A Pseudonymous Communications Infrastructure for the Internet

by

Ian Avrum Goldberg

B.Math. (University of Waterloo) 1995
M.Sc. (University of California at Berkeley) 1998

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Eric Brewer, Chair
Professor Doug Tygar
Professor Hal Varian

Fall 2000

The dissertation of Ian Avrum Goldberg is approved:

Chair Date

Date

Date

University of California at Berkeley

Fall 2000

A Pseudonymous Communications Infrastructure for the Internet

by

Ian Avrum Goldberg

Abstract

A Pseudonymous Communications Infrastructure for the Internet

by

Ian Avrum Goldberg

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Eric Brewer, Chair

As more and more of people's everyday activities are being conducted online, there is an ever-increasing threat to personal privacy. Every communicative or commercial transaction you perform online reveals bits of information about you that can be compiled into large dossiers, often without your permission, or even your knowledge.

This work presents the design and analysis of a Pseudonymous Communications Infrastructure for the Internet, which we call a Pseudonymous IP Network, or PIP Network. This system allows parties to communicate in real time over the Internet without being forced to reveal their identities, thus forming the basis for communications and electronic commerce systems that respect the privacy of the individual.

This work also presents the Nymity Slider, an abstraction that can be useful when talking about how much personally identifying information a given transaction reveals,

and when designing privacy-friendly technologies. We discuss why pseudonymity, rather than anonymity, is the goal of this project.

Finally, we introduce the primitive of the rendezvous server, which allows a system such as the PIP Network, which protects the privacy of the users of Internet services, to be turned around to protect the privacy of the providers of those services as well.

Professor Eric Brewer
Dissertation Committee Chair

To my friends, for all the good times.

To the makers of broken security systems, for helping me get to where I am today.

To my parents, advisors, and colleagues, for pushing me to finish this thesis.

And to the US Government, for at least keeping things interesting.

Contents

List of Figures

List of Tables

Acknowledgements

\x06

4500 0034 4fc9 4000 4006 ecf8 7f00 0001
7f00 0001 09cf 0016 37ea 0b4d 3828 7572
8010 7960 b653 0000 0101 080a 007d 26ab
007d 26ab

This thesis is the conclusion of a journey that spanned more than two decades. For the first time in 23 years, I am no longer a student. This is not to say, of course, that I will stop studying; many people have instilled in me the love of learning (and of teaching) that will serve me well in the future. I would especially like to thank my parents, for starting me out right, and providing me with the opportunities of which I was able to take advantage.

I would like to thank the Canadian Mathematics Competition, for providing a forum in which I could develop mathematical maturity, and to those who helped me get there; in particular, I could not have had a better mentor than Jean Collins, whose support, encouragement, and especially enthusiasm, took me from high school to the International Mathematical Olympiad.

The University of Waterloo provided me with both an excellent undergraduate education in Mathematics and Computer Science, as well as an immensely enjoyable time; I would like to thank my friends from U(W), and especially the Computer Science Club, for making my four-year stay so much fun.

The CS Division at the University of California, Berkeley has been invaluable to my development as a researcher. I would primarily like to thank my advisor, Eric Brewer, who

provided the guidance I needed in order to keep on track, and who tolerated my continual distractions into side projects. The residents of 445 Soda Hall were also an amazing set of people to bounce ideas off, to work with, and to hang out with. Armando Fox, Steve Gribble, David Wagner, Yatin Chawathe, Nikita Borisov, and Mike Chen ensured never a dull moment around the office.

More recently, I have moved to Montreal to work with Zero-Knowledge Systems, Inc. They have provided me with an immense amount of support, and I would like to thank Austin Hill, Hamnett Hill, Alex Eberts, and the outstanding original coding team, for convincing me to join them. I would also like to thank my many colleagues there who read early drafts of this thesis, and made helpful suggestions.

Finally, I would like to thank my many friends, most notably the Bab5 and PAIP crowds. My appreciation to them cannot adequately be expressed in words, but rather will be suitably revealed at the next Party At Ian's Place.

"It's Always Party Night When Ian's In Town."

Part I

Privacy-Enhancing Technologies

"*Any technology distinguishable from magic is insufficiently advanced*."

Chapter 1

Background

Recently the Internet has seen tremendous growth, with the ranks of new users swelling at ever-increasing rates. This expansion has catapulted it from the realm of academic research towards new-found mainstream acceptance and increased social relevance for the everyday individual. Yet this suddenly increased reliance on the Internet has the potential to erode personal privacies we once took for granted.

New users of the Internet generally do not realize that every post they make to a newsgroup, every piece of email they send, every World Wide Web page they access, and every item they purchase online could be monitored or logged by some unseen third party. The impact on personal privacy is enormous; already we are seeing databases of many different kinds, selling or giving away collections of personal data, and this practice will only become

more common as the demand for this information grows.

All is not lost. While the Internet brings the danger of diminished privacy, it also ushers in the potential for expanding privacy protection to areas where privacy was previously unheard of. This is our vision: restoration and revitalization of personal privacy for on-line activities, and betterment of society via privacy protection for fields where that was previously impossible. We want to bring privacy to the Internet, and bring the Internet to everyday privacy practices.

1.1 Definitions

A few definitions are in order at this point.

Privacy refers to the ability of the individual to control the distribution of information about himself. Note that this does *not* necessarily mean that your personal information never gets revealed to anyone; rather, a system that respects your privacy will allow you to *select* what information about you is revealed, and to whom. This personal information may be any of a large number of things, including your reading habits, your shopping habits, your nationality, your email or IP address, your physical address, or of course, your identity.

Anonymity and **pseudonymity** are two forms of privacy of identity (though often, in

common usage, they are conflated and are both referred to as simply "anonymity"). A system that offers anonymity is one in which the user gets to control who learns his identity (or other **verinym** ("true name"); see Chapter 4). In particular, it is the case that his identity is not automatically inserted in headers (or is easily derived from same), and also that it is difficult, if not impossible, for an adversary to "break" the system, and discover the user's identity against his wishes.

The distinction between anonymity and pseudonymity is that in the latter, the user maintains one or more persistent personae (**pseudonyms**, or **nyms**) that are not connected to the user's physical identity. People with whom the user interacts using a given nym can be assured that, although they do not know the physical identity behind the nym, it is in fact the same person each time. With anonymity, on the other hand, there is no such persistent identifier, and systems that provide strong (or unlinkable) anonymity leave no inherent way to tell whether any given message or transaction was performed by the same person as any other.[1]

The topics of anonymity, pseudonymity, linkability, and verinyms will be expanded upon in Chapter 4.

Forward secrecy refers to the inability of an adversary to recover security-critical information (such as the true name of the sender of a controversial message) "after the fact" (e.g. after the message is sent); providers of anonymity services should take

[1] Users of anonymity services should keep in mind that messages written by the same person tend to share certain characteristics, and that this fact has been used to identify the authors of anonymous works in the past [63].

> care to provide forward secrecy, which entails (for instance) keeping no logs.
>
> We currently see fairly regular affirmations in the legal system, that if a provider of a service does keep a log of the identity of the user of the service, then he is compelled to turn it over to satisfy even the most trivial of legal requests, such as a civil subpoena. This compulsion is often used by companies to track down people who criticize them on Internet message boards: the company files suit against the unknown poster, claiming libel or defamation, and uses the existence of the suit to force the company hosting the message board to reveal the identity of the poster. The suit is then dropped, and the company pursues its own action against the now-identified speaker (for example, by firing him if he is an employee).
>
> Therefore, to protect the privacy of one's users, the operator of such a service must ensure that he *has* no logs to turn over, and has no way to go "back in time" and reveal information about a past transaction.

1.2 Motivation

The threats to one's privacy on the Internet are two-fold: your online actions could be (1) monitored by unauthorized parties and (2) logged and preserved for future access many years later. You might not realize that your personal information has been monitored, logged, and subsequently disclosed; those who would compromise your privacy have no

incentive to warn you.

The threat of long-term storage and eventual disclosure of personal information is especially acute on the Internet. It is technically quite easy to collect information (such as a compendium of all posts you have made to electronic newsgroups) and store it for years or decades, indexed by your name for easy retrieval. If you are looking for a job twenty years from now, do you want your employer to browse through every Usenet posting you've ever made? If you are like most people, you have probably said something (however minor) in your past you would prefer to forget—perhaps an incautious word from your indiscreet youth, for instance. Long-term databases threaten your ability to choose what you would like to disclose about your past.

Furthermore, in recent years great advances have been made in technology to mine the Internet for interesting information [34]. This makes it easy to find and extract personal information about you that you might not realize is available. For instance, one of your family members might have listed information about you on their web page without your knowledge; Internet search engine technology would find this easily. Did you know your phone number, email address, and street address are probably listed on the Web? Or that your social security number is available on any of several for-pay electronically searchable databases? Most people probably do not want to make it easy for salesmen, telemarketers, an abusive ex-spouse, or a potential stalker, to find them.

In these ways, the Internet contributes to the "dossier effect", whereby a single query can

compile a huge dossier containing extensive information about you from many diverse sources. This increasingly becomes a threat as databases containing personal information become electronically cross-linked more widely. A recent trend is to make more databases accessible from the Internet; with today's powerful search engine and information-mining technology, this is one of the ultimate forms of cross-linking. For instance, phone directories, address information, credit reports, newspaper articles, and public-access government archives are all becoming available on the Internet. The "dossier effect" is dangerous: when it is so easy to build a comprehensive profile of individuals, many will be tempted to take advantage of it, whether for financial gain, vicarious entertainment, illegitimate purposes, or other unauthorized use.

Government is one of the biggest consumers and producers of dossiers of personal information, and as such should be viewed as a potential threat to privacy. The problem is that today's governments have many laws, surveillance agencies, and other tools for extracting private information from the populace [10]. Furthermore, a great many government employees have access to this valuable information, so there are bound to be some workers who will abuse it. There are many examples of small-scale abuses by officials: a 1992 investigation revealed that IRS employees at just one regional office made hundreds of unauthorized queries into taxpayer databases [3]; employees of the Social Security Administration have been known to sell confidential government records for bribes as small as $10 [58]; highly confidential state records of AIDS patients have leaked [4]. Finally, there is very little control or oversight, so a corrupt leader could easily misuse this in-

formation to seize and maintain power. A number of cautionary examples are available: FBI Director J. Edgar Hoover had his agency spy on political dissidents, activists, and opponents; the NSA, a secret military surveillance agency, has a long history of spying on domestic targets [9]; President Clinton's Democratic administration found themselves with unauthorized secret dossiers on hundreds of Republican opponents in the "Filegate" scandal.

Anonymity is one important form of privacy protection that is often useful.

We observe that anonymity is often used not for its own sake, but primarily as a means to an end, or as a tool to achieve personal privacy goals. For example, if your unlisted telephone number is available on the web, but can't be linked to your identity because you have used anonymity tools, then this might be enough to fulfill your need for privacy just as effectively as if you had kept the phone number completely secret. Many applications of online anonymity follow the common theme of "physical security through anonymity". For instance, political dissidents living in totalitarian regimes might publish an exposé anonymously on the Internet to avoid harassment (or worse!) by the secret police.

In contexts other than the Internet, anonymous social interaction is both commonplace and culturally accepted. For example, the Federalist papers were penned under the pseudonym Publius; many other well-known literary works, such as *Tom Sawyer*, *Primary Colors*, etc. were also written anonymously or under a pseudonym. Today, home HIV tests rely on anonymous lab testing; police tip lines provide anonymity to attract informants; journalists

take great care to protect the anonymity of their confidential sources; and there is special legal protection and recognition for lawyers to represent anonymous clients. The US Postal Service accepts anonymous mail without prejudice; it is well-known that anonymous voice calls can be easily made by stepping into a payphone; and ordinary cash allows everyday people to purchase merchandise (say, a copy of Playboy) anonymously. In short, most non-Internet technology today grants the ordinary person access to anonymity. Outside of the Internet, anonymity is widely accepted and recognized as valuable in today's society. Long ago we as a society reached a policy decision, which we have continually reaffirmed, that there are good reasons to protect and value anonymity off the Internet; that same reasoning applies to the Internet, and therefore we should endeavor to protect online anonymity as well.

There are many legitimate uses for anonymity on the Internet. In the long term, as people take activities they'd normally do offline to the Internet, they will expect a similar level of anonymity. In fact, in many cases, they won't even be able to imagine the extensive use this data could be put to by those with the resources and incentive to mine the information in a less-than-casual way. We should protect the ordinary user rather than requiring them to anticipate the various ways their privacy could be compromised. Moreover, the nature of the Internet may even make it possible to exceed those expectations and bring anonymity to practices where it was previously nonexistent. In the short term, there are a number of situations where we can already see (or confidently predict) legitimate use of Internet anonymity: support groups (e.g. for rape survivors or recovering alcoholics), online tip

lines, whistleblowing, political dissent, refereeing for academic conferences, and merely the pursuit of everyday privacy of a less noble and grand nature. As the New Yorker magazine explained in a famous cartoon, “On the Internet, nobody knows you’re a dog” [60]—and this is perhaps one of the greatest strengths of the Internet.

On the other hand, illicit use of anonymity is all too common on the Internet. Like most technologies, Internet anonymity techniques can be used for better or worse, so it should not be surprising to find some unfavorable uses of anonymity. For instance, sometimes anonymity tools are used to distribute copyrighted software without permission (“warez”). Email and Usenet spammers are learning to take advantage of anonymity techniques to distribute their marketing ploys widely without retribution. Denial of service and other malicious attacks are likely to become a greater problem when the Internet infrastructure allows wider support for anonymity. The threat of being tracked down and dealt with by social techniques currently acts as a partial deterrent to would-be intruders, but this would be eroded if they could use Internet tools to hide their identity. In many major denial of service attacks, the attacker obscures his IP source address to prevent tracing [23]. Widespread availability of anonymity will mean that site administrators will have to rely more on first-line defenses and direct security measures rather than on the deterrent of tracing.

1.3 Abuse

Another great challenge that faces future researchers in Internet privacy technology is the problem of abuse. As tools and infrastructure for anonymity become available, some will abuse these resources for illicit purposes.

We have some experience with handling abuse from the deployed remailers. Abuse only accounts for a small minority of remailer usage, but it is typically the most visible. One of the most common abuses of remailers is junk email, where senders hide behind anonymity to send vast quantities of unsolicited email (usually advertising) to a large number of recipients who find it unwelcome. Remailers today include simplistic alarms when they encounter a large volume of mail in a short time; then remailer operators can delete the spammed messages and source block the spammer (i.e. blacklist the sender). Harassment of a targeted individual is another common abuse of anonymous remailers. One countermeasure is to have targeted individuals install mail filtering software, or provide some other means (such as a challenge-response email system) for them to not receive unwanted email.

Remailers could also provide destination blocking services, but this raises many thorny issues: Should block lists be maintained centrally for coherency and coordinated fast response or separately to stop attacks based on falsified block lists? What is the policy for placing addresses on block lists? What parties are authorized to request that an email

address be destination blocked—the individual? his system administrator? his ISP?

The effect of this abuse is to place tremendous political and legal pressure on the remailer operator [44]. Of course, remailer operators receive no benefit themselves from providing anonymity services to the world, which makes it all the harder to justify spending much time, money, or effort to defend one's remailer. Each incident of abuse generates a number of complaints to the remailer operator, his ISP, and others who might be in a position to pressure them. This situation has become so acute that one of the greatest difficulties in setting up a new remailer is finding a host who will not give in to the political pressure.

Undoubtedly the magnitude and severity of abuse will increase when more infrastructure becomes available, and we will need to know how to deal with this problem. For instance, an uncontrolled completely anonymous communications infrastructure, be it an anonymizing Internet service, or an anonymous telephone call, potentially allows malicious hackers to break into a remote site untraceably. We can borrow some techniques from today's remailers. For instance, intrusion detection software at the point where the anonymous communication enters the "normal" network may detect some attacks, but it also has some serious limitations; we may also be able to use source blocking to shut out known trouble-makers. New techniques will probably be needed too. For example, some have suggested that requiring a small payment for the anonymity services would reduce spam, harassment, and denial of service attacks by making it too expensive to send large volumes of data; also, the resulting revenue might make it easier and more economical for

providers of anonymity services to handle abuse and stand up to political pressure. In any case, abuse management and prevention is likely to remain a central challenge for future anonymity technology.

1.4 Deployment

Perhaps the most important challenge facing Internet privacy advocates is to ensure that it sees widespread deployment. The issues include educating users about the need for special privacy protection to restore the privacy lost in an online world, building privacy software that is integrated with popular applications, winning over those who fear anonymity [10], and building systems that meet the needs of real users. It is important that this technology reaches the users who most need it. But more than that, the technology needs to be so pervasive that it reaches users that don't even need it.

Why this curious state of affairs? As the slogan for the Crowds project [64] goes, "Anonymity loves company". Even the best privacy-enhancing technologies, while hiding your identity, often do not hide the fact that you are *using* that technology, and in trying to track down a poster of an anonymous political rant, the adversary knows the culprit is one of the handful of people who use the particular privacy-enhancing technology. If only those who need it most, because of serious threats to their person, use these technologies, they do not gain as much protection from it as if the technology is just naturally used by ordinary

people in their daily activities. These people are said to provide **cover traffic** for the ones who need serious protection.

In order to protect the privacy, and often personal safety, of all manner of people, from the human rights worker in Asia, to the child left home alone, to the CEO preparing a merger, to the consumer browsing DVDs online, it is important that we see the creation of privacy-enhancing technologies that are widely deployed, are easy to use, and offer strong protections for users' personal information.

Chapter 2

Classification of Technologies

Alice wishes to send a message to Bob. Eve is an eavesdropper who wishes to use that fact for her own nefarious purposes. How can Alice make this hard for Eve to do?

The traditional answer is for Alice to protect the contents of her message; for example, by using **encryption**. If Alice uses a good encryption scheme, and good cryptographic protocols, she should be able to assure herself that only Bob can read the message destined for him; Eve will be unable to distinguish the contents of the message from random noise.

If this is Alice's goal, she should be successful. Encryption algorithms and cryptographic protocols are well-understood today.

However, what if it is not sufficient for Alice to simply hide the *content* of the message

from Eve? Suppose Alice and Bob are CEOs of big companies, and all of a sudden, they start exchanging encrypted email. Then, even though Eve cannot read the message, she can still gain useful information (for example, that Alice and Bob's companies may be involved in a future business deal or merger, or may be illegally participating in anticompetitive collusion) from simply knowing the *metadata*, such as the sender, the recipient, the time the message was sent, or the length of the message.

If Alice does not want Eve to find out who sent the message (or to whom the message is addressed), Alice needs to use techniques known as **privacy-enhancing technologies** [36]. These techniques allow Alice to not only hide the contents of the message, but also some amount of the metadata.

Suppose now that instead, Alice is a political dissident, and Eve is the government censor. Now, the mere *fact* that an email message was sent, may be cause for problems for Alice (especially if the message is encrypted). Alice needs to hide not only the contents and the metadata of the message from Eve, but in fact she needs to hide the entire *existence* of the message.

In order to solve this problem, Alice uses techniques known as **steganography**. These techniques allow her to hide the message inside of some other, innocuous, channel. Alice may send Bob an ordinary email that passes by Eve the censor without a problem, but, for example, the number of spaces after each period in the message could form the bits of another (usually encrypted) message to Bob, which Eve will not notice.

Level	What to protect	Method
3	Existence of message	Steganography
2	Metadata of message	Privacy-enhancing technologies
1	Content of message	Encryption
0	Nothing	None

Table 2.1: Levels of protection for a message

In summary, there are three levels of protection Alice can use on her message (excluding the trivial "none at all" protection): she can protect the data of her message using encryption; she can protect the metadata of her message using privacy-enhancing technologies; she can protect the existence of her message using steganography (see Table 2.1).

This work focuses on the second level of protection: securing Alice's privacy by protecting the metadata of her messages.

Systems that prevent an eavesdropper from learning the identities of the sender and recipient of the message can be divided into two broad categories:

Mutually revealing: Systems in which the communicating parties know who each other are, but the identities of one or both of the parties are hidden from the eavesdropper;

Not mutually revealing: Systems in which at least one of the communicating parties, in addition to the eavesdropper, does not know the identity of the other.

The examples given above were of the first category: Alice and Bob knew each other's

identities. However, in other situations, such as whistleblowing, pseudonymous posts to mailing lists or newsgroups (or replying to same), or using anonymous electronic cash, it is important that one of the communicating parties not learn the identity of the other.

Another direction in which privacy technologies can be divided is whether it is the identity of the *sender* or the *recipient* (or both) of the message that is hidden, certainly from the eavesdropper, and also possibly from the other party.

We note that a system that is not mutually revealing, and that is designed for ongoing communication, must provide at least somewhat for protection of *either* the sender or the recipient of a message. Suppose, for example, only the identity of the sender can be hidden, so Alice can send a message to Bob without Bob learning Alice's identity. But now in order for Bob to *reply* to Alice, he needs some way to deliver a message to someone whose identity he does not know; i.e. he needs a system that provides protection for the identity of the recipient (protection for the identity of the sender is not necessarily important, since Alice already knows Bob's identity).

The two main situations in which we can provide protection for the identity of one party (from the other) in an ongoing communication are:

Protection of the client: The identity of the party initiating the communication is protected; some (perhaps short-term) mechanism is provided for the recipient to reply to the initiator without knowing his identity. Note that the reply mechanism only has

to work for the one recipient (of the original message); it is not required that anyone else be able to use this mechanism.

Protection of the server: Clients send messages to a server using some long-term pseudonymous address, which hides the true identity of the server. In contrast to the above, the mechanism that protects the identity of the recipient of the message now usually needs to be able to accept messages from arbitrary, and even previously unknown, clients. Of course, a mechanism that protects the identity of the sender of a message is also required, in order for the server to reply to the client.

Finally, systems that provide for ongoing communications can be divided another way into two classes:

Store-and-forward: In this class, the sender transmits his message, and, perhaps after some time, it arrives at the recipient. Communications in this class of system include email and newsgroup postings.

Interactive: In this class, the sender and recipient are communicating in *real time*; large delays in transmitting the message are not acceptable. Communications in this class of system include the World Wide Web, online chat rooms, telephones, videoconferences, and most instances of electronic commerce.

Providing privacy protection for interactive systems turns out to be an extra challenge: the

low-latency requirement can often introduce timing correlations that an eavesdropper can use to defeat the privacy of the system. For example, if Bob always receives a packet within a fraction of a second of some other particular person transmitting one, and vice versa, Eve can be pretty sure that that person is the one communicating with Bob. (Attacks such as these will be discussed in more detail in Chapter 8.)

But solving this challenge is necessary if we are to achieve a system that robustly provides privacy for users of today's Internet, whether it be for web, chat, voice-over-IP, or electronic commerce.

In summary, then, privacy-enhancing technologies which provide for ongoing bidirectional communication can be divided along three independent axes:

Store-and-forward vs. Interactive: Is it acceptable for messages to be queued and shuffled in order to achieve the privacy protection, or do we need to communicate in real time?

Mutually revealing vs. Not mutually revealing: Are the identities of the participants in the communication hidden only from potential eavesdroppers, or from one or more of the participants themselves?

Protection of the client vs. Protection of the server: Is it the identity of the party initiating, or receiving, the communication that is hidden?

As we will see in the next chapter, there are many existing store-and-forward privacy-enhancing technologies, while the much of the communications over today's Internet are interactive. Therefore, the goal of this work is to design a set of interactive privacy-enhancing technologies.

In Chapter 4, we will see how a not mutually revealing technology can be converted easily into a mutually revealing one, but not vice versa. Therefore, our design will be not mutually revealing.

Finally, our primary motivation will be the protection of individual's privacy, and so our design will focus on the protection of the client; however, we will show how to modify the design so as to achieve protection of the server instead.

Chapter 3

Related Work

In this chapter, we will outline the history, current state of the art, and current trends, of privacy-enhancing technologies.

3.1 Past

In past years email was the most important distributed application, so it should not be surprising that early efforts at bringing privacy to the Internet primarily concentrated on email protection. Today the lessons learned from email privacy provide a foundation of practical experience that is critically relevant to the design of new privacy-enhancing technologies.

The most primitive way to send email anonymously involves sending the message to a

trusted friend, who deletes the identifying headers and resends the message body under his identity. Another old technique for anonymous email takes advantage of the lack of authentication for email headers: one connects to a mail server and forges fake headers (with falsified identity information) attached to the message body. (Both approaches could also be used for anonymous posting to newsgroups.) Of course, these techniques don't scale well, and they offer only very minimal assurance of protection.

The technology for email anonymity took a step forward with the introduction of anonymous remailers. An anonymous remailer can be thought of as a mail server that combines the previous two techniques, but using a computer to automate the header-stripping and resending process [5, 33, 40, 62]. There are basically three styles of remailers; we classify remailer technology into "types" that indicate the level of sophistication and security.

The `anon.penet.fi` ("type 0") remailer was perhaps the most famous. It supported anonymous email senders by stripping identifying headers from outbound remailed messages. It also supported recipient anonymity: the user was assigned a random pseudonym at `anon.penet.fi`, the remailer maintained a secret identity table matching up the user's real email address with his `anon.penet.fi` nym, and incoming email to the nym at `anon.penet.fi` was retransmitted to the user's real email address. Due to its simplicity and relatively simple user interface, the `anon.penet.fi` remailer was the most widely used remailer; sadly, it was shut down after being harassed by legal pressure [44].

The disadvantage of a `anon.penet.fi` style (type 0) remailer is that it provides rather weak security. Users must trust it not to reveal their identity when they send email through it. Worse still, pseudonymous users must rely on the confidentiality of the secret identity table—their anonymity would be compromised if it were disclosed, subpoenaed, or bought—and they must rely on the security of the `anon.penet.fi` site to resist intruders who would steal the identity table. Furthermore, more powerful attackers who could eavesdrop on Internet traffic traversing the `anon.penet.fi` site could match up incoming and outgoing messages to learn the identity of the nyms.

Cypherpunk-style (type I) remailers were designed to address these types of threats. First of all, there is no longer support for pseudonyms; there is no secret identity table, and remailer operators take great care to avoid keeping mail logs that might identify their users. This diminishes the risk of "after-the-fact" tracing. Second, type I remailers will accept encrypted email, decrypt it, and remail the resulting message. (This prevents the simple eavesdropping attack where the adversary matches up incoming and outgoing messages.) Third, they take advantage of *chaining* to achieve more robust security. Chaining is simply the technique of sending a message through several anonymous remailers, so that the second remailer sees only the address of the first remailer and not the address of the originator, etc. Typically one combines chaining with encryption: the sender prepends the ultimate destination address to the email, and encrypts the result with the public key of the *last* remailer in the chain. It then prepends the address of that remailer, and encrypts the entire resulting block with the public key of the next-to-last remailer in the chain. It

prepends the address of that remailer, and so on (see Figure 1).

After that has been done, the message will consist of an encrypted block that only the first remailer in the chain can read. The sender then emails the block to the first remailer, which opens the encrypted section, to discover the address of the second remailer, and an encrypted block (which the second remailer can read, but the first can't). It then forwards the encrypted block to the second remailer, and so on. The last remailer finds the address of the intended recipient of the message, as well as the (cleartext) message to send, and delivers the mail.

The advantage here is that every remailer in a chain must be compromised before a chained message can be traced back to its sender, as only the first remailer ever interacted with the sender, only the second one ever interacted with the first one, etc. This allows us to take advantage of a distributed collection of remailers; diversity gives one a better assurance that at least some of the remailers are trustworthy, and chaining ensures that one honest remailer (even if we don't know which it is) is all we need. Type I remailers can also randomly reorder outgoing messages to prevent correlations of ciphertexts by an eaves-dropper. In short, type I remailers offer greatly improved security over type 0, though they forfeit support for pseudonyms, and have some other limitations, which we discuss next.

The remailer message initially sent to **A**:

E_A [B, E_B [C, E_C [address, message]]]

A decrypts and sends to **B**:

E_B [C, E_C [address, message]]

B decrypts and sends to **C**:

E_C [address, message]

C decrypts and sends to **address**:

message

Figure 1: Providing sender anonymity: the structure of a chained remailer message

A, B, C are the remailers in the chain. E_x is public-key encryption with the public key of x.

3.2 Present

The currently most sophisticated remailer technology is the Mixmaster, or type II, remailer, based on the idea of a "mix network" by Chaum [19]. This technology extends the techniques used in a type I remailer to provide enhanced protection against eavesdropping attacks in a number of ways.

1. One always uses chaining and encryption at each link of the chain.

2. Type II remailers use constant-length messages, to prevent passive correlation attacks where the eavesdropper matches up incoming and outgoing messages by size.

3. Type II remailers include defenses against replay attacks. In these attacks, an adversary wishing to know where a given incoming message was headed would intercept the message, and send many copies of it to the remailer, which, if it were stateless, would end up sending many identical messages to the intended recipient. This is easily detectable. A type II remailer acts in a stateful manner, remembering which messages it has seen before, and not sending out the same message twice.

4. These remailers offer improved message reordering code to stop passive correlation attacks based on timing coincidences. [24] Because their security against eavesdropping relies on "safety in numbers" (where the target message cannot be distinguished from any of the other messages in the remailer net), the architecture also calls for

continuously generated random cover traffic to hide the real messages among the random noise.

Many of these protection methods will be seen again in Chapter 6.

Complementary to the sender anonymity of type I and type II remailers is the technology of the "newnym"-style nymservers. These nymservers are essentially a melding of the recipient anonymity features of a `anon.penet.fi` style remailer with the chaining, encryption, and other security features of a cypherpunk-style remailer: a user obtains a pseudonym (e.g. `joeblow@nym.alias.net`) from a nymserver; mail to that pseudonym will be delivered to him. However, unlike `anon.penet.fi`, where the nymserver operator maintained a list matching pseudonyms to real email addresses, newnym-style nymservers only match pseudonyms to "reply blocks": the nymserver operator does not have the real email address of the user, but rather the address of some remailer, and an encrypted block of data which it sends to that remailer. When decrypted, that block contains a symmetric key, the address of a second remailer, and a nested encrypted block. The remailer will *encrypt the message* with the given symmetric key, and pass the nested encrypted block and the newly encrypted message to the second remailer. That remailer in turn decrypts its block to find a symmetric key, the address of a third remailer, and a nested encryption block. It re-encrypts the message with the symmetric key, and passes the nested encrypted block and the (doubly) encrypted message to the third remailer, etc. Eventually, when some remailer decrypts the block it receives, it gets a symmetric key and

the real email address of the user. It will then encrypt the (by now many times) encrypted message with the symmetric key and send the result to the user (see Figure 2).

When the user receives the message, he decrypts it with each of the symmetric keys in turn (he kept a copy of them when he created the reply block), and discovers the message. The effect of all this is that *all* of the remailers mentioned in the reply block would have to collude or be compromised in order to determine the email address associated with a newnym-style pseudonym.

Another simple technique for recipient anonymity uses message pools. Senders encrypt their message with the recipient's public key and send the encrypted message to a mailing list or newsgroup (such as `alt.anonymous.messages`, set up specifically for this purpose) that receives a great deal of other traffic. The recipient is identified only as someone who reads the mailing list or newsgroup, but onlookers cannot narrow down the identity of the recipient any further. A "low-tech" variant might use classified advertisements in a widely read newspaper such as The New York Times. Message pools provide strong recipient anonymity, but of course the huge disadvantage is that they waste large amounts of bandwidth and pollute mailing lists with bothersome noise.

One could reasonably argue that the problem of anonymous email is nearly solved, in the sense that we largely understand most of the principles of building systems to provide email anonymity. However, email is not the only important application on the Internet. More recently, we have seen the beginnings of privacy support for other services as well.

The nym server receives an email message for `johndoe@nymserver.net` and finds the following reply block in its database:

`johndoe`**, A, E_A [K_A, B, E_B [K_B, C, E_C [K_C, user]]]**

The nym server sends the reply block and message to **A**:

E_A [K_A, B, E_B [K_B, C, E_C [K_C, user]]], message

A decrypts, recovers K_A, encrypts the message and sends to **B**:

E_B [K_B, C, E_C [K_C, user]], S_A [message]

B decrypts, recovers K_B, encrypts the message and sends to **C**:

E_C [K_C, user], S_B [S_A [message]]

C decrypts, recovers K_C, encrypts the message and sends to **user**:

S_C [S_B [S_A [message]]]

The user receives the multiply encrypted message, and decrypts it using his copies of K_C, K_B, and K_A (in that order).

Figure 2: Providing recipient anonymity: the structure of a reply block

A, B, C are the remailers in the chain. E_x is public-key encryption with the public key of x. K_x is a symmetric key shared between the user and x. S_x is symmetric-key encryption using K_x.

3.3 Future

When attempting to design anonymity support for web traffic, interactive text/voice/video chatting, remote `telnet` connections, and other similar services, we quickly see that we can approach the problem in two ways: either by building application-specific solutions, or by creating an infrastructure to provide privacy protection for general-purpose bi-directional interactive Internet traffic.

3.3.1 Application-specific solutions

Others have proposed some special-purpose applications for Internet privacy, though their use is not yet widespread.

One of the first online applications that received attention was naturally web browsing.

The “strip identifying headers and resend” approach used by remailers has been applied to provide anonymity protection for Web browsing as well. The Anonymizer [2] from Anonymizer.com is simply a web proxy that filters out identifying headers and source addresses from the web browser. This allowing users to surf the web anonymously without revealing their identity to web servers. However, the Anonymizer offers rather weak security—no chaining, encryption, log safeguarding, or forward secrecy—so its security properties are roughly analogous to those of type 0 remailers: just like `anon.penet.fi`,

the operator of the Anonymizer could, possibly under legal pressure, reveal the identities of users accessing particular sites. There are, in addition, a number of other implementations of the same approach; for example see [25, 29].

The Crowds project [64] is to the Anonymizer what type I remailers are to type 0: there are a number of web proxies distributed around the Internet (in fact, on the computers of the clients *using* the service), and web requests are randomly chained through a number of them before being forwarded to the web server hosting the requested data. The idea is that the web server will see a connection coming from some user of Crowds, but it is impossible for the server to determine *which* Crowds user initiated the request. Clearly, the more people that use the system, the more "mixed-up" the data will be.

Anonymous digital cash, or "ecash", is another upcoming technology for Internet privacy. As many observers have stressed, electronic commerce will be a driving force for the future of the Internet. Therefore, the emergence of digital commerce solutions with privacy and anonymity protection is very valuable. Two different sets of protocols, one from David Chaum [20], and one from Stefan Brands [12], have the strongest privacy protection of any developed payment system—they use sophisticated cryptographic protocols to guarantee that the payer's privacy is not compromised by the payment protocol even against a colluding bank and payee. These forms of electronic cash have many of the privacy properties of real cash (or money orders); most other deployed payment systems have only about as much privacy as checks or credit cards.

In a series of papers, Camp, Tygar, Sirbu, Harkavy, and Yee also consider a variety of electronic commerce transaction types that incorporate pseudonymity and anonymity [13, 14, 15]. Of course, the anonymous ecash protocols only prevent your identity from being revealed by the protocols themselves: if you send the merchant a delivery address for physical merchandise, he will clearly be able to identify you. Similarly, if you use pay using ecash over a non-anonymized IP connection, the merchant will be able to deduce your IP address. This demonstrates the need for a general-purpose infrastructure for anonymous IP traffic, as discussed later. (The other option is to pay by email, with which you can use the existing remailer infrastructure, to preserve your privacy.) In any case, security is only as strong as the weakest link in the chain, and we need strong anonymity (such as provided by Chaum's and Brands' protocols) in our payment system as well as strong anonymity in our data transport infrastructure.

There are currently a number of different proposals for providing *anonymous publication*. Ross Anderson's Eternity Service [1] is designed to provide long-term distribution of controversial anonymous documents, even when the threat model includes governments and other powerful parties, and this has been somewhat implemented by Adam Back in the form of Usenet Eternity [6].

Publius [77] is another system (under active development) for providing long-term document distribution. In contrast to Usenet Eternity, in which the documents are simply stored as Usenet (newsgroup) articles on Usenet servers around the world, Publius uses dedicated

servers to store documents. It also, unlike Eternity, does allow for the modification and/or deletion of documents, but only by the original poster.[1]

Other examples of systems for anonymous publication focus more on short-term publication, more akin to file-sharing [59, 54]. The MIT Free Haven project [31] and FreeNet [22] are two such systems, still in their early stages of development.

Mojo Nation [57] is a commercial product attempting to provide a service to achieve the same effect; an interesting feature of this system is that users are paid in an online currency called "mojo" for donating their disk space to the global online storage.

Many cryptographers have studied the problem of *electronic voting*, and cryptographic protocols abound [68]—but more practical experience with building and deploying large voting systems is needed. The need for more application-specific privacy-respecting systems will no doubt arise as the Internet continues to grow.

3.3.2 General-purpose infrastructure

Based on Chaum's mix networks [19], Wei Dai has described a theoretical architecture that would provide privacy protection based on a distributed system of anonymizing packet forwarders, analogous to today's remailer network; he called it "Pipenet" [27].

[1] Note that it is useful to be able to turn this feature *off*; i.e. to be able to publish a document that cannot be modified or deleted, even by yourself (and provably so). That way, no amount of coercion or force can compel you to remove the document.

Like the remailer network, Pipenet consists of a "cloud" of packet forwarding nodes distributed around the Internet; packets from a client would be multiply encrypted and flow through a chain of these nodes, much like in Figure 1. Pipenet is an idealized architecture, and has never been built, or even fully specified, in practice. It suffers from a number of flaws that would make such a network impractical:

- There is no method for nodes to learn the topology of the network.
- You can only securely communicate with nodes that are part of Pipenet, not just any Internet server.
- Packet loss or delay is extremely bad; Pipenet treats any packet delay as a potential attack, and responds by propagating that delay to the whole network, so as to prevent the attacker from learning anything from the variation in inter-packet timings.

In addition, there is no support for pseudonymity, but in Chapter 4 we will see that this is easy to fix.

There are two independent projects that provide a more mature implementation of a Pipenet-like architecture: Onion Routing [37, 38], and Freedom [11, 8], the latter of which is based on the protocols described in this work.

With Onion Routing, a user directs his applications to contact application proxies that form the entrance to the cloud of nodes (called Onion Routers). The application proxy will then

send an “onion packet” (so named because of the nested layers of encryption resembling an onion) through a string of Onion Routers in order to create a route through the cloud. The application proxy will then forward the application data along this route through the cloud, to exit on the other side, and be delivered to the server to which the user wishes to connect.

The Onion Routing design has some serious flaws, however; for example,

- There is no protection between the client and the application proxy; an attacker watching the network at that point would destroy the privacy of the client.
- Packet loss in the network causes a problem of network backlog and cascading re-transmits, since the nodes talk to each other via TCP.
- The client must completely trust the application proxy to choose a suitable route through the cloud; in contrast to the remailer network, where the user chooses the chain, and where choosing any one trustworthy node suffices to maintain the chain’s security, here, a single compromised node at the entrance to the cloud can remove all privacy protection.

It should be noted that these flaws have been addressed in the design of the second version of their system. In addition, Onion Routing also has no support for pseudonymity, but again, that is easy to fix. Also, and more seriously, it does not provide a way to manage, or in some cases, even prevent or detect, abuse of the system.

In brief — the details are the core of this thesis — this work avoids the problems of trust between the client and the rest of the cloud by having the client itself become *part of* the cloud. On the upside, this puts much more control over what parts of the network to trust into the hands of the individual users. On the downside, however, this requires the client to run special software, unlike Onion Routing, where the client need at worst reconfigure his application proxy settings.

We mitigate the TCP-in-TCP problems by use of IP-in-IP tunneling or UDP instead; in this way, the loss of a packet does not prevent subsequent packets from being transmitted.

We can detect and prevent abuse by the use of exit node application-layer proxies; we can also manage abuse through the use of pseudonymity, and reputation capital.

All of these pieces will be described in much more detail in the following chapters, which will discuss the theory behind anonymous and pseudonymous systems, give the design goals, and detail the design methodology of our system to provide a pseudonymous communications infrastructure for the Internet.

Part II

A Pseudonymous Communications Infrastructure for the Internet

a complex system
for achieving the goal of
pseudonymity

Chapter 4

The Nymity Slider

People engage in numerous forms of transactions every day. Some of these transactions are *communicative*; for example, sending a letter (or email) to a friend, reading a newspaper, posting to a newsgroup, or using an online chat room. Some are *commercial*; for example, buying a newspaper, selling stock, or trading baseball cards.

In each case, the participants in the transaction exchange some content: information in the case of communicative transactions, or value in the case of commercial transactions. But the transactions also involve the exchange (among the participants) or revelation (to outsiders) of meta-content: information *about* the participants, or about the transaction itself.

Some examples of meta-content may include the date and time of the transaction, the val-

ues of the items exchanged, the physical location at which the transaction was conducted, or information about the identities of the participants.

This chapter will be particularly concerned with this last piece of meta-content. We will define the **nymity** of a transaction to be the amount of information about the identity of the participants that is revealed. Note that transactions often have different nymity levels for different participants, and it may be that the nymity level with respect to a participant differs from that with respect to an outside observer; for example, the person with whom you are corresponding may know your identity, but that information is hidden from third-party eavesdroppers. The goal of this chapter is to catalogue various useful nymity levels that occur in common types of transactions, and to note certain properties of these values.

4.1 The levels of nymity

The amount of identity one chooses to, or is required to, reveal in a transaction (be it a commercial transaction or a communication) is variable, and depends on the particular situation. These different amounts of revealed identity can be placed on a continuum, which we call the “Nymity Slider”.

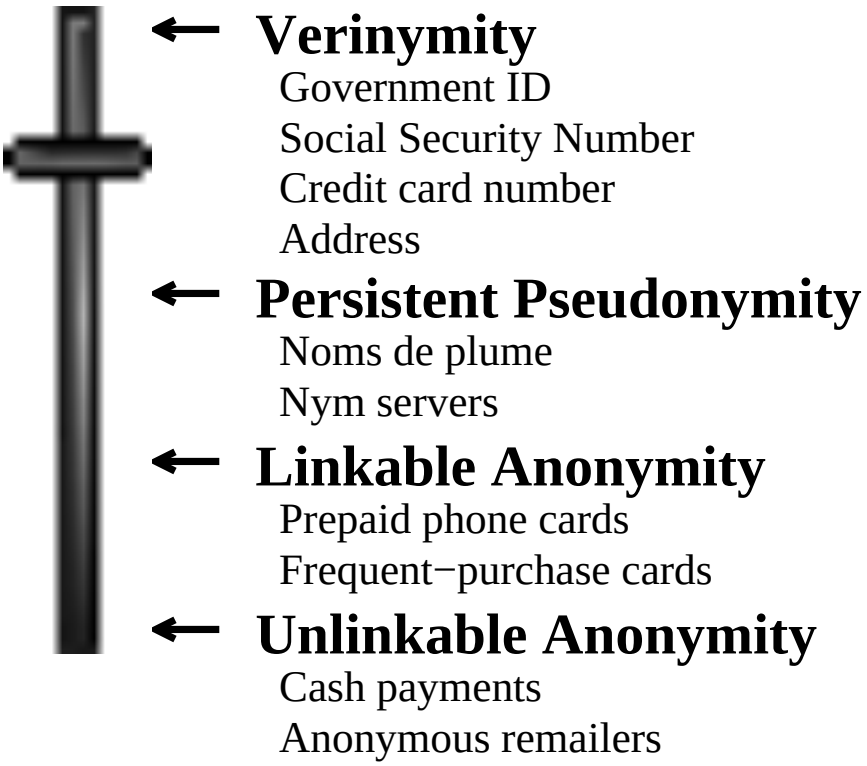

Figure 3: The Nymity Slider

4.1.1 The high end: verinymity

A *verinym* is a True Name [75]. But what do we mean by that? We *could* mean the name printed on your government-issue birth certificate, driver's license, or passport, but not necessarily.

By "verinym" or "True Name" we can also refer to any piece of identifying information that can single you out of a crowd of potential candidates. For example, a credit card number is a verinym. So can be a telephone number, or a street address. In the online

world, an email address or an IP address can also be considered a verinym.

The idea is that, if I know you are one of the people in a potential set of candidates, then if I get a verinym for you, I can figure out which one you are. Clearly, some attributes may or may not be verinyms, depending on the particular set of candidates I have in mind. For example, if the candidate set is rather small, then simply knowing that you work in Washington, DC may be sufficient to single you out; but that same piece of information is not sufficient if the candidate set is, say, the set of US Federal Government employees. For this reason, some forms of verinymous information are listed slightly lower on the Nymity Slider than other forms.

Transactions in which a verinym is revealed are said to provide **verinymity**. This forms the high end of the Nymity Slider.

Verinyms have two important properties:

Linkability: Any verinymous transaction you perform can be linked back to you, and thus, to any other verinymous transaction you perform. Verinymous transactions therefore inherently contribute to the dossier effect; they make it possible, if not easy, to construct a large dossier on you by cross-referencing large numbers of databases which are each indexed by one of your verinyms.

Permanence: Verinyms are, for the most part, hard to change, and, generally, even if you do change them, there is often a record of the change (thus linking your old name to

your new one).

These two properties are what makes *identity theft* so problematic: if an imposter uses one of your verinyms and sullies it (say, by giving it a bad name or a bad credit record), it's quite difficult to get the situation cleaned up, since you can't change the verinym you use (permanence), and you can't separate the transactions you made from the ones the imposter made (linkability).

Companies such as Verisign [74] want to bring verinymity to the Internet in a strong way by issuing Digital Passports that will provably tie your online activities to a real-life verinym, such as the name on your driver's license.

4.1.2 The low end: unlinkable anonymity

In contrast, at the extreme low end of the Nymity Slider are transactions that reveal no information at all about the identity of the participant. We say that transactions of this type provide **unlinkable anonymity**.

We use the term "unlinkable" to mean that not only is no information about your identity revealed, but also that there is no way to tell whether or not you are the same person that performed some given prior transaction.

The most common example in the physical world is paying for groceries with cash.[1] No information about your identity is revealed to the merchant (provided you do not willingly divulge extra information; more on that in Section 4.2), nor is the merchant able to determine which of the cash transactions over the course of a month are by the same person.

Technologies such as type I anonymous remailers (see Chapter 3) provide unlinkable anonymity for Internet email; there is no way to tell if two remailer messages are from the same person or not.

4.1.3 The middle: linkable anonymity, persistent pseudonymity

As usual, the most interesting parts of a slider are not at the extremes, but rather, in the middle.

Above unlinkable anonymity on the Nymity Slider is the situation where a transaction does not reveal information about the identity of the participant, yet different transactions made by the same participant can, at least in some cases, be linked together.

A simple example is when one purchases a prepaid phone card (using cash). Neither the

[1] Though note that even this leaks a tiny bit of information about your identity; for example, you very likely live in the same part of the country as the grocery store is located. Also, the store may have recorded an image of your face on a security camera. The Netherlands has even proposed having bank machines record the serial numbers of bills issued to you, and having merchants record the serial numbers of bills spent. Dutch money has barcodes printed on it ostensibly for this purpose.

merchant nor the phone company generally learns the identity of the purchaser; however, the phone company *can* link together all calls made with the same card. This is **linkable anonymity**. Similarly, many grocery stores offer frequent-purchase cards (the Safeway Club Card is a canonical example); you sign up for one of these, often using an obviously fake name (it turns out the stores don't care about your name), and then all purchases you make can be linked to the card, and thus to each other, but not to your identity.[2]

Merchants use the information gathered from these linkable anonymous transactions to determine, for example, that people who buy diapers also tend to buy beer and so may arrange to place them near each other in the store. (This curious situation apparently arises when Wife asks Husband to go to the store to pick up some diapers; Husband figures that while he's out, he may as well pick up some beer. [3])

When some authors or columnists publish their works, they do so under *noms de plume*, or *pen names*. The True Name of the author is not revealed, but it is assumed that when another work appears under the same pen name, it was written by the same author or group of authors. "Bourbaki" and "Dear Abby" are two well-known pen names that were used by a group of people, as opposed to a single person. In the digital world, we can in

[2]The Bay Area Cypherpunks attempt to foil the linkability measures of Safeway Club Cards by having invented the Safeway Club Card Exchange Protocol: once in a while, when a large number of them are in the same physical location, they will throw their cards into a big pile, and randomly extract a replacement [78].

[3]In fact, one time I related this example in a talk, a member of the audience came up to me afterwards and told me that just that day, his wife had asked him to go get diapers, and he had picked up diapers and beer. He was a little spooked. Notwithstanding the confirming example, though, the beer-and-diapers story is apparently an urban legend.

fact arrange something slightly stronger: unforgeability. That is, *only* the original person or cooperating group behind the pen name can use it; in that way, we assure that if some given name appears to have authored a number of works, we can be certain that the original author was responsible for them. This is called **persistent pseudonymity**, and sits between linkable anonymity and verinymity on the Nymity Slider.

In the online world, the nym servers (see Chapter 3) provide for persistent pseudonymity: two posts from `lrh@nym.alias.net` are assured to have come from the same person.

Because of this assurance of origin, persistent pseudonymity (sometimes simply referred to as "pseudonymity"; the persistence is implied) provides for what is known as *reputation capital*. If you perform transactions with a certain pseudonym (or "nym") repeatedly, be they commercial or communicative, that nym will gain a reputation with you, either positive or negative. For instance, you might come to believe that the nym pays promptly when purchasing things online; or that the goods he advertises in an online auction are generally of poor quality; or that he is very knowledgeable in the field of recreational sailing; or that he spouts off loudly about quantum mechanics, but he knows nothing at all about it — or even all of the above.

You form an idea of the kinds of commercial transactions you would be willing to perform with this nym, and of the kinds of communications you are willing to undertake with him, and of the kinds of things you will believe if he says them. This is that nym's *reputation* with you, and that reputation is useful, even though you may know nothing at all about the

person behind the nym.

This is in fact one of the most common levels of nymity on the Internet today. In news-groups or mailing lists, for example, you have no idea whether the person posting under the names “Tim May”, “Lucky Green”, or “Ian Goldberg” actually has a driver’s license or passport with that name written on it. Nor should you care. You merely decide for yourself what the person’s reputation with you should be; do you generally agree with the things he says, do you feel compelled to have political debates with him, or do you you simply ignore his mad rantings?

This level of nymity shares somewhat the property of linkability with that of verinymity: anything posted *under a given nym* can be linked to anything else so posted. However, a single person may have multiple pseudonyms, and this is where the usefulness of this level of nymity is most apparent. It is not generally possible to link transactions made under one pseudonym to those made under a different one. That is, given transactions from two different pseudonyms, it is usually very difficult, if not impossible, to tell whether the transactions were performed by (and therefore the pseudonyms represent) the same person or not.

This lack of permanence allows for a number of useful features; for example, you might use one pseudonym on a résumé web site while looking for a new job (so that your current employer can not tell you’re thinking of leaving), and a different pseudonym on a singles web site while looking for a date. There is no good reason these two pseudonyms should

be able to be tied together. You might use a pseudonym when you're young, so that when you go looking for a job twenty years later, your teenage posts to Usenet don't come back to haunt you. In this way, pseudonymity defeats the ability of others to compile dossiers on you in the manner described earlier.

The ability for nyms to acquire reputation, and to defeat the dossier effect, suggests that persistent pseudonymity is the desired nymity level at which to aim our system.

4.2 Properties of the Nymity Slider

One of the fundamental properties of the Nymity Slider is that, given any transaction that normally occupies a certain position on the slider, it is extremely easy to change the transaction to have a higher position on the slider (closer to verinymity): the participant merely has to agree to provide more information about himself. This is the situation, for example, where a consumer volunteers to use a frequent-purchase card at a grocery store to allow the merchant to link together all of the purchases he ever makes. Assuming the merchant does not require some proof of identity to obtain the card, this action moves the position of the Nymity Slider from "unlinkable anonymity" to "linkable anonymity". If the merchant *does* require a True Name in order to issue the card, the slider gets pushed all the way to "verinymity" — a pretty steep price for groceries.

Similarly, a poster using an anonymous remailer can sign his messages with the PGP key of a pseudonym, in order to turn unlinkable anonymity into pseudonymity. Or a user of a pseudonym can "unmask" himself by voluntarily revealing his own identity. In all cases, simply revealing more information at a higher layer of the protocol [46] is sufficient to increase the nymity of the transaction.

On the other hand, it is extremely difficult to move a transaction *down* the slider (towards unlinkable anonymity). If the only method of payment one has available is a credit card, for example (payments over the Internet currently fall into this category, to a first approximation), it is challenging to find a way to buy something *without* revealing that verinym. If any layer of your protocol stack has a high level of nymity associated with it, it is difficult to build a protocol on top of it that has a lower level of nymity.

For example, anonymous electronic cash protocols inherently have a very low level of nymity, but if you try to use them over ordinary TCP/IP, your IP address (a verinym, or close to it) is necessarily revealed, and the point of the anonymous application layer protocol is lost. In this sense, the Nymity Slider bears some resemblance to a ratchet: it is easy to move an existing protocol up, but it is hard to move down.

For this reason, when we design new protocols, at all layers of the stack, we should design them to have *as low a nymity level as possible*. If done in this way, protocols built on top of them will not be forced to have high nymity. Also, even if we *want* a protocol to have high nymity, it is best to design it with low nymity, and then simply provide the additional

identity information at a higher layer. The reason for this is that it enables us to easily change our minds later; although we may want some new system to provide verinymity or persistent pseudonymity, we may later decide that some (perhaps limited) uses of lower levels of nymity are desired. In order that we be able to avoid a complete redesign, we should design the system with a low nymity level from the start, add additional nymity now by simply displaying the additional information, and then just remove this added nymity if desired at a later time. We further note that this added nymity can be of whatever strength is required by the application; for example, a “screen name” could be merely sent as an additional field in a chat protocol, whereas an application requiring unforgeability would likely require some sort of digital signature, or other authenticity check.

Therefore, although the end goal of our Pseudonymous Communications Infrastructure for the Internet is to provide network communications which support persistent pseudonymity, we will design it from the start to provide unlinkably anonymous access. Then, to provide persistent pseudonymity, we will simply require that users of the system display their pseudonyms in order to gain access to it.

Chapter 5

Design Goals

The scope of this work is to design and analyze a system that will allow for individuals to communicate, and to utilize network services, pseudonymously. In this chapter, we will outline the various threats against which we may wish to defend, we will discuss some high-level design goals for the system, and finally, we will examine the tradeoffs we must be willing to make.

5.1 Overview of Threats

In making claims about the protection a privacy-enhancing technology offers, and in explaining the limitations of the technology, it is useful to examine some of the types of

people who may attempt to violate a user's privacy. Below, we briefly describe those attackers and our assumptions about their abilities. Later, we will chart the various attacks we believe these attackers can carry out, as well as some possible defenses. We note up front that we will not be successful in defending against all of these threats; rather, we will analyze which defenses are effective against which threats.

5.1.1 Web Site Operators

A web site operator can offer cookies to try to track a user. Many web sites will use various forms of encouragement to get personal information, such as asking for a ZIP code for weather reports, and then share that information with their advertising networks. An advertising network, such as DoubleClick [55], by placing ads on many sites, is able to gather a large profile of any given user. Internet sites using custom protocols, like RealNetworks [79], can also engage in tracking of users.

Web sites can also use active content, such as ActiveX, Javascript, and other languages, to cause a user's computer to send information to the site. This behaviour is less common than gathering profiles through cookies.

5.1.2 Systems Administrators and Internet Service Providers

Corporate systems and network administrators, and commercial ISPs, can variously read a user's mail, watch to where he makes network connections (such as web browsing), and generally monitor all his unencrypted online activities. A company sysadmin can read any files stored on network drives, and may also be able to access all the files on desktop or laptop computers. There may be local laws controlling this activity, though users may have signed away all of their rights under such laws as part of an employment contract. Also, in a corporate situation, other employees on the inside may have (legal or illegal) access to files and networks.

5.1.3 Search Engines

Search engines can discover a lot of information about people that they themselves, their friends and family, their employers, their school, and others in their lives may have placed online. A single slip up that links a pseudonym to a verinym, posted anywhere on the Internet, is easily discovered with a simple search.

5.1.4 Lawmakers and Law Enforcement

In democracies or other countries where the police are under the jurisdiction of civilian authorities, police threats are usually overt, in the form of attempts to obtain encryption keys to force data recovery, including identity information. This is usually involves warrants or court orders, but may also include psychological tactics or even physical intimidation.

In some countries, police may also operate covertly through actions such as emissions monitoring and “dumpster diving”. One cannot assume that all police actions are authorized or even legal, or that if authorized and legal, the regime that has authorized them is ethical and protective of human rights. Police in various countries have been known to use illegal means of gathering information, which they abandon when it leads them to a legal way of gathering information. [56]

Police departments often work as agents of the courts, who attack by way of warrants or subpoenas. The subject of a warrant or subpoena may be ordered to be silent about it.

Attacks by legislatures include declarations that keys must be escrowed, passing “Know Thy Customer” laws (which prevent certain kinds of transactions from occurring without the participants being strongly identified) and identity card laws, and other measures usually taken with the public’s interest ostensibly in mind, but from an authoritarian point of view.

5.1.5 System Crackers

System crackers will generally use search engines, trojan horse software, and network monitoring (much like a sysadmin) to gather information about someone. Depending on their level of interest, they have also broken into credit reporting agencies, police computers, and other systems with poor security in order to gather information.

5.1.6 National Intelligence

National Intelligence Agencies may operate wide net "vacuum cleaner" operations designed to gather huge amounts of electronic information based on keywords and header information. The Echelon system [16] is reputed to do this. They may also engage in more targeted methods where they gather information from colleagues and acquaintances of people, or in technical attacks, where they use techniques such as Van Eck monitoring [73] or hidden microphones to gather information.

5.1.7 Litigious Groups

There are a variety of organizations who, feeling their interests threatened, spend huge amounts of money threatening and filing lawsuits. This capability can allow them to force AIP operators, for example, to reveal any stored or logged information they may possess.

These lawsuits may need to be filed in a number of countries.

5.1.8 Organized Crime

Criminal organizations may attempt to either subvert the network, or the privacy of a nym. This type of attacker is more likely to use physical violence for employee subversion, theft, or breaking and entering. Also, in some cases, organized gangs can be better funded and equipped than police forces.

5.2 Goals

In this section, we list some design goals for our pseudonymous communications system. In Section 5.3, we will examine to what extent these goals are complementary, and to what extent they are conflicting.

Deployable over the existing Internet: This criterion allows us to "get there from here"; that is, it gives us the ability to leverage existing communications infrastructure in order to build a new system. This is certainly preferable to setting up our own network. However, it also requires us to inherit some of the less-desirable aspects of the existing Internet, whether the problems be unintentional (packet delays and losses are unpredictable), or malicious (there is near-zero security on the links between

nodes in the network; adversaries have the ability to read, write, modify, or delete traffic on those links).

Applicable and real-time: The system should be useful at the very least for the most common uses of the Internet today: web surfing, email, Usenet, and chat. This requires the ability to support a multitude of current Internet protocols, as well as easily adding new ones. In addition, in order to support many of these protocols, the system must have low latency; web-traffic packets must be able to be delivered pseudonymously without the hour-long delays typical of chaining remailers, for example. We would also like the user's "view of the net" to be affected as little as possible; that is, except for not revealing his identity, his use of Internet services with this system, should be as close as possible to his use of them without.

Resistant to attack: It should be difficult to trace the origins of a packet, or to find the user behind a pseudonym. Varying this degree of difficulty can lead to interesting tradeoffs, both design-time and run-time; see below. As well, it should be difficult to attack the infrastructure itself; there should be no single point of failure, where one particular node going down would cause the rest of the network to cease functioning, or worse, to forfeit privacy silently.

5.3 Tradeoffs

Some of our goals complement each other, whereas others conflict; for example, in order to achieve some of the functionality we may want, we may be forced to compromise security, or vice versa. In particular:

Using the Internet vs. Applicability: These two goals do not conflict; in fact, they complement each other. If our goal is to make existing Internet services available to users (in a pseudonymous fashion), it is no problem to run the traffic over the existing Internet; that is, after all, the way the users get access to those services today.

Resistance to attack vs. Using the Internet: Using the public Internet clearly opens the system to a number of attacks that may not exist on a private network. Not only do attackers not have to get access to a special network, but the properties of Internet traffic such as best-effort delivery, and unpredictable throughput, packet loss, and jitter, aid the attacker in more subtle ways; see Chapter 8 for much more detail. Against this, however, is the huge benefit obtained by being able to build on an existing network.

For now, we will resolve this tradeoff in favour of using the public Internet, possibly at the expense of security. A more mature system may wish to simply move the system hereunder described entirely to a private network, and instantly gain a number of security benefits.

Resistance to attack vs. Applicability: Today's Internet applications were, for the most part, not designed with privacy in mind. Some applications, in fact, actively violate the user's privacy by sending information about him or his computer to some other party over the Internet, without the user's consent, or even knowledge [42, 72].

It may be difficult, if not impossible, to make these applications continue to work, while still providing privacy to the user. We will resolve this tradeoff in favour of privacy; we will attempt to prevent attacks on the identity of the user, even if it comes at the expense of the user not being able to perform some operations or run some applications.

Resistance to attack vs. Resistance to attack: There are, in fact, more difficult tradeoffs to be made. Some design choices allow for attacks by certain classes of adversaries, and it so happens that sometimes two possible design choices merely let us select which class of adversary can attack the system, and not let us secure the system from attack entirely.

These tradeoffs we will make on a case-by-case basis, and we will go into more details about them as they arise in the description below.

Chapter 6

Design Methodology

What we wish to build is a **Pseudonymous IP (PIP)** network. However, if we recall the lesson of the Nymity Slider, it would behoove us to design a system for the *anonymous* transport of IP packets. This **Anonymous IP Infrastructure (AIPI)** can then have a pseudonymity layer added on top of it; the Nymity Slider says that this should be easy to do.

Therefore, we now go about designing an AIPI. The three components of an AIPI are as follows:

The IP Wormhole: The primary piece of the AIPI is what we term the “IP Wormhole”. This is an Internet service [35] with the following properties:

- A client can set up an IP Wormhole between himself and some other location on the Internet (the "exit node"). The choice of exit node is not arbitrary; typically, the client must select from one of a predefined list of endpoints, but this list could be large and geographically diverse.
- The client can "inject" IP packets into the IP Wormhole. After some delay (the *latency* of the IP Wormhole), the packet will (with some probability; this is a "best-effort" delivery mechanism, in the style of IP) be *transported* to the exit node. The exit node will insert the packet onto the Internet, which will proceed to route it to its final destination in the usual way.
- Packets sent by servers in reply to packets sent through the IP Wormhole will be routed by the Internet to the exit node. The exit node will then inject the reply back into the IP Wormhole, for transport back to the client.
- The IP Wormhole should, to the extent possible, hide the identity (in particular, the verinym with which we are concerned here is the IP address) of the client from the recipients of the client's packets, who may additionally be colluding with other adversaries.
- Clients using the IP Wormhole should be able to communicate with any Internet server; it must not be required that existing Internet servers run special software, or know about special protocols.

Note that the key here is this process of transporting IP packets from one end of the

IP Wormhole to the other. This needs to be done in such a way as to protect the client's identity from the adversaries. The design methodology for the IP Wormhole will be covered in Section 6.1.

The Network Information Database: The Network Information Database, known as the NIDB, maintains the list of the possible exit nodes (and intermediate nodes, which will be discussed below), along with their public keys and statistical information about them.

The NIDB can, in its simplest form, just be a central database that is queried for the pertinent information. However, there are dangers to doing this (both in terms of scalability and in terms of security), so a more distributed approach is warranted, as will be seen in Section 6.2.

Application-level proxies: The Nymity Slider says that if any layer of a protocol has inherently high nymity, then it is difficult to make the protocol as a whole have low nymity. The IP Wormhole will provide the anonymous transport at the low layers, but we still need mechanisms for anonymous or pseudonymous *applications*. To this end, the AIPI will provide for the ability to install application-level proxies.[1] These proxies will come in two types, with different functions:

Client-side proxies: The role of a client-side application-level proxy is to protect the identity of the client by removing identifying information from the high

[1] Note that, technically, the AIPI only provides the *hooks* for the application-layer proxies; the proxies themselves are not part of the AIPI, which is a transport-layer mechanism.

layers of application protocols.

Exit node proxies: The role of an exit node proxy is to protect the integrity and security of the PIP Network as a whole by preventing malicious (and possibly anonymous) clients from conducting undesirable behaviour on the network.

Further details on application-level proxies can be found in Section 6.3.

6.1 The IP Wormhole

This section will describe the methodology we will use to construct the IP Wormhole. We will begin by presenting a very simple design, and work up from there, each step defending against more powerful adversaries.

6.1.1 Protecting against IP address revelation to the server

In the normal course of operation of TCP/IP, a client's IP address is plainly present in the IP packets delivered to the server. We may design an extremely simple IP Wormhole to prevent the client's IP address from reaching the server, as follows.

Set up one or more dedicated hosts around the Internet; we call these hosts **Anonymous Internet Proxies** (AIPs). The basic idea will be that the client will select one of the

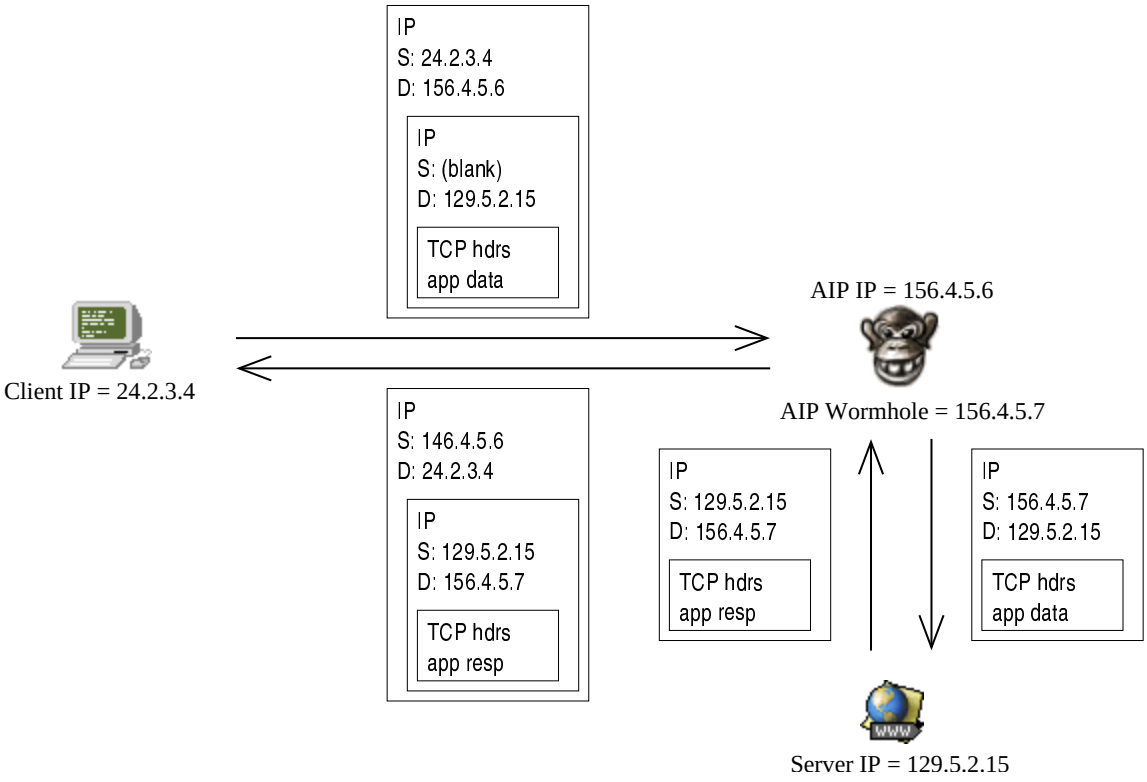

Figure 4: Protecting against IP address revelation to the server

AIPs (either at random, or perhaps a specific one close to himself, or close to the server with which he is communicating); the client will then send packets to the AIP, which will forward them on to the destination server.

In more detail (and see Figure 4):

- Each AIP has at least two IP addresses: its regular Internet address, and one or more Wormhole addresses, the purpose of which will be given below. Note that IP packets containing any of the AIP's addresses (Internet or Wormhole) must be routable to

the AIP.

- To send a packet, the client first removes the source address information from the packet; this packet will then have a blank source address, and the destination server's IP address as the destination address. It then encapsulates the (somewhat anonymized) packet as the payload of another IP packet; this outer packet will have the client's IP address as the source address and the AIP's IP address as the destination address. The client then sends the resulting packet, which will be delivered to the AIP by normal Internet routing methods. (Note that this is just IP-in-IP tunneling [69].)

- The AIP receives the packet from the client. If this is the first packet it has received from this client (or at least the first since some timeout interval), it needs to assign a Wormhole-IP address to the client. This can be done in one of two ways:

  – Assuming the packets are TCP or UDP packets, a unique (Wormhole-IP, port) pair can be assigned to each (Client-IP, port) pair seen on incoming packets. This is the method used by Network Address Translation (NAT) or IP Masquerading [32, 52]. This method works best when only one (or a very small number) of Wormhole-IP addresses are available to the AIP.

  – If many Wormhole addresses are available for the AIP to use (remembering that they all must be routed to the AIP, so private IP space is not usable here), the AIP can simply assign a unique Wormhole-IP address to every Client-IP

address seen on incoming packets. This method works best when the available IP space is large, such as with IPv6 [28].

Having selected a Wormhole-IP address (or (Wormhole-IP, port) pair) to assign to the packet, the AIP stores that choice in a table for use when subsequent packets from the same Client-IP (or (Client-IP, port) pair) arrive.

- The AIP extracts the encapsulated packet, and substitutes the blank source address with the Wormhole-IP address selected above. It then transmits the resulting packet, which will be delivered to the destination server.

- When the server replies, the destination address of the packet will be one of the AIP's Wormhole addresses. The AIP will accept this packet, and look up in its table the corresponding Client-IP address. It will then encapsulate the reply packet in another IP packet; this outer packet will have the AIP's Internet address as a source address and the Client's IP address as a destination address. The AIP will then transmit this packet, which will be delivered to the client.

- The client will receive the encapsulated reply packet (and note that in this way the client learns the Wormhole-IP address he was assigned by the AIP; this will become useful later on), and process it as if it were received directly from the server.

This simplest form of IP Wormhole certainly does what it claims, so long as the only adversary is the operator of the destination server, and that operator can only gain information

from his server's IP logs. More sophisticated attackers can attack this system very easily; simply monitor the network near any AIP, and just read the IP-in-IP packets to discover which client is communicating with which server. Of the threats outlined in Section 5.1, Internet Service Providers, System Crackers, and National Intelligence are the ones most likely to be of concern at this point.

Note that the above is really the figure of interest: the AIPs are handling traffic from many clients, and many servers; how hard is it for an attacker to determine which client is communicating with which server? In this case, the attacker simply needs to be able to monitor traffic passively anywhere between the client and the AIP it is using, in order to defeat the system completely.

6.1.2 Protecting against reading the IP-in-IP packets

In order to prevent the adversary from reading the IP-in-IP packets in order to discover the correlation between client address, Wormhole address, and server address, we use encryption.

Note that, in general, since the servers are not required to know about special protocols (see above), we cannot deliver encrypted data to them most of the time; the packets that arrive at the server must appear to be from an ordinary client, and not speak some other protocol, or be additionally encrypted.

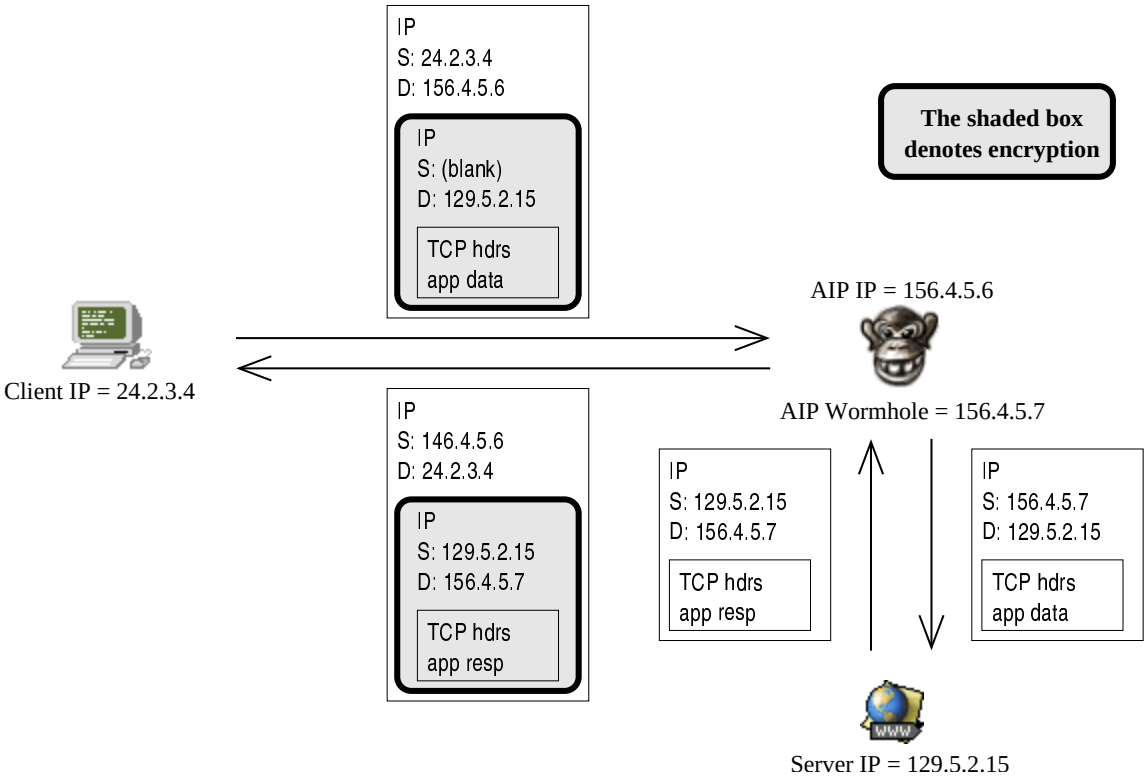

Figure 5: Protecting against reading the IP-in-IP packets

Because of this, we are limited to encrypting the data between the client and the AIP. Now, before the client encapsulates the original IP packet in a packet destined for the AIP, it first encrypts it in a way that only the AIP can decrypt (either with public-key or symmetric-key cryptography; more on this choice later). Similarly, when the AIP receives the reply packet, it encrypts it for the client before encapsulating it (see Figure 5).

Now the attacker can no longer read the contents of the packets flowing between the various clients and the AIP, although he can still read the headers of those packets, and he can also read the complete packets between the AIP and the servers. From the headers of

the packets between the clients and the AIP, he can determine which AIP a given client is using, and also the set of clients communicating via any given AIP. From the packets between the AIP and the servers, he can determine which servers are being accessed through the AIP, as well as the contents of the conversations.[2] This information is insufficient to determine which client is communicating with which server.

But it turns out that the adversary can still do some fairly trivial traffic analysis, if he can passively monitor both the traffic coming into an AIP and the traffic leaving it (for example, by snooping at the AIP operator's ISP): the adversary gets to see plaintext data between the AIP and the server, and also the corresponding ciphertext data between the client and the AIP. It turns out it is very easy to determine the *size* of the ciphertext, given the plaintext; if compression is not used, the ciphertext size is usually a constant overhead (possibly zero, depending on the algorithm) more than the plaintext size. Even if compression is used, the attacker usually has reasonable methods to approximate the size of the compressed plaintext, (for example, by compressing it, if the compression method involves no secret parameters) and therefore the size of the ciphertext. So now the attacker can easily pair up plaintext packets with their corresponding ciphertext packets, and then he knows which client is communicating with which server.

For example, if client A sends a pattern of small, medium, large, large, small packets to the AIP, and the AIP sends that same pattern of packets to some server S (and different

[2] This implies that the contents of the conversations must be stripped of identifying data, but that's already necessary, if the server is not to learn the identity of the client.

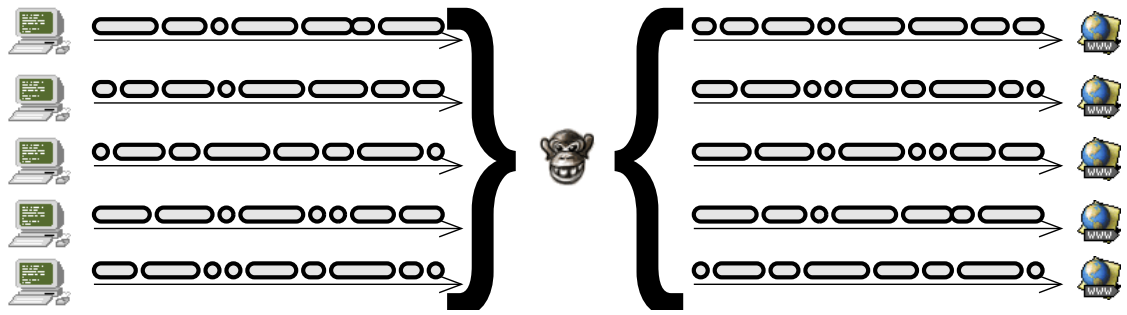

Figure 6: Which client is communicating with which server (correlations of size)?

patterns to different servers), it is fairly easy to pick out that client A is communicating with server S (see Figure 6).

6.1.3 Protecting against packet size correlations

In order to prevent the adversary from using the different patterns of packet sizes sent from different clients to distinguish one client from another, we can require that the patterns of packet sizes from each client be *the same*.

The simplest way to do this is to use *packet padding* to make each packet transmitted between the client and the AIP the same size. That is, the possibly compressed plaintext is padded out to some constant length before encryption, so that all encrypted packets are the same size, and there is no relationship between the sizes of the encrypted packets and the sizes of the plaintext packets. Note that the adversary still sees the true sizes of

the (unpadded) plaintext packets, since those packets are delivered to the server with no protection.

Padding all packets to the same size is somewhat inefficient, however; choosing constant sizes for packets is well-known to be a contentious issue [43]. We can somewhat alleviate this problem by having multiple packet sizes. Looking at HTTP traffic, for example, data packets from clients to servers mostly consist of relatively small HTTP requests, and data packets from servers to clients mostly consist of maximally sized bulk data transfer. In addition, dataless (and therefore very small) TCP ACK packets travel in both directions.

With multiple packet sizes available for the encrypted packets, outgoing packets can be padded into the smallest size in which they will fit; we can arrange, for example, to have very small packets that will accommodate mostly empty TCP ACKs, medium-sized packets for the HTTP requests, and large packets that will accommodate bulk data transfer.

We then arrange a pattern of transmitted packets in each direction consistent with the predicted usage patterns. We will arrange, for example, for the traffic from the AIP to the client to have a larger proportion of larger packets than the traffic in the other direction.

If we now arrange that all clients send the same patterns of sizes of packets to the AIP, the adversary has no way to distinguish the clients based on the sizes of the encrypted packets.

Unfortunately, this helps very little. Now, instead of correlating the *sizes* of the incoming and outgoing packets at the AIP, he correlates their *times*. The attacker will see an

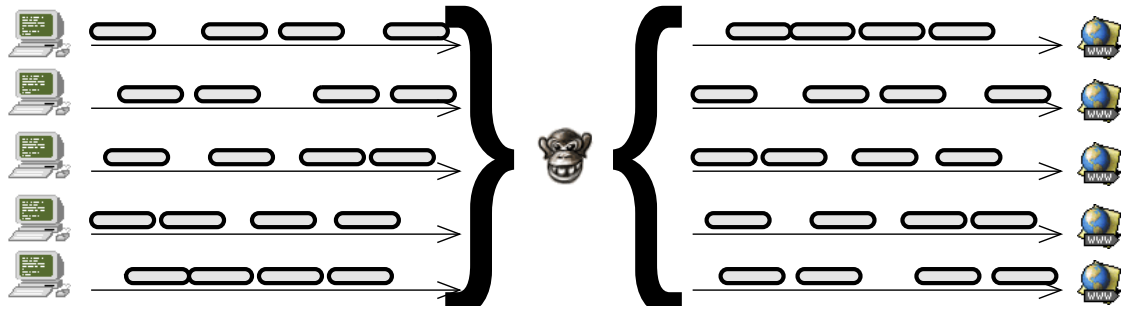

Figure 7: Which client is communicating with which server (correlations of time)?

encrypted packet arrive at the AIP from a particular client, and shortly thereafter, a decrypted packet will leave the AIP, headed for some server. After not too much data, the attacker should be able to correlate the clients with the servers (see Figure 7).

6.1.4 Protecting against packet timing correlations

We avoid this problem by adding *link padding*; that is, not only do we shape the sizes of the packets according to some fixed pattern, we also share their times.

Again, the simplest, but not the only, acceptable pattern is to send a constant number of each size of packet per unit time. So, for example, in each time slice, we might send 5 small packets, 4 medium packets, and 1 large packet from the client to the AIP. Note once again that the different directions of communication may suggest different distributions.

If not enough packets of a given size are available when it is required to send them, dummy packets consisting of random data will be sent instead of properly encrypted packets; at the other end of the connection, they will fail to decrypt to anything sensible, and will be discarded. Note that properly encrypted data is indistinguishable from random data by an eavesdropper who does not know the encryption key, so said eavesdropper cannot learn which packets are real and which are not.

If packets of a given size are ready to be transmitted at a higher rate than we want to transmit them, we can do one of three things:

- If the packets are small, and there are unused slots for larger packets, we can send one (or more) small packets in the frame of one large one.
- We could queue the packets for a short time, hoping the burst subsides.
- We could simply drop the excess packets, and let the higher-lever protocols, such as TCP, infer (correctly) that the link is congested.

As mentioned above, a constant rate is not the only acceptable link padding pattern to use. Any *data-independent* function is possible; i.e. any pattern of transmitting packets that does not depend on when we actually have data to send, and when we do not. For example, if your network connection uses a shared link, you may decide to use a link padding function that transmits more data at night (when the link is usually idle) than in

the middle of day. Traffic patterns follow fairly predictable cycles [39], so this is not too difficult to do.

The key is that each client using a given AIP should be using the same distribution of packet sizes and times. In this way, the adversary sees a number of *identical* packet distributions between numerous clients and the AIP. The packet patterns between the AIP and the servers are all different (as they are ordinary TCP/IP traffic), but now the adversary has no way to distinguish between the clients based on their traffic patterns, and so it is impossible for him to figure out which client is communicating with which server.

This methodology effectively solves the problem for the case of an adversary who is able to passively monitor the network (in particular, the parts of the network near any given AIP). In particular, referring to our taxonomy of threats in Section 5.1, Internet Service Providers, System Crackers, and National Intelligence are the most likely to be sniffing Internet traffic. But what if the adversary is somewhat more powerful, and in fact controls the AIP itself (or its operator)? Note that this can come about either because (unbeknownst to the clients) the adversary is behind the organization administering the AIP, or because a system cracker penetrated the security of the machine hosting the AIP.

Given omnipotent access to the innards of the AIP, the attacker now can decrypt the traffic between the clients and the AIP, and so can determine which packets are dummy packets, and can once again read the contents of the IP-in-IP packets. This will nullify the privacy of all clients using that AIP. This is a single point of failure for the system, and as such

must be addressed.

6.1.5 Protecting against an untrustworthy AIP

In order to remove the single point of failure, we utilize chaining: instead of hiding his identity behind a single AIP, a client can choose a chain of AIPs; his packets will be sent first to one AIP, and then to a second, and so on, until the last AIP sends the packet to the server. The server's replies will go to the last AIP, which will send them to the one before it, and so on, until the first AIP sends them to the client.

The most obvious way to do this is simply to concatenate multiple instances of the system described in section 6.1.4, so that each AIP in the chain just acts as the client for the next AIP in the chain. Unfortunately, that does not have the property we want. In a system like that, the client encrypts the packet to the first AIP, and delivers it there. The first AIP decrypts it, re-encrypts it for the second AIP, and delivers it there. The second AIP decrypts it, re-encrypts it for the third AIP, and so on.

In a system like this one, each AIP gets to see the contents of the packet, and that will reveal to it information about the destination server (whose IP address is contained in said packets). In particular, the first AIP knows who the client is, because they are in direct communication, and if it can also read the contents of the packets, it can determine who the server is, and so the client's privacy is broken, regardless of the number of AIPs in the

chain.

We make a slight modification to the encryption process to fix this problem. We can view the above system as having a number of AIPs in a chain, with a secure (strongly encrypted and authenticated and packet-padded and link-padded) link between each pair of adjacent AIPs in the chain (including one between the client and the first AIP). Note that we can't put a secure link between the last AIP and the server, because we are assuming we can't modify arbitrary servers on the Internet, and we want to be able to use the system with any server. IP packets are then sent along this chain; the problem is that they pop out of the encryption and padding protection at each hop in the chain.

Our slight modification is to add, in addition to the secure link between each adjacent pair of AIPs in the chain (which we will call the **link encryption**), a secure link between the client and each node in the chain (which we will call the **telescope encryption**); this secure link will be tunneled over the link-encrypted hops we had before.

Now the process of communicating anonymously is as follows:

- We assume there exist AIPs deployed across the Internet, and that there are secure links between them, forming some (not necessarily complete) graph, called the AIP graph.
- A client will be activated somewhere on the Internet and will start up a secure link to its nearest AIP.

- The client will pick an *exit AIP*; this will be the location at which he will appear to be when he communicates with servers. Sometimes the physical jurisdiction of the exit AIP is important to the client, and sometimes it can just be random. The client then picks a random path through the AIP graph from himself (or his nearest AIP) to the exit AIP.

- The client then (using public-key cryptography) establishes shared secrets between himself and each AIP in the chain. These secrets are used to build the telescope encryption.

At this point, the IP Wormhole (between the client and the exit AIP) has been created. Now, to send IP packets through the IP Wormhole:

- The client first prepares the IP packet by multiply encrypting it. The packet (after performing packet-padding by growing it to a constant size) is first encrypted with the secret shared between the client and the *last* AIP in the chain (the exit AIP); then with the secret shared between the client and the next-to-last AIP, and so on, finally with the secret shared between the client and the first AIP in the chain. It should be noted that this encryption can be arranged so that multiply encrypting a packet does not increase its size at each step.

- The client then drops the packet into the Wormhole. The packet will travel through the secure link to the first AIP (and in so doing, will be encrypted, sent as one of

the packets in a constant-rate stream, and decrypted by the first AIP), where the outermost layer of encryption on the IP packet will be removed.

- The result will then be dropped back into the Wormhole, to be delivered to the second AIP in the chain, over the secure link between the first AIP and the second AIP (and again will be encrypted, sent as one of the packets in a constant-rate stream, and decrypted by the second AIP). The second AIP will then remove the outer layer of encryption on the packet, and drop it back into the Wormhole.

- This process is repeated for each AIP in the chain.

- The last AIP will receive the packet, decrypt it, and (now that it is fully decrypted) send it to its intended destination.

Note especially that the last AIP knows the server, and the next-to-last AIP in the chain, but *does not know who the client is*. (Note, however, it does get to see the plaintext data, so it is important for the client to remove identifying information from the *data* portion of the protocol before injecting the packet into the Wormhole; see section 6.3.1, below.) Conversely, the first AIP knows who the client is, and also the second AIP in the chain, but does not know who the server is, or the contents of the packets. The AIPs in the middle of the chain know none of the data, the client, and the server, but only know the AIPs on each side of them in the chain.

This chaining method is particularly useful against an **incremental attack**; that is, an

attack where the adversary can potentially compromise AIPs, but for which the effort required to do so is highly non-trivial, so that the attacker will tend to direct his attacks against one node at a time, and only if it seems useful to do so. An example of an incremental attack is the **legal attack**, in which the adversary uses legal action (for example, subpoenas or lawsuits) to compel the operator of an AIP to decrypt certain packets, or to reveal encryption keys. Referring to the categories of attackers listed in Section 5.1, we see that Law Enforcement and Litigious Groups are the most likely to engage in a legal attack. Another example of an incremental attack is the **extra-legal AIP compromise**, in which the attacker coerces the AIP operator, or breaks into the machine running the AIP. We feel that System Crackers, National Intelligence, and Organized Crime are the groups most likely to effect this style of attack. We note that having an AIP monoculture (that is, every AIP is running the same software) makes this kind of attack easier; finding a single hole causes all AIPs to become compromisable.

In an incremental attack, the adversary generally observes the exit AIP which is being used by the client, and wishes to learn who the client is. By attacking the exit AIP, the attacker can only learn the identity of the previous AIP in the chain. The attacker then needs to compromise this next-to-last AIP in order to find out the one before it, and so on.

By using methods of forward secrecy, we can arrange that once the client shuts down his IP Wormhole, even compromising an AIP that was in the chain is no longer of any use; because of this, the attacker only has the lifetime of the Wormhole to compromise each

AIP in the chain in turn. By choosing AIPs in multiple legal jurisdictions around the world (and in different time zones), this kind of attack can often be completely prevented.

On the other hand, an attacker may attempt a **mass AIP compromise**, in which he attempts to control a large proportion of the available AIPs. If he is successful, then any client which forms a route using only AIPs he has compromised, will have his privacy destroyed. The classes of attackers likely to attempt a mass AIP compromise include System Crackers, National Intelligence, and Organized Crime. A mass AIP compromise can be defended against with standard methods for host security. As in the incremental attack, though, if the AIPs form a monoculture, they may be significantly easier to attack.

We can now construct IP Wormholes that protect against a range of strengths of attackers, from server operators, to passive network sniffers, to (incremental) AIP compromisers (see Table 6.1). In Chapter 8, we will examine the effect of adding powerful active adversaries (who can modify Internet traffic at will) to the threat model.

Power of Attacker	Protection Method
Server operator	Use an AIP (6.1.1)
Passive Internet sniffer	Encryption (6.1.2) Packet padding (6.1.3) Link padding (6.1.4)
Incremental AIP compromise	Chaining (6.1.5)
Mass AIP compromise	Host security for AIPs (6.1.5)

Table 6.1: IP Wormhole protection methods against varying strengths of attackers

6.2 The Network Information Database

The second piece of the AIPI is the Network Information Database (NIDB). This is an Internet-accessible database that stores the current state of the AIP graph.

6.2.1 Providing the AIP graph information

When a client sets up an IP Wormhole (as described above), it needs to know a number of things about the AIP graph:

- The list of nodes, so that it can pick:
  (a) the nearest AIP to itself, and
  (b) an exit AIP, perhaps one with specific properties (such as its physical jurisdiction)
- The pairs of nodes with a secure link between them, so that it can choose a path through the AIP graph from the nearest AIP to itself to the exit AIP it has chosen.

At least this much information, then, needs to be stored in the NIDB and made available to the client. We will assume for the moment that the AIP graph information is rarely changing.

As mentioned earlier, the simplest way to implement the NIDB is to have a centralized database that the client can query for the current state of the AIP graph. Note that if your threat model includes the ability for an adversary to compromise nodes, then it is important that the client retrieve the *entire* AIP graph, and not just ask about the parts of the graph it intends to use; otherwise, if the node from which the client obtains the NIDB information is compromised, the client leaks information about which AIPs it intends to use in its IP Wormholes. Further, even if your threat model is not as above (so you do not force the client to download the entire AIP graph), but does include Internet sniffers, the connection between the client and the NIDB needs to hide at least the contents, and likely the length, of the query and response, by using encryption and packet padding.

6.2.2 Removing the single point of failure

Unfortunately, having a centralized NIDB server provides for a single point of failure: if the server goes down, clients will be unable to learn the topology of the AIP graph, and therefore will be unable to construct IP Wormholes. Worse, if the centralized NIDB server is compromised, it could report inaccurate data; for example, if an attacker compromises the NIDB server and a handful of AIPs, he can cause the NIDB to report that only his compromised AIPs are currently available; all clients will then create Wormholes through only compromised AIPs, and their anonymity will be destroyed.

We attempt to distribute the NIDB in the following manner:

- As was assumed earlier, each AIP possesses a long-lived public-key signature key pair. The clients can learn these keys either through some PKI mechanism [61] or via a web of trust [67]. Note that the list of public keys for AIPs changes even less often than the AIP graph; even an out-of-band delivery mechanism is likely to be acceptable in this case.

- Each AIP is responsible for reporting its own status; that is, whether it is up (implicitly; clearly, if the AIP is down, it will merely report nothing), and to which other AIPs it has active secure links. It produces a signed neighbour list, which includes a timestamp, and the list of public keys of its active neighbours, all signed with its private signature key.

- Using a flood-fill algorithm similar to NNTP [49], the AIPs exchange the most recent signed neighbour lists they know about. In this manner, all AIPs learn the current status of the AIP graph.

- Clients can then query a random subset (or a trusted subset, if there are AIPs a particular client trusts more than others) of the AIPs to learn their views of the AIP graph, and combine them (by taking the most recent signed neighbour list for each AIP) to form its own view of the graph.

In this way, each AIP becomes a server for the NIDB, and there is no centralized point of

failure, or of attack. This method of distribution will be analysed in more detail in Chapter 8.

6.3 Application-Level Proxies

The third piece of the AIPI is the support for application-level proxies. As seen earlier, there are two kinds of application-level proxies in the system: client-side proxies, which protect the identity of the client, and exit node proxies, which protect the integrity and security of the AIPI (and PIP Network). We will examine these two kinds of proxies separately.

6.3.1 Client-side proxies

The IP Wormhole provides for the hiding of some of the *metadata* of Internet communication; namely, the identity of the client. It does not touch the *data* in any way. Some applications, however, insert information identifying the client into the application data itself. Some examples of application data that could identify a client are:

- "From" addresses in email or newsgroup postings
- HTTP cookies

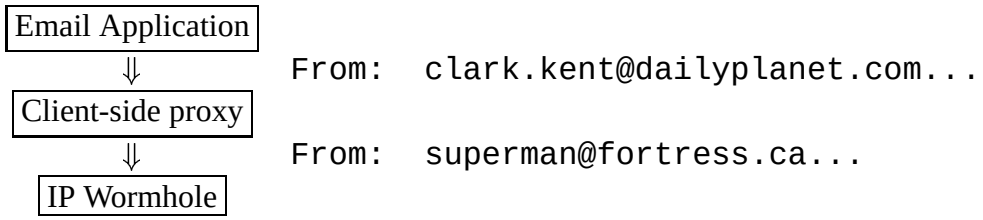

Figure 8: Using a client-side application-level proxy

- IP addresses in the FTP protocol

The purpose of the client-side application-level proxy is to remove all identifying information from any application data that is about to be sent over the IP Wormhole.

In addition, the proxy may also "sanitize" *incoming* data; for example, it may strip Javascript from web pages, remove inlined references to external images from email, or scan attachments for viruses (all of these are ways that have been used in the past to determine the IP address of an anonymous or pseudonymous user).

Whenever the client machine attempts to make a network connection to some service (POP mail, a WWW server, outgoing SMTP, etc.), the client software (which implements the AIPI) should transparently reroute the traffic to the appropriate client-side application-level proxy, which will sit between the application and the client end of the IP Wormhole (see Figure 8). The proxy can then sanitize the outgoing and incoming data, and prevent the client's identity from leaving the machine.

There are three ways to hook in a client-side proxy:

- Configure each application to pass its data to the proxy instead of to the network stack.
- Configure the socket library (such as `libsocket.so` on Solaris or Winsock on Windows) to pass data streams to a proxy (determined by the intended destination address and port) instead of to the network stack.
- Configure the network stack to redirect packets intended for certain destination addresses and/or ports, to a given proxy instead, in the manner of the Transparent Proxy feature of Linux.

Each of these methods is acceptable; which to use will depend on the operating environment of the client.

It is important to note that client-side proxies are part of the client's trusted computing base, and would usually run on the same machine as the rest of the client software. If an attacker compromises this machine, he has penetrated the trust boundary, and may gain access to all of the secrets of the client and the client-side proxies, which would usually include information about what pseudonyms are in use.

6.3.2 Exit node proxies

The purpose of the exit node application-level proxy is not to protect the client, but rather to protect the exit node, its operator, and the AIPI as a whole, from abuse.

Remember that, due to the design of the IP Wormhole, when a client using the AIPI contacts an Internet server, the IP address seen by the server is that of the exit node. Often, undesirable behaviours on the part of anonymous clients will get blamed on the exit node operator. To mitigate this risk, the exit node operators can set policies as to what kinds of traffic they will be willing to send to the Internet at large.[3]

Note that an exit node proxy is running for the benefit of the exit node operator; it is to no advantage to the operator to make the proxy malicious. In particular, any attack the exit node proxy would be able to perform, could equally well be performed by the operator of the Internet service to which the client is connecting, or to anyone with promiscuous access to the network between the exit node and that service. Since we believe that we have protected ourselves against attacks by the operators of the Internet services to which we connect, we face no further risk from the exit node proxies.

Examples of policies might include:

[3]Note that this risk does not arise if one is merely being an intermediate node in a chain, as the only systems which will see your IP address will be other participants in the AIPI. Further, as an intermediate node, you only get to see encrypted packets, so there's not much you could do about enforcing usage policies, anyway.

- Anonymous or pseudonymous connections shall not be allowed to certain sites, such as `whitehouse.gov`.
- All pseudonymous outgoing email must be properly digitally signed by the pseudonym in question. These emails can be counted in order to enforce a limit on the number of messages a given nym can send in a certain time period, thus preventing the use of the system for spamming.
- Completely anonymous connections may be allowed to some subset of sites; pseudonymous connections are required for others.
- Pseudonymous IRC access may be allowed, but the DCC SEND command (which is used to transmit large files from one IRC user to another) may be blocked, in order to prevent the system from becoming an automated copyrighted software or child pornography server.
- Only known protocols (SMTP, NNTP, HTTP, FTP, etc.) will be allowed through; unknown IP packets will be dropped. This prevents anonymous clients from performing various kinds of IP hackery [17, 18] against target sites.

Note that some of these examples assume the addition of the pseudonymity layer, to be described below.

The simplest way to interpose exit node proxies is to have the exit node send the decrypted IP packets not to the destination indicated in the packet, but rather to the proxy appro-

priate for the application to which the packet belongs (demultiplexed by port number, for example). The proxy (or the OS) will reconstruct the data stream, determine if the client's use of the application is valid, according to the exit node's policies, and if so, forward the application data to the true server.

The reply from the server will come back to the exit node proxy, which could even modify it, if it likes, before depositing the result into the Wormhole for delivery back to the client. One situation where this could be potentially useful is to do transparent compression of all data flowing from the exit node back to the client along the IP Wormhole; the exit node proxy will compress the data that it gets from the server, and the client-side proxy will uncompress it, and then pass the uncompressed result back to the client application, which will have no idea that anything funny happened at all.

6.4 Completing the PIP Network: Adding Pseudonymity

The IP Wormhole, the NIDB, and the application-level proxies together provide an effective Anonymous IP Infrastructure (AIPI). Our next task is to build a Pseudonymous IP (PIP) network, which, according to the principles of the Nymity Slider, should be relatively easy to build on top of the AIPI. As was seen earlier, the addition of persistent pseudonyms to the network will allow for reputation capital to be accrued by the pseudonymous users of the system, making it possible to have them participate more safely and trustworthily in

commercial and communicative transactions, without revealing their identities.

6.4.1 Obtaining pseudonyms

Clients can obtain pseudonyms in the following manner:

- A client will choose a unique (human-readable[4]) pseudonym, as well as generating a public-key signature key pair.
- The client makes an anonymous connection to a Certification Authority (CA) using the AIPI. The connection is anonymous so that the CA cannot correlate the pseudonym requested with the identity or IP address of the client.
- The pseudonym and the public verifying key are submitted anonymously to the CA, who ensures that no one else has previously chosen that same pseudonym, and then generates a certificate binding the verifying key to the pseudonym.

A client can, of course, generate more than one pseudonym. It would behoove him to submit them to the CA at different times, since otherwise the CA will learn that that particular group of pseudonyms are owned by the same person (although the identity of that person may remain unknown).

[4] "Human readable" means strings like "`superman@fortress.ca`" as opposed to public key hashes like "`e93bdfb9b26319e6c44cd3c3b53ad7c4b4837b5b`".

There are a number of variations on this design:

All-knowing CA: The CA may require some proof of identity before issuing a certificate for a pseudonym. In this case, it is not necessary for the client to communicate with the CA over the AIPI, but the communication should still be encrypted, or else an eavesdropper will learn which pseudonym the client is requesting. Note that this has a strong central point of failure: the CA can bind identities to pseudonyms, and so would make a "fat target" for legal or extra-legal attack.

Commercial CA: The CA may charge money for assigning a pseudonym. If the CA is not to know the identity of the client, some manner of anonymous electronic cash [20, 12] is required (see Section 3.3.1). The electronic cash is simply submitted along with the pseudonym and public key.

No CA: If human-readable names are not deemed necessary, the public key itself can act as the pseudonym, in the manner of SDSI [65]. Clients simply generate a public key pair, and there is no certificate at all.

The downside of having a CA is that, if the CA is shut down, no new pseudonyms can be created on the network (although existing pseudonyms will continue to work until their certificates expire). This can be somewhat mitigated by having many different CAs. The main benefit of having a CA is that is allows for unique human-readable pseudonyms, which can be important in environments such as email communication with people unfa-

miliar with the AIPI or PIP network. Also, having a CA allows it to set policy for obtaining a pseudonym; for example, the CA could require that the true identity of the client be escrowed or secret-shared amongst several trustees, even if the CA does not itself learn the identity. It is useful to note that this policy choice is independent of the rest of the design of the PIP Network — it is simply part of the added nymity tacked on when changing the anonymous AIPI into the pseudonymous PIP Network.

6.4.2 Using pseudonyms

In order to use a pseudonym, we make a tiny addition to the IP Wormhole setup protocol. After a client sets up an IP Wormhole between himself and some exit AIP (in the anonymous fashion, as before), he then has *the option* of presenting his pseudonym certificate (or just public key, if there is no certificate) and a fresh message signed with the pseudonym's private signature key to the exit AIP (over the IP Wormhole just constructed). If he does this, the exit AIP can now associate all packets leaving and arriving at that IP Wormhole with the presented pseudonym.

As noted in Section 6.3.2, the exit AIP may choose to require the presentation of a pseudonym in order to use certain protocols, or to access certain parts of the Internet. The exit AIP may also choose to honour some CAs, but not others, or to participate in a cooperative blacklist program with other nodes, which will enable him to block a certain pseudonym

if it has been determined that that pseudonym has been abusive of the network.

Note that even if the exit AIP learns the pseudonym of the client, it never learns the client's real identity. If the CA does not require the client to divulge his identity in order to obtain a pseudonym, then *no one other than the client himself* will know which pseudonyms are associated with which client.

Here is the main portion of the design:

- Clients choose a unique pseudonym (a "nickname"), as well as generating a public-key signature key pair.
- The pseudonym and the public verifying key are submitted to a Certification Authority, who generates a certificate, binding the verifying key to the pseudonym, after ensuring that no one else has previously chosen that same pseudonym.
- We make a small modification to our last IP Wormhole design, above, so that any AIP acting as an exit node will only accept a client-to-AIP secure link if the setup messages for that link are signed by a valid pseudonym's signature key, as evidenced by the pseudonym's certificate.

Now the only difference has become that the exit AIP, in addition to knowing the server to which the client is communicating, also knows the client's pseudonym (or, more correctly, *one* of the client's pseudonyms). Note that the exit AIP still does *not* know the client's

identity. In fact, if the Certification Authority does not require any sort of identification in order to produce a pseudonym's certificate, *no one* in the system (except for the client himself) will know which client is associated with which pseudonym.[5]

Once the exit AIP learns the pseudonym using a given IP Wormhole, it can make that information available to Internet servers in general, using a higher-level protocol such as ident [70]. Many Internet servers use the ident protocol today in order to determine the username of the client connecting to it; such servers would just end up obtaining the pseudonym of any client using the PIP network to connect to the server pseudonymously. This is perfect, since we require the end servers not be modified. Unfortunately, the ident protocol runs over insecure, unencrypted TCP/IP (and so the results could potentially be modified by an attacker in transit). To obtain higher assurance, a more secure version could be deployed, but then servers would need to be modified to use it.

[5] If the client is not careful, however, the CA may learn that some given pseudonyms are owned by the same person (without knowing who that person is). This can happen, for example, if the client submits several pseudonyms to the CA in rapid succession.

Chapter 7

Privacy Protection for Servers

The PIP Network as described provides privacy protection for clients of Internet services, so that the servers to which they connect do not learn information about their identities. As promised in Chapter 2, we will now demonstrate how, with little modification, we can provide for identity protection for the *providers* of the services, as well.

The key here is the existence of one or more **rendezvous servers**. These servers can, but need not, be part of the PIP Network; they can be operated by anybody, though, as we will see below, it is likely they will be of a transient nature, as volunteers bring them up and down (much like the current anonymous remailer network).

The goal of a rendezvous server is to act as a proxy in order to turn client-protected privacy into server-protected privacy. The general strategy (we will go into more details below) is

as follows. Here, Alice wishes to provide an Internet service without revealing her identity or location. We assume the existence of the PIP Network, as described.

1. One or more rendezvous servers come online and announce themselves.
2. Alice locates one of the rendezvous servers.
3. Alice (over the PIP Network) makes a connection to the rendezvous server, and gives it a service tag (a name for the service).
4. The rendezvous server assigns Alice an IP address (one that gets routed normally to the rendezvous server) and port.
5. The rendezvous server publicizes the (service tag, address, port) triple.
6. Bob wishes to use Alice's service, and looks up Alice's address and port using the service tag.
7. When packets from Bob arrive for Alice's port, they are encapsulated as data (headers and all), and sent to Alice over the PIP Network connection she has established.
8. Alice receives the packets and passes them to her service.
9. Response packets are encapsulated as data, and deposited into the PIP Network connection, for delivery to the rendezvous server.
10. The rendezvous server receives the response packets, decapsulates them, and sends the response to Bob.

7.1 The Difficulty of Naming

Before we delve into the details of the basic mechanism, we will first observe what is the inherent difficulty in providing a service without revealing one's identity or location. The fundamental problem is that you wish to advertise some way for clients to contact you, but that contact information should not be able to be used to identify you.

At first glance, it seems sufficient to simply use the address of some third party who is willing to offer the use of his address openly. But it is even harder than that: it is possible the service being provided is politically incorrect, or even illegal, in some jurisdictions. Simply using a third-party hosting service such as Geocities is inadequate, since the hosting service will come under pressure to terminate the service, and they have little incentive not to comply.

The problem is one of naming: the Internet has a number of different global-hierarchy unique naming schemes, such as DNS and IP addresses, and in order to send a packet to a service on the Internet, one needs to know the IP address for that service. If this address is fixed, the owner of the address may be compelled to remove the service.

So we endeavour to make the contact addresses non-fixed. The problem now is how does a service tell its potential clients what its address is? The usual answer is simply to use a well-known central nameserver, but again, this server can then come under pressure to not serve names associated with "unpopular" services.

We now go one step further, and decentralize even the name service. In order for Bob to learn the address-of-the-day for Alice, he queries any one of a distributed set of database nodes, which are operated by many different people and organizations, and so are unlikely to all be pressured to shut down (or to remove Alice's entry) all at once.

But now how does Bob find the nodes of this distributed database? To answer this question, we simply cheat. We use the fact that such decentralized databases exist, and that other people are worrying about those problems. We feel free to use systems such as Gnutella [54], FreeNet [22], or FreeHaven [31] as our nameserver. For concreteness below, we will suppose we are using Gnutella.

7.2 Details of the Basic Mechanism

In this section, we will give a detailed description of the basic mechanism for using rendezvous servers. We will also give a running example. Note that the IP addresses used herein are completely fictitious.

One or more rendezvous servers come online and announce themselves:

When a rendezvous server comes online, it announces itself by making available to Gnutella a file named `Rendezvous`, initially containing only the IP address and port it is listening on for service connections. In this example, suppose it listens on

the address `115.1.2.3:9114`.

Note also that if the distributed database in question supports queries by pattern or substring matching, it could more easily just publish an empty file with the name `Rendezvous-115.1.2.3:9114`.

Alice locates one of the rendezvous servers:

Alice queries Gnutella (over the PIP Network) for files named (or starting with) `Rendezvous`, and retrieves one or more of them at random. Alice reads the file (or filename) to determine the IP address and port of the rendezvous service (in this case, `115.1.2.3:9114`).

Alice makes a connection to the rendezvous server, and gives it a service tag:

Alice simply uses the PIP Network in the straightforward way to make a TCP connection[1] from herself to the rendezvous service. Note that the rendezvous server itself might not be part of the PIP Network, so:

(a) The rendezvous server learns the IP address of the exit AIP Alice is using.

(b) Data passes in the clear between the exit AIP and the rendezvous server.

We note that this is acceptable, since the goal here is to provide privacy of identity and location, not the contents of the communication (which will likely be in the clear, anyway, between the client and the rendezvous server). We note further that

[1] For simplicity, we will assume a TCP connection, but UDP would also work here, and would avoid issues with TCP-in-TCP multiple retransmit.

the rendezvous server is not given any sensitive information; it need not be trusted to keep secrets, nor can a legal or extra-legal attack on it result in any knowledge useful to an attacker.

The rendezvous server will see a TCP connection from the TCP/IP address/port `43.4.5.6:4386` which was assigned by the IP Wormhole used by Alice.

Alice also provides a **service tag**, which is simply a unique name identifying the service. It need not be human-readable (though it should be printable), so that, for example, a hash of a public key (which can also be used to sign this very message) can be used. We will suppose the service tag in this case is `BigBux`.

At this point, the rendezvous server may also require some form of per-service payment, in the form of anonymous electronic cash (for example, the schemes of Brands [12] or Chaum [20]). In this way, rendezvous server operators could be recompensed for the exposure they undergo. Note, however, that it is possible that in some jurisdictions, accepting payment may put the rendezvous server operator on shakier legal ground, if he is claiming that he is content-agnostic.

The rendezvous server assigns Alice an IP address and port:

The rendezvous server will assign Alice the address `115.1.2.4:7352`. Note that the IP address assigned to Alice may be different from the one used by the server itself; this is fine, so long as packets for both addresses get delivered to that machine.

The rendezvous server keeps an internal table correlating the service tag, the IP

address and port on which it is listening for packets from clients, and the IP address and port from which it received the connection to the rendezvous service. In this example, the rendezvous server's table will include the line:

`BigBux` ↔ `115.1.2.4:7352` ↔ `43.4.5.6:4386`

We note that the contents of this table are not sensitive, and are not useful to an attacker trying to locate Alice (all it contains is the IP address of the exit node of Alice's IP Wormhole, which is assumed to be secure). The rendezvous server could even choose to publish this table, either on a web page, or as part of its `Rendezvous` file. Choosing to publish in this way can be used to avoid being hassled by subpoenas wishing to extract the data.

The rendezvous server publicizes the (service tag, address, port) triple:

The rendezvous server publishes a file on Gnutella called `RendSvc-`*`tag`* containing the address and port it assigned to the service. As above, the address and port can also be put in the filename.

Bob looks up Alice's address and port using the service tag:

First, Bob needs to hear about the BigBux service somehow. He can do this either through browsing Gnutella for files starting with `RendSvc-`, or through Usenet, or (the most popular way nowadays) through spam email. The message would contain the (long-lived) service tag `BigBux`.

Having learned of the existence of the service, Bob finds the `RendSvc-BigBux`

file in Gnutella to locate the address and port of the rendezvous listener assigned to BigBux, and directs his web browser there. (In this case, the address would be `115.1.2.4:7352`.)

There may be a problem if people attempt to thwart the system by inserting many fake `RendSvc-BigBux` files. This can potentially be solved by having Alice sign the contents of the files with a key publicized in the spam (again, even the signature can be put in the filename, with short signature schemes). This of course adds onus on Bob to check the signatures, but that could conceivably be automated.

Packets from Bob arrive for Alice's port:

Bob's web browser will start by sending a SYN packet from some TCP port at his address, say `24.4.6.8:1027` to the rendezvous listener for BigBux, listening at `115.1.2.4:7352`. That SYN packet will arrive at the rendezvous server, which will simply snatch the entire packet off the network (unprocessed by its host OS; we want the SYN to set up a TCP connection between Bob and Alice, not between Bob and the rendezvous server), and send it over its open TCP connection to the IP wormhole at `43.4.5.6:4386`.

To be clear, this last packet will be a TCP/IP packet from `115.1.2.3:9114` to `43.4.5.6:4386`, which, as its data payload, will contain a TCP/IP packet from `24.4.6.8:1027` to `115.1.2.4:7352`. Right now, this inner packet will be a SYN packet, but later on, it may contain application data.

The rendezvous server could also charge Alice a bandwidth-related fee at this step, in order to offset its extra network costs.

Alice receives the packets and passes them to her service:

The program on Alice's machine that established the TCP/IP connection over the IP Wormhole to the rendezvous service (which we will call the "rendezvous program") now receives the inner packet from `24.4.6.8:1027` to `115.1.2.4:7352`. It then arranges to have the packet processed by the host OS and delivered to the application server she is running; how this is done will be OS-dependent. (On Linux, for example, this could be done via ipchains [66]; note that some IP address rewriting in the manner of IP Masquerading may be necessary, as well.)

In our case, the SYN packet would be processed by the OS, which would create a SYNACK packet in response. Future response packets would contain application data.

Response packets are returned to the rendezvous server:

Using similar OS-dependent mechanisms, the response packet is delivered to Alice's rendezvous program, which ensures that it is a TCP packet from the address `115.1.2.4:7352` to `24.4.6.8:1027`, and sends it over the TCP connection over the PIP Network to the rendezvous server.

The rendezvous server decapsulates the response packets, and sends them to Bob:

The rendezvous server will receive the response packet as data over the TCP con-

nection from the IP Wormhole, and simply drop it on the Internet, where it will be routed to Bob.

Note that in all this, we have not even given an example IP address for Alice herself, so clearly none of the participants could have learned it. Also, the only custom software was at Alice's machine and at the rendezvous server. Bob only needs a Gnutella client, and a web browser.

7.3 Cleaning Up

When the TCP connection between Alice and the rendezvous server closes, the rendezvous server can clean all state associated with it, including the port being listened to for clients, the table entry, and the publicized `RendSvc-BigBux` file. The rendezvous server can also use "keepalive" packets (possibly also containing payments) between itself and Alice to ensure that Alice's machine is continuously up and willing to accept connections; if Alice's machine dies suddenly, it can then time out the connection and clean up. Note that it must not clean up simply out of inactivity; Alice is running a server, and may receive client connections only rarely, but needs to be contactable whenever they arrive.

7.4 Adding Robustness

As noted above, any machine that is seen to be a contact point for an undesirable service will tend to come under attack, even though in this case, the rendezvous servers have no idea what kind of packets they are shuffling back and forth. We therefore assume that Alice should treat the rendezvous servers as transient, as they may cut her off (or disappear entirely) at any time.

The simplest thing for Alice to do is simply to use multiple rendezvous servers simultaneously. Clients connecting to any of the servers will have their packets routed to Alice. If a rendezvous server shuts down, clients using it will be disconnected, and will have to look up another rendezvous server for Alice, and reconnect to a new IP address (thus possibly destroying any session state the client had).

If rendezvous servers appear and disappear slowly, this is likely acceptable. However, if this happens quickly, it is not so good. For example, it may come about that a popular Gnutella client can also act as a rendezvous server. Then, one would expect rendezvous servers to be appearing and disappearing all the time. How can we deal with a situation like this?

We make some minor changes to the basic mechanism:

- Alice should check regularly for active rendezvous servers (by querying the data-

base), and should maintain active connections to several of them at any time. This number will of course depend on the rate at which the servers are appearing and disappearing, and should be chosen so that Alice can expect to remain connected to at least one or two of the servers before the next time she checks for new ones.

- The rendezvous server listens on one additional port, for connections from *clients*. Data sent to this port by clients will be of the form (service tag, IP packet). The rendezvous server will then simply forward the IP packet over the server connection associated with the service tag.

- While Bob is using Alice's service, he should regularly query the database to update his list of rendezvous listeners for the service. He also uses special software (note that Bob now needs special software as well, unfortunately) to intercept the IP packets destined for Alice's service, and send them instead to the client port of *any* rendezvous server listening for Alice.

With this mechanism, Bob can maintain a constant TCP connection with Alice, even if the rendezvous servers with which Alice is registered keep appearing and disappearing.

7.5 Bi-directional Privacy Protection

Now that we have an effective means to provide privacy protection for servers, what if we want bi-directional privacy protection, so that neither party reveals his identity, IP address, or location?

This is clearly simple: just proceed as above, but have Bob connect to the rendezvous server via the PIP Network himself. Now the role of the rendezvous server is simply to shuffle packets between the exit points of two IP wormholes. The data in these packets, again, is in the clear, but this end-to-end problem can easily be solved with an end-to-end cryptographic solution such as SSL.

Part III

Analysis

"In theory, there's no difference between theory and practice. In practice, there is."

Chapter 8

Analysis

In this chapter, we shall present an analysis of the Pseudonymous IP Network as described, and also some shortcomings that came to light while observing the constructed system. We will present fixes for a number of these problems, but we will note up front that some of them are fundamental, and fixing them would be at odds with one or more of the design goals of the system. This is obviously a disappointing situation, but we will do the best we can to ameliorate the conflict.

8.1 Client issues

In this section, we will discuss issues primarily related to the client side of the PIP Network. We note that this is necessarily an approximate taxonomy, as not all issues can be cleanly divided as to whether they pertain to the client, the network, or the AIPs.

8.1.1 Implementation Surprises

The client software is responsible for ensuring that information about the links between user and nyms is not leaked. This turns out to be surprisingly difficult. As an example, Netscape Navigator keeps track of the user's browsing history, collected across all nyms (Navigator, being an opaque unmodifiable binary, cannot be fixed to know about different nyms). It is bad enough that interactions in the browser cache may leak information about what pages have been visited by some other of the user's nyms, but Netscape's "What's Related" feature actively *sends* part of that history over the network.

Many problems of this type can be solved by a more highly-enforced separation boundary between data associated with different nyms; see Section 8.1.3 for more on this.

Other issues are trickier; for example, if the PIP Network suddenly became unavailable for some reason, it was sometimes the case that the TCP retransmit would go out in the clear over the Internet. Any packet leaks of this type could possibly compromise the nym.

IP packet fragmentation and reassembly was also an issue; what happens if one of the inter-AIP links has an abnormally small MTU? Should Path MTU discovery be honoured? If you're not careful, then watching for fragmented or very small packets emerging from an IP Wormhole can give you a clue as to which nyms are using the low-MTU link.

Just as Cheswick and Bellovin's Corollary 1.1 states, "A security-relevant program has security bugs," [21] it is also the case that a privacy-enhancing technology has privacy bugs. But in a case like this, we may have the unfortunate problem that any bug which associates a nym with a user causes that association to exist forever; fixing the bug cannot remove the association. This implies our implementation needs to be carefully audited and tested before seeing widespread use.

8.1.2 Trusted Computing Base

Anyone wishing to use this PIP Network clearly needs to have a local machine under his control that can intercept and encrypt packets bound for the Internet, establish routes through the PIP Network, and send the multiply-encrypted packets over the network.

This machine necessarily knows both information about the user's pseudonyms, as well as his real IP address, and likely other personal information about him. For this reason, this machine needs to be part of the user's trusted computing base [30], as it is desired that the user himself be the only one with the ability to reveal which pseudonyms he controls.

Certainly, if this machine were compromised by an attacker, the user's privacy would be compromised: all of his online actions, whether performed under his own name or that of a pseudonym, would be revealed.

For this reason, it is necessary that this machine be made as secure as possible against external intrusion. Depending on the operating system, it may be desirable to have access to the machine (both over the network and physically) restricted to the single user. Another user having superuser or administrator access to that machine definitely gives the superuser the ability to compromise the privacy of the nyms stored on it.

If this machine is the user's general-purpose workstation (as opposed to a gateway or a firewall machine, for example), extra care must be taken to prevent the infection of the machine by malware such as viruses, trojan horses, or remote-control applications such as Back Orifice [26].

8.1.3 Entanglement

We borrow the term **entanglement** from quantum mechanics to indicate the (undesirable) behaviour of information leakage between a user's real identity and one of his pseudonyms, or between two of his pseudonyms.

For example, two nyms can become entangled of one of them receives a password to a web

site, but a different one uses it. Similarly, if the user (as himself) reveals information he learned as one of his nyms, that could entangle his real identity with his nym. Stylometry [63] is a special case of this problem; here, the user's *writing style* leaks across to that of his nyms.

Socially, this is a difficult problem; the user needs to keep track of what information he learned as what nym, and have the other nyms plead ignorance to it.

Technically, it is also difficult. If the same system stores information learned from multiple nyms, it may accidentally leak (or be coerced into leaking) this information across nyms. In particular, if the system contains applications, such as a web browser or mail reader, that are not aware of the underlying PIP Network, then details such as:

- which pages or images exist in the browser cache, along with their last update times
- inlined images received in email by one nym, but viewed when the user is operating under another nym
- application history, version numbers, and capabilities

can all be used to link together information received by different nyms.

Preferably, the user should separate the information he receives under each of his nyms. Suggesting completely separate machines is somewhat extreme, although separate *virtual*

machines, such as VMWare [76] or Plex86 [53], may be appropriate and useful; the user would open the virtual machine corresponding to his desired current nym, and use the tools within it. Moving information between the virtual machines is then difficult, which is a feature, since it makes accidental entanglement correspondingly difficult.

8.1.4 Use of the Network

It should be noted that when a user connects to the PIP Network, his actions online remain private, but the *fact* that he is using the PIP Network can be known, for example, to anyone noticing the encrypted packets travelling between his machine and one of the AIPs.

This opens a potential attack wherein the attacker observes that a certain nym is only active rarely, say for a few minutes a month, and that a certain user only connects to the PIP Network for that same few minutes a month. In order to pull off this attack, the attacker would need to be watching many AIPs to see when certain nyms are in use, and when various clients connect to them.

We note that this seems to be contradictory to the earlier claims of security against a passive attacker. However, there, we went through trouble to make each client's network behaviour identical (by using packet padding and link padding, for example), in order that an attacker be unable to distinguish different clients.

Now, we see that, if users connect to and disconnect from the PIP Network, we cannot reasonably make their traffic patterns the same (especially if they are offline some of that time). This variance in when clients are online and connected to the PIP Network may allow the attacker to distinguish them.

This is a difficult problem to address; we cannot reasonably require (in most cases) that clients remain connected to the PIP Network at all times, though that would eliminate the passive attacker problem. It may be possible to propose some sort of "mobile agent" solution, in which an agent, operating on a connected part of the network, can use an otherwise rarely used pseudonym, even when the owner is disconnected from the PIP Network. This may foil some of the correlations of user connect time to nym use time, at the cost of the usual trust and security issues for mobile agents.

8.2 Network issues

8.2.1 Latency Variations

Earlier, we showed that if all clients send the same patterns of traffic, both in packet sizes, and in inter-packet timings, then an observer of the network should be unable to distinguish clients based on their (identical) packet behaviours.

However, observing the network in action, we found that this is not quite accurate, under slightly more general assumptions. We allow the attacker to observe the packets between the exit AIP and the Internet service with which the client is communicating, and we also allow the attacker to measure the *latency* of any link in the PIP Network.

Now the attack proceeds as follows: the attacker observes some pseudonym requesting some data from an Internet server (a web server, for example). It may so happen that this response contains data which will cause the client software to automatically perform some subsequent request (for example, fetching an inlined image, or even just the transmission of the TCP ACK for the received data). The attacker will then observe that second request, after some elapsed time.

The attacker can then correlate the time between the initial response and the subsequent request with the round-trip latency between his own location, and the client (over the PIP Network, of course). By being able to measure latencies of inter-AIP links, and latencies of the links between clients and their entry AIPs (the first AIP in the chain they have selected), the attacker may be able to determine the chain used, and the location of the client. As a practical example, Cliff Stoll used similar round-trip latency measurements to identify the physical location of a network intruder in the celebrated “Cuckoo’s Egg” case [71].

This attack is a “side-channel” attack, and is quite similar to other such attacks, such as reaction attacks [41], timing attacks [51], and power attacks [50]. As in all of these

attacks, even though we have made the various data under observation indistinguishable to the attacker, the coupling to the external world leaks information he can use.

One way to thwart this attack is to ensure all links in the PIP Network have the same latency. AIPs would collect all the packets that arrive in a certain time interval, then, at the end of the interval, send them all out (in a randomized order). This interval would be constant for all AIPs, and would have to be larger than the largest inter-AIP network latency. This approach clearly has severe performance implications, but may need to be implemented if this attack is part of your threat model.

8.2.2 Active Attacks

If the adversary is able to perform active attacks on the network, such as delaying, deleting, inserting, or modifying packets en route to their destinations, he can gain quite a bit of leverage.

As before, we wish the attack to be unable to distinguish clients based on their network behaviours, so we try to make them behave identically.

However, if the adversary has the power to attack the network actively, he can selectively delete packets from certain clients, and see which pseudonyms continue to function, and which abruptly stop.

The most extreme form of this is the “half-the-Internet” attack, where an extremely powerful attacker selectively shuts down half the Internet, and sees which nyms continue to operate. By repeating this process with different partitions of the Internet, he can likely learn all client-to-nym correlations. We are not claiming even that an adversary of this power exists; however, it shows that there are limits to the level of active attack any such network can withstand.

Interestingly, Pipenet [27] does offer a potential solution to the above problem, albeit at a very high cost: if a Pipenet node ever fails to receive a packet it was expecting (for example, due to packet loss), it assumes it is under attack, and stops transmitting packets entirely. This of course causes all its neighbours to stop transmitting, and very quickly the entire network is stopped. Thus Pipenet has the property that it is either performing perfectly, in the sense that packet sizes and times are perfectly distributed (and so no information from them can be extracted by an attacker), or else it is not performing at all (in which case again, no information can be extracted by an attacker).

This extreme resistance to information leakage will unfortunately have the tendency to keep the network down all the time; simple packet jitter will shut down Pipenet, even if there is no malicious attacker to do it. Although we tend not to worry too much about simple denial of service attacks, this is too extreme, as it is too easy to shut down the network completely.

Another form of active attack is for an attacker rapidly to create routes through the PIP Net-

work, consuming bandwidth and cryptographic processing power at the AIPs. Although this is somewhat of a simple Denial of Service, we can easily deal with it.

As a client builds a chain of AIPs through which to transmit his packets, each AIP in the chain will ask him for a token of some sort. This token can be in one of two forms:

Electronic cash: The token could be actual electronic cash, of an anonymous form such as [20] or [12]. In this way, the attacker needs to spend significant money in order to overload the network.

Client puzzles: The token could be the answer to a puzzle posed by the AIP; for example, in the manner of Hashcash [7]. These puzzles have the property that a significant amount of CPU time needs to be spent on solving them, so an attacker could not consume the majority of the resources of the network. RSA has proposed a similar scheme to combat denial of service in regular TCP connections to web servers [47].

8.3 AIP issues

8.3.1 Colluding AIPs

Care must be taken when selecting a chain of AIPs to use that you do not select a set of AIPs that are all working together, and sharing their information in order to compromise

the system.

Note that this is not saying that there must be one AIP in the chain that you trust; you need not in fact trust that any AIP in the chain is not acting maliciously; you need only trust that not all the AIPs in your chain are *colluding*. For example, you may suspect that one particular AIP is run by the US Government, and that one particular other AIP is run by the Libyan Government. Although you may trust neither of these organizations to keep your privacy safe, you might trust that at least they won't work together in order to out you.

8.3.2 First-and-Last Attacks

There are a class of attacks on the PIP Network called "First-and-Last" attacks. These attacks use the fact that the first AIP in your chain knows your real IP address, and the last AIP in your chain knows your pseudonym. The goal for the attacker is then:

- Compromise a number of AIPs.
- Collect IP address information from clients using one of the compromised AIPs as a first AIP.
- Collect pseudonym information from clients using one of the compromised AIPs as a last AIP.

- Try to correlate those two sets of information.

The various attacks in the class try to achieve the correlation in different ways. For example:

Net presence correlation: As noted above, correlate when certain IP addresses are online with when certain pseudonyms are active.

Active packet modification: Garble some incoming packets at the first AIP. At the last AIP, you will notice that the purported cleartext is also garbled; in this way, you can correlate the pseudonym that packet was supposed to go out under with the IP address from which the packet you garbled came.

This attack can be prevented by using integrity checking at each layer of the nested encryption. This can be done fairly quickly, for example, by using Integrity-Aware CBC mode [48], but it adds overhead to the packet at each layer.

Padding removal: If the client only does link padding between himself and the first AIP, then a compromised first AIP will learn what the *true* traffic pattern is. A compromised last AIP can then look for a pseudonym with that same traffic pattern, and link up the pseudonym with the client.

This attack can be prevented by doing end-to-end (actually, client-to-last-AIP) link padding instead of (or in addition to) client-to-first-AIP link padding. Then, only the

> last AIP ever learns the real traffic pattern (which it needs to know anyway, since it will send the traffic in the clear onto the Internet).
>
> However, the last AIP can still play tricks with the return path: even if we are doing client-to-last-AIP link padding, if the last AIP is malicious, it could just fail to do the link padding on the data coming back to the client, and possibly expose him in that way. In addition, client-to-last-AIP link padding adds substantial padding overhead to the entire network.

In particular, then, when selecting a chain of AIPs, choosing two non-colluding AIPs as your first and last in the chain can protect you from these attacks.

Chapter 9

Conclusion

9.1 Contributions

This work has presented the design and analysis of the Pseudonymous IP Network (PIP Network), a system which enables the use of persistent pseudonyms in order to communicate in real time over the Internet. This communication mechanism provides a basis for other privacy-friendly applications such as digital cash. Providing long-term pseudonymity is a difficult proposition, as a leak of the binding between verinym and pseudonym can have consequences arbitrarily far backwards and forwards in time.

In order to motivate the design of the PIP Network, we introduced the concept of the Nymity Slider, an abstraction which lets us evaluate the amount of personally identify-

ing information revealed by a given transaction. We saw that the "ratchet" nature of the Nymity Slider implies that we should design our technologies to leak as little information about the users' identities as possible; we can always have a higher-level protocol add more identity information, but it is very difficult to remove it when it is unwanted.

Taking this advice, we designed the PIP Network as an Anonymous IP Infrastructure (AIPI) with a pseudonymity layer added on top. In building the AIPI, we borrowed some useful pieces from other privacy-enhancing technologies, such as chaining from anonymous remailers, and used new techniques to deal with the specialized problems of an interactive environment.

In analyzing the PIP Network, we enumerated a number of potential classes of adversary, and estimated their power. In Table 9.1, we summarize the various attacks discussed in this work, as well as defenses, and the efficacy of those defenses against the various threats. We point out that this chart is meant to be illustrative, not definitive; there is no way, for example, for us to truly know the power of most of the groups listed. Further, the true strength of any adversary will be very dependent on the particular situation at hand. The table simply estimates what likely powers each adversary has, and whether the given defenses are likely to be effective. Also, a given attacking entity may potentially fall into more than one class of attacker; Organized Crime may utilize System Crackers to further their ends, for example.

Finally, we introduced the rendezvous server, a primitive which allows the privacy proper-

ties of a client-protecting system like the PIP Network to be extended to servers as well.

9.2 A Final Word

The Nymity Slider tells us that a system only provides as much privacy as the layer with the highest nymity: if any protocol layer, from Physical, through Network, up to Application (or above), reveals information about you, it is very difficult for other layers to re-hide that information.

Keeping this in mind, we realize that we must be careful when we design components of applications, and that we must proactively design privacy into the protocols. Certainly we must avoid explicitly designing behaviours that are detrimental to users' privacy; writing such explicit "spyware" will hopefully start to be seen as simply unethical.

However, we must equally not allow ourselves and our designs to be privacy-agnostic. The default behaviour of most systems is to leak personal information in the form of metadata: context, timing information, and headers can all be used to identify clients of protocols, and systems that leak information accidentally are often worse than those that reveal information by design, since at least sometimes there is the option to the user to turn off the latter behaviour.

Rather, designers of protocols, applications, and systems, at all layers of the ISO stack,

Class of Attack	Defense (if any)	1	2	3	4	5	6	7	8
Parsing log files	Use an AIP	●	○	●					
Trusted Computing Base penetration	Host security		○		○	●	●	○	●
	Encrypted local data		●		○	○	○	○	○
Finding entanglement	Care in separating nyms			●					
Forcing entanglement	Virtual machines	●				●	●	●	●
Correlating network usage times	Mobile agents		●		●	○	○	●	○
Internet sniffing	Encryption, packet padding, link padding		●			●	●		
Exploiting latency variations	Fixed-latency network	●	●			●	●		●
Active packet deletion	Propagate losses					●	●		
"Half-the-Internet"							○		
Create many routes	DoS tokens					●	●	●	●
Incremental AIP compromise	Chaining				●	●	●	●	●
	Heterogeneity					●	●		●
Mass AIP compromise	Security for deployed AIPs					●	●		●
Mass AIP collusion							○		○
"First-and-last"	Route choice		●			○	○		○
	Security for deployed AIPs		○			●	●		●

Threats (see Section 5.1)

1 Web Site Operators
2 Systems Administrators and Internet Service Providers
3 Search Engines
4 Lawmakers and Law Enforcement
5 System Crackers
6 National Intelligence
7 Litigious Groups
8 Organized Crime

- ● Attack applicable, defense effective
- ○ Attack applicable, defense ineffective

It should be noted that these attacks and defenses are not as black-and-white as the chart may indicate; sometimes certain attacks can only somewhat be carried out by the party in question, and given defenses can are more or less effective, depending on the circumstances.

Table 9.1: Summary of threats, attacks, and defenses

must enter a new mindset: Privacy is Important. What personal information about the user does your design reveal? To whom? Is the user notified of this? Can she turn it off meaningfully? What is done with this information? Where is it archived? Who has access? What are the protections against it being stolen or misused?

Both the European Union and the FTC in the United States have adopted guidelines including questions like the above. Ignoring privacy issues when designing protocols, writing applications, or deploying systems, is simply no longer acceptable practice.

When computers were new on the scene, the goal of every programmer was correctness: his programs had to work properly. Hot on the heels of correctness was performance: computers were still a scarce resource, and it was considered admirable for programs to consume as little of that resource as possible.

Time passed, and by the 1990's, computers had come out of the research labs, and for the first time, the Internet was seeing significant numbers of new users in months other than September. In this more open environment, security was the next important thing. Older, more trusting, protocols could no longer be used. Encryption, firewalls, and sandboxes started appearing, and programmers and crackers alike knew to watch out for the dread buffer overrun.

Now we are entering 2001. Computers are commonplace in the home, and the Internet is a commodity available to the general public for $19.95 a month. People are conducting more

and more of their personal lives on their computer, if not online. In this environment, the threat to personal privacy is real. Instead of worrying about teenage crackers stealing your CPU cycles, we must now concern ourselves with reputable software companies stealing your personal information when you install their software for teaching children to read.

In addition to the concerns of correctness, performance, and security taken up by the previous generations of designers and programmers, the current generation will need to address issues of privacy in the same manner. As today's users are forced to deal with the security choices made by generations past, the users of tomorrow will have to live with the privacy choices we make today. Let's not let them down.

Bibliography

[1] Ross Anderson. The Eternity Service. In *Proc. Pragocrypt 96*, pages 242–252, 1996.

[2] Anonymizer.com. http://www.anonymizer.com/.

[3] Gary Anthes. Irs uncovers bogus access to tax records (internal revenue service's atlanta office investigation). *Computerworld*, 27(32):15, August 1993.

[4] Associated Press. 19 September 1996.

[5] André Bacard. Anonymous remailer faq, 1999. http://www.well.com/user/abacard/remail.html.

[6] Adam Back. Eternity service. http://www.cypherspace.org/~adam/eternity/.

[7] Adam Back. Hash Cash: A partial hash collision based postage scheme. http://www.cypherspace.org/~adam/hashcash/.

[8] Adam Back, Ian Goldberg, and Adam Shostack. Freedom 2.0 Security Issues and

Analysis, December 2000.
http://www.freedom.net/info/whitepapers/Freedom_Security.pdf.

[9] James Bamford. *The Puzzle Palace*. Penguin Books, New York, 1983.

[10] Douglas Barnes. The Coming Jurisdictional Swamp of Global Internetworking (Or, How I Learned to Stop Worrying and Love Anonymity).
http://www.io.com/~cman/swamp.html, November 1994.

[11] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom Systems 2.0 Architecture, December 2000.
http://www.freedom.net/info/whitepapers/index.html.

[12] Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates — Building in Privacy*. PhD thesis, Eindhoven University of Technology, Netherlands, 1999.

[13] L. Jean Camp. *Trust and Risk in Internet Commerce*. MIT Press, 2000.

[14] L. Jean Camp, Michael Harkavy, J.D. Tygar, and Bennet Yee. Anonymous Atomic Transactions. In *Proc. 2nd Annual Usenix Workshop on Electronic Commerce*, pages 123–134, Oakland, CA, November 1996.

[15] L. Jean Camp, M. Sirbu, and J.D. Tygar. Token and notational money in electronic commerce. In *Proc. Usenix Workshop on Electronic Commerce*, pages 1–12, New York, NY, July 1995.

[16] Duncan Campbell. Interception capabilities 2000. *European Parliament Directorate General for Research Directorate A The STOA Programma*, 1999. http://www.nrc.nl/W2/Lab/Echelon/interccapabilities2000.html.

[17] CERT. Smurf IP Denial-of-Service Attacks, 13 March 2000. http://www.cert.org/advisories/CA-1998-01.html.

[18] CERT. IP Denial-of-Service Attacks, 26 May 1998. http://www.cert.org/advisories/CA-1997-28.html.

[19] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the Association for Computing Machinery*, 24(2):84–88, February 1981.

[20] David Chaum. Blind signatures for untraceable payments. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proc. CRYPTO 82*, pages 199–203, New York, 1983. Plenum Press.

[21] William Cheswick and Steven Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[22] Ian Clarke. The Free Network Project. http://freenet.sourceforge.net/.

[23] Elizabeth Corcoran. Hackers Strike at NY Internet Access Company. *The Washington Post*, page D09, 12 September 1996.

[24] Lance Cotrell. Mixmaster & remailer attacks, 1995. http://www.obscura.com/~loki/remailer/remailer-essay.html.

[25] Ray Cromwell. Welcome to the Decense Project, 1996. http://www.clark.net/pub/rjc/decense.html.

[26] Cult of the Dead Cow. Back Orifice 2000. http://www.bo2k.com/.

[27] Wei Dai. Pipenet 1.1. http://www.eskimo.com/~weidai/pipenet.txt, 2000.

[28] Stephen Deering and Robert Hinden. *Internet Protocol, Version 6 (IPv6) Specification*, December 1995. RFC 1883.

[29] Laurent Demailly. Anonymous Http Proxy (preliminary release). http://www.demailly.com/~dl/anonproxy.txt.

[30] Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria, December 1985. DOD 5200.28-STD (The Orange Book).

[31] Roger Dingledine. The Free Haven Project. http://freehaven.net/.

[32] Kjeld Egevang and Paul Francis. *The IP Network Address Translator (NAT)*, May 1994. RFC 1631.

[33] Arnoud Engelfriet. Anonymity and Privacy on the Internet, 26 January 1997. http://www.stack.nl/~galactus/remailers/index.html.

[34] Joe Hellerstein et al. Federated Facts and Figures. http://fff.cs.berkeley.edu/.

[35] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric A. Brewer. Scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, St. Malo, France, October 1997.

[36] Ian Goldberg, David Wagner, and Eric Brewer. Privacy-enhancing technologies for the Internet. In *Proceedings of IEEE COMPCON '97*, 1997.

[37] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In Ross J. Anderson, editor, *Information hiding: first international workshop*, volume 1174 of *Lecture Notes in Computer Science*, pages 137–150, Isaac Newton Institute, Cambridge, England, May 1996. Springer-Verlag, Berlin, Germany.

[38] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, February 1999.

[39] Steven Gribble and Eric Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, December 1997. http://www.cs.berkeley.edu/~gribble/papers/sys_trace.ps.gz.

[40] C. Gulcu and G. Tsudik. Mixing E-mail with BABEL. In *Symposium on Network and Distributed Systems Security (NDSS '96)*, San Diego, California, February 1996. Internet Society.

[41] Chris Hall, Ian Goldberg, and Bruce Schneier. Reaction Attacks Against Several Public-Key Cryptosystems. In *Proc. ICICS 1999*, 1999.

[42] Dana Hawkins. Privacy worries arise over spyware in kids' software. *U.S. News*, July 3, 2000. http://www.usnews.com/usnews/issue/000703/nycu/privacy.htm.

[43] Juha Heinanen. *Multiprotocol Encapsulation over ATM Adaptation Layer 5*, July 1993. RFC 1483.

[44] Johan Helsingius. Johan helsingius closes his internet remailer. http://www.cyberpass.net/security/penet.press-release.html.

[45] Douglas Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, New York, 1979. See also [45].

[46] International Standards Organization. The reference model of Open System Interconnection, 1982. ISO International Standard IS 7498.

[47] Ari Juels and John Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. http://www.rsasecurity.com/rsalabs/staff/ajuels/papers/clientpuzzles.pdf, February 2000.

[48] Charanjit Jutla. *A Parallelizable Authenticated Encryption Algorithm for IPsec*, November 2000. http://search.ietf.org/internet-drafts/draft-jutla-ietf-ipsec-esp-iapm-00.txt.

[49] Brian Kantor and Phil Lapsley. *Network News Transfer Protocol*, February 1986. RFC 977.

[50] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In Michael Wiener, editor, *Advances in cryptology — CRYPTO '99: 19th annual international cryptology conference, Santa Barbara, California, USA, August 15–19, 1999 proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1999. Springer-Verlag.

[51] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Lecture Notes in Computer Science*, 1109:104–113, 1996.

[52] Chris Kostick. IP masquerading with Linux. *Linux Journal*, 27, July 1996.

[53] Kevin Lawton. Plex86. http://www.plex86.org/.

[54] Michael Macedonia. Entertainment computing: Distributed file sharing: Barbarians at the gates? *IEEE Computer*, 33(8):99–101, August 2000.

[55] Jennifer Mack. DoubleClick privacy plan slammed. *ZDNet News*, 15 Feb 2000. http://www.zdnet.com/zdnn/stories/news/0,4586,2437611,00.html.

[56] John Miller. L.A. Not-So-Confidential. *ABC News*, 8 October 1998. http://more.abcnews.go.com/sections/tech/DailyNews/privacyla981007.html.

[57] Mojo Nation. http://www.mojonation.net/.

[58] The Nando Times, 20 November 1996.

[59] Napster. http://www.napster.com/.

[60] The New Yorker, 5 July 1993. page 61.

[61] Radia Perlman. An Overview of PKI Trust Models. *IEEE Network*, 13(6):38–43, November 1999.

[62] A. Pfitzmann and M. Waidner. Networks without user observability — design options. In Franz Pichler, editor, *Advances in cryptology: Eurocrypt 85: proceedings of a workshop on the theory and application of cryptographic techniques, Linz, Austria, April, 1985*, volume 219 of *Lecture Notes in Computer Science*, pages 245–253, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1985. Springer-Verlag.

[63] Josyula R. Rao and Pankaj Rohatgi. Can pseudonymity really guarantee privacy? In *Proc. Ninth Usenix Security Symposium*, Denver, Colorado, 2000.

[64] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[65] Ronald Rivest and Butler Lampson. SDSI–A Simple Distributed Security Infrastructure. http://theory.lcs.mit.edu/~cis/sdsi.html.

[66] Rusty Russell. Linux IP Firewalling Chains. http://netfilter.filewatcher.org/ipchains/.

[67] Jeff Schiller and Derek Atkins. Scaling the Web of Trust: Combining Kerberos and PGP to Provide Large Scale Authentication. In *Usenix Winter Conference Proceedings*, January 1995.

[68] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.

[69] William Simpson. *IP in IP Tunneling*, October 1995. RFC 1853.

[70] Michael St. Johns. *Identification Protocol*, February 1993. RFC 1413.

[71] Cliff Stoll. *The Cuckoo's Egg : Tracking a Spy Through the Maze of Computer Espionage*. Doubleday, 1989.

[72] Robert Vamosi. What Is Spyware? *ZDNet*, 2000. http://www.zdnet.com/zdhelp/stories/main/0,5594,2612053,00.html.

[73] Wim van Eck. Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? *Computers & Security*, 4:269–286, 1985.

[74] Verisign. http://www.verisign.com/.

[75] Vernor Vinge. *True Names*. Dell Books, 1981.

[76] VMWare, Inc. Welcome to VMWare, Inc. http://www.vmware.com/.

[77] Marc Waldman, Avi Rubin, and Lorrie Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proc. Ninth USENIX Security Symposium*, Denver, Colorado, 2000.

[78] Elizabeth Weise. Identity swapping makes privacy relative. *USA Today*, 2000. http://www.usatoday.com/life/cyber/ccarch/cceli017.htm.

[79] Wired News. RealNetworks in Real Trouble, 10 Nov 1999. http://www.wired.com/news/politics/0,1283,32459,00.html.