newcomer [31].

Academically, membership concealment networks have remained less explored than, and frequently confused with, related technologies such as *privacy*, *anonymity*, *unlinkability*, *unobservability*, *pseudonymity*, and *censorship resistance*.[1]

Unobservability

Related to anonymity, unobservability is usually endowed with one of two meanings. Pfitzmann and Hansen define the term to mean that a principal in an anonymity scheme cannot be "observed" to be sending or receiving a message (i.e. other nodes cannot

[1] For a thorough treatment of some of these terms, see [32].

determine whether a given node sent or received a message at any particular time) [32]. Some authors have interpreted this to mean that it is difficult to distinguish whether a principal participates in the network or not [33, 34]. The former clearly does not imply membership concealment: a scheme that is unobservable in this sense would remain unobservable if all principals periodically announced their participation. The latter sense is membership concealment in terms of an outsider-only attack, since it is generally necessary for *some* participants to be revealed to others in order for messages to be delivered. The extent of this exposure determines the level of membership concealment the network provides.

Pseudonymity and Anonymity

Pseudonymous credential systems [35, 36, 37] dissociate real-world identities from semi-persistent network identities (pseudonyms). A real-world identity is any information, such as participants' names, credit card numbers, or IP addresses, that may reduce the set of candidate identity-pseudonym pairings by a non-trivial amount. MCONs must use pseudonyms to address members, and *for a system to be membership-concealing it must be impossible, with overwhelming probability, to determine the real-world identity of a user given only that user's pseudonym.*

Anonymity, on the other hand, does not have the persistent identity property, but instead hides any and all identifying information. Consider the relationship between *anonymity* and *membership concealment*. The main goal of an anonymous network is to conceal who is communicating with whom. However, this *unlinkability* or "relationship anonymity" does not require concealment of *who participates in the overlay*, and a scheme with perfect relationship anonymity would not sacrifice this property if a complete list of participants was broadcast on a regular basis. MCONs clearly require some type of minimal pseudonymity to prevent a passive insider from simply harvesting identities — for example, messages should not include the real identity of the originator. However, MCONs do not guarantee or require anonymity or unlinkability, e.g. each message may contain the pseudonym of both its source and destination, destroying relationship anonymity but preserving membership concealment.

While aspects of some anonymity schemes in the literature can be seen as implicit efforts to provide membership concealment, e.g. Bauer's scheme seeks to hide the users

of a mix net among a larger set of web users [38], no deployed anonymity scheme explicitly claims to provide membership concealment, and it is largely accepted that sender anonymity (origin obfuscation) can be achieved without it [39, 40]. Some schemes, such as Tarzan [41], explicitly distribute a list of members. However, since this information simplifies certain variants of the intersection attack [42], recent P2P anonymity schemes such as Salsa [43] have mentioned hiding the membership list as a security goal. Unfortunately these schemes do not provide membership concealment under adversarial conditions.

While the goal of MCONs is different from that of censorship-resistant networks, *robust censorship resistance requires membership concealment*: if either an insider or an outsider adversary can determine the IP address of a given participant in a censorship-resistant system, that member can be blocked at the network border by an adversary that controls the network ingress/egress points. MCONs are designed to make this task difficult.

Censorship Resistance and Availability

Censorship-resistant networks are designed to prevent adversaries from denying users' access to a particular resource or file. Censorship resistance does not imply membership concealment: communication between two parties is blocking-resistant if they use a covert channel to communicate, but regularly announce that they are in contact with each other. Neither does membership concealment imply censorship resistance: MCON members may exchange unencrypted content, so censorship would simply require blocking messages that contain selected keywords, even if the identities of communicating nodes are hidden. This keyword-based blocking at the network layer is similar to the approach used by China's "Great Firewall" [44].

2.2 MCON Requirements

Informally, we define an MCON to be a *communication system that hides the identities of its members* from both insider and outsider attackers (network members and non-members, respectively), while retaining members' ability to communicate efficiently.

The goal is to reveal no information about the network participants that would allow them to be identified in the "real world." Honest nodes have one fixed network pseudonym, which allows other members to uniquely address them. (We will refer to overlay-level identities as "pseudonyms" and real-world identities as "identities" from now on.) For the purposes of this work, we assume that obtaining a node's network (IP) address is both necessary and sufficient to identify the real-world user of the network.[2]

In addition to hiding member information, this network must be robust to link failure and partitioning: we must maintain *availability* both in the presence of normal network events and attackers. (A related requirement is *node-equity*, i.e. no node is more important to the network than another.) It should also be scalable, allowing for the membership set to grow while maintaining routing efficiency and minimizing communication, computation, and storage overhead. Finally, it should provide efficient search functionality, which can reliably locate any information stored in the network within a predictable time window.

As Rhea *et al.* [45] point out, a rich set of overlay network operations can be built from a consistent put/get functionality. Thus a major functional requirement for an MCON is scalable support for the $put(\text{key}, \text{value})$ and $get(\text{key})$ operations. Scaling to large networks induces two secondary requirements. First, *put* and *get* should have low cost in terms of the network size to avoid high bandwidth costs — ideally, polylogarithmic in N. Second, any large group of users will experience some rate of churn — users going offline and coming back on — and thus the scheme should be resilient to churn, continuing to function in its presence.

2.2.1 Formal Definition of MCONs

We assume an adversary with the resources of a large ISP or state government. This means that the adversary can monitor or disrupt traffic on some fraction ℓ of links; can communicate with arbitrary nodes on the network; and can selectively "corrupt"

[2] If users voluntarily disclose their real-world attributes, then IP addresses become sufficient, but not necessary, to de-anonymize them.

or otherwise assume control of some fraction γ of selected nodes. We call this an (ℓ, γ)-adversary. Formally, we say that an overlay network protocol is (Λ, Γ, f)-membership-concealing if no (ℓ, γ)-adversary monitoring $\ell \leq \Lambda$ links and corrupting $\gamma \leq \Gamma$ members can identify more than $f(\ell, \gamma, N)$ members, where N is the total number of MCON participants. When $f(\ell, \gamma, N) = \Theta(\ell, \gamma)$ we call the protocol a membership-concealing network protocol. We note that no overlay protocol that permits communication between peers can be $(\Lambda, \Gamma, o(\Lambda + \Gamma))$-membership-concealing since at least one node must deliver messages to each corrupted or monitored identity, and an adversary can always choose to corrupt or monitor identities with no common neighbors.

2.2.2 "Open" vs. "Closed" Networks

As a brief aside, consider three types of membership schemes in access-controlled networks: *open* access, *controlled* access, and *closed* access. In an open network, any node can become a member at any time (usually just by contacting another member). In a controlled network, checks are performed to ensure that a node meets certain criteria for joining. In closed networks, only "approved" nodes can become members, e.g. some networks may require new members to be introduced to the network by existing members. Without loss of generality we can refer to a necessary condition for joining an MCON as having a non-replayable "token," which can be a proof of work, a certificate, or a voucher from an existing member. In the case of open networks, the only identifying feature of a node is its network (IP) address, but this information is not used to limit network membership. This situation reduces to defending against Sybil attack, which is provably impossible in the general case [15]. This leads us to the conclusion that it is not possible to design an open network with membership concealment properties.

2.3 Related Work

Arguably the first darknet was WASTE [46], released in 2003. It was designed to facilitate secure collaboration by small groups. Some file sharing applications have recently added darknet features [47, 48], and applications for "friend-to-friend" (F2F) sharing have been developed [49]. The latter scheme is meant to allow sharing through trusted intermediaries, preventing the disclosure of the uploader's identity. [48] and [49]

are also fundamentally different from previous darknet designs, since they hide the network member set even from other network members. Unfortunately, all of these systems share similar problems, such as forming partitioned groups instead of larger networks, scalability limitations, search efficiency issues, and security vulnerabilities.

2.3.1 Freenet

The system that is currently most similar to an MCON is Freenet [8]. It is a censorship resistant network which hides the publisher, querier, and storage location of files by obfuscating their names and contents, making it difficult for any party other than the querier to identify the content that is being retrieved. Moreover, Freenet uses recursive routing to reduce the number of nodes who are aware of each other's existence. (Recursive routing sends queries through a series of intermediate nodes instead of directly between source and destination.) Freenet version 0.7 is designed to allow for two modes of operation: "opennet" and "darknet." In opennet mode, nodes may freely connect to any other opennet node. Identities of other Freenet nodes are obtained through *announcements*, which are requests from peers to join some other node's routing table. A node replies to an announcement with its pseudonym and IP address if it has room to add the announcing node into its routing table, otherwise it responds with its IP address only and forwards the announcement through the network. Announcements can be made to arbitrary logical "locations" in the network, and since there are fewer nodes than available locations, a single node is responsible for responding to requests made to its own exact location as well as nearby locations. This response strategy is an inherent weakness in the membership concealment properties of Freenet; we show a proof-of-concept exploit in Section 2.4.

Darknet mode allows connections with other nodes only by prior out-of-band agreement, presumably based on mutual trust [48]. This provides protection from malicious nodes crawling Freenet for membership information. Note that because darknet nodes do not communicate with opennet nodes, there may be many disconnected darknets instead of one large network. Darknet mode incorporates *location swapping* as a part of its routing algorithm, wherein a node will compute the total distance between all of its friends and itself at the current location and at a new proposed location, to which it will potentially swap. The result of this distance measurement, along with a probabilistic

factor, is used to determine whether or not the swap will proceed. The swap is intended to improve search efficiency — decreasing the length of communication paths to peers reduces the number of logical hops required to reach a destination.

Routing and searching in Freenet is done by flooding. File transfers are recursive, and intermediate nodes will cache files. Freenet claims a search time of $O(\log^2 N)$, and a theoretical asymptotic minimum search time (in a long-running network) of $O(\log N)$ in an *open, non-darknet* network [48], but no concrete statistics are readily available. Moreover, since these calculations rely on caching (which allows nodes to service a request for content that they've seen before but are not the original storage point or location), locating rare files may prove extremely difficult in practice.

2.3.2 Tor Bridges

Tor [39] is a popular anonymizing network that offers sender anonymity. It employs a "client-server" overlay design, where a smaller, publicly known list of members relay traffic for a larger set of users, whose identities are neither explicitly revealed nor explicitly protected. The system is heterogeneous, with the server-like relays volunteering their bandwidth and processing power to forward data, and clients making use of these dedicated relays but not providing a service. The public nature of the router set, which is required for anonymity against various attacks, has implications for censorship: an adversary who wishes to deny access to Tor can download the relay list and block connections to the listed IP addresses. China did just that in September 2009 [14]. Tor designers have been actively working on "bridge" functionality [7] that would make Tor more difficult to block. Tor bridges are not dedicated relays; they are Tor clients who allow users in censored regions to contact them directly as a first step into the Tor network. Since bridges are client nodes, they are more numerous and experience higher churn than dedicated relays, so blocking them is a more difficult task. This is implicitly a membership concealment feature – one cannot selectively block what one cannot detect.

Tor currently relies on a publicly-known centralized authority or out-of-band (social) communication for distribution of bridge descriptors, but the authority can itself be blocked. Although the authority takes precautions to avoid providing bridge descriptors

en masse to anyone who asks,[3] the system is vulnerable to attack: an adversary who controls many IP addresses can query the authority repeatedly, pretending to be different nodes. While other defense mechanisms are in the works, they are not in place as of the writing of this document.

2.3.3 Other Systems

Other anonymity schemes [33, 34] have also attempted to provide "blocking resistance" by hiding their members among a larger set. However, even if an adversary cannot block access to all members of an overlay, he might be able to block queries for particular types of content. Since most storage networks provide an efficient lookup feature, an adversary knowing the identifying information of the content (hash, ID, etc.) can look up the node(s) storing that content and selectively deny access to those nodes. Censorship resistance requires blocking resistance, but it is orthogonal to membership concealment.

2.3.4 Using Social Networks to Bootstrap Trust and Mitigate Sybil Attacks

Social network-based Sybil-defense systems such as SybilLimit [16] and SybilInfer [16] attempt to distinguish Sybil identities from real users by exploiting hypothesized properties of social network graphs: if Sybil nodes form tight clusters in the network, these clusters may be detectable in a centralized or decentralized manner, and members of these clusters can be marked as malicious. SybilLimit, for example, conjectures that Sybil clusters are more sparsely connected to the rest of the social network than other, honest communities, and uses the number of inter-cluster edges as an implicit metric. Nodes with few such edges are believed to be malicious. This method requires assuming that social networks are fast-mixing — that when performing a random walk, the starting position is hard to determine with overwhelming probability from the position after a relatively small number of steps.

Danezis *et al.* also use social networks to bootstrap a Sybil-resistant DHT [50]. DHTs are structured overlay networks that allow for very efficient searching [51, 52, 53].

[3] If a client already knows a bridge descriptor, it can ask the bridge authority for the current IP address of that bridge. This will later become an important factor for an attack against the membership-concealing properties of bridges.

Each DHT node has a random pseudonym, and is responsible for responding to queries that are lexicographically close to that pseudonym. Each node maintains a routing table of $O(\log N)$ peers that allows it to efficiently identify the node responsible for a query in $O(\log N)$ hops. Based on the same assumptions as above — that adversaries are connected to a social network in few places compared to honest members — the Sybil-resistant DHT builds trust profiles for individual nodes along a query path and favors nodes who usually yield correct results. Since the majority of adversarial (Sybil) nodes will be connected to the DHT through very few honest nodes, those connection points will (with high probability) return Sybil nodes as next hops, eventually producing incorrect results when adversarial nodes misbehave. The trust profile of that honest node is then reduced, and little trust will be placed in results returned by queries going through that node in the future. By using multiple queries with different trust profiles, querying nodes can be confident that correct results will be returned.

Some systems use social networks as an explicit basis of trust — an edge between nodes implies a trust relationship. Freenet darknet [48] and Turtle [54] are examples of networks that *bootstrap from a social network that expresses mutual trust.* Queries are flooded and do not terminate until either every node in the network has responded or the maximal query depth is reached. Unfortunately, it is rare for social networks to explicitly express trust. Kaleidoscope [55] also uses social networks to distribute proxy information, mitigating Sybil attacks. However, Kaleidoscope also uses a centralized server to distribute information about proxies to newly-joining nodes, and so it represents the same centralized point of failure as the Tor bridge authority.

A recent system called OneSwarm aims to be a privacy-preserving P2P file sharing system by utilizing F2F and recursive routing in order to disclose who shares which files to only a trusted set of neighbors and yet allow searching the entire network for files shared by any user [49]. Unfortunately, the system shares some drawbacks with Freenet, Turtle, and Kaleidoscope — none of them is completely resistant to at least some membership-disclosure attacks. This is not entirely surprising, since most of those systems (except for Tor bridges, OneSwarm, and possibly Freenet) are not meant to be membership-concealing. We discuss possible attacks, as well as their results, in the next section.

2.4 Attacks on Existing Systems

The primary goal of MCONs is resistance to *member identification attacks*, in which either an insider (MCON member) or an outsider attempts to determine the "real-world" identities of network members. These attacks may take two forms: existential and targeted. In the former case, which can also be called the *harvesting attack*, an adversary attempts to determine the identities of as many network members as possible. The latter attack allows an adversary to match a network pseudonym with an identity, or to significantly reduce the number of candidate identities for a given pseudonym as a precursor to rubber-hose cryptanalysis.

Most existing systems are vulnerable to at least one type of harvesting attack. The simplest variant exploits systems that do not limit the number of identities that a single member can collect simply by querying the network repeatedly. Since the attack is active, it may be detected and the attacker could be blacklisted, but the adversary can always throttle or otherwise mask his actions to appear benign. A harder-to-detect variation is the *passive harvesting attack*: an adversary runs a network node that logs all direct communication attempts, learning the identities of all other nodes over a long-enough timeline. Both attacks become faster with more adversaries. Another example of a harvesting attack is the *multiple join*, or *bootstrap* attack, in which an adversary either sequentially joins the network multiple times at multiple logical locations (which are the Freenet equivalent of DHT IDs), or creates multiple (Sybil) nodes and simultaneously joins them to the network. Since every joining node must obtain the identity of at least one other network member, multiple joins allow the adversary to learn the IP addresses of a large fraction of network members.

The *celebrity attack* affects systems that use social networks to bootstrap trust, such as [8, 54, 55, 49]. If the social network topology can be discovered, then an adversary can choose to corrupt or monitor nodes with many friends, learning disproportionally many other network members. (Mislove *et al.* report node degrees of up to $10,000$ in many popular social networks [56].) Only a few very popular network nodes need to be corrupted or monitored in order to learn the vast majority of network members [57]. This can be generalized to attacks against "tasty targets," applicable when networks that bootstrap from social networks but do not "smooth out" node degree. It also

applies to networks with so-called "supernodes" — members who have more power than other members. An MCON should either not contain any targets of compromise that know disproportionally more member information than any other target, or should ensure that such nodes are difficult to identify.

Social networks also expose the constructed MCON to a graph overlap attack — Narayanan and Shmatikov have shown that anonymized graphs can be de-anonymized based only on topology knowledge and access to an overlapping non-anonymized graph [58]. This means we cannot anonymize a graph by simply replacing identities with pseudonyms; we must either restrict adversaries from constructing a complete view of the anonymized graph topology and/or perturb node degrees. While some of the above networks would resemble MCONs more closely if they were not vulnerable to the celebrity attack, most of them expose their topology while not enforcing node degree limits.

Another serious attack on a membership-concealing network is the *confirmation attack*. If an MCON requires nodes to respond in a distinctive way to connection attempts, then a non-member adversary can "cast a wide net" and identify a large number of nodes by attempting to connect to them. As an example, consider a network administrator at a large corporation who wishes to identify users on the internal network who are using a file-sharing application. Assume that the most popular application uses a certain port number in the default configuration. Our adversarial network administrator can "probe" each host on the internal network, connecting to that default port, identifying users by the tell-tale replies from the file-sharing client.

Finally, MCONs must resist *protocol identification attacks* when communicating with other network members as well as when joining, leaving, or inviting others to the MCON. Such an attack would allow passive identification of MCON users by monitoring communication patterns without peeking at content [59]. Consider once again our sneaky system administrator from above. Since there are only a small number of exit points from the internal network to the Internet, our adversary can monitor protocol traffic at those locations, identifying all users of the targeted protocol.

2.4.1 Attacking Freenet Opennet

The following attacks on Freenet opennet are instantiations of harvesting attacks described above, with the goal of breaking the hypothetical membership concealment properties. However, our attacks do not generalize to Freenet nodes running in darknet mode.

Freenet nodes have persistent *identities* and *locations*, which are the long-term pseudonym of a node and its logical placement in the Freenet network, respectively. Both are generated upon node creation and remain fixed throughout the existence of the node, but can be changed manually by editing the Freenet configuration file. Locations are represented by a double precision floating point number between zero and one, resulting in an address space of 10^{16} — far greater than the expected number of participating Freenet nodes.

We implemented a *passive harvesting attack* using well-behaved Freenet clients[4] in "opennet" low-security mode, whose only modified behavior is passive logging of communication with others.[5] Their pseudonyms and locations were randomly generated at creation time, and do not change throughout the experiment. We call these nodes "markers" because we use them to measure the success of our attack — since they have random pseudonyms, the time required to locate all marker nodes will be close to the upper time bound to find all nodes in the Freenet network.[6] We also implemented an *active harvesting attacker*, which announces itself to random logical locations in the network, collecting pseudonyms and IP addresses of responding Freenet nodes who are located "near" the announcement point in the Freenet logical coordinate space. To eliminate the effects of dynamic IP addresses we only counted node pseudonyms, which are unique and constant.

Figure 2.1 provides a comparison of each of our attacks on Freenet, using 80 marker nodes and a single announcer. Announcements were broadcast at a rate of once per second. When increasing to 10 per second, we did not observe significantly better results, likely due to the Freenet throttling feature, which inhibits replies to nodes that flood the network with requests. In fact, 50 announcements per second collected fewer

[4] Based on Freenet 0.7 Build #1204 r25665 (2-17-2009)

[5] Our attacks require no customization of the Freenet client other than to facilitate logging.

[6] There is no way to be certain that any given attack has uncovered every node, especially considering that some may be down for the entire duration of our experiment.

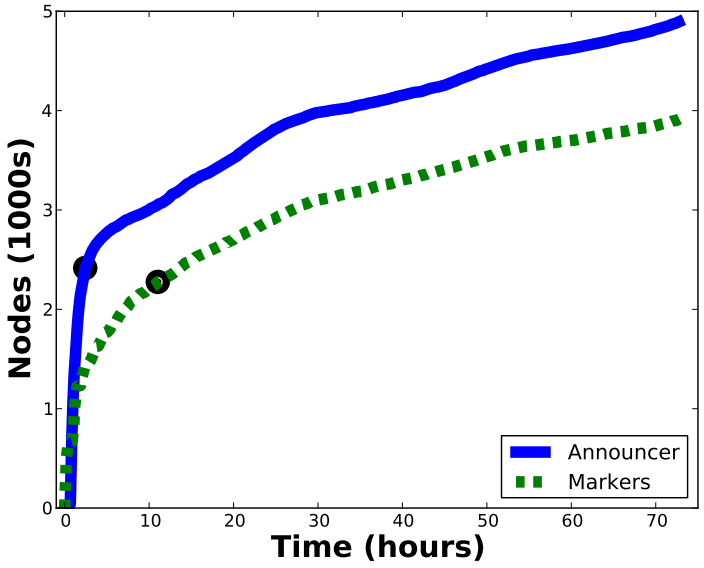


Figure 2.1: Total unique Freenet nodes found over time. Dots show the time when all marker nodes were found.

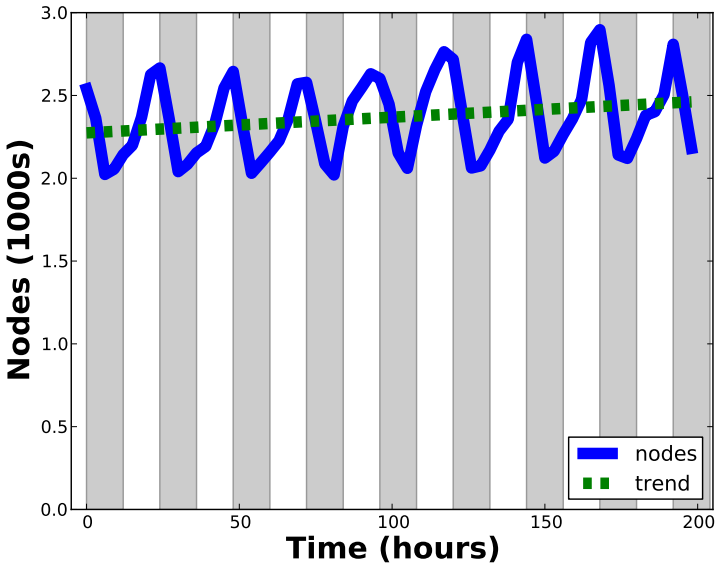


Figure 2.2: Unique running Freenet nodes for each 3-hour time period. The cycle is likely the result of day-time versus night-time usage patterns.

node references than the one-second attacker. (Data for the 10- and 50-announcement attackers is not included in the figure.) Both attacks collected new unique pseudonyms at an ever-decreasing rate, with a combined total of 7,794 unique pseudonyms found by the markers in 380 hours, and 7,317 pseudonyms found by the announcer in 200 hours.[7] Each individual marker node was only able to find an average of 51.03 of the other 79 marker nodes, with a range of 24 to 63 and standard deviation of 7.93. However, combining data from all markers shows that when taken together, they also found all other makers, but in 11.03 hours. The single active attacker outperforms all 80 passive attackers in terms of speed, but collection speed can always be increased by adding more attackers — the bandwidth and processing costs are not a bottleneck.

The dots in Figure 2.1 represent the time when each attack discovered all 80 marker nodes, signalling that we have likely enumerated the majority of running nodes. Our passive attackers were able to enumerate all markers in 11.03 hours, and the active attacker found all of them in 2.43 hours. Figure 2.2 shows the membership of opennet derived from snapshots of 3 hours each — since it took our announcer less than 3 hours to find our markers, we expect this graph to be an accurate measure of the membership of opennet at any given time. We observe an interesting cycle-like pattern, and if we assume that most users are online during the day and some users offline at night, we conjecture that most Freenet users are likely European or British, since the low part of the cycle is around 3am GMT and the high at around 8pm. (The shaded areas are 8pm to 8am GMT.) There is a slight upward trend of 0.95 new nodes every hour, possibly attributable to the expanding membership space of opennet as more users join the system.

We observed between 2,000 and 3,000 *running* opennet nodes at any given time. While the total number of *existing* opennet nodes cannot be counted accurately since a large number are likely to remain offline for the duration of the experiment, we discovered a total of 11,100 unique node pseudonyms. We expect that several users join transiently, only casually using the system. Note that these attacks do not generalize to darknet mode, whose interaction with opennet is not well-documented. It is therefore unclear how the interplay between opennet and darknet affects the connectedness or

[7] At around 200 hours into the experiment, our announcer becomes unstable because its peers disconnect themselves due to our ambitious announce rate.

security of Freenet.

It is interesting to note that Freenet *resists confirmation attacks* by requiring that any message to a Freenet node be encrypted with that node's public key, thus requiring either a copy of a message the node previously accepted, or prior knowledge of the node's identity before the node can be queried. Additionally, since Freenet uses UDP (a stateless, connectionless protocol), an adversary cannot even check if the default Freenet port is open on any given Internet-connected computer, since messages not encrypted with the node's key simply receive no reply.

2.4.2 Attacking Tor Bridges

We also launched a *passive harvesting attack* against Tor bridges using an unmodified Tor router,[8] configured as a middleman (non-exit) node and offering 100MB/sec of bandwidth (attracting disproportionally many client connections).[9] Our router should receive connections from clients, bridges, and other Tor routers. We weed out routers using several tests, including TLS handshake fingerprinting, querying all running directory authorities to see if they know the router, as well as by connecting back to the router and examining its descriptor [61]. Once we eliminate the routers we are left with clients and bridges. To differentiate between them we attempt a connection using common bridge ports. If our connection succeeds and we get a descriptor [7], we launch a *confirmation attack* by extracting the fingerprint from the descriptor and querying each directory authority. A router will appear in at least one, while bridges will not. (Note that these tests are all performed in real time.) Since we expect that all bridges will eventually connect to our router due to Tor's selection rules, we can eventually build a complete list of bridges. We collected 61 unique Tor bridge identities in 4 days, a clear vulnerability in the membership concealment capability of Tor bridges. Like the Freenet attack, we can increase the speed of collection by running multiple routers. It is once again impossible for us (or anyone without access to the Tor bridge authority) to definitively state that a given list of bridges is complete.

The Tor bridge specification [62] states that the bridge authority will respond with a bridge's descriptor when asked with a valid fingerprint, but we did not observe this

[8] Based on Tor 0.2.1.11-alpha r18192 (1-20-2009)

[9] Although directory servers cap advertised bandwidth at 10MB/sec [60].

behavior. Had the bridge authority been following specification, its response alone would have confirmed that the node is a bridge. If we cannot definitively determine that a given node is a bridge, it is not included in the bridge count.

2.5 Design

This section outlines three proof-of-concept MCON designs. As in several previous works [54, 55, 50], we use a social network based on offline relationships as a starting point. We bootstrap our MCON from a small fully-connected "seed" network of "social neighbors" (nodes connected by an edge in the social network). This allows us to mitigate the Sybil attack problem [15], based on the observation that that a single person will have trouble convincing others that he or she represents different real-world identities. To this end, we assume that for each physical person in the world there exists at most one unique social network member with high (but not overwhelming) probability. That is, we allow a fraction of people to have multiple identities in the social network. Social networks also allow us to exploit the small-world property to ensure that the distance between any two nodes is small.

The MCON grows by having existing members invite new nodes based on social relationships. When joining, nodes are assigned persistent pseudonyms and DHT IDs. After the MCON is built and after a period of DHT routing table discovery, we use a VRR-like protocol [63] to allow nodes to communicate with the DHT recursively through a small set of "physical neighbors." We define physical neighbors as those nodes who are allowed to directly communicate over IP. (This system may be considered a double overlay — we use DHT communication for efficient search, over a source-routing overlay, over IP.) To avoid celebrity attacks, every MCON node can only communicate directly with a constant k other nodes, since direct IP communication is sufficient to break membership-concealment. Most communication takes place through the DHT overlay, which connects any two nodes by $O(\log N)$ logical hops, where N is the total number of members. This contributes significantly to the scalability and efficiency of our system.

Many popular DHT designs use iterative routing, where a node will communicate with its DHT neighbor to ask for the IP address of the next DHT hop, with which it will then communicate directly. The process repeats until the desired node is found,

at which time the origin and the destination can communicate directly over IP. We cannot achieve membership concealment in the iterative scheme, since an adversary can learn the IP addresses of all intermediate nodes as well as the final destination by repeatedly searching the network. Moreover, as Mittal and Borisov [64] and Kang *et al.* [65] point out, iterative routing implies that an adversary who controls m nodes will see a $1 - (1 - m/N)^{\log N}$ of all queries; if $m = \Omega(N)$. This means iterative schemes cannot be made to work for MCONs, since a small number of adversaries will be able to monitor all queries and assemble a list of network participants. In recursive routing, nodes communicate only with their DHT neighbors, who forward requests to the next DHT hop on behalf of the originator. In this scheme, all communication between source and destination happens through multiple intermediaries. Sometimes recursive routing also carries the benefit of plausible deniability of query origin — a node receiving a message from a physical neighbor cannot distinguish whether that neighbor originated or forwarded the message.

Our design relies on a trusted central authority (the Membership and Invitation Authority, or MIA) to invite new nodes into the MCON and act as a key issuer. The MIA is also responsible for keeping track of node degree, ensuring that nodes do not exceed the global constraint. Since future designs will distribute the functionality of the MIA throughout the network, we want to minimize our current dependence on it. To that end, the MIA is only needed when a new node joins. Moreover, it does not have to respond in real time, and so can be offline and does not constitute a central point of failure *for denial of availability attacks*. Unlike the Tor authority, nodes need never contact the MIA directly, so it can remain hidden.

To prevent Sybil attacks, the authority can use existing systems such as SybilLimit or SybilInfer [16, 17], which use a social network to bound the number of Sybil identities accepted into the network. Note that *membership concealing properties of our scheme do not depend on the number of Sybil nodes in the network*, provided they are not connected to honest nodes. Since honest nodes will not directly communicate with anyone other than their neighbors, Sybil nodes without edges to honest nodes will only affect the robustness of routing in our network, not its security.

Below we present three MCON designs: the first is more efficient, the second is more robust in high-churn situations, and the third is a hybrid. They all function similarly,

and all functionality can be split into three major categories: *invitation and join*, *route discovery*, and *overlay routing*.

Invitation and join. The network is built by starting from a small "seed" and adding nodes one by one, expanding it to form the full MCON. While the seed can be an arbitrary group of social network nodes matching certain mutual connectivity parameters, growing that network is challenging. In our system, nodes who are already part of the MCON invite other nodes with whom they share connections in the social network. Nodes must receive multiple invitations in order to join the MCON, and the entire process must be somehow mediated to ensure admission control and key distribution for the MCON. While this is currently handled by a centralized entity, future designs will incorporate distributed computation of this information by MCON members.

Route discovery. Once node A has been admitted to the MCON, it must construct a DHT routing table for efficient communication. The routing table consists of source routes to other DHT nodes that share different prefix lengths with A. Routes are discovered by flooding requests over the "physical" network. Since nodes only communicate with their neighbors, route responses must conceal information about intermediate nodes. We accomplish this by using *private routing tokens*. A node building a routing table obtains information about the next hop (one of his direct neighbors), the destination (one of his DHT neighbors), and no information about the set of intermediate nodes, other than its size.

Overlay routing. Finally, once a node builds its DHT routing table, it can route to arbitrary DHT keys. As in route discovery, communication happens strictly through the node's "physical" neighbors, and DHT communication is recursively routed. MCON communication consists of two layers: *routing to a DHT hop*, and *routing between DHT hops*. In the first step, the node uses the collected private routing tokens to deliver a message to the first DHT hop. That DHT node will then use its routing table to transport the message to the next DHT hop, and so on until the destination is reached. Nodes never communicate directly with anyone other than their physical neighbors, and layers of encryption prevent the exposure of the DHT message as well as the source and destination. We ensure resistance to confirmation and brute-force scanning attacks by using *strong binding* — an MCON node will only communicate directly with her

physical neighbors, ignoring all messages from other nodes (enforced by cryptographic signatures and discussed further in Chapter 3).

2.5.1 Efficient Design

Network Construction

We start network construction with a clique of $\lceil k/2 \rceil$ social network neighbors, where k is the maximum number of allowed MCON physical neighbors. The MIA iteratively "grows" the network by finding nodes to invite. Node A can be invited if: a) A has at least $\lceil k/2 \rceil$ social friends in the current MCON, b) those friends have at most $k-1$ "physical neighbors," and c) A is not in the MCON, and has not been previously invited. Call A's friends satisfying (a) and (b) her potential physical neighbors. Once A has been identified, the MIA randomly chooses $\lceil k/2 \rceil$ of A's potential physical neighbors, tells them A's new pseudonym, and instructs them to (1) add A to their list of physical neighbors and (2) send an invitation to A with their IP addresses, MCON pseudonyms, public keys, and DHT ID. (We will later use the ID-based keys to emulate fuzzy identity-based encryption [66].) Once A receives the invitation and joins, the MIA assigns her a private key and a set of identity-based private keys [67]. Note that the MIA is in a position to both enforce that no node joins the MCON more than once for any given social network identity, and that every MCON member's node degree is no greater than k.

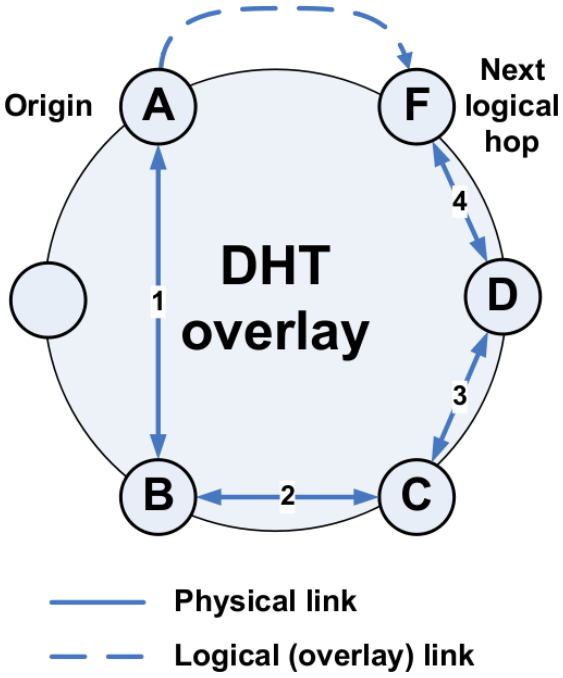


Figure 2.3: Routing to a logical hop over 4 physical hops

Route Discovery Request

After joining, A can discover her logical neighbors and build a routing table. The routing table is composed of source routes to DHT nodes whose pseudonyms share a common prefix with A's pseudonym. (For instance, in the Kademlia DHT protocol [53], A acquires K routing table entries for every i-bit prefix of A's pseudonym, for a total of $\log N$ entries. We must use a fixed number since we are not certain about the size of the MCON. For instance, if MCON pseudonyms are 128 bits wide, A might have 16 buckets, the first being a 16-bit match with her pseudonym, the second with a 32-bit match, and so on.) This is a difficult feat to accomplish while concealing a node's identity from the rest of the network. The solution is to only communicate with the node's neighbors — only nodes who know each other in the social network may communicate directly. So, source routes are discovered by scoped flooding over the physical links of the MCON, analogously to a wireless ad-hoc routing protocol, such as Virtual Ring Routing [63]. A continues sending discovery requests, increasing the scope of each by one, until her entire routing table is filled. (This constitutes a depth-limited breadth-first search of the network.) While expensive, floods are only needed during initial route discovery. A uses these source routes to establish onion routes to each of its routing table entries, similar to Tor tunnels [39]. Onion-wrapped source routes ensure that for most routes neither the source nor the destination learn anything about each other except that they share a common pseudonym prefix. Moreover, most intermediate nodes in a source route know neither the source nor the destination, making sender-receiver unlinkability a likely (but not guaranteed) property of this protocol. Note that messages using the same path are still linkable to each other by each node along that path using the path keys themselves,[10] but outsiders or non-adjacent nodes along the same path cannot determine whether two message they have seen are traveling over the same path.

The cryptographic requirements for discovery message are:

- The final recipient of the message must be authenticated as someone sharing a common pseudonym prefix (of a given size) with the originator
- The random route string must only be disclosed to the above-mentioned node and

[10] This is unavoidable while using private routing tokens, since they are not verifiable by any party other than their creator, and so could themselves be marked to link messages which use those tokens.

the first node on the return path

Route discovery messages are in the form of $(\mathrm{ID}, \mathrm{scope}, g^x, IBE_{\mathrm{prefix}}(z, R))$, where IBE is identity-based encryption [67], scope is the flood depth, g^x is a Diffie-Hellman half-key,[11] R is a route descriptor, z is a random number, and ID is $h(h(z))$, with h being a cryptographically secure hash function. IBE_{prefix} is an identity-based encryption to an i-bit prefix of A's pseudonym [67], meaning that only a node matching the search parameters can open the message. The route descriptor is a random bit-string of some fixed size. A stores z, the prefix, the route, and the DH half-key for later reference. When relaying a route discovery message, each node will decrement the scope by one, dropping messages whose scope is 0. Relaying nodes will also record the request ID and physical neighbor from whom it came. These records are kept either until a reply is received or a timer expires. If identical requests are received from multiple neighbors, all their identities are stored.

Route Discovery Reply

When node F receives a route discovery request which he can decrypt (meaning F's DHT ID contains the prefix to which the message is encrypted), he generates a DH half-key and composes a response in the form of $(\mathrm{ID}', g^y, R, E_k(z), \mathrm{MAC}_{k'}(E_k(z)))$, where k and k' are keys derived from the full DH key, i.e. $k = h(0, g^{xy})$ and $k' = h(1, g^{xy})$, and ID' is $h(z)$, the pre-image of the request ID. The response also includes R from the route discovery message (unchanged), and a message authentication code (MAC) of z. F sends this response to D, and also floods the original request, decrementing the scope. The resulting source route is shown in Figure 2.3.

The cryptographic requirements for the discovery reply message are:

- All nodes along the path of a discovery reply message and the originator of the discovery message can confirm that it was correctly decrypted by its intended destination node
- No node other than the one originating a route token should be able to derive meaningful information from it

[11] If the shared DH key is g^{xy}, where x and y are private keys, then one DH half-key is g^x and the other is g^y.

- The source and destination of the discovery message must derive the same shared key

Once D receives a route reply from F, he looks up $h(h(z))$ in a table of previous request IDs to verify that the request was correctly opened and to find the next hop where the response must be sent. He constructs a "route token," encrypts it with his public key, and prepends the resulting ciphertext to the route contained in the response. He also removes an equal-length token from the end of the route string. This "padded" route string works in the same way as padding for onion routing cells [68]: the originator creates a route data structure of a fixed length larger than required to store the returned route. When a node adds itself to the route it composes a routing token consisting of a hash of the next and previous hops along that route and a nonce, and encrypts it using his public key. He the prepends it to the beginning of the route data structure, removing the equivalent number of bits from the end. Since the initialized route string is longer than the actual route, the token removed from the end will be a random one. The source route received by the query originator will thus consist of a series of meaningful tokens followed by random tokens. (Note that each non-random token is only meaningful to the node that composed it.)

The new token identifies the next and previous hops along that route, and can only be decrypted by D. The encryption should include a random component to prevent every token that points to the same node from being identical, since the route is visible to intermediate nodes. D then sends the response to the appropriate physical neighbor. In this way, the originator of the request (A) will get back a series of tokens that are meaningless to her, but that comprise a source route to her logical neighbor. Additionally, since A generated the original random route string, she can ensure that it passes some rudimentary sanity checks: she knows the exact length of the route since she iteratively increased the scope of the flood, allowing her to check that only the correct number of routing tokens have been changed. An incorrect count would indicate that someone is not following the protocol, and the route should be discarded (this is possible since A would either get multiple replies for a single prefix match and can thus pick correctly-formed ones, or she can simply continue the search).

Since A is likely to get multiple replies to a route request, she must arbitrarily select full-bucket subset of all received replies. She cannot fill part of her bucket from scope

x and the rest from scope $x + 1$ since nodes who responded to requests with smaller scopes will again respond to requests with larger scopes, leading to duplicate bucket entries. We must be careful to defend against multiple adversarial replies, although any such countermeasure would ultimately prove futile since a single adversary "close" to A may respond on behalf of any and all adversaries in the network. To reduce the number of adversaries in A's routing table, she should select at least one reply from every physical neighbor, picking a random subset of that neighbor's replies. The intuition behind this strategy is that the distribution of DHT IDs should be similar independent of which "direction" in the network the request is routed, so the number of responses from each physical neighbor should be comparable. A large response set may indicate an adversarial node, especially if more than enough responses to fill a single bucket come from only a single node. We avoid wormhole attacks [69] since A already knows the cost of each route — the scope of the flood — and thus knows that each route returned for a given scope has the same cost upper bound.

After route discovery, A sets up shared keys with each node along the physical route in a process similar to constructing a Tor tunnel [39]. While A does not know the identities or pseudonyms of nodes along the route and cannot authenticate their DH keys, the final DH key (shared with F) is authenticated, since only a node with a given pseudonym prefix could have decoded the half-key. A can use this information to detect man-in-the-middle attacks and discard compromised routes. Key agreement proceeds recursively, starting with A contacting the first node in the return route and deriving a shared DH key. A will next provide that node its routing token, and initiate key agreement with the next node in the chain. The process repeats until A reaches the logical hop, with which it already shares a key, and can thus authenticate as the correct end-point. Note that a node performing a man-in-the-middle attack will be able to learn all the routing tokens. However, this attack would be detected, as key agreement will not terminate at the correct (authenticated) end node in the correct number of rounds, and the route will be discarded, invalidating the exposed tokens.

The cryptographic requirements for key agreement are:

- All nodes along the path of a discovery reply message must derive different shared keys with the source of the discovery message

- A route token should only be visible to the node originating it
- Man-in-the-middle attacks must be detected and the route discarded, since these attacks cause route tokens to be exposed

DHT Routing

Once the logical routing table has been built, MCON routing is identical to DHT routing, with the exception that messages to every logical hop must traverse a number of physical hops. To prevent examination and tampering, those messages are encrypted and MACed. Routing to a single logical hop in the efficient design is sketched in Figure 2.3.

When searching, A hashes the search term to determine the destination (X) and the closest logical hop in A's routing table (F). She then retrieves the DH key shared with F, along with the source route (B, C, D) and the associated keys shared with each physical hop. A uses identity-based encryption to encipher the hash of the search term so only a node logically close to X can open the message. (Authenticating the final logical hop prevents arbitrary DHT nodes falsely claiming responsibility for a given key.) She composes a message containing the resulting ciphertext, encrypts it to F using the shared DH key, attaches a MAC, and onion-wraps it such that each physical hop must remove a layer of encryption to forward the message. Route tokens are included in the onion-wrapped portions so that each hop only receives its own token.

DHT (overlay) messages are in the form of
$(\mathrm{ID}, E_{k_1}(\mathrm{token}_1, E_{k_2}(\mathrm{token}_2, \ldots (E_{k_s}(\mathrm{token}_\mathrm{s}, E_{k_s}(M))), \mathrm{MAC}_{k'}(ID, E_k(M)))))$, where k_i is the shared key between the source and the ith physical hop in the route and ID is the message identifier, followed by repeated layers of onion encryption containing route tokens. s is the path length (the scope of the message when it reached the replying node). The inner-most onion layer is composed using the verified DH key k shared with the logical hop and contains a message M for the final DHT hop. M is in the form of $L_1, IBE_{L_1}(L_2, IBE_{L_2}(\cdots (IBE_{L_n}(h(\text{search term})))))$, where L_i is the DHT (logical) destination prefix. Each logical hop will be able to remove a layer of identity-based encryption corresponding to its neighborhood to reveal only the next logical hop, but

not the message payload. Only a node in the same neighborhood as the query destination will be able to decrypt the full payload.

The cryptographic requirements for DHT messages are:

- No node along the source route should be able to derive any information about the contents of the message to the logical hop
- Only a logical hop in the neighborhood of the destination should be able to decrypt the DHT message payload
- No logical hop in a neighborhood other than the destination neighborhood should be able to derive any information about the contents of the DHT message payload
- A route token should only be visible to the message source, and the node that composed the token

Time Padding

Onion-wrapping messages and randomizing routing tokens provides some protection from A, B, and C linking messages or learning the MCON topology. However, any one of them may monitor the amount of time between query and response along a given source route, and can deduce the magnitude of the ID prefix match between the source and destination nodes. (Messages sent to an early logical hop along a route will take a long time to return a result, while messages sent from the last logical hop to the query destination would see an almost immediate response.) We can prevent this attack by asking each logical hop to delay query responses for a fixed amount of time. Since the number of logical hops required to complete a query is uniform for a given network size, and since each logical hop knows its logical distance from the destination, the required delay can be estimated within a factor of 2. This increases the total time required to receive a response to every query, but also ensures that the time is constant (so query failure is easy to detect).

If the query originator or a logical hop fails to receive a response (within a timeout period) from the next logical hop (which can be the result of failure at the logical hop or any physical hop along the way), it picks the next best logical hop and repeats the attempt, until it either succeeds (receives a response) or runs out of logical hops to try.

In the latter case, it would admit failure, and not return a response. While this is not a complete solution for churn, it does provide a certain level of robustness against offline nodes and packet loss. A more robust scheme is discussed in the next section.

2.5.2 Robust Design

Robustness is somewhat tricky to achieve in the efficient design, since a single offline physical hop along a source route renders the entire source route unusable. In this section, we discuss a design that trades increased robustness for decreased efficiency and larger number of disclosed IP addresses. We employ what we call a "skipping stones" approach:[12] in addition to sending a message to a single hop along a physical route, the message is sent to *each neighbor of that hop*. Each of those nodes sends to each of the neighbors of the next physical hop, and so on. This reduces the probability of failure because only one node per "neighborhood" needs to be honest and online in order for a message to get through. To that end, all MCON nodes must know not only the IP addresses and cryptographic keys of each neighbor, but also addresses and keys of each neighbor's neighbor. The MIA will reveal that information during the bootstrap phase. Furthermore, the entire neighborhood needs a shared key for use with route tokens. This key can either be given out by the MIA or agreed-upon by neighborhood members using a key agreement scheme such as in [70].

Although DHT routing does not change in the robust scheme, we must alter our physical hop routing and discovery to accommodate neighborhood-wide routing decisions. When a neighborhood receives a route discovery reply, a majority of neighbors must come to a consensus regarding the contents of their routing token (and thus the previous and next hops for the reply). They sign the agreed-upon token using a threshold signature scheme [71],[13] which requires m out of n nodes to partially sign a message before a full signature can be derived. For a simple majority, m would be $\lceil \frac{n+1}{2} \rceil$. Each node can then independently encrypt the signed token with the shared neighborhood key, prepend it to the route reply, and forward it. Note that majority agreement is required only during route discovery and during shared key exchange with the originator.

[12] A stone skipped over water makes contact with the surface repeatedly, creating ripples at each contact point.

[13] Threshold signatures allow some nodes to disagree or be offline during route discovery.

Multiple route replies are handled the same way as in the efficient scheme, but now they are far less likely to be malicious since multiple nodes must agree on the route — a malicious majority at one of the intermediate neighborhoods would be required to produce a compromised route.

In order to send a message in the robust scheme, the originator sends messages to each neighbor of the next physical hop along the route. When a message arrives, each neighbor can independently decrypt the enclosed routing token (using the neighborhood key) and verify the signature to ensure it is correctly formed. Since only the final destination can determine message validity by verifying the enclosed MAC, intermediate nodes will not know if a message is legitimate. If an intermediate node gets different messages with identical IDs, it must forward one of each copy, potentially increasing the number of messages proportionally to the number of adversaries encountered en route.

While offering superior robustness under heavy churn, this scheme has higher overhead than our efficient scheme. We face a constant-factor increase in the number of real-world identities every MCON member knows, since every node must now keep track of the IP addresses not only of its physical neighbors, but also of their neighbors. However, since the number of identities each node knows is still constant, this does not compromise membership-concealment. We also lose plausible deniability: any one of a node's neighbors can perform packet counting [72] and timing attacks [39] to determine if a message is being forwarded or originated. However, we can recover plausible deniability by using cover traffic.

2.5.3 Hybrid Design

The hybrid scheme maintains most of the robustness properties of the previous scheme while significantly reducing communication costs. We take a similar approach to Saia and Young [73] and modify our robust scheme such that nodes discover the identities of their neighbors' neighbors only if $h_1(ID_1) \mod m = h_2(ID_2) \mod m$ for some small constant m, where ID_1 and ID_2 are the DHT IDs of the two nodes. If the equality does not hold, nodes simply do not learn about each other. Since introductions are handled by the MIA, this invariant is trivial to enforce. Intuitively, this design probabilistically guarantees that every node of the next neighborhood receives at least one copy of each

message. As we increase the modulus m, fewer messages are sent and robustness decreases. However, this reduction is acceptable when we consider that message overhead (combining communication time, bandwidth, and cryptographic overhead) is reduced by a factor of m.

2.6 Theoretical Analysis

Our MCON designs do not share the flaws of existing schemes such as Freenet [8], Tor bridges [7], Turtle [54], or Kaleidescope [55]. The latter two, being based on social network, are susceptible to targeted corruption and celebrity attacks since nodes are not degree-constrained, and therefore some are "tasty targets" for compromise, which would lead to the discovery of a non-trivial fraction of network members.[14] Freenet opennet is vulnerable to the same attacks, and also to both passive and active harvesting. Bridges are vulnerable to confirmation and passive harvesting attacks. Moreover, our designs likely provide sender-receiver unlinkability, and our efficient design provides plausible deniability. Unlike Freenet and OneSwarm [49], our search completes within a guaranteed time bound while making rare files as easy to find as popular files.

2.6.1 Search Time

While Freenet claims a search time of $O(\log^2 N)$ in either open or darknet mode, and a theoretical asymptotic (in a long-running network) minimum search time of $O(\log N)$ for popular files in an *open, non-darknet* network, no concrete statistics are readily available. DHT, on the other hand, guarantees an average search time of $O(\log N)$ logical hops. When incorporating physical hops, this becomes $O(\log^2 N)$ in the worst case. Incorporating caching should help this situation, and should bring this value closer to $O(\log N)$ in a long-running network. Note, however, that the Freenet limit is a theoretical asymptotic limit while the DHT limit is a probabilistically guaranteed one, and that we do not require flooding during search to achieve this result.

[14] We note a celebrity could split her contact lists into many nodes with a small number of neighbors each, and remain a logically tasty target while maintaining a low target profile at the network layer. This attack is unlikely at the social layer, since a celebrity must maintain her celebrity status to get contacts, and any system that enforces a maximal node degree at the membership concealing layer will not create multiple pseudonyms from a single social network-level identity.

2.6.2 Membership Concealment Intuition

To verify that our designs do not fall victim to identity disclosure, we check that 1) only physical neighbors communicate directly over IP (preventing *harvesting*), 2) no adversary can query arbitrary Internet hosts or otherwise elicit an IP-level response identifiable as an MCON message (preventing *confirmation*), and 3) no adversary learns the identity of a node who does not directly connect to corrupted or monitored nodes (preventing information *leakage*). In our system, (1) and (2) are handled by the strong binding property — nodes will only respond to messages that are signed by their physical neighbors, and neither initiate nor respond to IP-level contact with any other nodes using the MCON network protocol. (3) presents a greater challenge: a powerful network-monitoring adversary may monitor not only individual nodes but entire networks, and use some encryption-oblivious fingerprinting technique to identify MCON members [59]. The defense is protocol-level obfuscation (steganography) such as used in [74], which, while not explicitly implemented in our current system, is a natural extension. While we impede graph de-anonymization attacks by perturbing the maximal MCON node degree, making it independent of nodes' social degree, our main defense is to prevent topology exploration by both insider and outsider adversaries. The success of the latter mechanism depends on the quality of traffic obfuscation.

Recall our (ℓ, γ)-adversary, who can monitor ℓ links and can corrupt γ network members. Since he can only learn k additional members from every member he corrupts or monitors, he is limited to learning at most $k\ell + k\gamma$ correctly-functioning members ($k^2\ell + k^2\gamma$ in the robust scheme). Without protocol obfuscation, we say that our network is $(\frac{N}{2k}, \frac{N}{2k}, f)$-membership-concealing for $f(\ell, \gamma, N) = k\ell + k\gamma$, since $f(\ell, \gamma, N) = \Theta(\ell, \gamma)$, where N is the total number of MCON participants. In the robust scheme, the network is $(\frac{N}{2k^2}, \frac{N}{2k^2}, f)$-membership-concealing for $f(\ell, \gamma, N) = k^2(\ell + \gamma)$. If we use protocol obfuscation then membership hiding properties will depend on the details of the steganographic system, but with perfect obfuscation our efficient network would be $(N, \frac{N}{k}, f)$-membership-concealing and our robust network would be $(N, \frac{N}{k^2}, f)$-membership-concealing for $f(\ell, \gamma, N) = k\ell$ and $f(\ell, \gamma, N) = k^2\ell$, respectively.

2.6.3 Churn

Churn, or the constant leaving and re-joining of nodes, causes problems in peer-to-peer networks — nodes in such networks are not expected to be long-lived, and if all of a peer's contacts go offline, the peer will be disconnected from the network and must re-join, discovering new (online) network contacts in the process. Churn is particularly problematic in MCONs because disconnected nodes are not allowed to acquire new MCON contacts and *any* level of churn reduces the efficiency of our routing scheme by invalidating some optimal routes whose member nodes are offline. Node degrees in the MCONs must be large enough to handle churn, and yet small enough to minimize identity exposure. We use a very strong churn model in our analysis: we do not assume any relationship between the online status of a node from one moment to the next, i.e. any node has the same probability of being offline at any time, independent of its previous online status. While we do not currently consider nodes who permanently leave the MCON, we can add an MIA-mediated revocation system — nodes who have been offline for a long time can have their keys revoked. Neighbors of those nodes can then be allowed to acquire more neighbors, since they will still not know more than k MCON members — the revoked node no longer counts among the member set. Neither do we consider social network churn because we do not use social network edges as trust relationships. Therefore, the loss of a social network edge need not affect the topology of the MCON. As for new edges, we support issuing invitations as long as all other conditions, such as node degree, continue to hold. The MIA can discover such edges as they are created.

Connectedness

A node becomes disconnected when all of his physical neighbors are offline. Assuming nodes come online and go offline independently of each other, the probability of disconnection is c^k, where c is the churn rate and k is the MCON node degree limit. This means that at $k = 7$, around 90% of MCON nodes have to be offline for a node to be disconnected half the time in our efficient scheme. The chance of disconnection in the

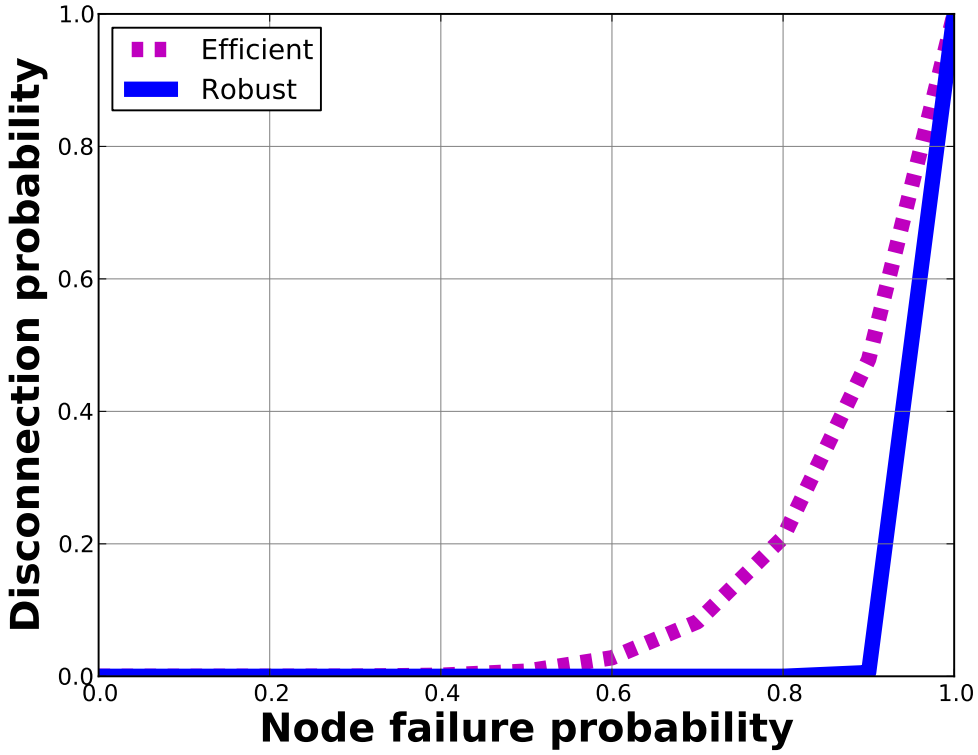


Figure 2.4: Estimated probability of node disconnection. Churn is the fraction of MCON members who are offline.

robust scheme is not significant when churn is less than 90%. This is shown in Figure 2.4. Analysis of Freenet data shows a churn rate in the vicinity of 70%,[15] meaning that we can be almost certain that nodes are always connected in our scheme. However, a problem can occur that causes nodes to permanently lose track of each other: if node A goes offline, and B, A's physical neighbor, goes offline sometime later, and they both change IP addresses before returning, they will have no way to communicate with each other when re-joining. The solution is to have nodes periodically publish their signed IP addresses to a known DHT location, combined with a random value, encrypted with their physical neighbors' keys. This ensures that physical neighbors can always keep track of each other while providing no information to unauthorized parties.

Reachability

Nodes cannot reach a network destination (even if both nodes are technically connected) if there is no DHT route between them. This may happen if all required DHT hops are down, or if they are not reachable through source routes. In our efficient scheme, the probability that all nodes along a given source route are up and forwarding packets is $(1-c)^d$, with offline probability c and route length d, which is at worst the network diameter. The robust scheme is more forgiving since it uses more resilient source routes — only *one neighborhood node* in every source route needs to be forwarding packets. In this scheme, our failure probability becomes $(1-(1-c)^k)^d$. Note that since k is constant, we cannot *guarantee* resilient routing, but this is unlikely to be a problem in practice — while we need $O(\log \log N)$ nodes per group for provable resilience, we can set k to 11 and support a network of 100 billion nodes. In the next section we present simulation results measuring reachability when re-routing is taken into account.

Denial of Service Attacks

An unfortunate sideeffect of plausible deniability in the efficient scheme is the inability to prevent nodes from flooding the network, since it is impossible to determine if a node legitimately initiated such a flood or is forwarding the message for another node. This leads to the problem of denial of service (DoS) attacks through network floods. We can

[15] discounting nodes that we see only once throughout the experiment

counter this using oblivious throttling, where neighbors of a node sending packets faster than a certain threshold will refuse to forward some of those packets, independent of their ultimate origin or destination. This prevents undue usage of network bandwidth but degrades the maximum possible performance of the network.

Even without plausible deniability, the robust and hybrid schemes fall victim to DoS due to the amplification factor of messages — for every message sent by a node, multiple message must be sent by recipients. While nodes can refuse to forward duplicate messages, adversarial intermediaries modifying messages will cause both the original and the modified messages to be propagated. With enough adversaries, the final destination could be overwhelmed with messages, all of which require decryption and verification. Rate-limiting messages at intermediate nodes could alleviate the problem, but would also limit the usable bandwidth of the network since distinguishing between malicious and honest traffic is difficult or impossible.

2.7 Simulation Results

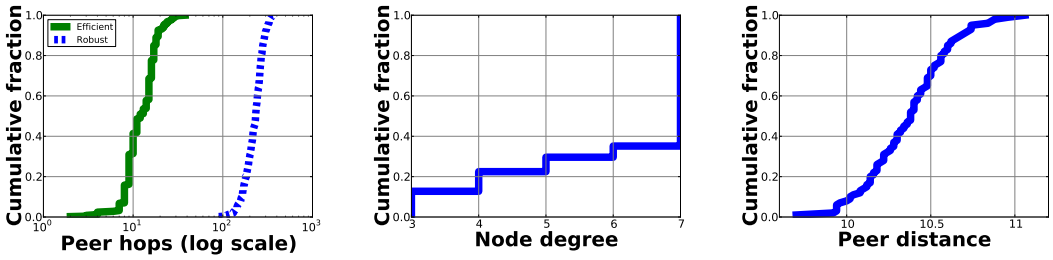
We simulated MCON construction and routing using the Orkut dataset from Mislove *et al.* [56]. The data contains $3,072,606$ nodes,[16] of which $75,229$ were eliminated due to asymmetric links, resulting in $2,997,377$ usable nodes, with an average node degree of 74. We assume this data to be representative of social networks in general.

2.7.1 MCON Construction

From the social network dataset, we constructed an MCON with a node degree limit of 7. The bootstrap protocol randomly selects an initial seed clique of four nodes ($\lceil \frac{7}{2} \rceil$) from the social network and iteratively adds nodes to the seed based on social network relationships. The final MCON contained just over 85% of the nodes in the social network. (3.61% of nodes in the Orkut dataset have node degrees too low to ever join.[17]) Slightly fewer than 35% of MCON nodes had under-full routing tables, resulting in

[16] Less than 12% of Orkut's network at the time of collection

[17] This situation can be repaired by allowing "leaf nodes," who only originate or consume traffic, and never serve as intermediates. Whether or not to accept leaf nodes is a decision best left to individual MCON members, who expose themselves to more danger by having more than the usual number of neighbors, and who bear the burden of servicing traffic from those neighbors.



(a) (b) (c)

Figure 2.5: Results of MCON simulations. Cumulative distributions of: (a) physical hops per DHT query with solid and dashed lines corresponding to efficient and robust schemes, respectively; (b) node degree; and (c) pairwise distance.

average node degree of 5.997 (Figure 2.5(b)), with an average pairwise physical distance of 10 (Figure 2.5(c)). The small pairwise distance implies that although we smoothed out the degree distribution, we still retain the small-world property. Setting $k = 5$ increases the number of nodes who can join the MCON to almost 90% of the social network nodes. The average node degree was 4, with 41% having under-full routing tables. While the above results are very consistent across different simulations using different random cliques when the maximum node degree is 5 or 7, when we set the maximum node degree to 9, the simulations became highly sensitive to seed locations — only 26% successfully constructed an MCON. The failure occurs early in the process, and can be fixed by selecting a different seed clique.

2.7.2 Routing and Search

Our DHT uses the Kad routing protocol (a variation of Kademlia [53]), using routing table of 16 buckets of 8 entries each. The average number of DHT hops between any two MCON nodes is 2.5, which translates to an average of 13 physical hops in the efficient case, and 26 hops in the robust case. (The probability distribution of physical hops per query with no churn is shown in Figure 2.5(a).) "Physical hops" is a misnomer in the robust case, but rather represents the number of nodes contacted, up to 8 in parallel. Note that due to the greedy nature of routing table construction, which preferentially incorporates the nearest node with a given prefix match, the average number of physical hops per logical hop is *lower* than the average number of physical hops between any two

random nodes in the MCON. Assuming average round trip times of 180ms (computed from the "King" dataset [75]), a search should complete in fewer than 2.5 seconds without time padding.

When a route fails, we select the next best route and continue trying until we succeed or reach 25 failed routes per node. The rates of DHT query failure with churn for all three schemes are shown in Figure 2.6. Data was collected using 500 independent trials, routing between two randomly-selected nodes. In the efficient scheme, the worst-case number of logical hops is 18 and the worst case for physical hops is 178, which translates to a query time of just under 33 seconds. In the worst case for the robust scheme, average performance is 127 logical hops and 395 physical hops, which would require 71 seconds on average. (Note that many of these physical hops are contacted in parallel, making the time estimate strictly pessimistic.) The efficient scheme reached its performance limit at 21% churn, and the robust scheme at 75% churn. Hybrid scheme performance depends on the modulus.

Freenet is reported to perform well when up to 30% of paths have failed. We measured Freenet churn by comparing consecutive 3-hour network snapshots to determine whether nodes in the earlier snapshot are still present in the latter one. Nodes were recorded by Freenet ID, and not IP address, to ensure that dynamic IPs do not interfere with our measurements. We recorded a churn rate approaching 71%, meaning that the efficient scheme would not be very effective. However, our robust scheme shows resilience to up to 75% churn, meaning it would perform well in Freenet-like conditions.

Routing performance in the efficient scheme shows only marginal improvement when k is raised from 7 to 9. Connectivity improvements are likewise not significant. Reducing k to 5 leads to small reduction in connectivity and routing efficiency. These results suggest that our initial guess of $k = 7$ is most likely optimal for the Orkut network in terms of performance, connectivity, and security.

Finally, the robust scheme may provide worse-than-expected resilience in certain topologies, such as when adversarial nodes form clusters in the MCON due to dense social network relationships among them. Clusters reduce the adversaries' knowledge of honest MCON nodes (since most of her neighbors are malicious), but impede routing — adversarial clusters have a high chance of forming neighborhoods with malicious majorities. However, assuming route discovery proceeds correctly, we only require *one*

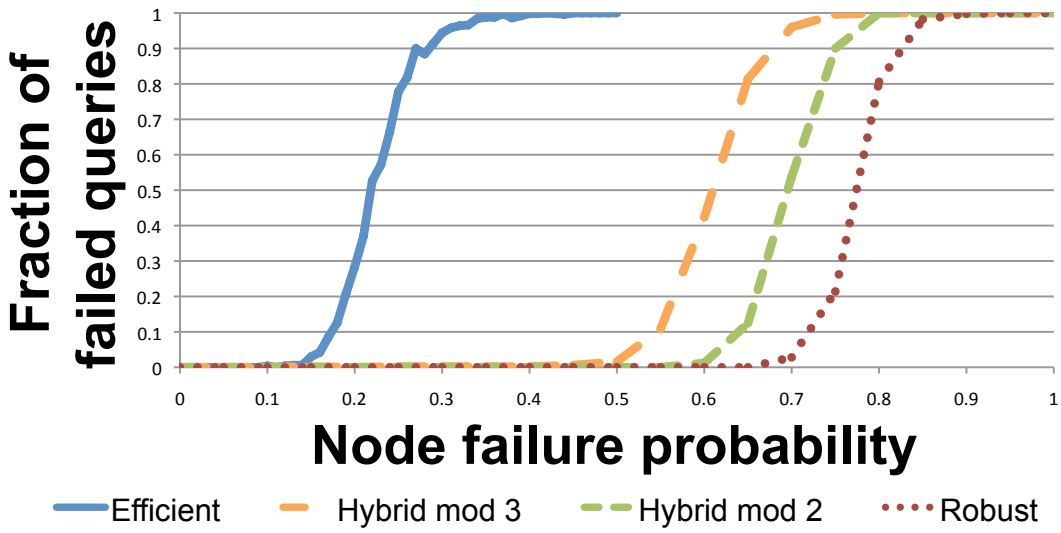


Figure 2.6: Measured probability of query failure in MCON simulations. Churn is the fraction of MCON members who are offline.

honest node per neighborhood for message forwarding to succeed.

Routing With Malicious Clusters

We had previously conjectured that malicious nodes (Sybil or otherwise) will form groups in the social network. It is then reasonable to assume that invitations will be issued in such a way as to preserve this "malicious cluster" behavior in the MCON. While we do not rely on it for protection from attack, it may prove detrimental for the purposes of routing, since even the robust scheme is vulnerable to malicious *groups*. To that end, we carried out routing simulations where neighbors of malicious nodes were preferentially malicious as well. The test topologies were constructed by taking a previously-constructed MCON and marking nodes as malicious following a growth pattern produced by a modified version of the algorithm in [76]. Topologies were constructed such that the network either had a large number of small clusters of malicious nodes, or a small number of large malicious clusters.

Figure 2.7 shows the results of our simulations. Note that the "x" axis now displays the sum of honest but offline nodes as well as malicious nodes (who never forward

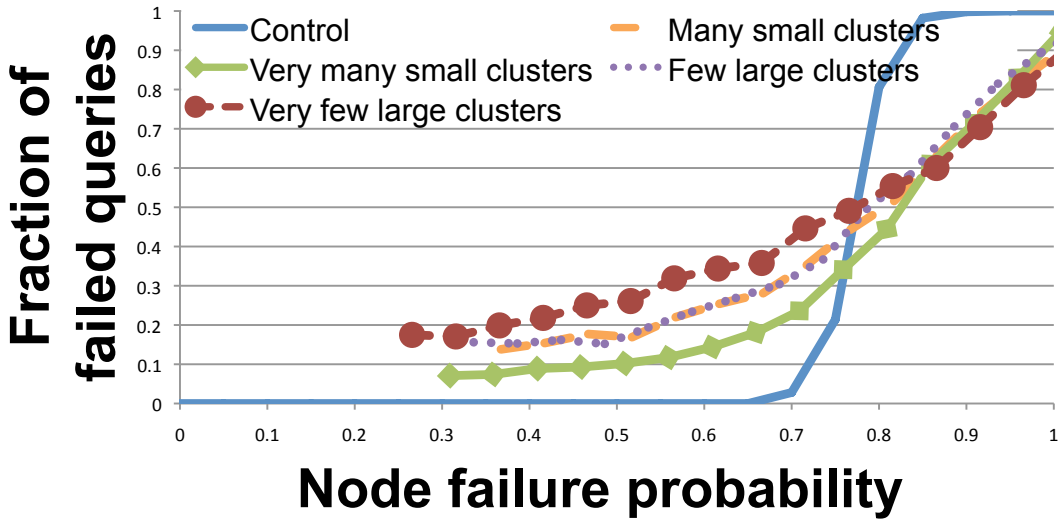


Figure 2.7: Probability of query failure in MCON simulations using clustered malicious nodes. Non-malicious nodes obey the standard churn model.

messages and can thus be considered offline). The results show that both clustering strategies are similarly effective in reducing the robustness of the network, with a small number of very large clusters being most effective (likely by isolating a small number of unlucky nodes) and very few small clusters being least effective, since a single honest node is enough to cause a query to succeed even if all surrounding nodes are malicious. When malicious nodes are clustered, the initial failure rate is higher, but the network is more robust at higher honest churn values. We therefore conclude that failure of randomly-placed nodes is more detrimental to long-run network performance than clustered malicious nodes.

Chapter 3

SilentKnock: practical, provably undetectable authentication [1]

[1] **A version of the work included in this chapter appears in [77] and is available at** `http://www.springerlink.com/content/427082v0r5300205/`**. The final publication is available at** `www.springerlink.com`**.**

As we have discussed in Chapter 2, MCON neighbors need to implement a form of authenticated communication in order to protect against hypothesis testing attacks, where adversaries connect to a node to check whether it will respond with an MCON protocol message. In this chapter we present an example of a covert authentication protocol that can be used for this purpose. We prove that a scheme which is secure in our model also resists forgery and provides replay attack protection against a global active adversary. We outline a protocol for a generic networking scheme, which makes rudimentary use of provably secure steganography [78], and show that this protocol satisfies our strong notion of security. Furthermore, we describe and analyze the security of SilentKnock, an implementation of our generic protocol for the Linux 2.6 TCP/IP stack. SilentKnock combines simple TCP steganography [79] with a very fast cryptographic message authentication code (MAC) [80] to provide efficient, provably secure connection authentication that integrates seamlessly with existing applications by hooking directly into the operating system kernel, eliminating the need to recompile. SilentKnock produces packets that are provably indistinguishable from those generated by the Linux 2.6 implementation of TCP, under the assumption that AES or HMAC-SHA1 (depending on the choice of MAC) and a variant of MD4 are pseudorandom functions.[2] No applications need to be altered, no shared libraries need to be replaced, and no potentially-conflicting protocols emerge. SilentKnock is lightweight, has minimal computational overhead, and is freely available for download [82].

We first discuss related work and the history of covert communication in Section 3.1. After formally defining secure port knocking in Section 3.2, we present the design of our scheme as both kernel- and user-space implementations, along with performance analyses. Finally, we discuss some limitations of this work in Section 3.4.

3.1 Related Work

Port knocking was originally used to deal with *port scans*: a kind of network attack (or attack precursor) in which an adversary attempts to connect to all, or some subset of,

[2] Linux 2.6 chooses TCP sequence numbers using 24 rounds of MD4 applied to the source and destination IP address, destination port, and 32 secret random bits, using an initially randomly-generated secret chaining value that changes every 5 minutes. See the functions `secure_tcp_sequence_number` and `half_md4_transform` in [81].

TCP and UDP ports at a given IP address. Port scans are useful to attackers because the results often indicate the operating system, architecture, and even a set of specific binaries that a host is running. This information can then be used for the purposes of a confirmation attack, to determine what software exploits should be used to attack the host, or what level of compromise might be likely.

Of course, if a server runs no vulnerable or secret software, a port scan is not a serious threat, but software security is a sufficiently hard problem that this cannot be seen as an immediate solution. A popular method of protecting against such network attacks is the firewall, which simply blocks all connection attempts to "internal" network hosts from "external" ones. Since there are many reasons why it might be desirable for a given service to be externally accessible — for instance, users may access a network service from a priori unknown network addresses depending on their physical location — this solution is not always satisfactory. Another option is virtual private networking (VPN), which is a layer of authentication allowing authorized users to get through the firewall and access internal services directly. However, VPN use is obvious at the network level and the VPN service itself presents a tempting target to an adversary.

Another class of proposed solutions to this problem is "port knocking": a firewall is deployed to protect a server, and before allowing a client connection to a particular port, that client must transmit a special "knock" that authenticates it. This knock may be either common to all authorized users of the system, or may be unique to a given user. Any attempt to connect that is not associated with the correct knock will be dropped; thus to an unauthorized user it should appear as if no network services are running on the server. Note that without a correct knock it is not immediately obvious to a naïve adversary that a port knocking service is being used. A variety of knocking methods have been proposed, such as a sequence of dropped connection attempts to closed ports [18], inclusion of a cryptographic authenticator in the initial connection request packet [19], "funny-looking" DNS lookups [20], and IPsec tunneling [21].

Many previous proposals for port knocking schemes have been accused of offering "security through obscurity," since it is trivially easy for an intelligent adversary to detect and steal knocks in non-cryptographic systems. By making the distinction between flawed *implementations*, which are only secure if the details of the system are unknown,

and the *concept* of port knocking (such that *even given the details of a port knocking scheme* one cannot tell if it is being employed), we argue that the *concept* of port knocking is *not* fundamentally flawed. Since revealing the presence of a service can only help an adversary — for example, by revealing which of a list of hundreds of exploits are most likely to succeed, thereby decreasing the cost of an attack — the notion of concealing services from unauthenticated users is a potentially useful one, when used in addition to regular network and software security measures. Separating authentication from applications is also a sound choice, since it enforces least privilege and economy of mechanism, in addition to easing deployment.

Given that the goal of a port knocking scheme should be to conceal the set of services running on a network host, all existing implementations exhibit a potentially undesirable oversight. Under relatively weak attack models, these schemes fail to conceal that a port knocking service itself is running. The fact that port knocking is active may aid the attacker in exploiting the port knocking service itself. Of course a port knocking service may well be a relatively simple program, but even simple programs may use libraries or language environments that make exploits possible. For example, at the end of 2009, the Open Source Vulnerability Database [83] listed 63 exploitable vulnerabilities in the popular OpenSSL cryptographic library, 16 in the popular zlib compression library, and so on. Moreover, even if a vulnerability in port knocking does not allow an attacker access to the system, it at least constitutes a denial of service attack if fail-closed semantics are used, and allows for port-scanning and other service identification in a fail-open design. Since the port knocking service is such a high-value target, we argue that *the presence of port knocking itself should not be detectable.*

The first published description of a port knocking scheme seems to be a 2001 mailing list post by Borss [84]. In the first academic treatment, Barham *et al.* [19] describe a scheme whereby a pass-phrase is transmitted (in cleartext) to a firewall either through a series of `SYN` packets, in a single "knock" packet, or as an option in the `SYN` packet. Krzywinski [18] describes a similar scheme where a client opens a port by attempting connections to a secret sequence of port numbers;[3] a number of similar systems are

[3] The server will monitor connection attempts on all closed ports and will open a port if a specific sequence of connection attempts is detected.

listed at [85]. Several authors [20, 86, 21] observed that knocks should be cryptographically protected and prevent replay attacks, but all resulting systems still involve the use of extra packets or nonstandard TCP options that allow detection of knocks (these systems provide authentication only, i.e. they make no attempt to hide the use of authentication mechanisms). deGraaf *et al.* [21] and Manzanares *et al.* [87] describe some other attacks and weaknesses of previous port knocking schemes, which our notion of security precludes — i.e. any scheme that satisfies our security notion is necessarily secure against the attacks mentioned in these papers.

There is extensive literature on TCP/IP steganography and covert channels [79, 88, 89, 90, 91], although Murdoch and Lewis [79] show that many of these schemes are easily detected. We introduce a cryptographic formulation of security similar to that in [78], and our notion of a secure port knocking scheme can be seen as a simple instance of covert computation [92] or the dining Freemasons problem [93]. We are, however, unaware of previous work relating steganographic computation and port knocking, or any previous work implementing the schemes of [92, 93]. We note that our system, like those in [92, 93], differs from covert channels alone because we provide covert one-way *authentication*, handle synchronization issues, and formally reason about what it should mean to hide an authentication service.

3.2 Formal Definition of Port Knocking

In this section we formally define a secure port knocking scheme and prove several relationships between our definition and some security properties mentioned earlier. While our scheme currently only works with TCP, it could be adapted to any network protocol that offers fields containing a sufficient quantity of entropy to be used for steganographic embedding. In the case of TCP, we use the initial sequence number (ISN) and the least significant byte of the timestamp.

While no definition of port knocking can be said to be definitive, we claim that any service that *hides the presence of other services running on a given machine* can be considered a port knocking scheme. While the term itself seems to indicate connection attempts to closed ports on a machine, the *spirit* of port knocking is to add an extra layer of authentication before revealing services offered by a protected host. We can

define a *generic* port knocking scheme as a system that manipulates network packets in application-agnostic ways to add such an authentication layer. (The number of packets that must be manipulated in this manner is design-specific, or can even be a configurable parameter.) One can think of our formal definition of port knocking as a generalization of traditional port knocking to any form of transport-layer connection authentication. Therefore, we abuse the terminology, which we previously used to identify port knocking designs (since we are no longer concerned about ports), and move the design of port knocking systems to join other connection authentication schemes such as IPsec [94], TCP-MD5 [95], as well as systems that have been traditionally designated as port knocking.

Definition 1. *A* TCP implementation *is a triple $\mathcal{P}$ of efficient probabilistic programs* Client*,* Server*, and* Init*. Both* Client *and* Server *take three arguments, consisting of a* state*, a* packet*, and a* command*, and output a new* state *and a new* packet*.* Server *will additionally output a message m.* Init *takes a single argument which is a string "*`client`*" or "*`server`*," and outputs an initial* state *suitable for use by the entity identified in the argument to* Init*.*

All TCP/IP connections are uniquely identified by the 4-tuple (source IP address, source port, destination IP address, destination port), with port information carried by TCP and IP address information by IP. TCP packets p have flags which signal the current state of the connection. Here we are concerned with the `SYN` flag, which signals the start of every new connection.

Most sequences of TCP Client commands consist of connection setup, message sending, and connection teardown. They are in the form of "connect from local port x to remote port y," "send message $\mathbb{M}$ through local port x connected to some remote port," and "disconnect local port x from the remote port." TCP Server commands are similar, and most are in the form of "listen for connections on port x," "receive data on connected local port x," and "close connection between local port x and remote port y." A *null* Client command causes it to output the next packet to be transmitted, while a *null* command causes the Server to output a packet acknowledging the receipt of data in the last received packet. Both Client and Server contain state about all active connections, buffered messages, and commands.

We define a standard session of a TCP implementation $\mathcal{P}$ with the command sequence $\mathcal{C} \in (command \times command)^*$ in the following manner. First the initial server state Ss_0 and initial client state Cs_0 are initialized: $Ss_0 \leftarrow \mathsf{Init}(\text{“}\texttt{server}'', \mathbb{E})$; $Cs_0 \leftarrow \mathsf{Init}(\text{“}\texttt{client}'', \mathbb{E}')$. Both the client packet Cp_0 and server packet Sp_0 are initialized to *null*. For each command pair $(Cc_i, Sc_i) \in \mathcal{C}$ let $(Cs_i, Cp_i) \leftarrow \mathsf{Client}(Cs_{i-1}, Cc_i, Sp_{i-1})$ and $(Ss_i, Sp_i, m_i) \leftarrow \mathsf{Server}(Ss_{i-1}, Sc_i, Sp_{i-1})$. The *output* $\mathcal{P}(\mathcal{C})$ is defined as the concatenation of messages output by Server: $m_0||m_1||\cdots||m_n$.

Definition 2. *A* TCP port knocking protocol *is a modification of the* TCP implementation *in Definition 1 such that* Client *and* Server *take two additional identical inputs: a shared secret* κ *and a set of public parameters* Φ*. Denote this modified protocol as* $\mathcal{H}_{\kappa,\Phi}(\mathcal{C})$*.*

We say that $\mathcal{H}$ *extends* TCP implementation $\mathcal{P}$ if for every command sequence C there is an efficiently-computable Φ and command sequence C' such that $\mathcal{P}(C)$ and $\mathcal{H}_{\kappa,\Phi}(C')$ are computationally indistinguishable for a uniformly-chosen random κ. This implies *backward compatibility* between our port knocking protocol and TCP. Note that for a null keyspace our modified implementation of TCP is trivially insecure. Moreover, note that unlike previous definitions of port knocking, we do not require additional packets to be transmitted, since this would compromise indistinguishability.

3.2.1 Security Condition

Informally, the idea behind our construction is that port knocking should not only hide the set of services running on a protected server, but also hide the presence of port knocking itself. This condition should hold even when an adversary is allowed to observe the network traffic for the server and the client as well as make connection attempts to the server. To that end, we define the security of our port knocking scheme, a modified TCP implementation $\mathcal{H}$, in terms of the adversary's inability to distinguish between two “worlds” in which he may find himself: in the first “hidden” world, the server is protected by port knocking and clients gain access by using $\mathcal{H}$ instead of $\mathcal{C}$; in the second “plausible” world, the client and server share a secure out-of-band communication channel through which the client notifies the server of its IP address and intent to make an inbound connection at a specific time. The adversary is given

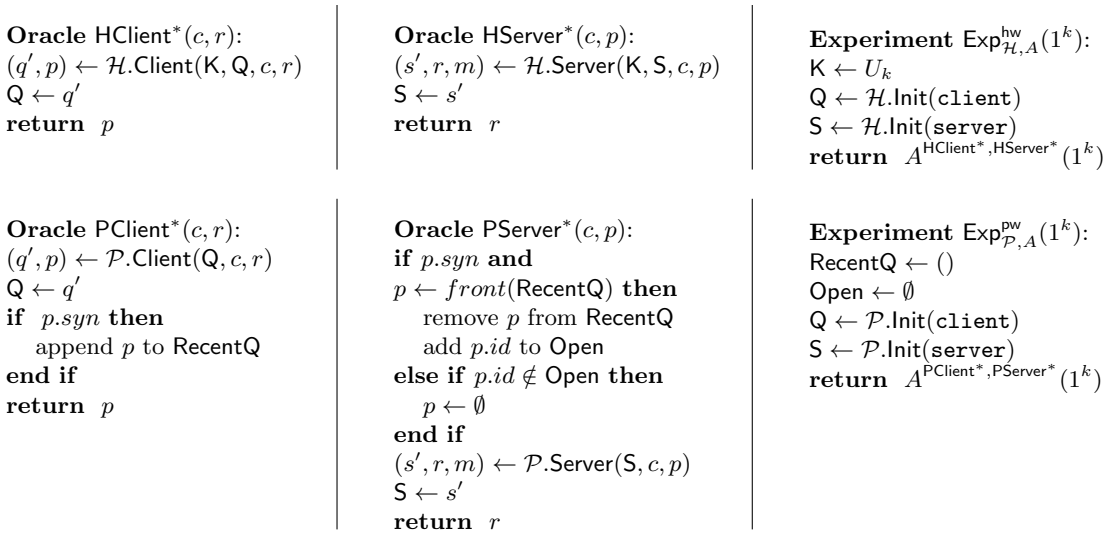


Figure 3.1: Definition of hidden world (top row) and plausible world (bottom row) experiments.

black-box access to previously-initialized Client and Server — although it cannot observe the internal state of either one, it is fully aware of their implementation details. The attacker may issue commands to either entity and observe the output.

In the hidden world, Client and Server use $\mathcal{H}$ initialized with a secret key and public parameters. In the plausible world, Client and Server use $\mathcal{C}$ but share a private queue Q. The server also maintains a list of active connections L. Whenever a client wishes to connect to the server, it inserts the id of the initial packet p into Q. When the server receives a connection request packet p, it allows the connection if $p.id \in Q$, removes the id form Q and adds it to L. When a server receives a packet p such that $p.id \notin Q \wedge p.id \notin L$ the packet is discarded. Once the client closes the connection, $p.id$ is removed form L. Notice that the output of the "plausible" world is identical to that of the "hidden" world provided the client and server share a key and public parameters. However, adversaries attempting to connect to the server will be unable to do so — their packets will be dropped. Both worlds reveal the presence of an authentication mechanism, but the inability to distinguish between the worlds prevents the adversary from determining the nature of that mechanism.

Formally we define a hidden-world experiment $Exp^{hw}_{\mathcal{H},A}$ and a plausible-world experiment $Exp^{pw}_{\mathcal{P},A}$ (see Figure 3.1) and define the *port knocking advantage of adversary A against $\mathcal{H}$ with respect to $\mathcal{P}$* as

$$\mathsf{Adv}^{\mathsf{pk}}_{A,\mathcal{H},\mathcal{P}}(k) = \Pr[\mathsf{Exp}^{\mathsf{hw}}_{\mathcal{H},A}(1^k) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pw}}_{\mathcal{P},A}(1^k) = 1].$$

We say that $\mathcal{H}$ is a (t, q_C, q_S, ϵ)-*secure port knocking scheme with respect to* $\mathcal{P}$ if for every time-t adversary A that makes at most q_C Client queries and q_S Server queries, $\mathsf{Adv}^{\mathsf{pk}}_{A,\mathcal{H},\mathcal{P}}(k) \leq \epsilon$. We call this a (t, q_S, q_C) adversary.

3.2.2 Related Notions

We have selected a security definition, but we have so far failed to ask whether it is the *right* definition. In this section we consider some security conditions which have been explicitly or implicitly stated as security goals, and show that our definition is in fact the strongest.

Hidden Server

The minimal requirement for a port knocking scheme is it prevents an unauthorized outsider from identifying Internet-facing services which may be vulnerable to attack. Note that service identification takes place at the TCP/IP layer — a server's existence may be determined by monitoring lower-level communication, such as ARP, but this does not reveal any information about the services offered. Since a port scan is just an instance of a $(t, 0, n)$-attacker, it is modeled by our security definition. The converse of this statement, that *hidden server security is implied by our definition* is clearly not accurate. For instance, any scheme that embeds an authentication field into TCP/IP (such as [95]) allows an adversary to distinguish between the plausible world, where packets lack an authenticator field, and the hidden world.

Malicious Server

A related notion is one where an adversary, masquerading as a server, cannot determine whether a given client is attempting to covertly authenticate. This is an instance of the "Dining Freemasons" problem [93], and is strictly weaker than our notion of security,

since we can model it as an $(t, n, 0)$-adversary. Once again we can see that the converse is not true — *malicious server security does not imply security under our definition.* A counterexample is a server who "reveals" the presence of a port knocking scheme by always choosing the same initial sequence number. An attacker who never queries the server does not learn whether port knocking is used, but any attacker making two queries (a $(t, q_C, 2)$-attacker) can distinguish between the hidden and plausible world with overwhelming probability.

Semi-Passive Adversary

An adversary that is allowed to arbitrarily modify communication taxes our "plausible" argument of an out-of-band channel. Instead, we use the intermediate notion of a *semi-passive adversary* who always correctly relays communication but can send his own messages to the client or server. Our framework can model this type of adversary, and thus implies security against semi-passive adversaries. Yet again, the converse statement that *semi-passive adversaries imply security under our definition* is not true. As in the *Hidden Server* model, client packets may contain authentication fields in the hidden world that make them distinguishable from the plausible world. In a more extreme example, every packet sent from the client may simply include the string "I am a port knocking client," allowing an adversary making a single client query to distinguish between the two worlds.

Replay Attacks

A number of previous port knocking schemes have been vulnerable to replay attacks that allowed adversaries to simply re-emit a previously seen packet or packet sequence that lead to a successful client connection. A replay attack can be used to distinguish between the plausible and hidden worlds: in the plausible world experiment `Server` removes "allowed" packets from the queue as soon as they are accepted, so replay attacks never succeed. Therefore, the probability of a successful replay attack in a (t, q_C, q_S, ϵ)-secure port knocking scheme by time-t adversaries should be at most ϵ.

Version Hiding

A desirable property of secure port knocking schemes is that observing a port knocking session should not be sufficient to reveal which port knocking scheme is in use. It is straightforward to show that our condition implies implementation hiding. Suppose $\mathcal{H}$ and $\mathcal{K}$ are port knocking schemes that implement network protocol $\mathcal{P}$. We define the version-distinguishing advantage of an adversary A as

$$\mathsf{Adv}^{\mathsf{vn}}_{A,\mathcal{H},\mathcal{K}}(k) = \Pr[\mathsf{Exp}^{\mathsf{hw}}_{\mathcal{H},A}(1^k) = 1] - \Pr[\mathsf{Exp}^{\mathsf{hw}}_{\mathcal{K},A}(1^k) = 1] \ .$$

Then any (t, q_C, q_S) adversary with version-distinguishing advantage ϵ gives an adversary for $\mathcal{H}_1$ or $\mathcal{H}_2$ with advantage at least $\frac{\epsilon}{2}$, since

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{pk}}_{A,\mathcal{H},\mathcal{P}}(k) &= \Pr[\mathsf{Exp}^{\mathsf{hw}}_{\mathcal{H},A}(1^k) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pw}}_{\mathcal{P},A}(1^k) = 1] \\
&= \mathsf{Adv}^{\mathsf{vn}}_{A,\mathcal{H},\mathcal{K}}(k) + \mathsf{Adv}^{\mathsf{pk}}_{A,\mathcal{K},\mathcal{P}}(k) \\
&= \epsilon + \mathsf{Adv}^{\mathsf{pk}}_{A,\mathcal{K},\mathcal{P}}(k)
\end{aligned}$$

Differentiated Service Hiding

When dealing with a multi-user environment, it may be prudent to not reveal the presence of some services to a subset of *authenticated* users. Our design is all-or-nothing, meaning that any port knocking-authenticated user is allowed to enumerate all services running on the server. Differential service hiding, while desirable, is orthogonal to the goals of this work, and a variety of techniques are known explicitly for this purpose [96, 97].

3.2.3 Generic Provably Secure Port Knocking

Here we define a generic design for a provably secure TCP-based port knocking system (see Figure 3.2). Let $\mathcal{P}$ be a TCP implementation that satisfies at least the following properties:

```
Program Client(K, q, c, r):
ρ1 ← Uk; ρ2 ← U*
(q', p) ← P.Client(q.pst, c, r; ρ1||ρ2)
if p.syn then
   τ = M.Tag_K(q.ctr, p.id)
   (q', p) ← P.Client(q.pst, c, r; τ||ρ2)
   Q.ctr = q.ctr + 1
else
   Q.ctr = q.ctr
end if
Q.pst ← q'
return Q, p
```

```
Program Server(K, s, c, p):
S.ctr = s.ctr
if p.syn and
M.Ver_K(s.ctr, p.id, p.iv) then
   add p.id to Open
   S.ctr = s.ctr + 1
else if p.id ∉ Open then
   p ← ∅
end if
(s', r, m) ← P.Server(s.pst, c, p)
S.pst ← s'
return S, r, m
```

```
Program Init(t):
S.ctr = 0
S.pst = P.Init(t)
return S
```

Figure 3.2: Generic protocol definition, simplified to assume a random *iv*.

- There exists a TCP field *iv* which is included in the connection initiation packet (those with only the `SYN` flag set) which is uniformly random [4] and independent of client state that can be efficiently computed by an external entity.

- $\mathcal{P}$.Client can be modified so that on any command that results in the output of a packet with only the `SYN` flag set, 1) the first k bits of a random input are copied directly into *iv*; and 2) all remaining fields of the packet are independent of the first k bits of the random input.[5]

  Note that 2) can be somewhat relaxed if the remaining fields can be efficiently computed by the client, but not by an external entity, from the first k bits of the random input. They would then need to be adjusted in the packet that incorporates the random input.

We use the initial sequence number (ISN) as the *iv* field in implementations of TCP that produce uniformly random ISNs. Some implementations produce *mostly* random ISNs, with some bits either set to invariant values or dependent on other bits. These can be adjusted when copying k into the ISN, provided that enough entropy is available to accommodate k. The core idea is to compute a MAC over some invariant fields of packet p and encode the resulting tag into $p.iv$. The nonce is a monotonically-increasing counter shared by the client and server, but incremented independently.

[4] The entire field may not be required to be uniformly random as long as the non-random bits are known by the packet originator and receiver.

[5] If some parts of *iv* are not uniformly random, they would be skipped when doing a bitwise copy of the random input into *iv*.

For our generic construction of a secure port knocking scheme relative to $\mathcal{P}$ we use a secure nonce-based message authentication code $\mathcal{M} = (\mathsf{Tag}, \mathsf{Verify})$. Tag is called with a k-bit key K, a nonce N, and a message m; the output is a tag τ. Verify takes a k-bit key K, a nonce N, a message m, and a tag τ, and outputs `True` if τ is a valid tag with respect to K, N, and m; `False` otherwise. We define the security of the message authentication code (MAC) against a *nonce-respecting* adversary[6] A such that it has advantage $\mathsf{Adv}^{\mathsf{uf-cma}}_{A,\mathcal{M}}(k) \leq \epsilon$. Additionally, we require that tags produced by Tag should be pseudorandom — i.e. an adversary querying a MAC oracle cannot distinguish between valid responses and random bit-strings with advantage $\mathsf{Adv}^{\mathsf{prt}}_{A,\mathcal{M}}(k) \leq \epsilon$ when running in time t and making q queries. This requirement is incompatible with unforgeability, but is provided by many MAC schemes, including Carter-Wegman-Shoup authenticators [98], and MACs that are pseudorandom functions, such as NMAC [99].

Theorem 1. *If $\mathcal{M}$ is (t, q_C, ϵ_F)-secure against forgery and (t, q_C, ϵ_R)-pseudorandom, then $\mathcal{H}$ is a $(t, q_C, q_S, \epsilon_R + q_S\epsilon_F)$-secure port knocking scheme with respect to $\mathcal{P}$.*

Proof. Consider an arbitrary adversary A running in time t and making at most q_C and q_S queries to Client^* and Server^*, respectively. We can use A to create adversaries A_F and A_R against the unforgeability and pseudorandomness of M, such that

$$\mathsf{Adv}^{\mathsf{pk}}_{A,\mathcal{H},\mathcal{P}}(k) \leq q_S \mathsf{Adv}^{\mathsf{uf-cma}}_{A_F,\mathcal{M}}(k) + \mathsf{Adv}^{\mathsf{prt}}_{A_R,\mathcal{M}}(k) \ ,$$

which implies the theorem.

Consider the hybrid experiment $\mathsf{Exp}^{\mathsf{hyb}}_{\mathcal{H},\mathcal{P},A}(k)$, which runs adversary A with the plausible world client and server oracles, with $\mathcal{H}.\mathsf{Client}$ and $\mathcal{P}.\mathsf{Server}$. Since the only packets that are dropped by this hybrid world's Server^* oracle and not dropped by the hidden world's Server^* oracle are those with forged MACs, we define adversary A_F: given a tagging oracle T_K, use it to simulate the Client^* and Server^* oracles of the hybrid experiment to A; let $p_1, p_2, \ldots, p_q$ be the sequence of packets A submits to Server^* and let ctr_i be the number of times MAC verification has been executed when p_i is submitted. A_F picks $j \in \{1, \ldots, q_S\}$ uniformly at random and returns $(\mathsf{ctr}_j, p_j.id, p_j.iv)$ as its forgery. An inductive argument (as in [67]) shows that although our simulation becomes inconsistent with $\mathsf{Exp}^{\mathsf{hw}}_{\mathcal{H},A}(k)$ after a forgery, the probability of finding a forgery

[6] Recall our notion of a semi-passive adversary

is the same. Also, when a forgery is found by A in this simulation, A_F will output this forgery with probability at least $1/q_S$. Thus $\mathsf{Adv}^{\mathsf{uf-cma}}_{A_F,\mathcal{M}}(k) \geq 1/q_S \left(\Pr[\mathsf{Exp}^{\mathsf{hw}}_{\mathcal{H},A}(k) = 1] - \Pr[\mathsf{Exp}^{\mathsf{hyb}}_{\mathcal{H},\mathcal{P},A}(k) = 1]\right)$.

We similarly define the pseudorandomness adversary A_R to simulate the hybrid world to A using its sequence of inputs in place of the MAC tags computed in $\mathcal{H}.\mathsf{Client}$. Since we assumed $p.iv$ is uniformly distributed in the cases of interest to $\mathcal{H}.\mathsf{Client}$, then when A_R's input is random, this experiment will be identically distributed to $\mathsf{Exp}^{\mathsf{pw}}_{\mathcal{P},A}(k)$, and when A_R's input is a sequence of MAC tags, the experiment will be identically distributed to $\mathsf{Exp}^{\mathsf{hyb}}_{\mathcal{H},\mathcal{P},A}(k)$. Thus if A_R simply outputs the result of A at the end of the simulation, we will have $\mathsf{Adv}^{\mathsf{prt}}_{A_R,\mathcal{M}}(k) = \Pr[\mathsf{Exp}^{\mathsf{hyb}}_{\mathcal{H},\mathcal{P},A}(k) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pw}}_{\mathcal{P},A}(k) = 1]$, completing the proof.

□

3.3 System Design

In this section we introduce SilentKnock, our implementation of the generic secure port knocking primitive introduced in the previous section, and show that this implementation correctly instantiates the provably-secure system defined above. We first discuss several adaptations necessary for secure and reliable interaction with TCP/IP, such as replay attack protection, client/server synchronization, and indistinguishability. We then analyze a number of possible attacks on our implementation. Finally, we present results demonstrating the performance of our system on real-world hardware.

SilentKnock is designed to be an application-agnostic, backward compatible transport-level authentication layer. It resists forgery and replay attacks while leaking no information about the authentication method employed. It is composed of two parts: a server-side packet monitor and a client-side packet-mangling proxy. The server-side component is available in two flavors: a user-space daemon, or a loadable kernel module. (The user-space module greatly simplifies the design and limits damage from programming errors, while the kernel module offers increased speed and efficiency.) We provide an implementation of a previously proposed operating system-specific steganographic embedding scheme for TCP/IP [79] and use it to embed authentication information into TCP headers. Specifically, we use keyed MACs as secure authenticators to resist

forgery and a two-part counter to counteract replay attacks and ensure that the client and server stay synchronized even in the presence of moderate packet loss. We avoid requiring loadable libraries or recompilation by hooking directly into the kernel. This ensures that applications do not need to explicitly support our system in order to benefit from it.

3.3.1 Universal Compatibility

Our application-agnostic design is trivial to use for end-users (who do not even need to know of its existence) and simple to set up for system administrators, who must only compose a configuration file with shared parameters and server names. Since we hook into the kernel rather than directly modifying it, we avoid requiring operating system or software changes to accommodate our design. This approach allows any application to transparently use SilentKnock provided that the network protocol used by the application has a steganographic embedding/extraction method supported by SilentKnock. We note that for certain protocols, such as TCP, where different implementations of the protocol may have subtle differences, each implementation may require a different steganographic embedding routine to preserve indistinguishability. Our goal is to seamlessly support as many transport protocol implementations as possible, although currently only TCP under Linux 2.6 is supported. A plugin-like architecture could be developed in order to ease the writing of protocol- and operating system-specific packet mangling routines.

3.3.2 Design Choices

Our implementation is designed for the Linux 2.6 kernel. We chose Linux 2.6 due to our familiarity with the system and the availability of the netfilter/libIPQ API [100], which allowed us to implement one version of the server component entirely in user space. We use Poly1305-AES [80] as our MAC since it is explicitly optimized for network packets and is implemented in optimized assembly for a number of popular platforms.[7] We

[7] For the kernel module version, we must use a function that is exported by the Linux kernel, so we employ SHA1 HMAC instead of importing Poly1305-AES into the kernel. The speed decrease from using a slower hash function is more than offset by the reduction of context switches between kernel and user modes.

implement Murdoch and Lewis' system [79] for embedding steganographic information into TCP initial sequence numbers (ISNs) and use the TCP timestamp option (enabled by default in Linux 2.6) to embed an additional byte of information into the timestamp, delaying packets when needed. For additional details on the adjustments necessary to make random ISNs consistent with the Linux 2.6 network stack, see [79].

3.3.3 Protocol

The SilentKnock algorithm is outlined in Figure 3.1. A SilentKnock client initiates a connection (composes a TCP `SYN` packet) to a SilentKnock-enabled server and steganographically embeds an authentication token into the packet. When a server receives a `SYN` packet, it extracts the authenticator from the ISN and timestamp and attempts to verify it. If verification is successful, the server allows the connection to continue, otherwise the packet is dropped. Connections are authenticated on a per-flow instead of per-source basis, so every `SYN` packet to a SilentKnock-enabled server must have an embedded authenticator token, even if it comes from a client that has previously authenticated successfully.

Instead of a sequence of knocks, we use a keyed MAC for client authentication, applying it to the source and destination (IP, port) tuples as well as a counter, so every connection attempt is guaranteed to contain a unique MAC. The embedding algorithm and resulting packet header structure are described in Figures 3.2 and 3.3, respectively. The client and server share a key, as well as a counter which is incremented for every client connection attempt (we discuss counter synchronization later). This counter prevents replay attacks by ensuring that every `SYN` packet sent by the client is different from any packet sent previously, and is also used as the nonce required by Poly1305-AES, our MAC of choice. Assuming that AES is a pseudorandom permutation, an adversary should not be able to compose a valid MAC (or even distinguish one from random bits) for the next `SYN` packet without knowing the key, even if we assume that all other factors are public information.[8] The key, initial counter, and resynchronization interval are exchanged out of band, since negotiation is impossible in case of one-way

[8] For the kernel module, we are forced to use only those cryptographic functions already available in the Linux kernel, so SHA1 HMAC is used, so the security of the kernel module is reduced to the security of SHA1 HMAC as implemented in Linux 2.6.

Algorithm 3.1 The pseudocode for SilentKnock. A is the server, B is the client, ctr_P is a per-IP-address counter maintained by principal P, k is a value derived from B's IP address and a symmetric key shared between A and B, m is a TCP flow identifier, and ft is a failure-tolerance parameter.

```
   B → A:
   SYN(MAC_{k,ctr_B}(m))                          {encoded in TCP/IP headers}
 2: ctr_B ← ctr_B + 1 A:
 3: for i = 0 to ft do
 4:     if MAC_{k,ctr_A−1+i}(m) = MAC_{k,ctr_B}(m) then
 5:         ctr_A ← ctr_A + i + 1                  {resynchronizes if client is ahead}
 6:     end if
 7: end forA → B:
 8: SYN-ACK
 9: goto 16 B:
10: if SYN-ACK received then
11:     goto 16                                    {connection was successful}
12: end ifB:
13: if SYN-ACK not received then
14:     goto 1                                     {connection failed}
15: end ifA,B:
16: proceed with TCP connection
```

communication. The server must also know the operating system of the client, since initial sequence numbers are composed differently by different implementations of the TCP stack (more on this below).

Steganography and Indistinguishability

The authentication token is embedded in the initial sequence number and timestamp fields of the TCP `SYN` packet [79]. Unfortunately, we cannot include the complete MAC, as our current implementation only allows a total of 32 bits to be embedded (24 bits in the sequence number and 8 bits — the least significant byte — in the timestamp), assuming Linux sequence numbers(see Figure 3.3).[9] Since we must not allow distinguishability based on discrepancy between the observed packet dispatch time and the packet timestamp, we delay packet transmission to coincide with the modified timestamp (average delay of 128ms, a maximum of 255ms). Although 32 bits is a relatively short MAC, an adversary would still have to compose 2^{32} packets, on average,

[9] OpenBSD has 30 bits of entropy available in the sequence number, while Linux 2.6 only has 24 bits. Therefore, OpenBSD could support 38-bit authenticators.

Algorithm 3.2 The steganographic encoding protocol. Decoding is performed by reversing the operations in this protocol.

P	{TCP `SYN` packet}
$P_{seq} \leftarrow \{S_1, S_2, S_3, S_4\}$	{Sequence number of packet (4 bytes)}
$P_{ts} \leftarrow \{T_1, T_2, T_3, T_4\}$	{Timestamp of packet P (4 bytes)}
$m \leftarrow (IP_B, source\,port, IP_A, destination\,port)$	{Authenticator}
$MAC_{K,ctr}(m) \leftarrow \{M_1, M_2, \ldots, M_n\}$	{n byte MAC}
$S_2 \leftarrow M_1, S_3 \leftarrow M_2, S_4 \leftarrow M_3$	
$T_4 \leftarrow h_M(\{T_2 \| T_3\})$	{n-Universal hash function}

to break the authentication (requiring, for example, 6 weeks to transmit over a T1 link), since the attack must be carried out online. Furthermore, a successful guess only leads to a single valid packet, forcing adversary to start over if it wishes to compose another valid packet. We remark that standard methods to deal with online guessing attacks can also be applied, such as account freezing or processing delays.

Another issue in using the TCP timestamp field (rather than just the ISN) to encode MAC data is the possibility of lost `SYN` packets. For instance, if a client generates a SYN packet but a `SYN-ACK` from the server does not arrive, the client must re-transmit the `SYN` packet. However, TCP requires that re-transmitted `SYN` packets have the same sequence number but different timestamp [101], so we can no longer encode the same stegotext in the timestamp: an adversary observing all `SYN` packets would detect that the least significant byte of the timestamp in the original and re-transmitted `SYN` packets are identical. The probability of this is only $\frac{1}{256}$, so the adversary could conclude that SilentKnock was in use.

To solve this problem, we ensure that the last byte of the timestamp looks random to our adversary, even when we are trying to re-transmit the same MAC. We can use two existing properties of our system to help us, the first having originally caused this problem: the higher order bytes of the new timestamp must be different from the ones in the original `SYN` packet. Secondly, we do not transmit the entire MAC (only the first 32 bits), so the adversary has no knowledge of the rest. We use these undisclosed MAC bytes to key an n-universal hash function (e.g. $h_{\vec{a}}(x) = a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_{n-1}x + a_n$) [102], which is applied to the higher-order bytes of the new timestamp [10] to determine the last byte of the timestamp, ensuring that any n or fewer distinct

[10] In reality, we only use the middle two bytes of the timestamp, since the upper byte is extremely unlikely to change, and the bottom byte will be replaced by stegotext.

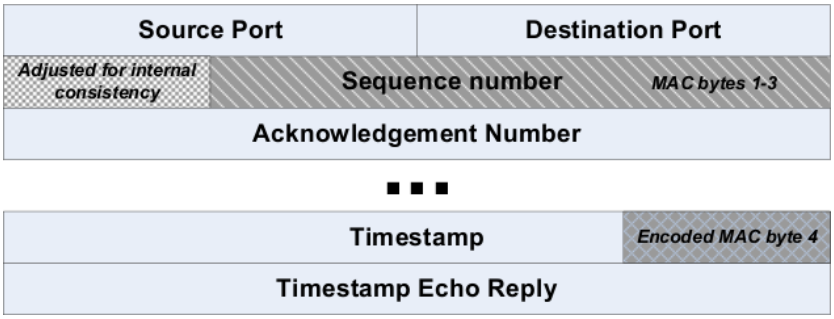


Figure 3.3: The TCP `SYN` packet after steganographic embedding. The "internal consistency" adjustment in the sequence number is performed to keep the modified sequence number consistent with what Linux is expected to produce.

timestamps have a last byte that is indistinguishable from random.[11] Since the server computes the same MAC, the server can reverse this process and extract the stegotext. Therefore we preserve the integrity and indistinguishability of stegotext in our timestamp even for re-transmitted packets. (Note that a packet will again need to be delayed so transmission time is consistent with the new timestamp.)

Counter Management

To protect against replay attacks, we employ a per-user counter, incremented after every connection attempt. If a given user has never before accessed a SilentKnock-protected server, the counter is initialized to 0 by both the client and the server. The counter poses additional challenges, such as what happens when client and server counters become desynchronized. This can occur in two ways: either the client's `SYN` packet never arrives at the server, leading to the client having a counter higher than the server's, or the server's `SYN-ACK` is lost (meaning the client and server are actually in sync, but the client does not know this). A client would have a hard time attempting to resynchronize after a failed connection, since it cannot differentiate between the two situations. To mitigate this, we allow some counter drift, and ensure that the protocol automatically

[11] By default, Linux 2.6 TCP only attempts to re-transmit a failed `SYN` packet five times, so 5-universal hashes are sufficient. If this number were to change, both the client and the server would need to modify their hash function (for n retransmissions, an m-universal hash function must be used, where $m \geq n$).

re-synchronizes after a certain time period.

We enforce the invariant $ctr_{server} \leq ctr_{client}$ by having the client always increment its counter when sending a `SYN` packet. The server, however, will only increment its counter upon successful MAC validation in order to prevent malicious desynchronization by an adversary sending bogus packets to the server while spoofing the client's IP address. In the naïve scheme of insisting the counter be exactly right, the server and client may never again get into sync once desynchronized, since the client will increment its counter on each connection attempt, but the server's counter remains the same. Checking more than one consecutive value of the counter as part of MAC verification would make desynchronization from non-malicious network events unlikely, but would also degrade security linearly, since it would allow multiple MACs to be valid at any given time. If multiple counters are checked, the server should saves counter that matches the MAC that verified successfully, and increments that counter for use next time. This way, the server and client should be in sync for the next connection attempt. (The number of alternate CTR values checked by the server is specified by the ft parameter in Figure 3.1.)

To counteract adversary-induced permanent desynchronization, we adopt a two-part counter design. Using a 64-bit counter, the first 32 bits (called the RESYNC field) are initialized to 0 at the time of first connection, and are incremented once for every configured unit of time (such as every hour, day, month, leap-year, etc.). The time period is a shared parameter and is agreed-upon during out-of-band setup. The latter 32 bits (called the CTR field) are always set to 0 when RESYNC is incremented. Using this two-part design allows resynchronization to occur automatically once the RESYNC increment time elapses. If there is substantial relative clock drift between the client and server, it is possible that client connections will fail (or even become desynchronized) when the client starts a connection at a time when one entity has incremented RESYNC and reset CTR but the other has not. However, this is extremely unlikely and would repair itself during the next RESYNC increment.

3.3.4 System Architecture

The SilentKnock system is composed of two separate programs - "`sknockd`" (running on the server), and "`sknockproxy`" (running on the client). The server component

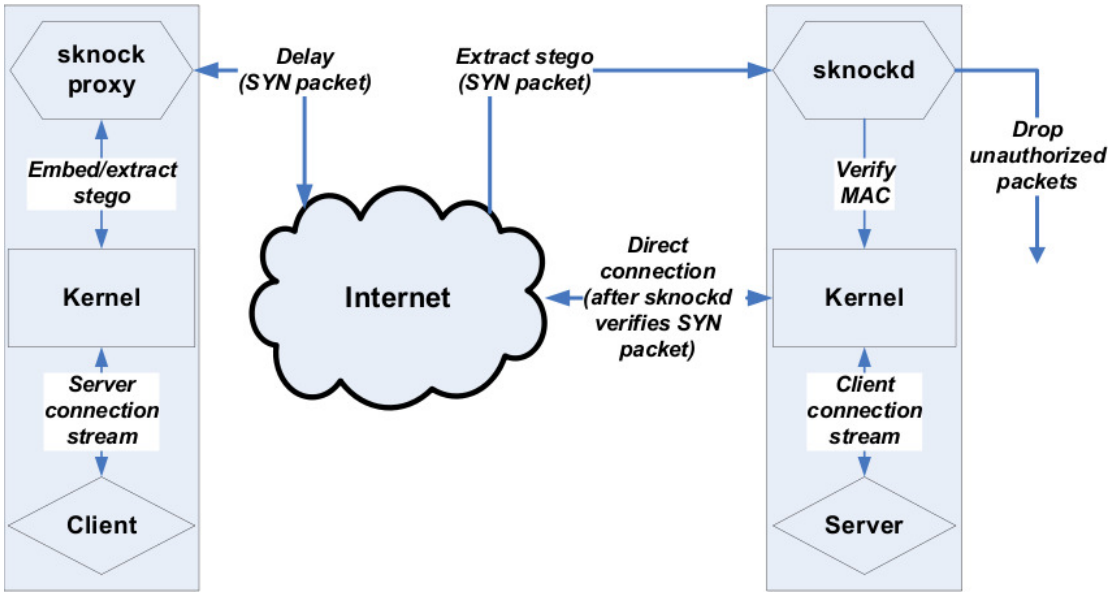


Figure 3.4: The architecture of SilentKnock. The client-side application initiates a connection to a server in the usual manner. The kernel composes a `SYN` packet, but `sknockproxy` intercepts the packet before it is sent, and embeds a MAC into the ISN and timestamp fields. The server receives the packet, and `sknockd` examines it before passing it to the kernel. If `sknockd` successfully extracts and verifies the MAC, the packet is passed to the kernel; otherwise it is dropped. Once the `SYN` packet is accepted, the user-space `sknockd` no longer examines other packets for that connection (except for terminating packets `FIN` and `RST`), for the sake of efficiency. The `sknockd` kernel module inspects every packet, but the overhead of fast-path processing for all but `SYN` packets is minimal (a few dozen machine instructions). `sknockproxy`, however, is forced to rewrite every incoming and outgoing packet for the connection to prevent the client TCP stack from getting confused due to a sequence number mismatch.

comes in user-space and kernel-space variants. The design of each of these programs is described below.

User-Space Knock Daemon

`sknockd`, the server side of the SilentKnock system, listens for connections on ports defined in its configuration file (the ports offering services, e.g. SSH on port 22), and examines incoming `SYN` packets on those ports before the TCP/IP stack sees them. When a packet is received, `sknockd` checks the source IP address of the packet and retrieves the secret key as well as the counter for that IP address from its configuration file (per-user shared keys are also supported). Using the TCP steganographic algorithm, `sknockd` extracts stegotext from the packet, and attempts to verify it as it would with a MAC. If verification succeeds, the packet is accepted and passed on to the TCP/IP stack; otherwise the packet is dropped. `sknockd` then increments the per-IP connection counter (CTR). This is the extent of `sknockd`'s involvement with the connection — all other packets are processed directly by the network stack in the kernel and are not seen by `sknockd`. The exception is `FIN` and `RST` packet, required to detect connection closing.[12] This removes the client's (IP, port) tuple from the list of active connections and reclaims memory. We use the libIPQ API and netfilter connection tracking [100] to register interest in packets with certain flags and (IP, port) tuples with the kernel, and those packets are rerouted by the netfilter system to user-space. The process is very efficient since our chosen MAC is very fast and only two or three packets per connection are examined in user space, are copied only once (from kernel to user space), and do not have to be copied back since they are not modified.

There is a small trick to preserving indistinguishability when we are intercepting only a subset of packets: we must prevent the adversary from being able to set the `SYN` flag on a packet that is part of a previously authenticated stream, because if `sknockd` drops that packet (due to incorrect MAC) the adversary will be able to conclude that SilentKnock is in use. Therefore, when `sknockd` tells netfilter to allow a certain connection (after verifying the MAC), we insert the ALLOW rule into netfilter *before* the rule that forwards `SYN` packets to `sknockd`. Thus, authenticated streams (having a

[12] Detecting connections closed due to TCP timeouts will be supported in the future, in case connection termination packets are lost.

known source (IP, port) tuple) are never again processed by `sknockd` (with the exception of detecting connection closure, as mentioned above, which does not alter TCP stack behavior), even if they incorrectly contain `SYN` packets, preserving default TCP stack behavior. The number of initial static netfilter rules is linear in the number of SilentKnock-protected services, and future dynamic rules scale linearly with the number of active connections to protected services. While the number of rules may become large with many active connections, this cannot be avoided and we must rely on the efficiency of the underlying packet filter implementation to scale gracefully under load. Memory requirements for per-user keys are linear in the number of users configured, and per-IP counter storage is linear in the number of client IP addresses.

Additionally, we must be careful to correctly process adversarial `SYN` floods (where many `SYN` packets are sent in succession for the same connection). To do this, we are careful to avoid a race condition where `sknockd` accepts the first packet to arrive, even though many instances of the `sknockd` packet handler may be spawned on a multi-processor machine. The overhead of this check rises linearly with the number of active connections to a given SilentKnock-protected service, but using hash tables to store active connections would allow this overhead to be reduced to a constant factor.[13]

Kernel Module Knock Daemon

Unlike user-space `sknockd`, the kernel module does not use a configuration file, but takes command-line arguments at load-time to determine protected ports and authentication keys. It intercepts all incoming connections and waits to see a `SYN` packet destined for a protected port, while accepting all other packets. (This "fast" path for all packets in which `sknockd` is not interested only requires a few dozen machine instructions.) `sknockd` then checks to see if the `SYN` packet belongs to an active connection. If it does, the packet is accepted in order to let the TCP stack correctly handle this case. Otherwise, `sknockd` extracts the stegotext from the packet and verifies the MAC. If verification succeeds, the packet is accepted, otherwise the packet is dropped before processing by the TCP stack. Since the kernel-space `sknockd` works independently from the firewall, no rule manipulation is required, and overhead is reduced to statistical

[13] However, using hash tables would force us to re-think our process synchronization system, since barriers do not make sense when using hash tables — see Section 3.3.5.

insignificance (as discussed in Section 3.3.6).

The kernel-space `sknockd` uses netfilter connection tracking [100] to determine the status of connections, and cleans up stale connection descriptors during idle time (eliminating the need to examine `FIN` and `RST` packets). To this end, we implement a novel low-contention process synchronization system that heavily prioritizes our packet handling code over our cleanup code. This system is described in Section 3.3.5.

Knock Proxy

`sknockproxy` reads a configuration file to find out which servers support SilentKnock, and for which services (listed by destination (IP, port) pairs). The configuration file also includes the key shared with the server, and the last value of the connection counter. If this is the first time connecting to that server, the counter is initialized to 0. `sknockproxy` registers interest for all `SYN` packets going from localhost to those (IP, port) pairs. When it receives such a `SYN` packet (generated by the local TCP/IP stack), it computes a MAC using the server shared key and steganographically encodes the information in the TCP initial sequence number and timestamp. It then registers interest for all incoming and outgoing packets for that (IP, port) tuple, increments the associated connection counter, and sends the packet over the wire.[14] Since we have modified the sequence number from what the local TCP stack expects it to be, we must modify it again in the return packets before the TCP stack sees them, otherwise we will confuse the stack and reset the connection. Likewise, we must continue to modify all future outgoing packets for that connection, otherwise the remote host will reset the connection when it detects a sequence number mismatch. Once the connection is closed, `sknockproxy` de-registers interest in that tuple (connection closure is detected the same way in the user-level implementation of `sknockd`). The number of initial netfilter rules is linear in the number of SilentKnock-protected services that might be contacted; future dynamic rules scale linearly in the number of active authenticated connections. Note that the client produces entirely backward-compatible packets — `sknockproxy` can be used when communicating with either an unprotected or a SilentKnock-enabled server.

This system is completely transparent to all applications running on the client and

[14] The packet may be delayed, depending on the modification made to the timestamp field.

server, and thus applications do not need to be modified to take advantage of it. Furthermore, since the configuration files for both `sknockd` and `sknockproxy` specify only the (IP, port) pairs and not the specific applications that use the system, the SilentKnock itself does not differentiate between applications. However, SilentKnock would need to be modified to support different network protocols (only TCP is currently supported). This modification would be a rather daunting task, as new steganographic embedding and extraction routines would need to be individually written for each new protocol and operating system. However, we may implement a plugin-like system in the future such that steganographic functions would still need to be written, but the required modification for `sknockproxy` to use those new functions would be minor.

3.3.5 Prioritized Synchronization With Minimal Contention

Our SilentKnock system must keep several lists of active, authenticated connections in order to prevent an adversary from setting the `SYN` flag on a non-`SYN` packet and seeing anomalous results: since the packet now has the `SYN` flag set, it would be processed by our code and dropped due to the incorrect MAC, allowing an adversary to distinguish connections using SilentKnock. These structures become stale after a connection has been closed, and need to be cleaned up. During idle time, a cleanup subsystem iterates over the list of connections, checks their status, and removes stale ones. However, since the subsystem monitoring incoming packets must have low latency, it is imperative that this background task not interfere with the near-real-time foreground task. This situation is a special case of the producer-consumer problem, with variable numbers of each thread (we use operating system-specific techniques to ensure that only one consumer is active at any given time, and a lock to keep more than one producer from concurrently modifying the data).

Synchronization is made easier by the specific behavior of our system: we only have two threads: the high-priority packet handler which reads the list but only appends entries to the end, and a second thread which removes entries from anywhere in the list but does not add to it. The cleanup thread must have strictly lower priority than the packet handler, since the latter must act in near-real-time. Priority-sensitive semaphores at first seem like a good solution, but do not turn out to be applicable, since we would like the consumer to work in an opportunistic fashion (no FIFO requirement), rather

than wait on blocked resources.

Our system is composed of mutex locks (one per connection list) and barriers. Each mutex controls access to its list, while barriers (associated with each list entry) prevent the consumer thread from reading past the "stable" end of the list, and into a partially constructed list entry. Figure 3.5 provides a graphical representation of our data structures, and Algorithm 3.3 shows pseudocode for the two threads. Note that neither lock-free [103] nor wait-free algorithms [104] are appropriate, since we want a specific thread to preferentially make progress, rather than any particular thread. We expound upon the use of each primitive below.

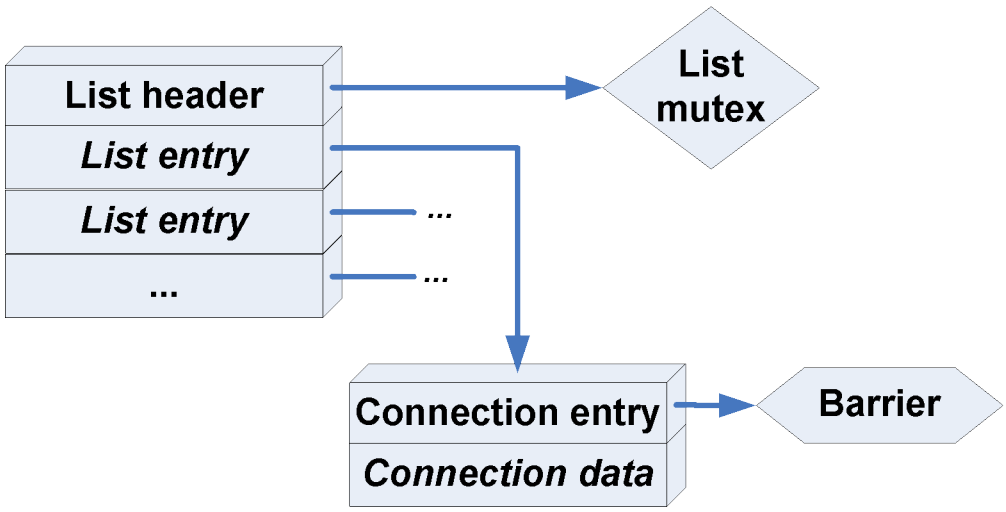


Figure 3.5: The shared data structure. Each list has an associated mutex, and each list entry has an associated barrier. A producer will not write to a list whose mutex is locked, and a consumer will not read a list past an entry with a barrier that is "closed."

Barriers

The last data structure in the list always has a barrier set to "closed." When the producer appends to the list, it does not lock the list, but instead constructs a new connection entry object (which has a closed barrier) and appends it to the end of the list. Only then will the producer open the barrier on the previous object.

Algorithm 3.3 Prioritized process synchronization

```
packet_handler:
 1: lock(list)
 2: read(list)
 3: unlock(list)
 4: create(list_object)
 5: add_to_tail(list, list_object)
 6: unlock_barrier(previous(list, list_object))
end packet_handler

cleaner:
 1: for each list_object in list do
 2:     if barrier_closed(list_object) then
 3:         break
 4:     end if
 5:     if stale(list_object) then
 6:         interrupts_disable()
 7:         if try_lock(list) = success then
 8:             remove(list, list_object)
 9:             unlock(list)
10:         end if
11:         interrupts_enable()
12:     end if
13: end for
end cleaner
```

Mutexes

While the producer task iterates over the list, it holds a lock on that list. Once it finishes reading, it releases the lock. The consumer thread also iterates over that list, but without holding the lock (this is safe, since the only objects the producer constructs will be added after a barrier, beyond which the consumer never reads). New objects are constructed by the producer without holding the lock, since they will be added only after the barrier. If the consumer tests an object and discovers that it is stale, the thread attempts to acquire a lock on the list holding the stale object. If the lock is unavailable, the consumer continues scanning the list. If the lock is acquired, the stale object is removed from the list, and the lock is released. This way, the only time the consumer thread is holding the lock is when it is actually in the process of removing an object (something that takes on the order of a few dozen machine instructions), giving priority to the producer thread. Additionally, the consumer always disables interrupts before cleaning an object, ensuring that it is never re-scheduled while holding

a lock, eliminating priority inversion. This is again safe since only a few dozen machine instructions are needed to perform the removal.

Our system provides exceptionally attractive properties, such as concurrent reads (since only the producer must hold a lock when reading the list, the consumer is free to read at any time), prioritized operation of one thread over another during both read and write operations, and resistance to priority inversion (since the consumer is never pre-empted while holding a lock).

3.3.6 Timing Analysis

The indistinguishability of SilentKnock relies on the adversary gaining no information through timing attacks — if `sknockd` takes an overly long time to process packets, a smart attacker with knowledge of traffic timing before SilentKnock was installed on a server would realize that additional processing is occurring (but not necessarily that SilentKnock is in use). If the difference in timing is large enough, it makes for a good distinguisher for SilentKnock in practice, even though timing information is not included in our formal model. On the other hand, if the timing difference is small (compared to timing noise between the adversary and the server — delays imposed by slower or overloaded routers, etc.) or the adversary lacks precise knowledge of the timing characteristics of the server, this "side channel" will not lead to a good distinguisher in practice. Timing considerations were the primary motivation in implementing the `sknockd` kernel module.

Results of our timings tests are shown in Table 3.1. We measure the time a server running `sknockd` takes to process `SYN`, recording the time period between receiving a `SYN` packet (containing a valid MAC) and emitting a response (`SYN-ACK`). Although information leakage (thought timing information) occurs in practice in the user-level implementation, the amount of information revealed is minor. Even using the user-space `sknockd`, an adversary located a few hops away and having perfect knowledge of the server timing distribution without `sknockd`would need to witness several hundred accepted sessions to gain a significant advantage in distinguishing `sknockd` from

Experiment	Baseline	nf_conntrack	sknockd	
			kernel-space	user-space
Average response time (μs)	95.76	110.48	123.54	249.35
St. Dev. (μs)	3.69	8.99	8.02	20.87
Slowdown (over baseline)	1.00	1.15	1.29	2.60
Slowdown (over conntrack)	N/A	1.00	1.12	2.02

Table 3.1: Average time difference between receiving a `SYN` packet and emitting a `SYN-ACK` packet. The second experiment uses kernel modules (nf_conntrack and nf_conntrack_ipv4) to help clean stale connections. The third and fourth experiments use only the user-level `sknockd`, and the `sknockd` plus netfilter connection tracking modules, respectively. The time difference between the connection tracking modules alone and the connection tracking modules with `sknockd` is not statistically significant.

a dynamic firewall.[15] To further minimize this difference, we implemented AES precomputation for Poly1305-AES nonces [80]. Currently we only precompute the initial counter value, but we can precompute and store values for the next several counters, allowing for verification to be performed without any online cryptographic computation. Unfortunately, this optimization is not possible with the kernel-level `sknockd`, which uses SHA1 HMAC.

The timing channel in the kernel-level implementation of SilentKnock is significantly less informative to a potential adversary. While there is a statistically significant difference between a SilentKnock-enabled server and a non-SilentKnock-enabled server, there is no statistical difference between the SilentKnock-enabled server and one running the nf_conntrack and nf_conntrack_ipv4 modules only. Therefore, an adversary does not know if SilentKnock is being used, or if the server simply tracks connection metadata for auditing.

While we do not test the client-side proxy for timing distinguishability, we conjecture that the use of `sknockproxy` would be much more difficult to detect than `sknockd`. Since the processing of `SYN` packets occurs before any observable event, and processing subsequent packets in a flow requires no manipulation of kernel data structures and no cryptographic computation, observable timing differences would be very small. If a

[15] 90% of Internet flows experience a standard deviation of 1ms or more in round-trip time [105], while the magnitude of timing difference in the case of user-space `sknockd` is about 0.15 ms.

remote adversary were to test for the presence of `sknockproxy`, the largest observable effect would be in the re-transmit timeout, which may be altered by the packet delay imposed by timestamp modification. However, since retransmit clocks have granularity measured in seconds [101] and our timestamp modification has millisecond granularity, detection is highly unlikely.

3.4 Discussion

3.4.1 Limitations of SilentKnock

Here we would like to note a number of limitations of our system. First, we only attempt to authenticate the start of a connection, but provide no guarantee that connections stay authentic. In other words, our system does not protect against connection hijacking, a well-known problem in TCP security [106]. We believe it is up to the application to provide connection hijacking protection and relevant user authentication (e.g., SSH [107]). Furthermore, due to the limited bandwidth for authentication, SilentKnock can only support symmetrically-keyed authentication with only 32 bits of a larger MAC. Since different operating systems have different TCP initial sequence number properties, the amount of data that can be embedded in the `SYN` packet is highly dependent on the OS composing the packet. Thus it is necessary that the server know the OS of the client in order to correctly extract the stegotext; alternatively, the server can attempt multiple extractions, but this will increase overhead and degrade security linearly in the number of OSes supported.

Finally, existing port knocking schemes which do not manipulate the TCP packet directly have the advantage that of unprivileged user-space implementations on the client side, allowing end-users to run such systems without the intervention of a system administrator. Additionally, destination port numbers and other user-specified fields are unlikely to be modified by other systems such as IPsec [94], that may mangle other TCP fields such as sequence numbers and timestamps, making user-space solutions more robust to in-transit tampering. However, this advantage comes at the price of indistinguishability model — while SilentKnock is indistinguishable from a TCP connection, traditional port knocking would at best be indistinguishable from a limited port scan, which is in itself a notable network event.

Identities, Addresses, and NAT

In any distributed authentication system it is necessary to decide what the identities in a system mean. Three natural choices are to let identities correspond to network addresses, to physical hosts, or to human users. Our current implementation supports two options: identities (keys) may be associated either with IP addresses or users. Each option has different consequences for usability and security. When identities are bound to IP addresses, we must assume that only a single client machine will be accessing a SilentKnock-protected server from a given IP address, since a single counter is used for each identity. This assumption breaks down in the presence of network address translation (NAT) or similar devices. Therefore, in this scenario, we must limit our system to only one client per NAT. We stress, however, that unlike previous implementations where NATs presented a security problem [87], adversaries sharing a NAT with a valid `sknockproxy` client gain no advantage.

We support per-user identities by issuing keys to each user and checking the MAC on each `SYN` packet against each authorized user key. This can be done at essentially no extra computational cost due to the design of the Poly1305 MAC [80], which is computed by adding a keyed non-cryptographic hash of the message to the AES encryption of a nonce mod 2^{128}. Suppose we assign different AES keys (but a shared non-cryptographic hash) to different users, and precompute the AES encryption of different users' counters for the next ft values. Then for any given packet p with embedded tag t we can check whether $t = \mathrm{MAC}_{K,r}(p,n) = \mathrm{Poly1305}_r(p) + \mathrm{AES}_K(n) \mod 2^{32}$ for some user's key K and counter n as follows. We first compute $t - H(p) \bmod 2^{32}$, and then search for the resulting value in our table of precomputed encrypted nonces; if the value is found, we accept the packet and remove older encrypted counters for the same user. This search can be implemented in constant time (with respect to the number of users) using a number of approaches such as hash tables or tries. After accepting the packet, we insert the next precomputed nonce for the same user into the table. While this solves the NAT problem mentioned above, it causes security loss proportional to the number of users (and thus the number of user keys) due to the requirement that we check the MAC against all keys. Alternatively, once IPv6 is a viable alternative to IPv4, we may be able to use unique *target* IP addresses as part of the key, such that a server running `sknockd` has one IPv6 address per user.

Denial of Service

While we have implemented some measures to prevent distinguishing or denial of service attacks due to packet dropping, our scheme is vulnerable to a selective denial of service attack. An adversary who modifies *all* packets on a network by consistently rewriting sequence numbers or timestamps can cause MAC verification to fail at `sknockd`, while not affecting the status of most standard TCP traffic. We note that this attack is both expensive, in that it requires the attacker to touch every packet in — and maintain per-flow state for — all connections on a network, and may effect other protocols that authenticate the TCP header, such as IPsec [94] or TCP-MD5 [95]. Additionally, such selective denial of service is much easier for other port knocking or general IP service authentication schemes, as in those cases it is easy to identify knock sequences or authenticated packets and drop them, while maintaining no other state. Finally, if the server logs failed connection attempts, it will be easy to notice such attacks since, for instance, altering the timestamp will still give a 24-bit MAC match in the sequence number, which is unlikely.

Distinguishability Attacks

SilentKnock is vulnerable to a distinguishability attack by a non-nonce-respecting adversary. Recall our notion of a semi-passive adversary, who is not allowed to modify packets in transit but only allowed to drop them or issue new packets. Now consider a packet-modifying (fully active) adversary that can arbitrarily modify packets. Such an adversary may alter the ISN of a `SYN` packet in transit to a SilentKnock-protected server and observe the result. There is no reason that a server should not accept a TCP packet with an arbitrary ISN unless it is either not offering a service on the target port or is protected by SilentKnock. If previous connections from a given source are observed to succeed, but a connection using a packet with a modified ISN fails, our active adversary can conclude that SilentKnock is in use. Note that this adversary is particularly powerful, since it must be able to block a TCP packet before this attack can be carried out.

Network Performance Considerations

Finally, there may be unexpected drawbacks of delaying packets in order to use the low-order byte of the timestamp option field to embed stegotext, such as decreased network timeout times (since packets are delayed *after* being processed by the TCP/IP stack), which may adversely impact TCP congestion control.

Chapter 4

Censorship-resistant overlay publishing system

Having previously constructed a membership-concealing communication layer in Chapter 2, and a covert authentication channel for MCON members in Chapter 3, we now extend our work to create a robust censorship-resistant storage system which we call CROPS (censorship-resistant overlay publishing system). This allows individuals to not only communicate with each other, but exploit the one-to-many nature of file sharing systems to distribute information to anyone who wants it. We define the requirements for such a system in Section 4.1, discuss related work in Section 4.2, and present our design in Section 4.3. Finally, we offer a theoretical evaluation in Section 4.4.

4.1 CROPS Requirements

Censorship-resistant systems must provide transport, storage, or both, such that these services are not subject to targeted censorship by insider and/or outsider attackers. They must also be useful, supporting efficient file storage and retrieval, scaling to a large number of participants and data objects, and surviving non-malicious member churn, such that nodes in the system may go offline without disrupting system functionality.

4.1.1 Security Requirements

All security requirements of a censorship-resistant system derive from two primary goals: availability of content, and identity privacy for clients accessing that content. Availability implies resistance to a broad spectrum of DoS attacks at all levels of the protocol stack, from the lower-level communication protocols (e.g. SSL/TLS) to upper-level application protocols (e.g. file sharing). The adversary must not be able to deny service to most system users individually, nor broadly block the protocol based on its network-level properties. Identity privacy implies that clients should not be observed as accessing specific content, and publishers should not expose their own real-world identities when posting content to the system. Both clients and publishers require identity privacy to resist coersion and self-censorship; publishers also need identity privacy to protect themselves from rubber-hose cryptanalysis. However, we require the additional protection — even authors should not have the ability to remove or modify content once published to the system, since removing content constitutes censorship, and authors can be coerced or have their cryptographic keys compromised. This allows us to derive the following

top-level requirements:

- *Identity privacy.* When censorship is primarily enforced through social deterrence (by applying post-facto punishment to those who attempt to circumvent it), it can be defeated by technology that prevents the identification of the individuals who supply or access censored materials. Membership concealment prevents linking online identities to real-world counterparts (e.g. IP addresses) or identifying participants, concealing the connection between online pseudonyms and real-world identities.

- *System availability (existential blocking resistance).* If a censor can locate all the participants of a censorship-resistant network, it can block access to the individual nodes at the network level. This necessitates the use of membership concealment in order to ensure service availability. However, when considering such a sophisticated censor, membership concealment is not sufficient to ensure availability; the censor may also attempt to disrupt service at higher levels as well, such as by examining a packet stream to identify the application being used. To defend against application blocking, censorship-resistant schemes must use either protocol obfuscation (e.g. [34]) or encryption and padding.

- *Content availability (targeted blocking resistance).* An adversary that is either unwilling or unable to prevent individuals from accessing a system may nonetheless attempt to remove specific content from the system, denying access to that content only. This can be achieved through multiple means: technological countermeasures against access to content with specific keywords have been deployed by multiple nations [44, 4]; social means such as legal action (or threats of legal action) are used in a number of Western democracies as well [5, 6, 4].

4.1.2 Targeted Blocking

Censorship-resistant systems must provide transport, storage, or both, such that these services are not subject to targeted censorship by insider and/or outsider attackers. An adversary should be prevented from selectively removing content from the system or blocking particular users from publishing or consuming content. Potential outsider

attacks include deep packet inspection to block content, as in the case of China's "Great Firewall" [44]. A robust censorship-resistant system requires protocol-level encryption or obfuscation to prevent such keyword-based attacks. Encryption alone, however, is vulnerable to encryption-oblivious fingerprinting [108, 109], so a robust system will require protocol-level padding.

Membership concealment prevents targeted blocking of individual users. However, even if a publisher of a given piece of content can be located, the act of publishing should be irreversible, since authors are vulnerable to rubber-hose cryptanalysis and can be rather forcefully persuaded "un-publish.". Furthermore, if content is modifiable, the system should at the very least support versioning, such that both the modified and previous versions of the data would persist and be easily retrievable [110]. In other words, the system must be resistant to both blocking specific content at the network level, as well as blocking content by locating the content publisher or server.

4.1.3 Existential Blocking

Censorship-resistant systems must also resist existential blocking, i.e. attempts to block the system outright. Some designs that were created to be censorship resistant, such as Tangler [11], are only robust against targeted censorship: the Tangler protocol cryptographically links all stored content, such that if some stored data is deliberately made unavailable, most content would be rendered irretrievable. Ironically, this makes it trivial for any adversary of even low power to deny access to the entire system, and thus all data. Systems that aim to resist an adversary which is willing to engage in complete blocking must be *membership-concealing* — it should be difficult for either an insider or an outsider to learn the real-world identities of a non-trivial fraction of system participants in bulk (compare to targeted censorship where adversaries attempt to learn the identities of parties publishing, consuming, or storing a given piece of content). This implies that *no centralized or semi-centralized system can be censorship-resistant* since by definition the number of storage servers to block would be low. Therefore, the system must be peer-to-peer, where all clients contribute to routing messages and potentially storing content. These systems must resist concurrent failure of a large fraction of participants, and gracefully recover using robust routing and/or content replication.

4.1.4 Functional Requirements

The basic requirement for any data storage system is support for replicated storage and robust retrieval. Assuming an existing routing infrastructure (such that arbitrary nodes can communicate with each other), storage can be implemented using only two basic operations — *put* and *get* [45]. Thus the primary functional requirements for robust censorship-resistant storage are the security requirements above, combined with scalable support for the put(key,value) and get(key) operations. Since CROPS is constructed as an overlay above an MCON, it is a simple matter to extend MCON's underlying distributed hash table (DHT) to provide storage capabilities and efficient lookup like PAST [111] or OceanStore [112]. In order to be useful in practice, a censorship-resistant storage system should satisfy the following additional requirements:

- *Fully-distributed architecture.* No centralized or semi-centralized system can serve as a robust censorship-resistant system since by definition the number of failed (or blocked) servers the system can tolerate is low. Therefore, we must use a distributed peer-to-peer (P2P) design. P2P architectures use collaborating members with similar responsibilities and rights, and generally no member node is more important than another to the continued correct functionality of the system.[1]

- *Plausible deniability for storers.* To prevent prosecution of storers on the basis of hosted content, node-local content must be opaque to real-time or post-hoc examination by an adversary and even the storer itself. It is sufficient to ensure that while stored content *can* be examined, the process is too resource-intensive to apply to a large fraction of content. The storer should also be prevented from determining the content of a specific query to which it is responding, i.e. the storer knows that the query matches data stored locally (which the storer returns in response), but cannot determine the clear-text content of the query or the data.

- *Scalability and efficiency.* We want the system to be scalable both in terms of the number of supported users (searchers and storers) and in terms of the amount of storage the network is able to provide. However, the network must replicate

[1] However, to support as large a user base as possible, the network should not require clients to store significant quantities of data in return for using the network, supporting a *heterogeneous node set.*

content across enough storers to guarantee, with high probability, that it will be retrievable even with high storer churn and attempted blocking. Furthermore, we should distribute storage and traffic loads equitably across peers for the purposes of fault tolerance as well as to avoid unduly taxing any particular peer. Robust replication and storage/bandwidth efficiency is a fundamental trade-off in such a system [113].

- *Resistance to contamination and storage exhaustion attack.* In a network that does not support content removal and only offers a fixed amount of storage, an adversary could pollute the results of any given search (by keyword) by repeatedly publishing faulty data marked the keyword in question. The adversary also could publish large volumes of content to exhaust storage space in the entire network, independent of keywords. Unfortunately, this problem is exceedingly challenging. Although there are verifiable fair storage schemes such as tit-for-tat [114] and storage accounting [115], the way to punish offending nodes in such systems is generally the removal of their data from the network. This is not only not an impediment to a censor, it is beneficial: if an adversary can "persuade" a given publisher to stop storing data in a tit-for-tat scheme, the publisher's data would be dropped from the network, censoring it. Since it is difficult to determine whether published data is relevant to a given keyword, and whether that data is accurate, the network will need to employ a variant of voting and/or editing scheme, while paradoxically not allowing an editor to explicitly censor content.

4.1.5 Adversary Models

Different censorship-resistant designs assume various adversary models, some far more powerful that others. We assume a strong adversary model e.g. of the level of a nation-state, with the associated monetary resources. The attacker may choose to be completely passive (monitoring only), affecting censorship through post-facto punishment (e.g. arrests, fines, or rubber-hose cryptanalysis). This type of attack may remove content, but can also induce self-censorship if widely publicized [116], causing selected material to never be posted in the first place. Alternatively, an active attacker can block access to

targeted content through a number of technological means such as deep packet inspection [44] or encryption-oblivious protocol fingerprints [108, 109]. The attacker may also opt for wholesale blocking of a given service, website, or protocol [13, 12, 14].

Before defining our adversaries, it is important to be clear on the nature of the client of our censorship-resistant system. The client is the party interested in gaining access to content, attempting to defeat or bypass the adversary's blocking methodologies. An adversary will attempt to prevent the client's access to data, using several real-time methods briefly outlined below. From this point forward we will not consider post-hoc punishment or forensic analysis of user equipment, especially considering the properties provided by the MCON underlay — even if the MCON identity of a client or data provider is known, as well as the nature of accessed/provided information, an adversary would first have to break the underlying membership concealment properties in order to locate specific participants.

- *Computer-local attacker.*
  - *Unprivileged access.* The attacker controls an unprivileged user account on the same physical computer as the client. The adversary may or may not be aware that the censorship-resistant software is being used. He may have permissions to read and write various files on the computer, possibly those related to the censorship-resistant software.
  - Privileged access (full control). The attacker controls all aspects of the client's computer, including being able to actively or passively examine all memory and intercept instructions dispatched to the CPU. This attacker may be a hardware manufacturer (Intel, AMD, Nvidia) or computer manufacturer (Dell, Lenovo, Apple) [117]. The attack vector may also be software running as a privileged user, such as Green Dam [118].
- *LAN-local attacker.* The attacker controls one or more computers on the client's local LAN (or cable Internet segment). He can observe all local traffic and send traffic to the client. Depending on the nature of the LAN, he may be able to launch interception/man-in-the-middle (MitM) attacks, or DoS the client by consuming local bandwidth or jamming the physical channel.

- *Service-specific attacker.*
  - *General Internet services.* This attacker controls one or more services which the client uses as a sideeffect of using the censorship-resistant system specifically, or the Internet in general. These attackers include DNS providers and immediate upstream Internet service providers (ISPs). Notice that such an attacker can trivially carry out a DoS attack by simply blocking all Internet access.
  - *Censorship-resistance service.* This attacker controls one or more members of the censorship-resistant network, such as a malicious storage server. It may control all members of a semi-centralized service, or be the only member of a centralized service (e.g. `Anonymizer.com` [9]), the latter allowing it to monitor and/or block all client communication through the service, potentially causing the client to make the request in the clear [119].
- *Global attacker.* This attacker controls Internet access for a large number of clients. It can use BGP maliciously to observe aggregated traffic from any/all other ASes. It can trivially locate and block all traffic to a given centralized or semi-centralized system either passively or by making malicious BGP announcements, causing other ASes' traffic to flow through the attacker [13].

4.1.6 System Types and Parties of Interest

Other than the client, our censorship-resistant system can have several types of participants, any one or more of which may be controlled by the attacker. Their roles are briefly sketched below (we re-define the client for completeness) and individual designs will be discussed in depth in the next section.

- *Client.* This is the party that would like to gain access to content. Depending on the system design, the client may be a *peer*, participating in the system by routing messages and/or storing data.
- *Publisher.* This is a subset of *client*, who publishes (stores) data to the network.
- *Intermediate service provider.* These are potentially multiple parties providing services not directly related to anonymity or censorship-resistance, but rather

support the underlying Internet communication infrastructure. These entities include ISPs and ASes, as well as DNS providers, fitting the definition of *general Internet services*, above.

- *Censorship-resistance service provider.* These entities are included in the above taxonomy as *censorship-resistance services*. We go into greater detail here.

  - *Storage-only systems.* These designs *provide robust storage infrastructure only*, leaving it up to the client to determine how to safely access the content. Content location services (indexing, search) may or may not be provided. Such designs are either centralized or semi-centralized, meaning that data is stored on multiple servers controlled by the same logical entity (e.g. WikiLeaks [10]), or on a relatively small number of servers controlled by independent individuals (e.g. Eternity [120]). Other examples include Free Haven [121], Publius [122], Tangler [11], and Serjantov's design [123]. The primary drawbacks of these designs are:

    * Limited to no resistance to legal and technological blocking of a limited server set
    * Limited publisher and/or client anonymity in the presence of compromised servers
    * Threats to content integrity/longevity in the presence of compromised servers
    * Questionable incentives for continued server operation

  - *Transport-only systems (anonymity provider).* These designs *provide transport services which clients use to request content*, leaving it up to the client to determine where to obtain the content (which content provider to choose, how to locate the content of interest, etc.). Such systems may be entirely centralized (e.g. `Anonymizer.com` [9]), semi-centralized (e.g. Tor bridges [7]) or completely decentralized (e.g. Infranet [124, 34][2]). These systems are essentially proxies, providing a layer of indirection between the client and the content provider. Fully decentralized systems rely on P2P designs, where

[2] This system requires limited cooperation from content providers and so is not entirely transport-only.

clients not only request content but also act as proxies for others. The primary issues with this type of design are:

  * Limited to no resistance to legal and technological blocking of a limited relay set
  * Limited client anonymity in the presence of compromised relays
  * Lack of scalability and/or inefficient routing
  * Questionable incentives for continued server operation

- *Storage and transport (censorship resistance provider).* These designs combine storage and transport into a single system. They are typically fully distributed, i.e. clients serve as both relays and content providers. Content location services (indexing, search) are generally not provided. Examples include: Freenet [8], GNUnet [125], and a design by Fiat and Saia [126]. The primary issues with this type of design are:

  * Limited to no resistance to legal and technological blocking of an enumerable peer set
  * Limited publisher and/or client anonymity in the presence of compromised servers
  * Lack of scalability and/or inefficient routing
  * Difficulties in locating rare content within the system (popular content is generally given preference)

4.1.7 Formal Definition of Censorship Resistance

There are several partially conflicting definitions of censorship-resistant systems. Danezis and Anderson define censorship as an external entity's attempt to impose on a set of nodes a particular distribution of files [127]. Perng *et al.* define censorship susceptibility as the likelihood a third-party can restrict a targeted document while allowing at least one other document to be retrieved [128]. Fiat and Saia uses the term "censorship resistant" in the sense that even after adversarial removal of an arbitrarily large constant fraction of the nodes in the network, all but an arbitrarily small fraction of the remaining nodes can obtain all but an arbitrarily small fraction of the original data

items [129]. We subscribe to the third definition since it incorporates Danezis and Anderson's definition and does not limit the adversary from removing access to a system entirely.

Formally, we say that a network is $(\Lambda, \Gamma, f', \rho)$-censorship-resistant if no (ℓ, γ)-adversary monitoring or disrupting $\ell \leq \Lambda$ links and corrupting or otherwise controlling $\gamma \leq \Gamma$ members can block more than $f'(\ell, \gamma, N)$ nodes from accessing more than a constant fraction ρ of content where N is the total number of network members. When $f'(\ell, \gamma, N) = \Theta(\ell + \gamma)$ we call the network $(\Lambda, \Gamma, f', \rho)$-censorship-resistant. ($\rho$ must be defined as sufficiently close to 1 for the network to be useful.) Note that any formal definition for a censorship-resistant network is closely tied to the definition of membership concealment. We refer the reader to our formal characterization of MCONs in Section 2.2, where we define a (ℓ, γ)-membership-revealing adversary, and observe that since the goal of an MCON is to prevent identification and subsequent blocking of nodes, $f'(\ell, \gamma, N) \leq f(\ell, \gamma, N) \leq \Theta(\ell + \gamma)$ for some ρ.

4.2 Related Work

A number of naïve and state-of-the art censorship-resistant systems are briefly described below, and a high-level comparison may be found in Table 4.1.

4.2.1 Limitations of Naïve Approaches

Unfortunately, many naïve schemes have been deployed in repeated attempts to create censorship-resistance systems. Such approaches tend to focus on thwarting specific censorship techniques, and can be easily defeated if deployed widely-enough to be noticed by the adversary (such as ignoring TCP RST packets [130]). These systems include:

- *Fast flux and domain flux.* Phishing sites have been known to use DNS fast flux [131] (returning different IP addresses for repeated DNS queries to the same domain) or domain flux to contact different domain names used as control centers, which change over time. These strategies are easily overcome by either shutting down the DNS server [132, 133], taking over current or future domains [132, 133], or blocking all the domain queries or URLs at the ISP level.

- *Steganography.* There is extensive literature on TCP/IP steganography and covert channels, including HTTP-embedded messages and tunneling information through DNS queries [20, 79, 88, 89, 90, 91], which can be used when accessing censored content. However, Murdoch and Lewis [79] show that many of these proposals are easily detected. Even when using provably secure steganography [78], the location of censored documents has to be somehow disseminated, giving the adversary an easier target. Improperly using keyed steganography falls victim to the same insider attack as a password-protected website.

- *Using file-sharing networks.* Likewise, using general-purpose file-sharing networks (e.g. Kad [134] or BitTorrent [135]) or public document storage sites (e.g. Google Docs) for censorship resistance is not secure against malicious insiders who can either block access to content [44] or disrupt the entire file sharing network [136].

A common issue with all these schemes is that in order to work they would require sophisticated key management to prevent insiders from learning most or all possible keys and subsequently removing content. It is currently unclear whether this key management problem can be solved in a manner allowing the use of public file-sharing systems, domain flux, or keyed steganography.

4.2.2 The State of the Art

Many of the current censorship-resistant publishing systems have tried to implement the Eternity service proposed by Anderson [120]. Such designs can be mostly partitioned into two sets: those that provide the communication infrastructure, and those that provide storage. Storage providers can be further subdivided into semi-centralized and decentralized systems. We will focus on low-latency approaches since they are far more likely to be accepted by the user community. Moreover, high-latency systems such as mixes are still vulnerable to malicious insiders and blocking by enumeration. All such systems can be broken down into the following rough taxonomy:

- *Anonymizing proxies.* These systems provide a layer of indirection between the client and the content provider: the client will communicate with a proxy, who will then forward the request on behalf of the client, hiding the identity of the client

from the provider. Likewise, if communication with the proxy is encrypted, the identity of the provider will be hard to determine for anyone monitoring the client's network communication. These systems include `Anonymizer.com` [9], Crowds [40], Nonesuch [33], Tarzan [41], Salsa [43], a large number of mix systems [137, 138, 139, 140, 38], and any open proxies that may still be available on the Internet. A major drawback is that proxy identities must be distributed in a secure manner that does not fall victim to the same attacks that allow centralized systems to be blocked.

- *Censorship-resistant systems.* These designs provide a robust permanent storage infrastructure. Some attempt to shield the client's identity from the data provider, or even the exact nature of the content the client accesses. These systems are either (semi-)centralized, comprised of one to a few thousand servers, or are completely decentralized, requiring clients to both store data and proxy connection requests.

  - *Centralized and semi-centralized systems.* Data access is provided either by multiple servers controlled by the same logical entity (e.g. WikiLeaks [10]), or a small to moderate number of servers controlled by individuals (e.g. Eternity [120], Tor bridges [7]). Other examples include Free Haven [121], Publius [122], Tangler [11], and Serjantov's design [123]. These systems can be easily blocked at the network border since every client must know the IP addresses of the servers.
  - *Decentralized systems.* A fully-distributed system spreads the information distribution role over far more entities, and is thus potentially more resistant to DoS attacks. Examples include Fiat-Saia [126], LOCKSS [141], Turtle [54], Kaleidescope [55], Freenet [8], and GNUnet [125]. Another scheme by Feamster *et al.* adds a decentralized component to Infranet [34]. These systems are also more robust to insider attack, since malicious nodes will affect clients with only a small probability. Unfortunately, these services can still fall victim to member enumeration by either an insider or outsider adversary (depending on the specific design), and so can be blocked at the network layer (see Chapter 2).

	Attack resistance		
Design	**Protocol-generic**	**Protocol-specific**	**Efficient**
Ad-hoc systems			
Domain flux	no: passive outsider	no: active outsider	yes
DNS fast flux [131]	no: passive outsider	no: passive outsider	yes
Open proxies	no: passive outsider	no: passive outsider	yes
File sharing	no: passive outsider	no: passive outsider	yes
Steganography	no: passive outsider	no: active outsider	yes
Centralized storage systems			
Eternity [120]	no: passive outsider	no: active insider	yes
Free Haven [121]	no: passive outsider	no: active insider	yes
Infranet [124]	no: passive outsider	no: active insider	yes
Publius [122]	no: passive outsider	no: active insider	yes
WikiLeaks [10]	no: passive outsider	no: active insider	yes
Anonymizing systems			
`Anonymizer.com` [9]	no: passive outsider	no: passive insider	yes
Crowds [40]	yes	no: active outsider	yes
Nonesuch [33]	no: passive outsider	no: active insider	no
Salsa [43]	yes	no: active insider	yes
Tarzan [41]	yes	no: active insider	yes
Tor [39]	no: passive outsider	no: active insider	yes
Censorship-resistant systems			
Fiat-Saia [126]	no: passive outsider	no: active insider	yes
Freenet [8]	no: passive outsider	no: active insider	no
GNUnet [125]	no: passive outsider	no: active insider	no
Improved Infranet [34]	yes	no: active outsider	yes
Kaleidescope [55]	yes	no: passive insider	yes
LOCKSS [141]	no: passive outsider	no: active insider	yes
MCONs (Chapter 2)	yes	yes	yes
OneSwarm [49]	no: passive outsider	no: passive insider	no
Serjantov [123]	no: passive outsider	no: active insider	yes
Tangler [11]	no: passive outsider	no: active insider	yes
Tor bridges [7]	yes	no: passive insider	yes
Turtle [54]	yes	no: passive insider	yes

Table 4.1: Summary of attack resistance of current censorship-resistant systems.

Infranet [124] partially addresses the enumeration-and-blocking problem by using steganographic techniques to hide content requests and responses. However, it requires active participation of a number of web servers and falls victim to the same attacks as other steganographic schemes mentioned above. In [34], Feamster *et al.* extend Infranet by adding an extra layer of indirection in the form of untrusted messengers, who pass requests to a forwarder, who then fetches the actual censored content. The latter is an excellent example of using transport providers to connect clients to storage-only systems. Another example is Tor bridges [7], which add limited membership-concealment functionality to the Tor anonymous overlay [39], allowing it to be used for censorship resistance.

Tor employs a *client-server* overlay design, where a publicly known list of nodes who relay traffic for a larger set of users who are neither explicitly revealed nor explicitly protected. The public nature of the relay set, which is required for anonymity against various attacks, implies that Tor itself can be blocked by enumerating participating nodes. In response, Tor designers introduced *bridges* [7], which are Tor clients (not relays) serving as gateways into the Tor network. Bridges allow other clients access to the Tor network even when the clients are using an adversarial ISP that blocks all known Tor relays. Tor then utilizes a centralized agent — the *Bridge Authority* — to hand out references to available bridges in a way that limits the number of nodes that an adversary can obtain. Since bridges are client nodes, they are more numerous and experience higher churn, so blocking them is a more difficult task. We note that this is actually a membership concealing feature — one cannot block what one cannot enumerate. However, both Tor bridges [7] and Infranet [124] with untrusted intermediaries [34] are still vulnerable to attack. Section 2.4 details a successful enumeration attack on Tor bridges. Infranet is additionally vulnerable to blocking at the network border using a load-balancing protocol proxy. The messenger groups are assigned based on the client's IP address, and will only service requests from the same source address. A load-balancing HTTP proxy may request a messenger descriptor using one IP address, but route subsequent communication through a different address, ensuring that messengers will not respond to the fetched using one IP address, but route subsequent communication through a different address, ensuring that messengers will not respond to the fetched descriptor.

4.2.3 Robust Distributed Storage

There is much literature on the subject of robust distributed storage resistant to malicious tampering. Storer, Greenan, and Miller provide a good taxonomy of *long-term archival systems* and potential security issues in [142]. These systems are a subset of distributed storage since they must provide robustness and security properties for extended periods of time, unlike distributed filesystems optimized for short- to medium-term storage. Furthermore, these systems are generally meant for *archival* storage, meaning data access is likely infrequent. The authors discuss a number of core threats including loss of secrecy, loss of file integrity, unauthorized access, slow prolonged attacks, and data migration issues.

Secrecy concerns are relevant to systems which attempt to only allow access to given pieces of content to authors and/or authorized parties only. While key management may be considered out of scope by some designs, it is nonetheless a relevant issue since long-term storage exposes systems to data which is encrypted with outdated, compromised, or lost keys. Some systems use secret sharing, splitting the file between multiple storage servers [143, 144, 145] such that none learn anything about the file unless enough servers cooperate to obtain greater than a threshold number of shares. User authentication is a related issue, encompassing challenges such as key revocation for old users of compromised keys, and re-encryption of files with keys that have been revoked.

Integrity is a concern in any storage system, either against benign corruption or malicious tampering. This is particularly crucial in archival systems, as one must be sure that a file being accessed is identical to the one that was stored an arbitrarily-long time ago. *Opportunistic scrubbing*, or periodically checking integrity of stored files and repairing if needed, has been shown to be a good middle ground between over-zealous integrity checks, which waste resources, and late verification, which may occur when file recovery has already become impossible [146]. However, opportunistic scrubbing is generally performed at file access time, making it a bad fit for read-maybe systems. Therefore, proactive scrubbing and replication are required to maintain integrity [144, 147].

Systems providing protection against Byzantine failure [148, 149, 150, 147, 151, 152, 153] guarantee that any file can be recovered as long as the maximum number

of failed hosts is lower than a threshold fraction. However, the number of hosts that must be contacted to store a file becomes prohibitive as the number of storage servers increases [149], making these systems a bad fit for P2P designs. Additionally, due to the long lifetime of these archives, there is no bound on how long their security properties must be retained. Such systems have to be concerned about slow attacks in which long-lived adversaries slowly acquire key shares such that eventually they may decrypt certain files: the failure threshold must only be exceeded briefly to break Byzantine guarantees even if the number of malicious hosts is lower than maximum failure threshold throughout most of the system lifetime [142].[3]

4.3 System Design

CROPS is designed to be a write-once, read-maybe long-term archival system, implemented as a storage layer on top of an MCON 2, thus providing both storage and transport. Since we build on prior research into distributed membership-concealing systems, we are protected from adversarial blocking (unlike previous systems). CROPS implements robust storage while preserving the membership-concealing properties of the routing layer. However, availability guarantees are far more difficult given that nodes in the system may be malicious and will undergo churn — since system members are not dedicated high-uptime servers, they will oscillate between online and offline states, possibly in an unpredictable manner. Since we cannot assume that nodes will behave correctly, we must use proactive replication and reactive repair to ensure availability, as opposed to relying on cooperative behavior such as load-balancing and data handoff.

Since we do not aim to keep files confidential, we avoid most of the issues mentioned in [142]. In fact, the only problems we face are file integrity and key management. The former is greatly simplified by the *requirement that stored data be immutable* — it allows us to use simple encryption and signature schemes to ensure authenticity and publisher-continuity (if a publisher's pseudonym is a hash of her public key, files encrypted and signed by the author are self-authenticating). We can thus forego the expense of generalized Byzantine fault-tolerant designs, which are prohibitively costly

[3] [153] is a notable exception, where all versions of data whose consistency cannot be guaranteed are maintained by the system as separate entries.

in terms of computation and communication in large systems [149]. Furthermore, since "modified" data represents a completely new document, we can avoid the complexities of a versioning filesystem [110]. Note that since we use only self-certified pseudonyms we cannot ensure file integrity — there is no way to determine if a given file has been modified since publication because the publishing party is a priori unknown. However, we do ensure that files signed by a given publisher were indeed produced by that person or a collaborator with access to the publisher's private key.

Combining key management and plausible deniability is a difficult task: if an archival file system is self-contained (meaning no external resource is required to access stored plaintext), keys as well as content must reside within the archive. Both must be retrievable, but to preserve plausible deniability we must ensure that if a peer has access to a key, she should not automatically be able to find the file, and vice versa. A number of systems [143, 144, 145] avoid encryption completely by using secret sharing — the file is split across multiple servers such that no single server holds enough secrets to reconstruct the file, learning nothing about its contents. Other designs rely on storing encrypted data, leaving key management to the clients [112, 122], specialized hardware [111], or directory servers [154]. None of these approaches can be used with our system since client-based key management would make search and retrieval difficult, and neither specialized hardware nor directory servers are secure against a powerful adversary. Fortunately, we do not aim to maintain persistent secrecy of file contents[4] or keywords, rather only separate them to ensure storer plausible deniability. In fact, we would like files to be accessible (searchable, downloadable, and decryptable) by as large a population of users as possible. We describe a way to achieve both plausible deniability and easy file lookup later in this section.

4.3.1 Robust DHT-based Storage

Since MCONs already incorporate DHTs, searching for content becomes trivial. However, DHTs are quite vulnerable to both targeted and existential censorship — adversaries can overwhelm clusters of nodes with requests and cause them to cease functioning, and thus stop serving hosted content. If the attack targets are all (or most)

[4] However, any two parties choosing to communicate over MCON/CROPS can use out-of-band key management to maintain secrecy.

nodes who host a particular piece of content, the attack can be called "targeted." Targeted censorship is a particular problem if all replicas of a given file are in the same DHT "neighborhood," as in systems like Kademlia [53]. While adversaries cannot easily attack nodes from outside the network (recall that MCONs make determining the IP address of a given node prohibitive), a number of colluding insiders can launch a denial of service attack *within* the network. A flood of adversaries requesting content hosted by a relatively small number of nodes in a given neighborhood will overwhelm those nodes, disrupting availability of all the data they host. Adversaries who control network gateways can also disable access to large sections of the Internet with the intention of censoring data that may be hosted at a physically-isolated location. Therefore, a more robust strategy is needed.

4.3.2 Resisting Massive Correlated Failures

Much like prior systems such as [152] and [113], we use erasure codes and a very high replication factor to defeat both targeted and existential attacks, providing better robustness than replication alone can. Our approach is most similar to that of Glacier [113], a DHT-based filesystem designed to be robust against *massive correlated failures*. However, we must adjust the design somewhat, since Glacier is not an archival storage system but is geared for content-aware collaborative filesystems. While DHT queries are churn-resistant by design, traditional DHTs such as Kademlia [53] are unlikely to maintain data availability in case entire "neighborhoods" fail — clusters of nodes with similar logical identities. Glacier adds additional resistance against network failure in case of large-scale failure events (such as the loss of a majority of all network participants) by using aggressive replication and storage at pseudorandom DHT locations (as opposed to any given object being stored in only one DHT neighborhood).

We expect the operating environment of CROPS to be significantly different from that of Glacier in terms of network latencies and member node churn. As we expect to operate in significantly higher-churn environments than Glacier, and cannot rely on long-term cooperation from any node, we must take a different approach. Furthermore, CROPS peers are explicitly content-oblivious, eliminating some optimizations that are possible with Glacier. Finally, we must protect against DoS attacks which are out of scope of the Glacier design, such as the storage exhaustion attack.

4.3.3 The CROPS Protocol

We use erasure codes to ensure file availability: we apply an m-of-n code such that out of n post-encoding blocks, only m need to be retrieved to reconstruct the original file. These blocks are stored in the network based on the hash of the block content [155], meaning that blocks are all self-verifying — fetching a block allows a client to verify that the block was fetched correctly because the hash of the block is the block identity [156]. The parameters of the erasure code specify the replication and storage overhead — an m-of-n code imposes a factor of $\frac{n}{m}$ increase on the amount of data stored in the system. Due to the nature of our publishing protocol (discussed below), it is possible to support publisher-specified erasure code algorithms and parameters. However, we will not discuss this at length and will instead focus on selecting m and n values that are "good-enough" to store some particular amount of data indefinitely.

Publishing

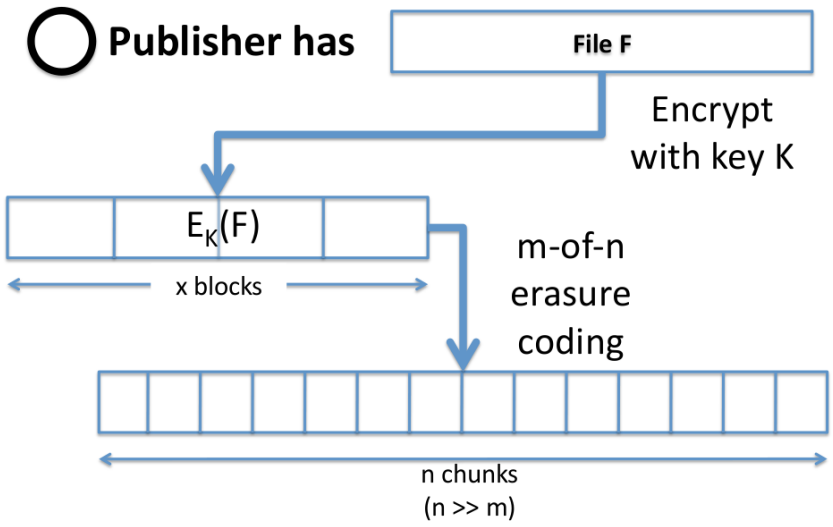


Figure 4.1: A publisher encrypts a file and applies an m-of-n erasure coding scheme.

We assume that publisher $\mathcal{P}$ has a persistent MCON pseudonym based on the result

of hashing the public key of an asymmetric key pair with a pre-image- and collision-resistant hash function h. Therefore, let the public and secret keys be PK and SK, respectively, and the pseudonym be $h(PK)$. Before publishing a file F, $\mathcal{P}$ compiles a list of keywords $\mathcal{K}$ that describe the file contents. The publisher selects a random integer K of arbitrary bit-length and encrypts the file F using a symmetric cipher keyed with K, producing $E_K(F)$. Let the length of $E_K(F)$ be a multiple of x blocks of bit-length b. We apply an m-of-n erasure code to each block, yielding a total of $\frac{xn}{m}$ encoded blocks, producing $EC_m^n(E_K(F))$. Note that erasure coding is performed on already-encrypted file blocks. Each of the n encoded blocks can now be inserted into the DHT under the key $h(n)$. The process is diagrammed in Figure 4.1.

File manifest. Each file has an associated *manifest*, or metadata file containing various identifying and authenticating information. A publisher must compose and publish two manifests: one for the key, and one for the file content. One can think of them as "rich content pointers," similar to simple content pointers used in Kad [134], that contain information about the network nodes storing the erasure-coded chunks of the desired file. Note that the manifest could hold publisher-selected values for m, n, and the erasure code algorithm.

A content manifest has the following format (see Figure 4.2:

- $h(E_K(F))$, a hash of the encrypted file content to verify that a file was successfully reconstructed from multiple chunks
- a list of hashed keywords chosen by the publisher for searching
- $h(F)$, a hash of the file content before erasure coding to verify that someone decoded and decrypted a file successfully
- $C_1, C_2, \cdots, C_n : \forall C \in EC_m^n(E_K(F))$, the list of chunk storage locations
- $h(PK)$, a hash of the publisher's pseudonym and public key for authentication and publisher linkability
- $Sig_{SK}(\mathcal{M})$, a signature over the entire manifest by the publisher for manifest integrity protection

A key manifest is essentially identical, except that the list of chunk storage locations is replaced with a single plaintext copy of K, and $h(F)$ is replaced with $h(K)$. These manifests are not erasure-coded, but rather stored whole and aggressively replicated. Each manifest is indexed by the contained keywords as well as replica numbers, such that if a given manifest contained 3 keywords and the replication factor was 3, the storage locations would be: $h(1, h(\text{keyword}_1)), h(2, h(\text{keyword}_1)), \cdots, h(3, h(\text{keyword}_1))$,
$h(1, h(\text{keyword}_2)), h(2, h(\text{keyword}_2)), \cdots, h(3, h(\text{keyword}_2))$,
$h(1, h(\text{keyword}_3)), h(2, h(\text{keyword}_3)), \cdots, h(3, h(\text{keyword}_3))$.
Note that in order to deny access to a file it is sufficient to deny access to all copies of either its key manifest or content manifest, therefore a particularly aggressive replication strategy is needed to ensure manifest availability.

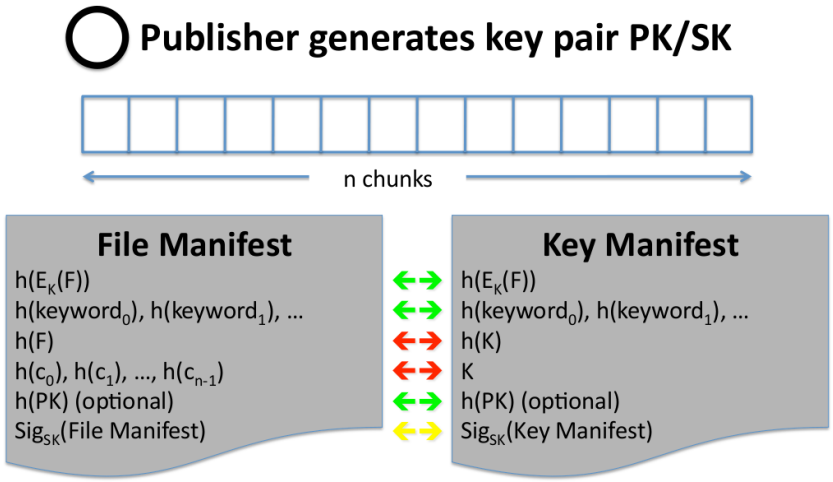


Figure 4.2: A publisher composes a file manifest containing the identities of all erasure-coded file chunks, and a key manifest containing the key itself.

Manifest "guarantors." Each peer in the network who had a copy of a manifest is referred to as a *manifest guarantor*, and is responsible for maintaining the replication factor of both the manifest and the content to which the manifest refers. Although manifest-holder knows the location of every file chunk and manifest, it cannot determine the content of the file since it only has access to hashes of keywords. Furthermore, fetching and reconstructing the file does not expose its contents to the manifest-holder,

since the file has been encrypted and the manifest-holder does not know the key. The converse is true for the holder of a key manifest — while it knows the key, it does not know which file the key decrypts. The properties of the manifest-holder are:

- Can re-assemble an entire erasure-coded encrypted file
- Cannot obtain the file plaintext
- Cannot obtain the file keywords
- Cannot alter the manifest without breaking the signature scheme used to sign it
- Cannot remove file chunks or manifests from the network other than dropping its own

File Searching and Retrieval

To retrieve a file from the DHT, a searcher first obtains the file manifest by hashing meaningful search terms and fetching manifest replicas stored at those hash keys. (Note that since search terms are hashed, everyone forwarding the search request or storing the results has plausible deniability as to the target of the query.) Since each keyword query will return a number of manifests, the searcher takes the intersection of all returned manifest to find the one that matches all the keywords in the query. The searcher repeats the process with the key manifest. Key manifests can be matched with file manifests because they will each contain an identical $h(E_K(F))$, and can be distinguished from file manifests since they do not include a list of chunks. Once both manifests have been retrieved, the searching node can determine the location of all the content chunks in the network and fetch m of them to reconstruct the encrypted file. She can verify that the file has been correctly reconstructed by checking that the results match $h(E_K(F))$. She then uses the key K (contained in the key manifest) to decrypt the content and verify that it matches $h(F)$. Note that the key manifest and content manifests only come together at the searcher, and the only required piece of knowledge to initiate a search is the list of keywords related to the desired content.

Content Storage, Maintenance, and Retirement

Plausible Deniability Like in GNUnet [125], files in the CROPS network are not stored whole, but are divided into blocks, which are then distributed throughout the network based on the cryptographic hash of the block itself. Even if a peer is storing multiple blocks from the same file, he has no way to determine if that is the case, nor can he learn the file content since the files were encrypted before storage. Manifest holders are likewise protected even if they are also storing chunks of the file, because reconstruction of the file would yield encrypted data and not the file cleartext. However, if a peer stores both the key manifest and the content manifest, he can link those manifests together, reconstruct and decrypt the file, learning the cleartext. We note that it is not in the best interest of an honest storer to be in possession of both manifests, since she would lose plausible deniability if her computer were to be confiscated. Storing both manifests does not benefit malicious nodes, since they could simply search for keywords of interest. Therefore, peers should refuse to store key manifests if they already hold a content manifest, and vice versa. Peers can determine whether the manifests correspond to the same file by checking that their file hash $h(E_K(F))$ matches, and refuse storage of the second manifest. Forward-secure encryption must be used by the publisher when making the storage request, since otherwise an adversary may record the transaction and destroy the storer's deniability. If the publisher itself is malicious, the storer can claim to simply be following the protocol. Furthermore, even if an adversary knows the network pseudonym of a particular storer, locating that storer in the real world is difficult due to the properties provided by the underlying MCON layer.

We provide storer protection even stronger than plausible deniability, which we call *probable ignorance*: while plausible deniability implies that a party may or may not have known something, probable ignorance implies that a party had little chance of having that knowledge. In our case, determining the nature of stored files is essentially impossible,[5] and content manifest-holders would have to either crack the file encryption key K or invert the keyword hashes in order to infer file content. Key manifest-holders would also have to invert the keyword hashes in order to determine the file to which their stored key corresponds.

[5] Timing attacks may be used to identify correlated chunks, but the content of the decrypted file cannot be determined.

Garbage collection. Content storers keep a timestamp associated with every locally stored block (initialized with the original publication time of that block), and update the timestamp every time that block is accessed by another user in the network. During idle times, storers lazily examine their stored blocks and probabilistically discard blocks that have timestamps older than τ', clearing storage space for new blocks. Recall the problem of pollution attacks (discussed in Section 4.3.2), where adversaries can to overwhelm the storage capacity of the entire network by inserting garbage. A number of archival schemes use periodic refresh to maintain content in a network and purge old or unpopular files [8, 115]. This garbage collection scheme ensures that un-refreshed data does not remain in the network for a long time, requiring adversaries to continue inserting or accessing their data in order to prolong the pollution attack.

Content-oblivious robust replication. Our replication assurance strategy relies on mutual cooperation of honest nodes to achieve resilience. While files are broken up into chunks, manifests are kept whole. Each node storing a manifest is "responsible" for the file to which the manifest points. Note, however, that the manifest holder does not know the content of the file since he cannot decrypt it without key K. To ensure availability in cases of failure of a large number of nodes, manifest holders constantly monitor the replication factor of the manifest target. Every time period τ, every node examines all stored manifests and downloads and reconstructs every encrypted file from chunks stored in the network. Applying the erasure code, the node can obtain copies of all chunks that should be stored. By searching for a sample of those chunks, the manifest holder can probabilistically determine the *current replication factor* of a file and compare it to the *desired* replication factor. If the difference is significant, the node inserts all missing blocks back into the DHT. The node does the same thing with the manifest itself, checking to see if enough replicas are available, and creating new replicas if not. Manifest-holders, through checking replication factor and accessing content chunks, implicitly serve to refresh timestamps of files in the network without requiring action on the part of the publisher.

Like the garbage collection scheme described above, manifests that are not being actively accessed or are over-replicated can be probabilistically discarded. (We do not discard deterministically since we may encounter the extremely unlikely situation wherein

all manifest holders simultaneously check the replication factor, discover that a manifest is over-replicated, and drop it.) As manifests are dropped, references to content chunks are lost with them, and content will no longer be accessed, eventually also being discarded. Note that *a single honest manifest-holder is sufficient to maintain the replication factor* of any piece of content with overwhelming probability, as long as the he can successfully retrieve the minimum number of blocks required to reconstruct the file at least once.

Although malicious manifest holders are a concern, the worst attack they can carry out is corrupting or dropping their own copy of the manifest. As long as they cannot differentiate between a manifest query for reconstruction purposes and a query for retrieval purposes they could not preferentially deny service to one rather than the other. This is equivalent to simply dropping the manifest — an honest guarantor will increase the replication factor to compensate.

Emulating WikiLeaks

CROPS is similar in concept to a massively distributed version of the WikiLeaks service [10]. MCONs are an especially attractive building block for CROPS considering the nature of WikiLeaks: the high latencies in the MCON makes real-time communication difficult, but since WikiLeaks is mostly used as a storehouse for large documents, latencies between 30 and 90 seconds should be quite acceptable, much like in file sharing systems.

In keeping with the spirit of WikiLeaks, we chose to use an *editor-facilitated* publishing model. While a free-for-all model is attractive (where all published content is maintained indefinitely), it suffers from a number of drawbacks such as unregulated content quality, pollution and collision attacks, and storage space concerns. We also reject automatic data filtering by popularity, such as that used by [125], since this can lead to vulnerabilities [157]. Moreover, unpopular content may be just as important, if not more important, than popular content. CROPS will be bootstrapped with a set of hard-coded *editor public keys*, the private counterparts to which are held by a select group of pseudo-administrators. Editor keys can be used to sign manifests to add a "stamp of approval."[6] Of course, an editor-assisted model introduces its own set

[6] Both the content and key manifests must be signed.

of problems, such as attacks on editor time. To address this issue, we use a hybrid model such that both editor-approved and editor-unapproved data (whether explicitly or simply for lack of examination) can still be stored and retrieved, but editor-approved data has special protection in that it will not be dropped from the network. Honest manifest-holders do not apply pruning to editor-signed manifests, but rather keep them forever, independent of the last time they were accessed. Since manifest-holders refresh content blocks in the network, the content is safe as well. Manifests not carrying editor signatures will be garbage-collected as descried above. Note that we are not particularly concerned with malicious editors — an editor cannot explicitly remove content from the system, so even a single honest editor is sufficient to ensure that important content makes it past the editorial process.

In order to bring newly-published documents to editors' attention,[7] notifications of document publication should be inserted to a number of designated network locations, for instance DHT IDs that correspond to the hash of editor public keys combined with the current date. The details of this scheme are still unclear and will require future work in order to determine the correct way to resist attack on those dedicated network IDs.

4.4 Theoretical Analysis

The authors of Glacier [113] derive the following equation for the *durability* of a block when using an m-of-n erasure-coding scheme:

$$D = P(s \geq m) = \sum_{k=m}^{n} \binom{n}{k} (1 - f_{max})^k \cdot f_{max}^{n-k}.$$

Note that this assumes that each block n is stored at a different network node. Therefore, if b blocks are stored in the network then the probability that every file is recoverable, or the *robustness of the censorship-resistant system* is $\rho(b) = 1 - D^b$. In other words, a 1-robust censorship-resistant storage system *always allows for the retrieval of every file*, assuming a node can successfully connect to the system. In CROPS, clients must first retrieve key and content manifests before fetching content blocks. To simplify analysis

[7] A publisher who wants to see her work widely disseminated has a strong incentive to get it signed by an editor.

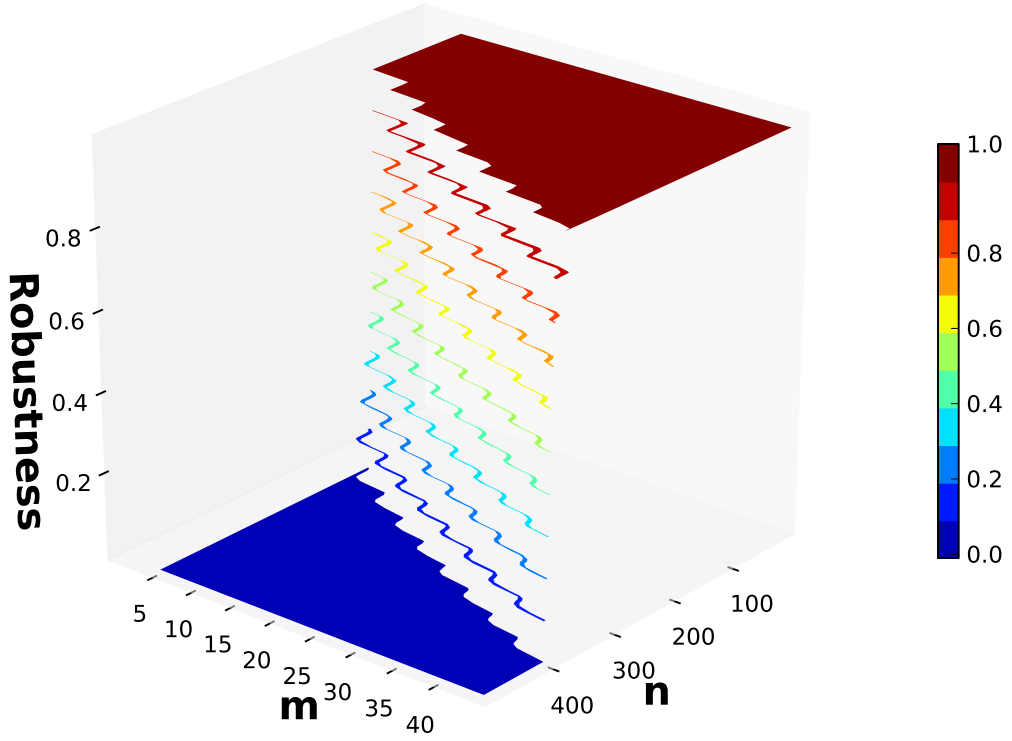


Figure 4.3: Determining the optimum erasure-coding variables to support up to 100 petabytes of network storage. n and m are erasure code parameters, while the "y" axis is the robustness ρ of a censorship-resistant system, showing the fraction of nodes that must be offline or malicious before data loss begins to occur.

we assume that the manifest replication factor is the same as that of the erasure code used.

Figures 4.3 and 4.4 show that nearly a factor of 10 storage overhead (in terms of erasure code parameters) is required for a robust censorship-resistant system that can support 1 exabyte of total storage, which provides 100 petabytes of *useful storage* when accounting for erasure coding overhead. The figures also show that we do not gain much advantage by moving to a lower overhead, since robustness drops quite sharply at about factor of 8 overhead. The measurements were obtained from uniformly sampling m and n values from 1–50 and 5–500, respectively. Tables 4.2, 4.3, and 4.2 (found at the end

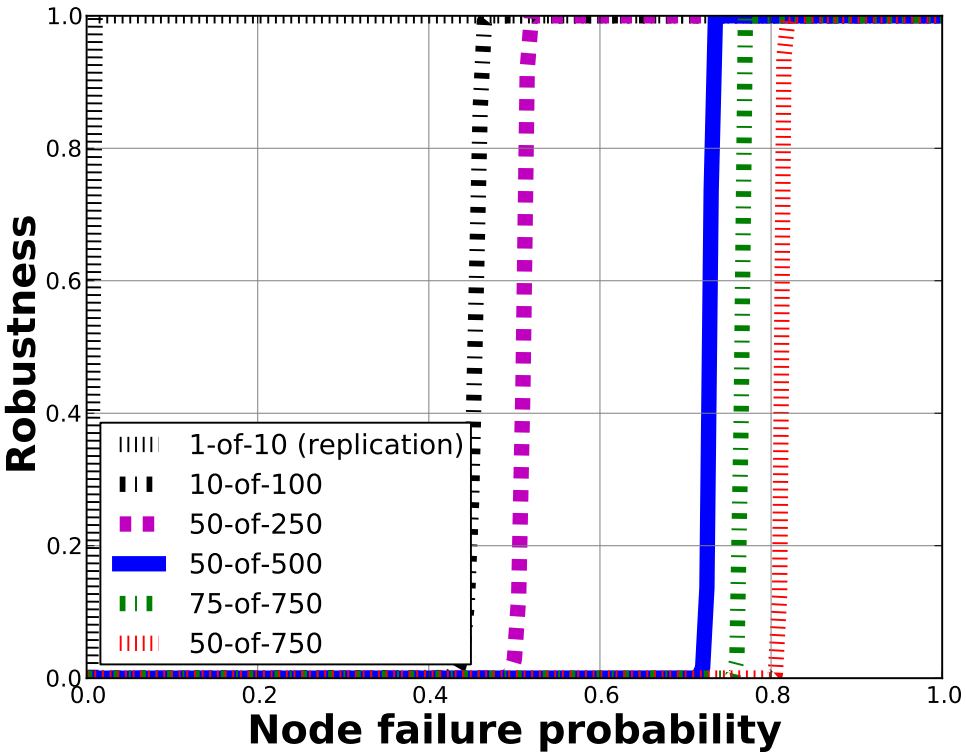


Figure 4.4: The robustness ρ of the censorship-resistant system using a given erasure code configuration or simple replication. The 50-of-500 configuration is a good tradeoff between overhead and robustness, as is 75-of-750. Both impose a factor of 10 storage overhead.

of this chapter) support this observation, showing the network robustness for several values of m and n for a given amount of total storage in the network. We observe that nothing less than an erasure code requiring a 10-fold increase in storage gives us a reasonable robustness value. Fortunately, that storage overhead allows us a practically unlimited storage: between 1 exabyte and 1 zettabyte.

4.4.1 Formal Statement of Censorship Resistance

Recall our theoretical analysis of a censorship-resistant network from Section 4.1.7. Since CROPS must necessarily be at least $f(\ell, \gamma, N)$-membership-concealing, the instantiation

of our scheme with robust MCON routing (see Section 2.6.2), 50-of-500 erasure coding, and zettabyte capacity is $(\frac{N}{2k^2}, \frac{N}{2k^2}, f', 1.33 \cdot 10^{-6})$-censorship-resistant for $f'(\ell, \gamma, N) = k^2(\ell+\gamma)$ without protocol obfuscation, and is $(N, \frac{N}{k^2}, f', 1.33 \cdot 10^{-6})$-censorship-resistant for $f'(\ell, \gamma, N) = k^2\ell$ with perfect obfuscation.

Petabyte of storage						
	25-of-100 erasure coding		30-of-200 erasure coding		50-of-500 erasure coding	
$E[f]$	D	ρ	D	ρ	D	ρ
0.00	1.0	0.0	1.0	0.0	1.0	0.0
0.05	1.0	0.0	1.0	0.0	1.0	0.0
0.10	1.0	0.0	1.0	0.0	1.0	0.0
0.15	1.0	0.0	1.0	0.0	1.0	0.0
0.20	1.0	$3.48 \cdot 10^{-18}$	1.0	0.0	1.0	0.0
0.25	1.0	$1.76 \cdot 10^{-11}$	1.0	0.0	1.0	0.0
0.30	1.0	$3.62 \cdot 10^{-6}$	1.0	0.0	1.0	0.0
0.35	0.999	0.075	1.0	0.0	1.0	0.0
0.40	0.999	1.0	1.0	0.0	1.0	0.0
0.45	0.999	1.0	1.0	$1.37 \cdot 10^{-17}$	1.0	0.0
0.50	0.999	1.0	1.0	$6.05 \cdot 10^{-11}$	1.0	0.0
0.55	0.999	1.0	0.999	0.000036	1.0	0.0
0.60	0.9994	1.0	0.999	0.97	1.0	0.0
0.65	0.99	1.0	0.999	1.0	1.0	0.0
0.70	0.89	1.0	0.999	1.0	1.0	$1.27 \cdot 10^{-12}$
0.75	0.54	1.0	0.9998	1.0	0.999	0.0060
0.80	0.13	1.0	0.97	1.0	0.999	1.0
0.85	0.0061	1.0	0.53	1.0	0.9996	1.0
0.90	0.000013	1.0	0.016	1.0	0.52	1.0
0.95	$1.82 \cdot 10^{-11}$	1.0	$8.71 \cdot 10^{-8}$	1.0	$3.58 \cdot 10^{-6}$	1.0
1.00	0.0	1.0	0.0	1.0	0.0	1.0

Table 4.2: Expected fraction of failed, malicious, or blocked nodes (left); the durability of a given erasure-coded block in the network (D); and the network robustness in terms of censorship resistance (ρ).

Exabyte of storage						
	25-of-100 erasure coding		30-of-200 erasure coding		50-of-500 erasure coding	
$E[f]$	D	ρ	D	ρ	D	ρ
0.00	1.0	0.0	1.0	0.0	1.0	0.0
0.05	1.0	0.0	1.0	0.0	1.0	0.0
0.10	1.0	0.0	1.0	0.0	1.0	0.0
0.15	1.0	0.0	1.0	0.0	1.0	0.0
0.20	1.0	$3.56 \cdot 10^{-15}$	1.0	0.0	1.0	0.0
0.25	1.0	$1.80 \cdot 10^{-8}$	1.0	0.0	1.0	0.0
0.30	1.0	0.0037	1.0	0.0	1.0	0.0
0.35	0.999	1.0	1.0	0.0	1.0	0.0
0.40	0.999	1.0	1.0	0.0	1.0	0.0
0.45	0.999	1.0	1.0	$1.42 \cdot 10^{-14}$	1.0	0.0
0.50	0.999	1.0	1.0	$6.193 \cdot 10^{-8}$	1.0	0.0
0.55	0.999	1.0	0.999	0.036	1.0	0.0
0.60	0.9994	1.0	0.999	1.0	1.0	0.0
0.65	0.988	1.0	0.999	1.0	1.0	$6.78 \cdot 10^{-21}$
0.70	0.89	1.0	0.999	1.0	1.0	$1.30 \cdot 10^{-9}$
0.75	0.54	1.0	0.9998	1.0	0.999	0.998
0.80	0.13	1.0	0.97	1.0	0.999	1.0
0.85	0.0061	1.0	0.53	1.0	0.9996	1.0
0.90	0.000013	1.0	0.016	1.0	0.52	1.0
0.95	$1.82 \cdot 10^{-11}$	1.0	$8.71 \cdot 10^{-8}$	1.0	$3.58 \cdot 10^{-6}$	1.0
1.00	0.0	1.0	0.0	1.0	0.0	1.0

Table 4.3: Expected fraction of failed, malicious, or blocked nodes (left); the durability of a given erasure-coded block in the network (D); and the network robustness in terms of censorship resistance (ρ).

Zettabyte of storage						
	25-of-100 erasure coding		30-of-200 erasure coding		50-of-500 erasure coding	
$E[f]$	D	ρ	D	ρ	D	ρ
0.00	1.0	0.0	1.0	0.0	1.0	0.0
0.05	1.0	0.0	1.0	0.0	1.0	0.0
0.10	1.0	0.0	1.0	0.0	1.0	0.0
0.15	1.0	$5.08 \cdot 10^{-21}$	1.0	0.0	1.0	0.0
0.20	1.0	$3.64 \cdot 10^{-12}$	1.0	0.0	1.0	0.0
0.25	1.0	0.000019	1.0	0.0	1.0	0.0
0.30	1.0	0.98	1.0	0.0	1.0	0.0
0.35	0.999	1.0	1.0	0.0	1.0	0.0
0.40	0.999	1.0	1.0	$3.15 \cdot 10^{-19}$	1.0	0.0
0.45	0.999	1.0	1.0	$1.45 \cdot 10^{-11}$	1.0	0.0
0.50	0.999	1.0	1.0	0.000063	1.0	0.0
0.55	0.999	1.0	0.999	0.999	1.0	0.0
0.60	0.9994	1.0	0.999	1.0	1.0	0.0
0.65	0.988	1.0	0.999	1.0	1.0	$7.27 \cdot 10^{-18}$
0.70	0.89	1.0	0.999	1.0	1.0	$1.33 \cdot 10^{-6}$
0.75	0.54	1.0	0.9998	1.0	0.999	1.0
0.80	0.13	1.0	0.97	1.0	0.999	1.0
0.85	0.0061	1.0	0.53	1.0	0.9996	1.0
0.90	0.000013	1.0	0.016	1.0	0.52	1.0
0.95	$1.82 \cdot 10^{-11}$	1.0	$8.71 \cdot 10^{-8}$	1.0	$3.58 \cdot 10^{-6}$	1.0
1.00	0.0	1.0	0.0	1.0	0.0	1.0

Table 4.4: Expected fraction of failed, malicious, or blocked nodes (left); the durability of a given erasure-coded block in the network (D); and the network robustness in terms of censorship resistance (ρ).

Chapter 5

Future work

While we have been able to show that extremely robust censorship resistant systems are possible even when taking very powerful adversaries into account, many interesting areas remain under-explored. This chapter highlights some notable examples, and hints at potential approaches.

5.1 Efficient MCON Formation and Routing

Our bootstrapping design for MCONs need to be streamlined both in terms of efficiency and usability. Our current "infection" approach to constructing MCONs, adopted to mitigate bootstrap attacks, is fairly cumbersome. Indeed, an ideal design would not make use of a social network at all, either as a connectivity graph of for Sybil attack resistance. A better approach would be one that allows individuals to ask for membership in the MCON, without the use of a social network, while preserving security. It is currently unclear how to achieve this goal.

Although our Membership and Invitation Authority can remain offline and hidden, it still represents a central point of failure[1] — if it were compromised then the complete membership of the network would be discovered. In principle, all functionality of the MIA can be carried out using secure multi-party computation, but the relevant *efficient* algorithms remain to be developed.

5.2 Implementing a Usable CROPS

We are working on improving the editing mechanisms of CROPS. While we will always need human editors, we can take steps to make their jobs as easy and efficient as possible. First, we require a mechanism for interested publishers to notify editors that a document has been submitted for their approval. Since the volume of content is likely to overwhelm a small community of dedicated volunteers, we need a secure way to extend the editor pool as well as revoke editing credentials of individuals who are no longer active. This needs to be done in a way that does not fall victim to the tyranny of the majority, so that even unpopular editors may have their say. A likely next step in the direction of lightening editor load is to have editors vet the relevance of documents to a particular

[1] with the exception of availability attacks

keyword, as opposed to evaluating the entire document. Thus editor signatures may bind documents to keywords, but would not vet documents entirely. Content dropping policies for these manifests are an additional unsolved problem.

We are continuing work on this project by developing a CROPS implementation that can use an overlay other than an MCON for communication. This implementation should be as user-friendly as possible, making censorship-resistant communication available to those that wish to use it, regardless of their technical abilities. Unfortunately, our current implementations, where available, are lacking in even basic usability features. For instance, our implementation of SilentKnock must be manually configured, and requires knowledge of the other party's operating system. We do not expect non-experts to easily use such a system. In order to see wide deployment, CROPS will need to be easily configurable and present a friendly and intuitive user interface. It is unclear what method of user interactions CROPS should employ. The "browser" paradigm is clearly a bad fit due to high latencies in the system, but an interface that mimics file sharing applications may be easy for non-technical users to understand. Latencies still pose a significant challenge, as users may be required to wait for long periods of time for search and/or download results. While current file-sharing networks do not provide instantaneous results, users nonetheless expect faster response that CROPS will be able to provide. We are looking into potential user interface designs, and at developing content-oblivious caching (at intermediate nodes) as a potential mitigation mechanism.

Bibliography

[1] Philip Elmer-Dewitt. First nation in cyberspace. *TIME magazine*, 49, 1993. Archived at `http://web.archive.org/web/20080526061939/http://www.chemie.fu-berlin.de/outerspace/Internet-article.html`.

[2] Ronald J. Deibert, John Palfrey, Rafal Rohozinski, and Jonathan Zittrain. ONI Home Page — OpenNet Initiative. `http://opennet.net/`, 2009.

[3] Ronald J. Deibert, John G. Palfrey, Rafal Rohozinski, and Jonathan Zittrain. *Access Denied: The Practice and Policy of Global Internet Filtering (Information Revolution and Global Politics)*. MIT Press, 2006.

[4] Ronald J. Deibert and Nart Villeneuve. *Firewalls and Power: An Overview of Global State Censorship of the Internet*. Cavendish Publishing London, 2004.

[5] Cinnamon Stillwell. Libel tourism: Where terrorism and censorship meet. *San Francisco Chronicle*. August 29, 2007.

[6] Doreen Carvajal. Britain, a destination for "libel tourism". *The New York Times*. January 20, 2008.

[7] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system. `https://www.torproject.org/svn/trunk/doc/design-paper/blocking.html`, 2007.

[8] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[9] Anonymous web surfing by Anonymizer. `http://anonymizer.com/`, 2010.

[10] WikiLeaks. `http://wikileaks.org/`, 2008.

[11] Marc Waldman and David Mazières. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2001.

[12] China 'blocks' iTunes music store. *BBC News*. August 22, 2008.

[13] Pakistan blocks YouTube website. *BBC News*. February 24, 2008.

[14] phobos. Tor partially blocked in China. `https://blog.torproject.org/blog/tor-partially-blocked-china`, 2009.

[15] John Douceur. The Sybil Attack. In *Proceedings of the International Peer To Peer Systems Workshop (IPTPS)*, 2002.

[16] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. SybilLimit: A near-optimal social network defense against Sybil attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.

[17] George Danezis and Prateek Mittal. SybilInfer: Detecting Sybil nodes using social networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2009.

[18] Martin Krzywinski. Port knocking: Network authentication across closed ports. *SysAdmin Magazine*, 12(6), 2003.

[19] P. Barham, S. Hand, R. Isaacs, P. Jardetzky, R. Mortier, and T. Roscoe. Techniques for lightweight concealment and authentication in IP networks. Technical Report IRB-TR-02-009, Intel Research Berkeley, 2002.

[20] D. Worth. CÖK: Cryptographic one-time knocking. In *Black Hat USA*, 2004.

[21] Rennie deGraaf, John Aycock, and Michael Jr. Jacobson. Improved port knocking with strong authentication. In *Proceedings of the Computer Security Applications Conference (ACSAC)*. IEEE Computer Society, 2005.

[22] Elli Androulaki, Mariana Raykova, Shreyas Srivatsan, Angelos Stavrou, and Steven M. Bellovin. PAR: Payment for anonymous routing. In Nikita Borisov and Ian Goldberg, editors, *Proceedings of the International Symposium on Privacy Enhancing Technologies (PETS)*. Springer, 2008.

[23] Tsuen-Wan "Johnny" Ngan, Roger Dingledine, and Dan S. Wallach. Building incentives into Tor. In Radu Sion, editor, *Proceedings of Financial Cryptography (FC)*, 2010.

[24] Mark Achbar and Peter Wintonick. Manufacturing consent: Noam Chomsky and the media, 1992.

[25] Constitutional Convention. United States Constitution, 1787. Available at `http://memory.loc.gov/cgi-bin/query/r?ammem/bdsbib:@field(NUMBER+@od1(bdsdcc+c0801))`.

[26] First Federal Congress of the United States. Amendments to the Constitution, 1798. Available at `http://www.loc.gov/rr/program/bib/ourdocs/billofrights.html`.

[27] United Nations General Assembly. The Universal Declaration of Human Rights. December 10, 1948.

[28] The whistleblower protection program. `http://www.osha.gov/dep/oia/whistleblower/index.html`, 2010.

[29] Whistleblower disclosures. `http://www.osc.gov/wbdisc.htm`, 2010.

[30] Society of professional journalists: SPJ code of ethics. Society of Professional Journalists, 2010.

[31] P. Biddle, P. England, M. Peinado, and B. Willman. The darknet and the future of content distribution. Technical report, Microsoft Corporation, 2002.

[32] Andreas Pfitzmann and Marit Hansen. Anonymity, unobservability, and pseudonymity: A consolidated proposal for terminology. Draft, 2000.

[33] Thomas S. Heydt-Benjamin, Andrei Serjantov, and Benessa Defend. Nonesuch: a mix network with sender unobservability. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES)*, 2006.

[34] Nick Feamster, Magdalena Balazinska, Winston Wang, Hari Balakrishnan, and David Karger. Thwarting web censorship with untrusted messenger delivery. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET)*. Springer-Verlag, LNCS 2760, 2003.

[35] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[36] David Mazières and M. Frans Kaashoek. The Design, Implementation and Operation of an Email Pseudonym Server. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. ACM Press, 1998.

[37] Josyula R. Rao and Pankaj Rohatgi. Can pseudonymity really guarantee privacy? In *Proceedings of the USENIX Security Symposium*. USENIX, 2000.

[38] Matthias Bauer. New covert channels in HTTP: Adding unwitting web browsers to anonymity sets. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES)*, 2003.

[39] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the USENIX Security Symposium*, 2004.

[40] Michael Reiter and Aviel Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), 1998.

[41] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2002.

[42] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. An analysis of the degradation of anonymous protocols. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*. IEEE, 2002.

[43] Arjun Nambiar and Matthew Wright. Salsa: A structured approach to large-scale anonymity. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2006.

[44] Jonathan Zittrain and Benjamin Edelman. Internet filtering in China. *IEEE Internet Computing*, 7(2), 2003.

[45] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: a public DHT service and its uses. *SIGCOMM Computer Communication Review*, 35(4), 2005.

[46] Justin Frankel. waste, 2003. Archived at: `http://slackerbitch.free.fr/waste/`.

[47] Eliot Van Buskirk. LimeWire adds private file sharing. `http://blog.wired.com/business/2008/12/lime-wire-adds.html`, 2008.

[48] Oskar Sandberg. Distributed routing in small-world networks. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2006.

[49] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Friend-to-friend data sharing with OneSwarm. Technical report, University of Washington, 2009. `http://oneswarm.cs.washington.edu/f2f_tr.pdf`.

[50] George Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and Ross Anderson. Sybil-resistant DHT routing. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2005.

[51] Ian Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*. ACM Press, 2001.

[52] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of*

IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). Springer, 2001.

[53] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 258. Springer, 2002.

[54] Bogdan C. Popescu. Safe and private data sharing with Turtle: Friends team-up and beat the system. In *Proceedings of the Cambridge International Workshop on Security Protocols*, 2004.

[55] Yair Sovran, Alana Libonati, and Jinyang Li. Pass it on: Social networks stymie censors. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[56] A. Mislove, M. Marcon, K.P. Gummadi, Peter Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the ACM SIGCOMM conference on Internet Measurement (IMC)*. ACM Press, 2007.

[57] George Danezis and Bettina Wittneben. The economics of mass surveillance and the questionable value of anonymous communications. In *Proceedings of the Workshop on The Economics of Information Security (WEIS)*, 2006.

[58] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2009.

[59] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[60] Roger Dingledine and Nick Mathewson. Tor directory server specification. `http://www.torproject.org/svn/trunk/doc/spec/dir-spec.txt`, 2009.

[61] Roger Dingledine and Nick Mathewson. Tor protocol specification. `http://www.torproject.org/svn/trunk/doc/spec/tor-spec.txt`, 2008.

[62] Roger Dingledine and Nick Mathewson. Tor bridges specification. `http://www.torproject.org/svn/trunk/doc/spec/bridges-spec.txt`, 2008.

[63] M. Caesar, M. Castro, E.B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by DHTs. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*. ACM Press, 2006.

[64] Prateek Mittal and Nikita Borisov. Information leaks in structured peer-to-peer anonymous communication systems. In Paul Syverson, Somesh Jha, and Xiaolan Zhang, editors, *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. ACM Press, 2008.

[65] Brent ByungHoon Kang, Eric Chan-Tin, Christopher P. Lee, James Tyra, Hun Jeong Kang, Chris Nunnery, Zachariah Wadler, Greg Sinclair, Nicholas Hopper, David Dagon, and Yongdae Kim. Towards complete node enumeration in a peer-to-peer botnet. In *Proceedings of the International Symposium on Information, Computer, and Communications Security (ASIACCS)*. ACM Press, 2009.

[66] Anant Sahai and Brent Waters. Fuzzy identity-based encryption. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*. Springer, 2005.

[67] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3), 2003.

[68] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In Ross Anderson, editor, *Proceedings of the International Workshop on Information Hiding*. Springer-Verlag, LNCS 1174, 1996.

[69] Yin-Chun Hu, Adrian Perrig, and David B. Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, 2003.

[70] Y. Kim, Adrian Perrig, and Gene Tsudik. Communication-efficient group key agreement. In *Proceedings of the IFIP TC11 Working Conference on Information Security: Trusted Information: The New Decade Challenge*. Kluwer Academic Pub, 2001.

[71] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2), 2002.

[72] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2003.

[73] Jared Saia and Moti Young. Reducing communication costs in robust peer-to-peer networks. *Information Processing Letters*, 106(4), 2008.

[74] X. Fu, B. Graham, R. Bettati, and W. Zhao. On countermeasures to traffic analysis attacks. In *Proceedings of the Information Assurance Workshop*, 2003.

[75] Thomer M. Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. The "King" data set. `http://pdos.csail.mit.edu/p2psim/kingdata/`, 2005.

[76] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the ACM Symposium on Theory of Computing*, 2000.

[77] Eugene Vasserman, Nicholas Hopper, John Laxton, and James Tyra. SILENT-KNOCK: Practical, provably undetectable authentication. *International Journal of Information Security*, 8(2), 2009.

[78] Luis von Ahn Nicholas J. Hopper, John Langford. Provably secure steganography. In *Proceedings of the International Cryptology Conference on Advances in Cryptology (CRYPTO)*, 2002.

[79] Steven J. Murdoch and Stephen Lewis. Embedding covert channels into TCP/IP. In Mauro Barni, Jordi Herrera-Joancomartí, Stefan Katzenbeisser, and Fernando Pérez-González, editors, *Information Hiding*. Springer, LNCS 3727, 2005.

[80] D. J. Bernstein. The Poly1305-AES message authentication code. In *Fast Software Encryption*, 2005.

[81] Matt Mackall and Theodore Ts'o. random.c – A strong random number generator. Linux 2.6.17.13 kernel source, drivers/char/random.c.

[82] Eugene Vasserman, Nicholas Hopper, John Laxson, and James Tyra. Silentknock, 2007. http://www.cs.umn.edu/~eyv/knock/.

[83] Jake Kouns, Chris Sullo, Brian Martin, David Shettler, Steve Tornio, and Kelly Todd. OSVDB: The Open Source Vulnerability Database. http://osvdb.org/, 2008.

[84] Christian Borss. DROP/DENY vs. REJECT. http://web.archive.org/web/20060901114422/http://www.lk.etc.tu-bs.de/lists/archiv/lug-bs/2001/msg05734.html, 2001.

[85] Martin Krzywinski. Port knocking. http://www.portknocking.org/, 2008.

[86] John Graham-Cumming. Practical secure port knocking. *Dr. Dobb's Journal*, 2004.

[87] Antonio Izquierdo Manzanares, Joaquin Torres Marquez, Juan M. Estevez-Tapiador, and Julio Cesar Hernandez Castro. Attacks on port knocking authentication mechanism. In *International Conference on Computational Science and Its Applications (ICCSA)*, volume 3483. Springer, LNCS 3483, 2005.

[88] D. Kundu PK. Ahsan. Practical data hiding in TCP/IP. In *Proc. Workshop on Multimedia Security at ACM Multimedia*, 2002.

[89] Craig H. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday*, 2(5), 1997.

[90] Conehead. Stego hasho. *Phrack*, 9(55), 1999.

[91] Todd MacDermid. Stegtunnel. http://www.synacklabs.net/OOB/stegtunnel.html, 2008.

[92] Luis von Ahn, Nicholas Hopper, and John Langford. Covert two-party computation. In *Proceedings of the ACM symposium on Theory of computing (STOC)*. ACM Press, 2005.

[93] Mike Bond and George Danezis. The dining freemasons: Security protocols for secret societies. In *Thirteenth International Workshop on Security Protocols*, 2005.

[94] S. Kent and R. Atkinson. IP authentication header. *IETF Internet Draft*, 1998.

[95] A. Heffernan. Protection of BGP sessions via the TCP MD5 signature option. *IETF Internet Draft*, 1998.

[96] Greg Hoglund and Jamie Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005.

[97] Sandra Ring and Eric Cole. Taking a lesson from stealthy rootkits. *IEEE Security and Privacy*, 2(4), 2004.

[98] Victor Shoup. On fast and provably secure message authentication based on universal hashing. In *Proceedings of the International Cryptology Conference on Advances in Cryptology (CRYPTO)*. Springer-Verlag, 1996.

[99] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of the International Cryptology Conference on Advances in Cryptology (CRYPTO)*, 1996.

[100] Harald Welte, Jozsef Kadlecsik, Martin Josefsson, Patrick McHardy, Yasuyuki Kozakai, James Morris, Marc Boucher, and Rusty Russell. The netfilter.org project. `http://www.netfilter.org/`, 2008.

[101] Jon Postel, *ed.* Transmission control protocol. *IETF Internet Draft*, 1981.

[102] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the ACM symposium on Theory of computing (STOC)*. ACM Press, 1977.

[103] John D. Valois. Lock-free linked lists using compare-and-swap. In *Proceedings of the ACM symposium on Principles of Distributed Computing (PODC)*. ACM Press, 1995.

[104] Maurice P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 1988.

[105] Jay Aikat, Jasleen Kaur, F. Donelson Smith, and Kevin Jeffay. Variability in TCP round-trip times. In *Proceedings of the ACM SIGCOMM conference on Internet measurement (IMC)*. ACM Press, 2003.

[106] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *SIGCOMM Computer Communication Review*, 19(2), 1989.

[107] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. In *Proceedings of the ACM conference on Computer and communications security (CCS)*. ACM Press, 2002.

[108] Andrew Hintz. Fingerprinting websites using traffic analysis. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET)*. Springer-Verlag, LNCS 2482, 2002.

[109] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-Bayes classifier. In *Proceedings of the ACM workshop on Cloud computing security (CCSW)*. ACM Press, 2009.

[110] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A.N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised systems. *Foundations of Intrusion Tolerant Systems*, 0, 2003.

[111] Peter Druschel and Aantony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of Workshop on Hot Topics in Operating Systems*, volume 0. IEEE Computer Society, 2001.

[112] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 35(11), 2000.

[113] Andreas Haeberlen, Alan Mislove, and Peter Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the*

USENIX Symposium on Networked Systems Design and Implementation (NSDI), volume 5, 2005.

[114] L.P. Cox and B.D. Noble. Samsara: Honor among thieves in peer-to-peer storage. *ACM SIGOPS Operating Systems Review*, 37(5), 2003.

[115] Ivan Osipkov, Peng Wang, and Nicholas Hopper. Robust accounting in decentralized P2P storage systems. In Mustaque Ahamad and Luís Rodrigues, editors, *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 2006.

[116] Google censors itself for China. *BBC News*. January 25, 2006.

[117] S.T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[118] Scott Wolchok, Randy Yao, and J. Alex Halderman. Analysis of the Green Dam censorware system. `http://www.cse.umich.edu/~jhalderm/pub/gd/`, 2009.

[119] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[120] Ross Anderson. The Eternity service. In *Proceedings of Pragocrypt*, 1996.

[121] Roger Dingledine, Michael J. Freedman, and David Molnar. The Free Haven project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, 2000.

[122] Marc Waldman, Aviel Rubin, and Lorrie Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proceedings of the USENIX Security Symposium*, 2000.

[123] Andrei Serjantov. Anonymizing censorship resistant systems. In *Proceedings of the International Peer To Peer Systems Workshop (IPTPS)*, 2002.

[124] Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing web censorship and surveillance. In *Proceedings of the USENIX Security Symposium*, 2002.

[125] Krista Bennett, Christian Grothoff Tzvetan Horozov, and Ioana Patrascu. Efficient sharing of encrypted data. In *Proceedings of the Australian Conference on Information Security and Privacy (ACISP)*. Springer-Verlag, 2002.

[126] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the ACM-SIAM symposium on Discrete algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2002.

[127] George Danezis and Ross Anderson. The economics of censorship resistance. In *Proceedings of Workshop on Economics and Information Security (WEIS)*, 2004.

[128] Ginger Perng, Michael K. Reiter, and Chenxi Wang. Censorship resistance revisited. In *Proceedings of Information Hiding Workshop (IH)*, 2005.

[129] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the ACM-SIAM symposium on Discrete algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2002.

[130] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the great firewall of china. In George Danezis and Philippe Golle, editors, *Proceedings of the Workshop on Privacy Enhancing Technologies (PET)*. Springer, 2006.

[131] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.

[132] Tim Cranton. Cracking down on botnets. `http://blogs.technet.com/microsoft_blog/archive/2010/02/25/cracking-down-on-botnets.aspx`. February 25, 2010.

[133] Atif Mushtaq. Smashing the Mega-d/Ozdok botnet in 24 hours. `http://blog.fireeye.com/research/2009/11/smashing-the-ozdok.html`. November 11, 2009.

[134] eMule Team. eMule project. `http://www.emule-project.net/`, 2009.

[135] BitTorrent, Inc. BitTorrent. `http://www.bittorrent.com/`, 2009.

[136] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper, and Yongdae Kim. Attacking the Kad network — real-world evaluation and high-fidelity simulation using DVN. *Security and Communication Networks*, 2009.

[137] Ceki Gülcü and Gene Tsudik. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*. IEEE, 1996.

[138] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of Information Hiding Workshop (IH)*. Springer-Verlag, LNCS 1525, 1998.

[139] Anja Jerichow, Jan Müller, Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. Real-Time MIXes: A Bandwidth-Efficient Anonymity Protocol. *IEEE Journal on Selected Areas in Communications*, 16(4), 1998.

[140] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[141] Petros Maniatis, Mema Roussopoulos, TJ Giuli, David S. H. Rosenthal, and Mary Baker. The lockss peer-to-peer digital preservation system. *ACM Transactions on Computer Systems*, 23(1), 2005.

[142] Mark W. Storer, Kevin Greenan, and Ethan L. Miller. Long-term threats to secure archives. In *Proceedings of the ACM workshop on Storage security and survivability (StorageSS)*. ACM Press, 2006.

[143] Arun Subbiah and Douglas M. Blough. An approach for fault tolerant and secure data storage in collaborative work environments. In *Proceedings of the ACM workshop on Storage security and survivability (StorageSS)*. ACM Press, 2005.

[144] Jay J. Wylie, Michael W. Bigrigg, John D. Strunk, Gregory R. Ganger, Han Kiliçcöte, and Pradeep K. Khosla. Survivable information storage systems. *Computer*, 33, 2000.

[145] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. POTSHARDS: secure long-term storage without encryption. In *Proceedings of the USENIX Technical Conference (ATC)*. USENIX Association, 2007.

[146] Qin Xin, J. E. Thomas, S. J. Schwarz, and Ethan L. Miller. Disk infant mortality in large storage systems. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE Computer Society, 2005.

[147] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4), 2002.

[148] Dahlia Malkhi and Michael K. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2), 2000.

[149] D. Dolev, M.J. Fischer, R. Fowler, N.A. Lynch, and H. Raymond Strong. An efficient algorithm for Byzantine agreement without authentication. *Information and Control*, 52(3), 1982.

[150] Jean-Philippe Martin and Lorenzo Alvisi. Fast Byzantine consensus. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2005.

[151] Barbara Liskov and Rodrigo Rodrigues. Tolerating Byzantine faulty clients in a quorum system. In Mustaque Ahamad and Luís Rodrigues, editors, *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 2006.

[152] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. Low-overhead byzantine fault-tolerant storage. In *Proceedings of 21st ACM SIGOPS symposium on Operating systems principles (SIGOPTS)*. ACM Press, 2007.

[153] Jinyuan Li and David Mazières. Beyond one-third faulty replicas in Byzantine fault tolerant systems. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2007.

[154] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the symposium on Operating systems design and implementation (OSDI)*. ACM Press, 2002.

[155] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, volume 31. ACM Press, 2001.

[156] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the Conference on File and Storage Technologies (FAST)*, volume 4, 2002.

[157] Dennis Kügler. An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET)*. Springer-Verlag, LNCS 2760, 2003.