On the Anonymity of Anonymity Systems

Andrei Serjantov

University of Cambridge
Computer Laboratory
Queens' College

March 2004

This dissertation is submitted for
the degree of Doctor of Philosophy

Publication history:
Submitted 18 March, 2004.
Revised 21 June, 2004.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

This dissertation does not exceed the regulation length of 60 000 words, including tables and footnotes.

To my grandfather, Nikolai Vassilievich Kirsanov, and to those who build technologies which "balance the equation".

Acknowledgements

First and foremost, I want to thank my supervisor Peter Sewell for keeping me on the right track through my time in Cambridge, and for all the good advice.

Secondly, I thank my colleagues and co-authors who contributed to the development of my thoughts in this field; read, understood and questioned my results; forced me to justify myself and ruthlessly destroyed some of my ideas. In particular they are George Danezis, Paul Syverson, Roger Dingledine, Claudia Diaz, Richard Newman and Ira Moskowitz. Thanks are also due to Richard Clayton who read and commented on this thesis (and almost all of my papers), and always remarked upon the practical aspects of my work. Quite aside from the technical issues, Keith Wansbrough helped *a lot* with LaTeX and huge numbers of practical matters, Steve Bishop was always fun to gossip with and the security group was an excellent place to discuss new ideas. I also gratefully acknowledge the support from EPSRC grant GRN24872, EC FET-GC project IST-2001-33234 PEPITO, and APPSEM 2.

A special thank you goes to my parents who have been patient and supportive during my entire time in Cambridge.

Work is only fun when the play is fun, and I have thoroughly enjoyed my 3 years of PhD thanks to my friends. I've had unforgettable climbing adventures with Daniele Varacca and John Peterson as well as the Cambridge University Mountaineering Club. Queens' college provided a social atmosphere, and, of course, without the academic support of Dr Walker through my undergraduate years, the PhD would not have been possible. My Russian friends were fantastic company on my frequent trips to Moscow and in Cambridge.

Above all, to those friends who for one reason or another have on this occasion to remain *anonymous*.

On the Anonymity of Anonymity Systems

Andrei Serjantov

Summary

Anonymity on the Internet is a property commonly identified with privacy of electronic communications. A number of different systems exist which claim to provide anonymous email and web browsing, but their effectiveness has hardly been evaluated in practice. In this thesis we focus on the anonymity properties of such systems. First, we show how the anonymity of anonymity systems can be quantified, pointing out flaws with existing metrics and proposing our own. In the process we distinguish the anonymity of a message and that of an anonymity system.

Secondly, we focus on the properties of building blocks of mix-based (email) anonymity systems, evaluating their resistance to powerful *blending* attacks, their delay, their anonymity under normal conditions and other properties. This leads us to methods of computing anonymity for a particular class of mixes – timed mixes – and a new *binomial mix*.

Next, we look at the anonymity of a message going through an entire anonymity system based on a mix network architecture. We construct a semantics of a network with threshold mixes, define the information observable by an attacker, and give a principled definition of the anonymity of a message going through such a network.

We then consider low latency connection-based anonymity systems, giving concrete attacks and describing methods of protection against them. In particular, we show that Peer-to-Peer anonymity systems provide less anonymity against the global passive adversary than ones based on a "classic" architecture.

Finally, we give an account of how anonymity can be used in censorship resistant systems. These are designed to provide availability of documents, while facing threats from a powerful adversary. We show how anonymity can be used to hide the identity of the servers where each of the documents are stored, thus making them harder to remove from the system.

Contents

Chapter 1

Introduction

> "*Praise the humanities, my boy. That'll make them think you are broadminded.*"
> — Winston Churchill's advice to R. V. Jones (who had just been appointed as Professor of Physics)

1.1 Privacy and Anonymity

Privacy is viewed by many, if not most, as a desirable feature of modern society. Naturally, there is no complete consensus as to the definition of privacy, although there are a few popular alternatives. Roger Clarke suggests that "Privacy is the interest that individuals have in sustaining a 'personal space', free from interference by other people and organisations" [Cla99]. An alternative definition from the Oxford English Dictionary is "Privacy is the the state or condition of being alone, undisturbed, or free from public attention, as a matter of choice or right; freedom from interference or intrusion". What will be of a particular interest to us is the notion of communications privacy: "Individuals claim an interest in being able to communicate among themselves, using various media, without routine monitoring of their communications by other persons or organisations. This includes what is sometimes referred to as 'interception privacy' " [Cla99].

In this thesis we will be looking at a modern communications infrastructure – the Internet – and examining how individuals can use it to communicate privately. Whenever one communicates using the Internet, one leaves "tracks" which identify where the message (packet) came from and was going to. To be more specific: every email message, every access to a web page, every file download contains information as to which IP address the email came from, or which IP address requested the web page, or downloaded the file. IP addresses are allocated to individuals or companies by their Internet Service Providers (ISP), who keep this information and thus ensure that it is possible to track every Internet communication down to an individual or at the very least an organisation[1]. Naturally, some IP addresses are more linkable to individuals than others; there are practical steps one can take to make oneself more or less traceable. Using an Internet cafe is a good example, though some countries require users in such establishments to present their identity cards or simply mount surveillance cameras on the premises. Whether one can become practically untraceable by taking steps to "cover one's tracks" is a subject of a PhD in its own right [Cla04]. Here, we are merely concerned with the situation of the average individual. His communication privacy is virtually non-existent: every packet travelling on the Internet contains its sender and receiver, any part of the communications infrastructure that it travels through (routers, switches, firewalls) can determine who is communicating to whom and (usually) what they are saying, hence enabling easy routine monitoring and surveillance of communications by governments, companies and even individuals in charge of parts of the infrastructure. It may be worthwhile to contrast this scenario with that of a person heckling from the back of a crowd, posting an anonymous letter or making a phone call from a telephone box, where privacy is more or less preserved.

And yet there are many cases when, on the Internet, communications privacy is not merely desirable, but essential. For instance, a web-based electronic survey connected with subjects such as AIDS, alcoholism, cancer, etc needs to make sure that even the very fact that an individual is responding to such a survey is kept secret from third parties. Failure to do so can be damaging to the individual (it may increase insurance premiums, cause price discrimination, social tension, etc). Currently, the fact that an individual has accessed the survey can be trivially obtained by a system administrator of the company the individual works for, the ISP, and in some cases even another user of the same network. The current solution is to trust the agency doing the survey to anonymize the results, and the communications infrastructure not to collect data about who is responding to the survey. In reality, the arrangement is often open to abuse: commonly results of surveys are sent in plaintext over the Internet and thus it is possible to see not only who responds to the survey, but also what their responses are.

Thus, using the Internet as a basic medium of communications means that, by default, privacy is destroyed. Protecting it should ensure:

- Secrecy of the contents of private communications, the inability of third parties

[1] Thus, some countries have taken the view that access logs (in particular, web server access logs) constitute personal information. Hence, data protection legislation requires that companies take due care of such information and, upon request, provide access to the individual who browsed the website.

to read what is being said in private.

- Anonymity of such communications, the inability of third parties to determine the participants of private communications.

Secrecy of communications is relatively easy to achieve by using one of a wide variety of tools (e.g. PGP [PGP]) employing well-known encryption techniques [RSA78, RSA02, AES01, DR02].

Anonymity of communications is independent from their secrecy: an email message may be encrypted, but it will be obvious that it is going from Alice to Bob; on the other hand, it may be possible to hide the fact that a message is going from Alice to Bob without hiding its contents: “Meet at 10am on Red Square”.

Hence, to even begin to develop an environment in which communications privacy can be preserved, there is a need for a communications infrastructure (on top of existing Internet protocols) which allows information to be transmitted from one party to another without revealing either their identity or the nature of the communication. In other words, we need to build an infrastructure to enable anonymous communications on the Internet. The infrastructure may comprise several application-specific *anonymous communication systems or anonymity systems*, e.g. for email and web browsing[2].

Anonymity systems, due to their various requirements and depending on user actions, will provide different degrees of anonymity. Working out precisely how much is the focus of this work.

The thesis that this dissertation aims to support is that *analysis of anonymity systems is feasible, necessary and somewhat practical.* A broader consequence of the work is that *systems providing different desirable degrees of anonymity on the Internet, each with an associated cost, are a practical option rather than a theoretical possibility.*

To analyse anonymity systems, we must first find an appropriate way of quantifying anonymity and then try to work out the anonymity in situations of interest. However, trying to calculate directly the level of anonymity of a user sending an email or retrieving a webpage via an anonymity system is far too difficult – such systems are very complex. Instead, we will build models of these systems along with ways to calculate anonymity from these models. In some cases this will enable us to calculate the anonymity analytically, in others this will prove too difficult and we will have to resort to simulations.

The anonymity of individuals using various system designs will necessarily depend on what the anonymity systems are designed to protect against, i.e. how powerful

[2]On top of an anonymous communications infrastructure we can build up an appropriate identification and authentication infrastructure as necessary.

the disturber, the intruder, or the interfering body may be. How should we decide who to design our anonymity system against? There are several possible approaches.

The first approach is to decide which kinds of intruders we want to protect the users against and how much anonymity they should have first, and then design and build anonymous infrastructure keeping this in mind. Our analysis techniques should then enable us to show that the systems do indeed meet the requirements. The other approach is to investigate how much anonymity *it is possible to provide users with* first, and make a decision as to how much anonymity to actually provide users with later. In this thesis we follow the second approach: we will design anonymity systems which are as strong as possible, and thus resist very powerful attackers. Having investigated stronger attack models, we will be in a better position to design and build systems tailored to specific anonymity requirements when such a need arises.

It is appropriate to note at this point that privacy conflicts with the ability of law enforcement agencies to obtain evidence of communications of suspects who may or may not have committed a crime. So how much privacy should users have? This is an ongoing debate, to which we wish to make no contributions in this work. We merely note the fact that it may be possible to modify anonymity systems to provide anonymity which is revocable on a case by case basis [CDG$^+$]. Thus the ideas presented in this thesis do not depend on the outcome of the privacy vs law enforcement debate.

So far we have considered generic user privacy as the main motivation for the design, development and deployment of anonymity systems. In the next section, we look at the motivation behind providing anonymity to users in more detail, keeping the generic "communications privacy" idea in mind.

1.2 Uses of Anonymity

1.2.1 Web Browsing and Email

The most popular uses of the Internet over the last few years have been email and web browsing. Therefore, to ensure communications privacy, we should build an anonymity infrastructure to enable users to perform these activities free from intrusion by various attackers.

We have already illustrated the motivation for anonymous web browsing with the electronic survey example in the previous section. There are many other similar scenarios – browsing websites about specific illnesses or disorders, reading political opinions, etc.

Turning to anonymous email, we claim that just as in the past correspondence between individuals by mail has been free from routine surveillance both in terms of content and participants, so should be correspondence by email. This may include emails about families, work, the advantages and disadvantages of products or services provided by various companies, political speech and other subjects which are potentially of interest to parties which are not involved in communication.

Another important application of anonymous email is whistle blowing. Essentially, this is the ability of an individual to report a wrongdoing and thus expose an irregularity or a crime, often by a person in charge or simply someone with authority over the whistle blower. Clearly, one way to enable whistle blowing is to have a very secure (email) anonymous communications system deployed, thus enabling anyone to send a message relatively easily without fear of being tracked.

Finally, both anonymous email and anonymous web browsing can be used to distribute news and hold online discussions without fear of surveillance in countries with repressive regimes.

1.2.2 Electronic Voting

Electronic voting is often viewed as a good application of anonymity. One fundamental requirement of an electronic voting system is to ensure that it is not possible to determine who votes for whom. Another is is to ensure that the votes are 'receipt-free', i.e. the voter is unable to prove to a third party which way they voted. This is necessary to prevent votes from being sold. E-voting is a very contentious area with some experts believing that it is currently technically infeasible to provide a secure system with strong anonymity properties in order to guarantee that an election is conducted fairly. Certainly the vast majority of the deployed systems have major shortcomings in their security, user interfaces, or reliability [Mer03, KSRW03]. Nevertheless, if electronic voting systems are ever to be deployed, they will probably need to use the same techniques as anonymity systems.

1.2.3 Censorship Resistance

Censorship resistance is the ability to publish a document on a system which ensures that it will be available for a long time, despite powerful adversaries trying to prevent its distribution. Anonymity is a useful tool in censorship resistant systems. It enables publishing to be done anonymously, so the author cannot be tracked (thus removing the fear element which often discourages people from posting controversial documents). Even more importantly, it prevents the machines which store a file from knowing what the file they are storing is, thus removing potential burden of filtering that may be placed on them by organisations. We explore this issue in more detail in Chapter 8.

1.3 Structure of the Thesis

Having shown the motivation for anonymous communication infrastructures in the previous section, we will first examine the basic principles behind achieving anonymity. In Chapter 2 we introduce the scheme commonly used to build strongly anonymous communication systems, basic concepts such as mixes and onion encryption, and discuss the different architectures of anonymity systems. We then distinguish two basic types of anonymity systems – (email) message-based systems and real-time connection-based systems. We look at existing designs and the shortcomings of previous work; and set the scene for our research.

We then go on to examine the problem of measuring anonymity (Chapter 3), and by way of example show how the anonymity of two mixes can be compared.

In Chapter 4 we turn our attention to message-based systems. We look at individual mixes – the building blocks of anonymity systems – focusing in particular on how resistant these are to the powerful blending attacks. These are active attacks which involve the adversary flooding the mix with his own messages in order to single out a message he wants to trace and thus deduce its destination.

In the next chapter, we continue analysis of single mixes, and devise a framework in which they can be expressed and compared and propose yet another mix, the binomial mix, with better properties.

Analysing single mixes is useful, but a strong anonymity system should consist of several mixes. The mixes can be combined in various ways. One of these is a mix network. In Chapter 6 we look in detail at how one would go about analysing an anonymity system based on a mix network architecture and provide a formal definition of the anonymity of a message going through such a system.

Next we look at connection-based systems, the attacks on them and how they can be protected against without using dummy traffic. In particular, we look at the attacks which a passive adversary can mount.

Finally, we look at how anonymity systems can be used to improve censorship resistant systems.

The meaning of various variables in Chapters 3-7 of the thesis is summarised in Appendix A.

1.4 Contributions of the Thesis

In this work we strive to introduce the idea of quantitative probabilistic analysis of anonymity systems. In the process, we make the following contributions:

- First, we look at the concept of anonymity set – the metric used in most previous analyses of anonymity systems – and show that it is inadequate for expressing the anonymity of a large class of real anonymity systems. We propose our own metric and show how it can be used in analysing single mixes – the building blocks of anonymity systems.

- We give an account of the $n-1$ attack which can be mounted by a powerful adversary and was thought to defeat anonymity systems. We show how this attack can be made much more observable and expensive by using more sophisticated mixes. This result quantitatively supports the intuition of the designers of the Mixmaster network.

- We present a framework in which existing mixes can be expressed, and design a new mix which is more resistant to blending attacks.

- We present methods of anonymity analysis for a variety of single mixes.

- We tackle the long-standing problem of choosing the best architecture – cascade or network – for an anonymity system. While mix cascades are easy to build and analyse, they scale badly and are more easily subjected to $n-1$ attacks. Mix networks, on the other hand, are hard to analyse, but were thought to have better properties (and scale well). We define the anonymity of a message travelling through a mix network and thus make substantial progress towards solving the mix cascade vs mix network problem.

- We also look at connection-based systems, which are designed to provide low-latency anonymous communication. Some of these can be used for everyday tasks such as web browsing [DMS04, FM02], while others are more specialised; addressing anonymous file sharing [BG03]. There are several implementations of these, with the most popular of these, JAP, having tens of thousands of users. Astonishingly, virtually none of these (except, perhaps, Crowds [RR97, WALS02, Shm03]) have a rigorous analysis of the anonymity they provide, with some (GNUnet) lacking even a consistent description of the system itself. We refine the standard threat model (this has been long overdue!) and present very simple attacks which can be used against most of the existing systems and propose ways of protecting against them. Unlike the relatively mature area of analysis of message-based anonymity systems, connection-based systems are very much in need of analysis, and we merely scratch the surface here.

- Finally, we look at censorship resistant systems and show an anonymity system can be used to help design a Peer-to-Peer architecture which protects against rubber-hose cryptanalysis, previously thought to be hard or impossible.

1.5 Publication History

The vast majority of the material presented in this document has been published in proceedings of peer-reviewed workshops or conferences. Some of Chapter 3 is based on the paper *Towards an Information Theoretic Metric for Anonymity*, [SD02] co-authored with George Danezis. This paper won the 2002 Award for Outstanding Paper in the field of Privacy Enhancing Technologies. The material of Chapter 4 is a selection of the work presented at the 5th Information Hiding Workshop in the paper *From a Trickle to a Flood: Active Attacks on Several Mix Types* [SDS02] co-authored with Roger Dingledine and Paul Syverson and at the Workshop on Privacy and Anonymity Issues in Networked and Distributed Systems *On the Anonymity of Timed Pool Mixes* [SN03] with Richard E. Newman. Chapter 5 is based on work published at PET 2003 *Generalising Mixes* [DS03b] with Claudia Diaz. Work on the anonymity of mix networks is yet to be submitted at the time of writing. Our thoughts on connection-based systems of Chapter 7 were published at ESORICS 2003 *Passive Attack Analysis for Connection-Based Anonymity Systems* [SS03] co-authored with Peter Sewell, and the work on censorship resistance was presented at the International Workshop on Peer to Peer Systems, 2002 *Anonymizing Censorship Resistant Systems* [Ser02].

Naturally, all the material presented is the author's contribution towards these publications or material which has been discovered independently by more than one person.

Not all the work of the author during the 3 years at Cambridge appears in this thesis – some was not related to analysis of anonymity systems [SSW01a, SSW01b, WNSS02, SL03], while other work [NMSS03] was too collaborative. The thesis was written as of October 2003, and does not include references to work (both my own and by other people) carried out since.

Chapter 2

The Basics of Anonymity Systems

"*ZHQM ZMGM ZMFM*[1]"

— G. Julius Caesar

In this chapter we introduce some basic mechanisms designed to provide anonymity on the Internet, looking at threat models and basic terminology. We leave the technical definitions of anonymity until Chapter 3, where we will have the chance to examine them in more detail.

2.1 Background and Historical Developments

To design effective anonymity systems we need to consider how real users wish to communicate on the Internet. The two most popular activities of users on the Internet are email and web browsing. Thus, research in the anonymity community has focused upon how to design, build and deploy systems which enable anonymous email and anonymous web browsing. It is important to notice that whatever the way the users are communicating, they may well be concerned about hiding different aspects of their communication patterns, all of which would fall under the general term anonymity.

First of all, users might wish to be undetectable when they *send* messages (or initiate communications) through the anonymity system. Alternatively, they might be very

[1] W and J were added to the Latin aphabet in the middle ages. Although there is some uncertainty as to when the letters Y and Z were introduced into the Latin alphabet, it has been previously suggested that they were already present at the time of Caesar[And].

worried about being observed to *receive* information from within the system. Finally, they might be quite happy about being observed to send information and receive information as long as it is hard to tell who the other communicating party is in either case.

The natural question to ask now is who they might want this information hidden from. The attacker could be local – observing the network connection between the user and the Internet, larger – the user's ISP, observing some part of the network, including parts of the anonymity system or global – observing the entire anonymity system and all the communications between the users and it. In addition, a powerful attacker would probably compromise parts of the anonymity system to try to deanonymize its users. Finally, an attacker can be actively trying to compromise anonymity either by modifying network traffic (and thus potentially being observable) or passively, offline, having previously captured the data required.

The development of anonymity systems started with a paper by Chaum in 1981 proposing a scheme for untraceable electronic mail [Cha81]. This is the basis of most modern anonymity systems, as we will discuss in detail below. Seven years later, Chaum proposed a protocol (Dining Cryptographers) for anonymous broadcast which was designed to hide the sender of the message [Cha88]. Pfitzmann and Pfitzmann made substantial improvements to it in 1990 [PP90]. The next significant development was a proposed scheme to anonymize telephone calls based on the (at the time) new ISDN infrastructure – ISDN mixes [PPW91]. It identifies many of the important and difficult problems of the field of anonymity systems and addresses them in the context of circuit-switched networks.

In the early 1990's the first practical email anonymity systems inspired by Chaum's work started appearing – the Type I remailers. These were vulnerable to a number of attacks (broadly, the ones described in Section 2.3.3), which was fixed in the Type II remailer design [MCPS03, Cot94]. The network of Type II remailers, Mixmaster [Mix], is operational and handles tens of thousands of emails per day. This system, however, lacks the ability to reply to anonymous email. After all, if the user receives an *anonymous* email, he does not know who to reply to. The scheme for anonymous reply addresses dates back to the original Chaum paper [Cha81], but has proved hard to implement securely. Indeed, the first system to do so is the recent Mixminion, [DDM03].

The first developments in anonymous web browsing began with the Onion Routing project [OR], although simple HTTP proxies appeared around the same time. In recent years, many projects have appeared, most notably JAP [JAP], Tarzan [FM02] and MorphMix [RP02] (the last two are based on a Peer-to-Peer design). More recently, the field has turned to robust implementation and quantitative (often probabilistic) analysis of anonymity systems. The Onion Routing and JAP projects are good examples of the former, while a series of papers by Danezis [Dan03b, Dan03c], Diaz [DSCP02], Shmatikov [Shm03], Wright [WALS02, WALS03] and myself are representative examples of the latter.

2.2 Terminology

Before describing any concrete ways of achieving anonymous communication, we introduce by example some standard terms which are useful in describing anonymity systems. We broadly follow the proposal of terminology by Pfitzmann and Köhntopp [PK00].

Let us first consider a multi-party asynchronous communications protocol in which participants send messages to each other. Some of these messages are *real*, i.e. contain information which the *sender* of a message wanted to transmit to the *receiver* of it. The aim of the *attacker* or the *adversary* is to discover senders and receivers of these messages[2]. Other messages, called *dummy messages* (or just *dummies*), do not contain information and are just sent to confuse the attacker. After some time (a run of the protocol), some real messages were sent by the senders and received by the receivers. The protocol provides *sender untraceability (unobservability)* if the adversary is not able to determine whether a particular participant has sent any real messages or not (see Figure 2.1). On the other hand, if the attacker is unable to determine whether a particular participant has received any real messages or not, the protocol provides *receiver untraceability* (see Figure 2.2). A very strong protocol might provide both sender and receiver untraceability.

Finally, if the set of senders and the set of receivers is known, but the attacker cannot determine which of the senders sent messages to which of the receivers, the protocol provides *unlinkability.* Note that unlinkability is weaker than both sender untraceability and receiver untraceability.

The above definitions apply to synchronous protocols as well as asynchronous ones. In the synchronous case, messages correspond to the communications, senders to initiators of the communications and receivers to responders.

In this thesis we will be dealing mainly with systems which provide unlinkability. A diagram of such a system is shown in Figure 2.3.

There are two styles of attack that the adversary can perform on such systems (see Figure 2.4). Broadly, a *traffic confirmation attack* is one in which the adversary only examines the communication patterns from senders to the system and from the system to the receivers. A *traffic analysis attack*, on the other hand, also takes into

[2] Of course, sophisticated attackers may not need to discover this with certainty, they may be content with information about the correlation between senders and receivers.

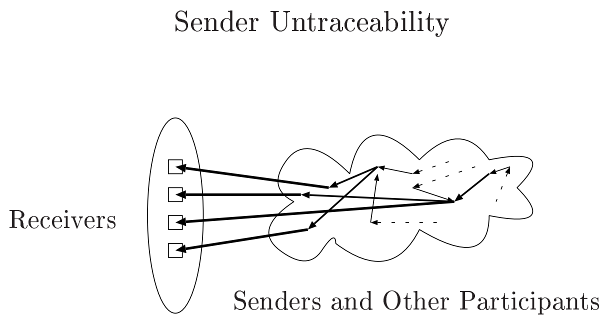


Figure 2.1: A system which provides only *sender untraceability* as seen by the adversary.

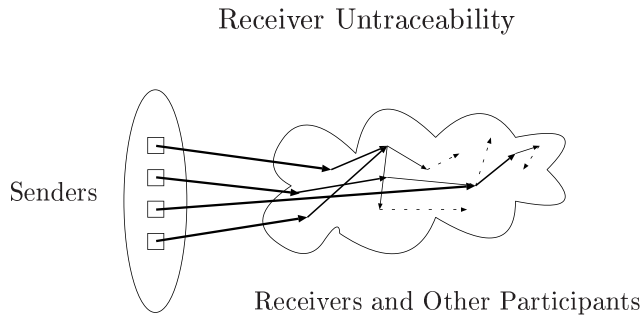


Figure 2.2: A system which provides only *receiver untraceability* as seen by the adversary.

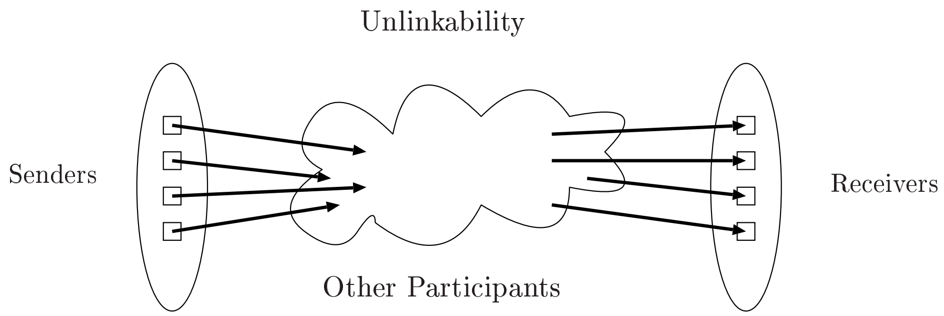


Figure 2.3: A system which provides *unlinkability* as seen by the adversary.

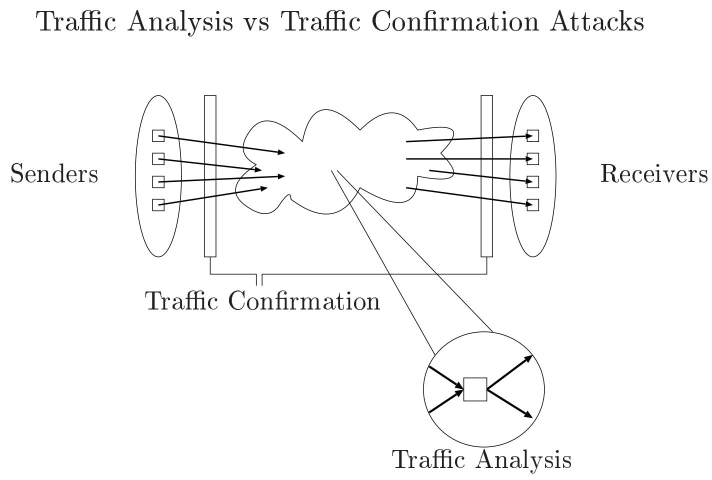


Figure 2.4: A traffic confirmation attack examines only the communications between the senders and the anonymity system and the anonymity system and the receivers. A traffic analysis attack also makes use of traffic patterns within the anonymity system.

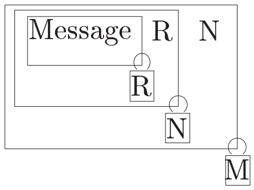
account the traffic inside the anonymity system. The latter style of attack is usually more powerful but requires more computational resources as well as more data.

Having equipped ourselves with some terminology, let us look at, perhaps, the most practical way of achieving anonymity (more specifically unlinkability) – mix systems.

2.3 Unlinkability via Mix Systems

As we already mentioned, the field of anonymity was started with a classic paper by Chaum [Cha81] which introduces the concept of a mix system and explains how a message (think of an email message) can be sent from a sender to a recipient via several proxies, thereby providing unlinkability against an attacker who watches the entire network.

The following may serve as a useful analogy for how these systems operate. Sarah, (the sender) having written her letter, puts it into an envelope and addresses it to Richard (the receiver). She then puts it in an envelope addressed to a friend Nick with a note asking him to post the envelope. She then takes the envelope addressed to Nick, and puts it in an envelope addressed to Mike, with a similar note. Finally, she sends the (by now large) envelope to Mike. Mike and Nick work at two different post offices and get many letters asking them to forward inner envelopes every day. They receive all the letters addressed to them at the post office, take them home, unwrap all the envelopes during the night, and post all the inner envelopes the next day. Thus even Amy, the boss at the central post office who records the sender and



$$\{\{Message\}_{k_R}, R\}_{k_N}, N\}_{k_M}$$

$$|M\#k_R|R\#k_N|N\#k_M$$

Figure 2.5: "Onion encryption" shown diagrammatically and in two standard notations

recipient of every letter (but cannot see what Mike and Nick do at home) cannot tell that Sarah sent her letter to Richard.

We are not aware of a permanent post office processing general purpose mail which operates in this way in the real world, though apparently the town of Loveland, Colorado performs remailing services similar to the ones described above for Valentine's day every year! In any case, the above description corresponds fairly well to how a mix system works. This is explained in more detail below.

The system which provides an anonymous email service consists of a number of *remailers* or *mixes*. A sender S wishing to send an anonymous message to a receiver R decides on a sequence (e.g. $[M, N]$) of anonymous remailers (mixes) via which the message should travel. Each remailer has a public/private keypair[3]; the public keys are retrieved by the senders in advance. The sender prepares the message by padding it to B bytes – a size used by all senders – and, if R has a public key, encrypting the padded message with it. She now appends the address of R, encrypts with N's public key, appends the address of N, encrypts with M's public key, and finally sends the whole structure to M. The structure, now commonly called an *onion* (the name is apparently due to Syverson [GRS96]) is illustrated in several different notations from [Cla03] in Figure 2.5.

When the onion is sent to the first mix (run by Mike), M, it is decrypted, and placed inside the mix. A *flushing algorithm* is then executed to decide whether to send on (some of) the messages inside the mix. The simplest example of a flushing algorithm is waiting until n messages are in the mix before reordering them and sending them all out to their next hop. A mix executing such an algorithm is called a *threshold mix* (we will go on to investigate other flushing algorithms in Chapters 4 and 5). When the message does leave the mix, it will be forwarded to the next mix (N, run by

[3]The reader who is not familiar with public key encryption, is advised to refer to a standard textbook on computer security, e.g. [And01].

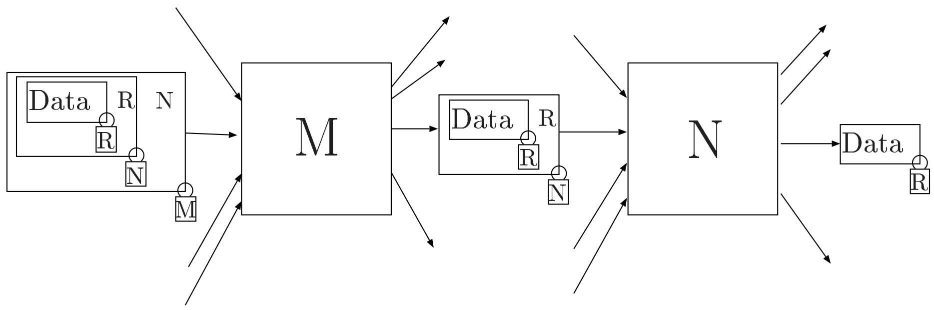


Figure 2.6: A message travelling through anonymous remailers

Nick), and so on backwards along the sequence of mixes chosen by the sender (in our example just $[M, N]$), and eventually to the receiver R. This is illustrated in Figure 2.6. Note that when passing through every mix, as far as the attacker can see, the message was "confused" with several (n in the case of the threshold mix) others. Of course, if the mix is *compromised* by the attacker, i.e. the attacker has the private key of the mix, no "confusion" is created.

This is more or less the scheme proposed by Chaum in [Cha81]. Alternative schemes, are examined in Section 2.6.

2.3.1 Basic Properties of Chaum's Construction

We now give an account of the basic properties of Chaum's construction.

1. First of all, it provides *bitwise unlinkability between inputs and and outputs of a mix.* By this we mean that an attacker who is watching the input and the output channels of a mix (but does not know its private key) cannot correlate incoming and outgoing messages just by looking at the bits which comprise the messages.

2. Secondly, it provides *bitwise unlinkability against any pair of non-consecutive (in the sequence) collaborating mixes.* Thus, if the first and the last mix decide to collaborate, they cannot tell which of the messages processed by the first mix corresponds to which of the messages processed by the last just by looking at the bit sequences comprising the messages.

It is interesting to note that the onion construction is not necessary to provide bitwise unlinkability between inputs and outputs of a mix. Instead, Sarah (the sender) could simply encrypt the message with a key she shares with Mike (the first mix). Mike, upon receiving the message, decrypts it and encrypts it with a key he shares with

Nick, etc until the message arrives at Richard[4]. The reason why onion encryption is used in mix systems is to provide bitwise unlinkability against collaborating mixes.

It should be noted that the scheme provides bitwise unlinkability against active attackers as well as passive ones. Should an attacker try to change the message (and propagate this change through some mixes), in an attempt to discover where this message goes, anonymity should be preserved. This is done automatically by the mixes in the scheme described above: if the message was changed in transit, the mix which receives it will not be able to decrypt it correctly and, thus, will have no option but to discard it. Hence unlinkability is not compromised. In real anonymity systems, much more care has to be taken to defend against these attacks [DDM03] as there, for efficiency reasons, only the header of the message is encrypted with the public key.

2.3.2 Threat Models

Now that we have seen something of how mix systems work, let us examine the threat models against which they are designed to protect.

There are several main threat models that anonymity researchers consider. Some of them will be considered in more detail later in this thesis.

1. **The global passive attacker.** This is, perhaps, the most common threat model in the literature. The adversary is able to observe (but not modify) all network traffic, and is unable to see inside any of the mixes.
2. **The global active attacker.** This adversary is able to observe and modify all network traffic. In particular, he is able to inject an arbitrary amount of traffic into the system in a very short time and delay traffic for an arbitrary length of time.
3. **The global passive attacker with many compromised mixes.** This is a very strong attacker model used by, for instance, Berthold, Pfitzmann and Standtke in [BPS00]. The only requirement is that there is at least one honest (uncompromised) mix on the path of the message in question[5]. Recall that by "compromised" we mean that the attacker knows the private key of the mix or can otherwise determine the correspondence between the incoming and the outgoing messages of the mix. If the attacker is the superuser on the machine running the mix, he is an active attacker.

[4]The mixes need not see the message itself either – Sarah could encrypt it with Richard's public key.

[5]The serious problem with this threat model is that it relies upon the user "magically" choosing a route through at least one honest mix, or suffering the consequences if all the chosen mixes are compromised.

4. **The global active attacker with many compromised mixes.** A combination of the latter two threat models.

5. **A sub-global attacker** This is a large class of attackers that have the ability to monitor some links in the anonymity system, and possibly have some of their own nodes forwarding traffic. All the real attackers almost certainly fall into this category. However, it is difficult to pin down precisely what a real attacker might look like within this class.

At this point it is appropriate to introduce some more terminology. The adversary is sometimes interested in compromising the anonymity (unlinkability) of a single message (e.g. the message from sender to the first mix or from the last mix to the receiver); we call it the *target message*. When we say "compromising unlinkability" we mean determining (or at least establishing some correlation between) the sender and the receiver of this message. We leave a detailed account of quantifying anonymity till Chapter 3.

In addition to the capabilities described above, each attacker may have some additional knowledge about what happens in the system. For instance, if we consider some run of a system, he might also know facts like "a message which arrived at R (hopefully this was not the target message!) was sent by A" or, "a message which arrived at R was not sent by B or C". Such knowledge might arise, for example, if the message to R was written in Mongolian and we definitely knew that neither B nor C had learnt Mongolian.

The basic construction described in Section 2.3 already provides unlinkability against the global passive attacker. However, we need to understand the scheme in more detail to consider whether it protects against some of the more powerful adversaries.

2.3.3 More on Mix Systems

In just giving Chaum's basic construction we omitted many details which are important to ensure that the system provides anonymity and some choices available to designers of anonymity systems. We will now briefly mention some of these.

Let us first of all recall that Sarah's message travelled through Mike's and Nick's remailers before reaching Richard. How were these remailers chosen? This is one of the most important questions in the design of anonymity systems, and various systems have dealt with it differently. Some do not allow any choice of routes at all. All messages have to travel via the same sequence of mixes (route). Such a system is called a *mix cascade*. A *mix network* on the other hand, allows users to choose their routes through the network. A *free route mix network* allows users to choose arbitrary routes (via any sequence of available nodes) while in a *restricted route mix* *network*, users are somewhat constrained.

This design choice has many implications for the rest of the anonymity system. Note, for instance, that messages will decrease in size when they travel through a mix. Thus, in the mix network as presented above, messages would only be confused with other messages of the same size. More precisely, message μ would only be confused with message μ' at mix M if μ and μ' are to pass through the same number of mixes after M. To ensure that *all* messages passing through a mix get mixed together, we need each mix to add a certain amount of padding (random bits) to make all messages *the same size*. Naturally, all the contents of messages must also be the same size as we mentioned previously.

In a mix network, we must also make sure that mixes do not know their position within the sequence the message is travelling through. Otherwise, a strong attack (by the global passive adversary with many compromised mixes) is possible. See [BPS00] for details.

Yet another attack is based on replaying messages. A global active adversary wants to trace a message travelling through a mix. To do this, he first records all the incoming and outgoing messages to this mix. He then takes the incoming message he wants to trace and sends it to the mix again. The mix flushes, and then if the attacker compares the outgoing messages from the mix to the ones he has recorded before, he will get exactly one match he knows that this is the reinserted message. He can now see where this message is going. Protection against this attack is easily possible by including timestamps in messages and storing details (more precisely cryptographic hashes) of the messages which have been seen in the last few days. Indeed modern mixes (e.g. Mixminion, [DDM03]) ensure that any replayed messages will be discarded automatically.

2.3.4 Cascades vs Networks

As we have seen in the previous section, the architecture of a mix system affects other choices in the design. Here we examine the properties of mix networks and mix cascades slightly more deeply in order to motivate some of the work presented in the later chapters of the thesis.

Mix cascades These are relatively easy to analyse. For instance, if a cascade is made up of threshold mixes with threshold n, it is clear that each message is mixed with $n-1$ others. It is also clear that if one of the mixes in a cascade stops working, so does the entire cascade. Finally, we note that the scalability of a mix cascade is entirely limited by the scalability of the machines on which the mix software is run. On the other hand, a mix cascade maintains a level of anonymity against *a* *global passive attacker with many compromised mixes*[6] [BPS00]. In other words, as long as one mix remains uncompromised, some anonymity is achieved. Indeed, if we consider a mix cascade made up of threshold mixes with threshold n, then even if all the mixes but one are compromised, the anonymity provided by the cascade is not reduced[7] .

Mix networks Mix networks are much harder to analyse. First of all, their properties are heavily dependent on how routes are chosen. This is often done by the users, which are hard to model. Nevertheless, it is clear that the scalability and reliability of mix networks are better than that of cascades, though these properties have not been rigorously quantified or compared. Finally, quantifying the anonymity that a mix network provides has proved elusive for several years. In this thesis we make considerable progress towards this goal. An efficient way of calculating the anonymity of a mix network would enable us to look at properties of mix cascades and mix networks and make a detailed comparison. It has generally been considered that mix networks are *secure against the global passive attacker*; though it has been shown in [BPS00] that they are *not secure against the global passive attacker with many compromised mixes*. The reader may find the above statements odd – what is "secure" in the context of an anonymity system? What we mean by "an anonymity system is secure against X" is that the adversary X does not significantly reduce the anonymity of the system from that which it was designed to provide. Or, more simply (though more subjectively), a significant amount of anonymity is maintained against threat model X.

2.3.5 The $(n-1)$ Attack

One strong attack, which is still subject of ongoing research, is the $n-1$ attack. Consider a threshold mix which flushes when n messages have arrived. If, of the n messages in the mix, $n-1$ come from the attacker (and are therefore recognisable to him at the output), he can deduce the remaining message and observe where it has gone to. This is a very powerful attack; one can seemingly do little to protect against it. Indeed, most methods try to reduce it to a denial of service attack in which the system stops working, without anonymity being compromised. Alternatively, one can try to make the attacker as observable as possible. We explore this subject in much more detail in Chapter 4.

[6] As long as no $n-1$ attacks (see Section 2.3.5) are possible. Methods of protecting cascades against this attack do exist, see, for instance, the first part of the Stop-and-Go mixes paper [KEB98].

[7] Although we have not examined this in detail, it is likely that the anonymity provided by a cascade of more sophisticated mixes, e.g. pool mixes, does go down as the number of compromised mixes goes up. Of course, it never goes down below the anonymity of a single pool mix.

At the very beginning of this chapter we mentioned that to design effective anonymity systems we need to tailor them to the particular method of communication. Thus, having now examined the basics of mix systems, we look at both email and web browsing anonymity systems in more detail.

2.4 Message-based Systems for Anonymous Email

There are three characteristics of email traffic which are important for anonymizing it:

1. **High Latency.** It is likely that users would tolerate a delay of minutes, hours, and sometimes even a few days if strong anonymity guarantees are provided to them. This is crucial for anonymity because when a message passes through a mix, a delay is introduced. As we shall see in Chapter 4, the more delay, the more anonymity.

2. **Narrow size distribution.** To a first approximation, all email messages are of a similar size. Typically, they are a few kilobytes long, for instance, around 1.1% of the 8130 emails in my inbox at the time of writing were bigger than 100K. Hence, a typical email will fit into one Mixmaster or Mixminion packet (if it does not, strong intersection attacks are possible [KAP02]).

   This is very important for anonymity because all messages need to be made the same size to avoid being trivially recognised within the system.

3. **Low bandwidth.** It is useful to observe that although email is very important in everyday life, the bandwidth used up in transporting it is very low compared to other forms of traffic. The relevance of this to anonymity is that sending dummy messages (which must be of a similar size to real messages) is *cheap*.

The second property gives rise to the term under which we use to denote email anonymity systems: *message-based systems*.

Another crucial property that message-based anonymity systems have to satisfy is that of usability. While senders of email who wish to preserve their anonymity might be happy to download and run special software, people who might receive such email are unlikely to do so[8]. If this is the case then anonymous email can be sent to anybody without requiring them to go to any extra effort to read it. A remark is in order here. Back in Chapter 1, we talked about hiding the content of the message as opposed to hiding the sender and the receiver of the message. A message-based anonymity system can be used to send a message to someone anonymously (without

[8] To hide the contents of the traffic from the observer when the message goes from the last mix to the receiver, the latter will need basic decryption capabilities.

requiring them installing special software) only if the content of the message itself does not reveal the sender and the receiver of the message. Alternatively, the content may be hidden by encrypting the message from the sender to a receiver using either symmetric or public key cryptography.

We have already mentioned that message-based anonymity systems work much along the lines of Chaum's original scheme. We do not go into any further details of the protocols of any of the remailer networks – what we have described about mix systems is sufficient for the purposes of this thesis. For more details about protocols and implementations of message-based systems see [MCPS03, DDM03].

2.4.1 Open Problems

We have already mentioned some open problems in this area when we were examining the theory of mix systems in Section 2.3. These include:

- Dealing effectively with the $n-1$ attack. Our approach to achieving this is presented in Chapter 4. We evaluate several mixes with respect to their resilience to the $n-1$ attack. Some are likely to provide enough for the attack to be detected (and hence the anonymity not compromised). Other methods of dealing with this attack also exist [KEB98], which have not yet been analysed in a satisfactory way. There is also some recent work on dealing with the attack using active defences (i.e. dummy traffic) [DS03a].

- Analysing and comparing the effectiveness of single mixes. We do this for active attackers in Chapter 4 and more generally in Chapters 3 and 5. As always, some open questions still remain.

- Analysing the anonymity of a mix network and comparing it to that of a mix cascade. This is a difficult problem; we make a substantial contribution towards solving it in Chapter 6 by presenting a definition of the anonymity of a message in a run of the network.

- Dealing with the issue of repeated communication. If the attacker knows that Alice is communicating with a fixed set of receivers over a long period of time, he can try to deduce this set. Such activity is generally referred to as *intersection attacks*. There is a substantial body of work on this issue; [KBS02, KAP02, AKP03, Dan03c]; we do not deal with this problem in the thesis.

2.5 Anonymous Web Browsing

Web browsing imposes a much stricter set of requirements on the anonymity system:

1. **Low Latency.** These systems must ensure that retrieving a webpage anonymously takes no longer than a few seconds. Otherwise, anonymous web browsing becomes unusable.

2. **Large size distribution.** The files which users commonly wish to download from the Internet can be anywhere between several kilobytes (small text files) and tens or even hundreds of megabytes (images and video). Note that fragmenting large files into small packets and reassembling them at the other end enables strong intersection attacks [KAP02].

3. **High bandwidth.** A user browsing the web uses up much more bandwidth than the one who sends and receives email. Hence using dummy traffic to help create anonymity is in general *expensive*: for instance, making a dummy request for a retrieving a file might result in a download of several megabytes. One way of limiting bandwidth costs is to make a closed system for a known set of users, but this, of course, also severely limits anonymity.

The usability requirement for anonymous web browsing systems is similar to that of the message-based system: a user (who downloads client software) must be able to browse material on existing web servers who are unaware of the anonymity system.

The reader should by now have the impression that these systems face strict requirements. Thus, we might expect them to be able to withstand only significantly weaker threat models and provide less anonymity than message-based systems. Indeed this is the case. It is much less clear how to build an anonymity system for web browsing which satisfies all of the above requirements. Nevertheless, some attempts have been made using Chaum's scheme as a broad basis.

A very broad description of a low-latency anonymous communication system is as follows. A number of *users*, running anonymity clients, connect through the system to some *web servers* (not running special software). HTTP requests and responses both travel through the system. The system itself consists of a number of *nodes*. Some designs have a 'classic' architecture, with relatively few nodes, whereas others have a 'Peer-to-Peer' architecture, with nodes run by each user. Each node has logical *links* to some (not necessarily all) other nodes, along which it forwards traffic. Links are implemented above inter-node TCP connections between IP/port addresses, link-encrypted. Nodes also accept connections from end-user clients. To protect against node compromise, data from each of the clients passes through several nodes before it reaches the web server. The first design choice made in all systems that we are aware of (Onion Routing, [OR], Tarzan [FM02], MorphMix, [RP02] and JAP [JAP]) was to make such systems *connection-based*. This means that the client establishes a bi-directional connection through the system and all the data travels (up and) down the same path. Another common feature of such connection-based systems is that they split up the traffic into small (from 128 bytes to 1K) packets. Note that such systems are not only useful for web browsing, but for any protocol running over TCP. Indeed, IRC and ssh are also possible (though the anonymity of a user running such protocols is likely to be low – it is likely that effective application-protocol specific attacks can be mounted).

While Onion Routing, Tarzan and MorphMix are based on a mix network architecture, JAP is a cascade. However, as we shall see in the next section, both classes of systems admit similar attacks.

2.5.1 Attacks and Open Problems

We will now briefly outline some well-known attacks on connection-based anonymity systems. These are presented here to give the reader the flavour of the problems with connection-based systems. The first of these attacks can be dealt with relatively easily; the others are strong attacks which are hard to protect against. We do not explore them further in the thesis. Instead, in Chapter 7, we look at two relatively simple attacks which defeat current systems and examine ways of protecting against them.

Suppose the adversary is a global passive attacker and the anonymity system is a free route network. If the attacker watches all the incoming (from users to nodes) and outgoing (from nodes to webservers) connections of the anonymity system, then because webpages vary so markedly in size, the number of bytes transferred is likely to be sufficient to identify which connection is which. Note that this is a traffic confirmation attack; it only requires observing the senders and the receivers rather than the entire anonymity system.

One of the ways anonymity systems can try to protect against this attack is by employing traffic padding from the client software to the first node on the route. What the best kind of padding to apply here is currently an open issue, as is the protection that such schemes can provide. We do not address this further in this thesis.

A global active adversary can also mount the following attack. He targets a user, wanting to find out which website he is accessing via the anonymity system. He then injects a 'timing signature' into the traffic travelling from the user to the first node, i.e. introduces a series of delays into the stream which he hopes to recognise when this data travels from the last node to the web server. He does the same (though the other way) when the website data starts travelling back to the user. Note that this is also a traffic confirmation attack (though the adversary is active).

Finally, an adversary who owns the first and the last node of a connection through the anonymity system, can, by simply watching the total volume of data passing through a connection, correlate the start and the end times of the connection. This is very similar to the first attack we discussed, though, unlike that attack, it cannot be protected by dummy traffic from user to the first node. Indeed, more generally, we note that an attacker who owns two compromised nodes may use time and traffic volume information to establish whether a connection passes through both these nodes. This is in sharp contrast to message-based systems, where such an attack is not possible (all messages are the same size).

On a rather different note, connection-based anonymity systems are much more expensive to run than message-based ones, and will hence be harder to deploy. While there were enough volunteers who devote their spare time to running remailers, this may not be the case with connection-based systems. Hence, an open problem is looking at ways to encourage people to run nodes, behave honestly or even merely use the system. Acquisti, Dingledine and Syverson have started looking at this problem, [ADS03], but many open questions remain.

2.6 Other Schemes – Related Work

So far we have only described mix systems as a way of achieving anonymity. In fact, there are other ways as well.

2.6.1 Crowds

Crowds [RR97] is a system which is designed to maintain sender (but not receiver) untraceability. The scheme is very simple: to initiate a communication, a participant of the system forwards his request to another member of the crowd. They then flip a biased coin; if the outcome is heads, they forward the communication to the responder, if it is tails they forward it on to another randomly chosen member of the crowd. Data from the responder travels down the same path in the opposite direction.

This system was designed to enable co-operating users on a medium-sized LAN (say, within a company) to protect themselves against the threats out on the Internet. Hence, there is no mixing, padding or onion encryption. Therefore an adversary who observes the entire network can trivially find the initiator – the initiator sends out a packet without having received one, while all other participants receive a packet first, forwarding it a short time later. Another problem with Crowds is the fact that the system is inherently not very scalable – in the original design every node had to share a key with every other node.

Several comments are in order. First of all, it is clear that the Crowds scheme of anonymous communication is less secure than a mix system for a message-based system. Secondly, although the Crowds scheme for connection-based systems has not been adequately compared to onion-based schemes, the latter offer more potential in the future because of the bitwise unlinkability properties offered by onion encryption. Indeed, that comparison is an interesting piece of work to do in itself – there are many attacks which can be mounted on both systems and coming up with a realistic threat model and an onion-based scheme which does not suffer from devastating attacks (some of which are outlined in Chapter 7) in that threat model is challenging in itself.

Furthermore, researchers have shown that even within the threat model (no global adversary, some number of compromised nodes) it is possible to mount effective attacks against Crowds [WALS02, Shm03]. However, these basically amount to long-term intersection attacks which are likely to be effective against any anonymity system.

An interesting extension of the Crowds system is Hordes [LS02]. It is essentially an efficiency improvement over crowds where data travels back to the initiators via a multicast group which they had initially selected.

2.6.2 Dining Cryptographers Networks

Schemes which provide sender and receiver untraceability (and thus unlinkability) are usually based on some sort of broadcast mechanism. Hence, unlike in mix systems, the adversary is unable to determine who is sending or receiving messages. The first is a scheme called “Dining Cryptographers” [Cha88] which is commonly illustrated by considering a number of cryptographers sitting round a table flipping (fair) coins. They arrange themselves in such a way that each could see their own outcome of the coin toss as well as that of the person sitting to the right of them. Each would broadcast whether the two coins they saw were the same or different. The protocol is executed repeatedly. A person may transmit a bit by inverting his broadcast if he is transmitting a “1” or act as above if he is transmitting a “0”. (If two people start to transmit, they will soon notice). Naturally, this is a good scheme only if the underlying communications medium offers broadcast as a basic service or sending lots of traffic is not too expensive. Examples of these include local LANs (or dinner tables at restaurants!). The work on DC-nets was continued by Waidner and Pfitzmann [WP90].

Yet another recent anonymous communication scheme based on broadcast is $\mathcal{P}^5$ [SBS02]. It presents an efficient way of setting up a broadcast-based communication (as a binary tree). As a result, the authors claim that the system is secure against a global passive adversary. Several comments are in order. First of all, the scheme does not satisfy the usability requirement we have argued essential above (Sections

2.4, 2.5). The scenario involves a set of users opting to co-operate and using the $\mathcal{P}^5$ scheme between themselves. Hence, if this scheme was to provide anonymous web browsing, users would only be able to browse websites which run on $\mathcal{P}^5$-compatible webservers. This, in our view, is unacceptable. Secondly, the system admits packet loss (this makes anonymity easier to provide, but the vast majority of anonymous communication systems avoid it if at all possible). The bandwidth requirements are also quite high, although, as expected, much better than the naïve broadcast protocol.

Other work in this area includes XOR-trees [DO00] and Herbivore [GRPS03].

2.6.3 Buses

A related way of doing anonymous communication is to exploit persistent communication patterns [BD03]. A variety of protocols with various time and communication complexity are explored in this work. They all rely on the co-operation of all the parties in the network and are not likely to lead to the development of practical systems.

The simplest of the protocols based on this idea is as follows. Imagine a bus driving around town. It stops in front of each participant who checks for any messages that were sent to him and changes all the "seats" allocated for his messages either to random junk or to messages he wants to send to other participants.

2.6.4 Location Privacy

A different kind of connection-based system was presented by Al-Muhtadi and co-authors [AMCK$^+$02]. It is aimed at achieving location privacy, the inability of the adversary to determine someone's location, and does not have a very strong threat model. The architecture of the system is very rigid, with packets being passed "up" and "down" the tree. Each node trusts its Lighthouse node. There is very little analysis of the anonymity this systems provides.

Most other work on location privacy also assumes weaker threat models. An interesting example with some anonymity analysis is presented by Beresford and Stajano [BS03].

2.7 Summary

In this chapter we have looked at the terminology often used to describe various aspects of anonymity systems, examined the basic constructions which underly anonymous communications, including a fairly detailed look at mix systems, considered how users might wish to communicate on the Internet and finally examined some other (mostly broadcast-based) schemes.

We now go on to look at ways of measuring anonymity.

Chapter 3

Measuring Anonymity

> "*Primitive man regards his name as a vital portion of himself and takes care of it accordingly.*"
> — Sir James George Frazer, The Golden Bough, Chapter 22.

In the previous chapter we presented some schemes for anonymous communication, but did not *calculate* how much anonymity they provide. Indeed, anonymity is hard to measure or quantify – the users of the system do not see how high or low their anonymity is directly. Naturally, they may see the effect of this later! This chapter is devoted to discussions of how to measure or quantify anonymity: until quite recently, there has been no consensus as to which metric should be used to quantify anonymity. Arguably this is still the case.

Let us briefly remind ourselves of the reasons for quantifying anonymity. First of all, if we can work out the amount of anonymity to be provided by a system, then each of the users can assess whether this system suits their requirements and thus decide whether to use it or not. Secondly, we need a consistent metric of anonymity to be able to compare different anonymity systems. Finally, we should also be able to use the metric to compare the effectiveness of attacks against anonymity systems.

In this chapter we first look at how anonymity has been measured previously, then point out the problems with this approach. We then propose a better metric. In all the examples we choose to abstract away from the application level issues of anonymous communication. Examples of this include preventing the attacker from embedding URLs pointing to the attacker's webpage in messages in the hope that the victim's browser opens them automatically [CDK01]. Instead, we work with the example of message-based mix systems, broadly at the abstraction level presented in Section 2.3. The threat model employed in this chapter is the global passive adversary.

Perhaps the most intuitive way of quantifying the anonymity of a message M in a mix system is to just count the number of messages M has been mixed with while passing through the system. However, doing so would produce a bad metric as the case of the $n-1$ attack (Section 2.3.5) shows – here the message is confused with n messages, but really has no anonymity at all.

Another popular metric of anonymity is the notion of anonymity set. We now look at how anonymity sets have previously been defined in the literature and what contexts they have been used in.

3.1 Previous Work: Anonymity Sets

3.1.1 Anonymity Sets in Dining Cryptographers Networks

The notion of anonymity set was introduced by Chaum in [Cha88] in order to quantify the "security" of Dining Cryptographers' (DC) networks. The anonymity set of a message is defined as the set of participants who could have sent that particular message, as seen by a global observer who has also compromised a set of nodes.

The size of the anonymity set reflects the fact that even though a participant in a Dining Cryptographers network may not be directly identifiable, the set of other participants that he or she may be confused with can be large or small, depending on the attacker's knowledge of particular keys. Chaum argues that the size of the anonymity set is a good indicator of how good the anonymity provided by the network really is. In the worst case, the size of the anonymity set is 1, which means that no anonymity is provided to the participant. In the best case it is the size of the network, which means that any participant could have sent the message.

3.1.2 Anonymity Sets in Stop-and-Go Mixes

In analysing Stop-and-Go mixes, [KEB98], the authors also use sets as the measure of anonymity. Given a message, they define the anonymity set of users as those who had a non-zero probability of having the role r (sender or recipient) for that message. The size of the set is then used as the metric of anonymity. Furthermore, *deterministic anonymity* is defined as the property of an algorithm which always yields anonymity sets of size greater than 1.

3.1.3 Standard Terminology – Anonymity Sets

In an effort to standardise the terminology used in anonymity and pseudonymity research publications and clarify different concepts, Pfitzmann and Köhntopp [PK00] define anonymity itself as:

> "Anonymity is the state of being not identifiable within a set of subjects, the *anonymity set*."

In order to further refine the concept of anonymity and anonymity set and in an attempt to find a metric for the quality of the anonymity provided they continue:

> "Anonymity is the stronger, the larger the respective anonymity set is and the more evenly distributed the sending or receiving, respectively, of the subjects within that set is."

The concept of "even distribution" of the sending or receiving of members of the set identifies a new requirement for judging the quality of the anonymity provided by a particular system. It is not obvious any more that the size is a very good indicator, since different members may be more or less likely to be the sender or receiver because of their respective communication patterns.

The reader will note that the definitions of anonymity sets look very similar, though we will later (in Section 3.2) show that there are subtle differences in the way they apply to the systems they are defined in the context of.

We slightly rephrase the definitions presented in the Stop-and-Go mixes work [KEB98] to introduce the notions of sender and receiver anonymity sets. For a particular message, and a particular attacker, we can try to compute all the possible senders who could have sent this message (from the point of view of the adversary). This is the *sender anonymity set*. Similarly, the *receiver anonymity set* is the set of receivers who could have received this message. Note that we are usually interested in the sender anonymity set when we know who received a message, and the receiver anonymity set if we know who sent it. If both the sender and the receiver anonymity sets contain only one element, *unlinkability is compromised* (we know who sent the message and who received it).

Let us consider an example of some messages travelling through a mix network made up of threshold ($n = 2$) mixes and see what the sender and receiver anonymity sets of different messages are. It is often helpful to illustrate a *run* of a mix system on a *mix network run diagram* such as the one shown on Figure 3.1. Here, A, B, C, D, E are the senders and Q, P, R, S are the receivers. Each arrow represents one message (sent from the entity at the base of the arrow to the entity at the head of the arrow), while each grey box represents a firing of a mix. Time runs from left to right, so the message from A to the uppermost mix has arrived before a message was sent from

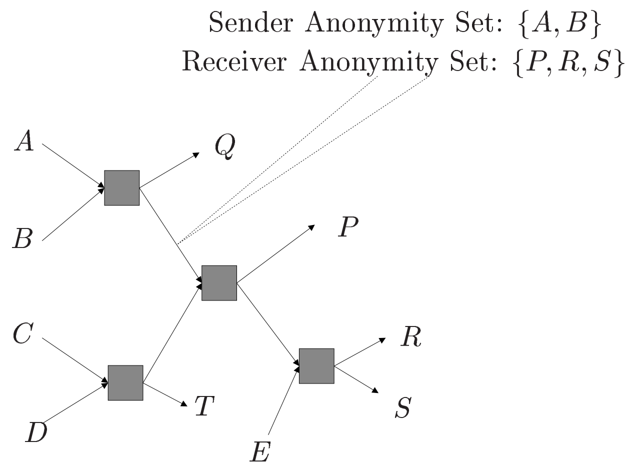


Figure 3.1: Example: Sender and Receiver Anonymity Set of a message

the uppermost mix to Q. Note that it does not matter whether the message from A or from B has reached the uppermost mix first – it cannot flush until both of them have arrived. Note that arrows from right to left would represent messages travelling back in time, and thus cannot appear on mix network run diagrams.

It is worth pointing out the contrast between anonymity and cryptography. When an adversary is attacking a cipher, he has typically 2^{64} or 2^{128} possible keys which could be the key the message was encrypted with. On the other hand, when trying to compromise the anonymity of a target message, he has merely the people in the anonymity set as possible candidates. This might be a few thousand participants or even less – much smaller than 2^{64}. Of course, it is much harder to do an exhaustive search of those participants!

3.2 Difficulties with Anonymity Set Size

Let us examine how the definitions of the anonymity sets were applied to the context they were defined in. The attacks against DC networks presented in [Cha88] can only result in partitions of the network in which all the participants are still equally likely to have sent or received a particular message. Therefore the size of the anonymity set is a good metric of the quality of the anonymity offered to the remaining participants. In the Stop-and-Go system [KEB98] definition, the authors realise that different senders may not have been equally likely to have sent a particular message, but choose to ignore it. We note, however, that in the case they are dealing with (mix cascades in a system where each mix verifies the identities of all the senders), all senders have equal probability of having sent the message. In the standardisation

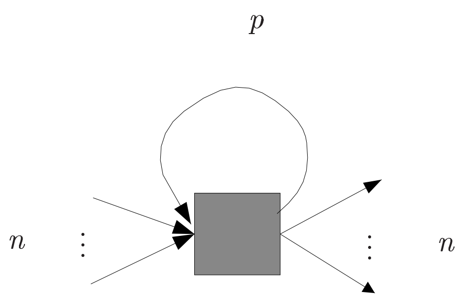


Figure 3.2: A Pool Mix (one round). Note that this is not a mix network run diagram, it simply represents the operation of the mix

attempt [PK00], we see that there is an attempt to state, and take into account this fact in the notion of anonymity, yet a formal definition is still lacking.

We have come to the conclusion that the issue of potentially different probabilities of different members of the anonymity set actually having sent or received the message has not been adequately addressed in previous work. Yet the fact that the probabilities are different gives a lot of extra information to the attacker. We proceed to illustrate this with an example.

3.2.1 The Pool Mix

To further emphasise the dangers of using sets and their cardinalities to assess and compare anonymity systems, we note that some systems have very strong "anonymity set" properties but may not provide intuitively good anonymity. One such system is a *pool mix*.

What is a pool mix? A pool mix differs from the threshold mix we described in Chapter 2 only in its flushing algorithm. This mix always stores a pool of p messages (see Figure 3.2). When n incoming messages have accumulated in its buffer, it picks p messages randomly out of the $p + n$ it has, and stores them, forwarding the others in the usual fashion. The process of accumulating (in this case n) messages, then processing them and finally sending messages out is called a *round*. There is always a non-zero probability of any message which has ever entered the mix not having left it. Hence the sender anonymity set of a message M passing through the pool mix includes the senders of all the messages which have passed through that mix before M, i.e. the anonymity set grows without limit. At this point we must consider the anonymity provided by this system. Does it really give us very strong anonymity guarantees or is measuring anonymity using sets inappropriate in this

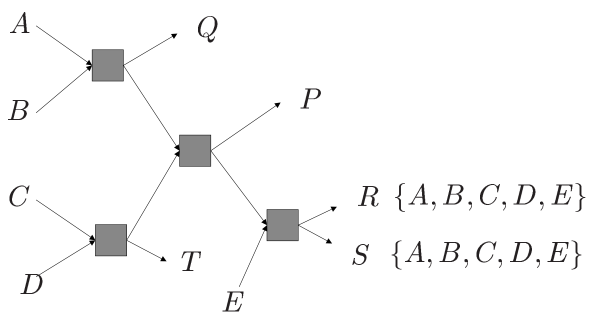


Figure 3.3: Vulnerability of Anonymity Sets

case? Our intuition suggests the latter[1], especially as the anonymity set seems to be independent of the size of the pool, p.

3.2.2 Knowledge Vulnerability

Yet another reason for being sceptical of the use of anonymity sets is the failure of the metric to take account an attacker's additional knowledge. Consider the mix network run diagram in Figure 3.3 (some of the messages arriving at receivers are labelled with their sender anonymity sets).

Notice that if the (global passive) attacker somehow establishes the fact that, for instance, A is communicating with R, he can derive the fact that S received a message from E. Indeed, to expose the link $E \to S$, all the attacker needs to know is that one of A, B, C, D is communicating to R. And yet this is in no way reflected in S's sender anonymity set (although E's receiver anonymity set, as expected, contains just R and S).

It is clear that not all senders in this arrangement are equally vulnerable to this. Other mix network run diagrams may be less (or at least more evenly) vulnerable to additional knowledge of the attacker. Although we have highlighted the attack here by using mixes with threshold of 2, it is clear that the principle can be used in general to cut down the size of the anonymity set.

[1] A side remark is in order here. In a practical implementation of such a mix, one might like to put an upper limit on the time a message can remain on the mix with a policy such as: "All messages should be forwarded on within 24 hours + K mix flushes of arrival".

3.3 Entropy

We have now discussed several distinct and, in our view, important difficulties with using anonymity sets and their cardinalities for quantifying anonymity. We have also demonstrated that there is a clear need to reason about information contained in probability distributions. One could therefore borrow mathematical tools from Information Theory [Sha48]. The concept of entropy seems particularly appropriate as it is designed to describe the amount of information contained in a probability distribution, and should therefore let us measure the amount of uncertainty the attacker has about who sent or received the message. We now proceed to define our anonymous communication model and use entropy to describe its quality. The model is very close to the one described in the Stop-and-Go mixes paper [KEB98].

Definition 1. *Given an attacker and a set of all users Ψ, let $r \in \mathcal{R}$ be a role for the user ($\mathcal{R} = \{sender,\ recipient\}$) with respect to a message M. Let $\mathcal{U}_{(M,r)}(u)$ be the attacker's a-posteriori probability of a user $u \in \Psi$ having the role r with respect to M.*

In the definition above we do not have an anonymity set but an r *anonymity probability distribution* $\mathcal{U}_{(M,r)} : \Psi \rightarrow [0,1]$ s.t. $\sum_{u \in \Psi} \mathcal{U}_{(M,r)}(u) = 1$. In other words, given his view of a run of the system, the attacker takes a message M and calculates as best as possible, the probability distribution of its senders and receivers. Of course, $\mathcal{U}_{(M,r)}$ may assign zero probability to some users which means that they cannot possibly have had the role r for the particular message M. For instance, if the message we are considering was seen by the attacker as having arrived at Q, then $\mathcal{U}_{(M,receiver)}(Q) = 1$ and $\forall S \neq Q \ .\ \mathcal{U}_{(M,receiver)}(S) = 0$.[2] If all the users that are not assigned a zero probability have an equal probability assigned to them, (as will be the case of a DC network under attack) then, as Chaum suggests, the size of the set could be used to describe the anonymity. The interesting case is when users are assigned different, non zero probabilities.

Definition 2. *We define $E_{(M,r)}$, the r anonymity of a message M, as the entropy of an r anonymity probability distribution $\mathcal{U}_{(M,r)}$. In other words*

$$E_{(M,r)} = -\sum_{u \in \Psi} \mathcal{U}_{(M,r)}(u) \log_2(\mathcal{U}_{(M,r)}(u))$$

We also define the effective anonymity set size *as 2^E.*

[2] Alternatively, we may choose to view the sender/receiver anonymity probability distribution for a message M as an extension of the underlying sender/receiver anonymity set to a set of pairs of users with their associated (non-zero) probabilities of sending or receiving it.

It is trivial to see that if one user is assigned a probability of 1 then the effective size of the anonymity set is 1, which means that the attacker already has enough information to identify the user.

There are some further observations:

- It is always the case that $0 \leq E \leq \log_2 |\Psi|$.
- If $E = 0$ the communication channel is not providing any anonymity.
- If $E = \log_2 |\Psi|$ the communication channel provides perfect anonymity.

Let us now use our new metric to analyse the anonymity of a message which has passed through the pool mix.

3.4 Analysis of the Pool Mix

Recall from Section 3.2.1 that a pool mix stores p ($p > 0$) messages and receives n messages every round. It then combines the stored and received messages and outputs n of them (chosen randomly). When the mix first starts operating, p messages are generated by the mix and placed in the pool. We call this round zero. The time period during which the next n messages arrive and the mix flushes is called round one, and so on.

First, we calculate the sender anonymity set and its cardinality for a run of the pool mix. Let the sender anonymity sets associated with the n messages arriving at round i be $K_{i,1} \ldots K_{i,n}$ and let $\hat{K}_i = K_{i,1} \cup \ldots \cup K_{i,n}$. The sender anonymity set $\hat{K}_0$ consists of only one sender – the mix itself (all the messages in the pool have been placed there by the mix). Let A_i be the sender anonymity set of the outgoing messages after round i; it is equal to the union of the anonymity sets of the messages stored in the mix (all of which are the same and are equal to the anonymity sets of all the messages which left the mix at the previous round) and the messages which arrived from the network.

$$A_0 = \{\text{mix}\}$$

$$A_i = A_{i-1} \cup \hat{K}_i$$

Now, assume that all of the messages arrive at the pool mix directly from senders, which are all different from each other. This will allow us to calculate the maximum possible anonymity of a pool mix, which we can compare to that of other mixes. Formally, assume $\forall i, j, l, m.(i \neq l \vee j \neq m) \Rightarrow K_{i,j} \neq K_{l,m}$. This implies that the size of the set after round k is

$$|A_k| = n \times k + 1$$

and in the limit[3] $k \to \infty$

$$\lim_{k \to \infty} |A_k| \to \infty$$

It is clear once again that the size of A_k does not provide us with a useful insight on how well this mix performs. In particular, it does not capture the fact that senders of past rounds have smaller probabilities of being the senders of messages that come out of the mix at the last round. Finally, this metric does not allow us to effectively compare the pool mix with other mixing architectures, including conventional threshold mixes.

We now calculate the entropy of the sender anonymity probability distribution of a message coming out of the pool mix.

The probability that a message which comes out of the mix at round k was introduced by a sender in the mix at round $0 < i \leq k$ is

$$p_{round_i} = \frac{n}{n+p}\left(\frac{p}{n+p}\right)^{k-i}$$

$$p_{round_0} = \left(\frac{p}{n+p}\right)^{k}$$

Definition 3. *Assume that each message arrives at the mix directly from its sender and all senders only send one message. After round 0, the only sender involved is the mix itself. The entropy of the anonymity probability distribution at round k is*

$$E_k = -\sum_{i=1}^{k}\left(\frac{n}{n+p}\left(\frac{p}{n+p}\right)^{k-i}\log_2\left(\frac{1}{n+p}\left(\frac{p}{n+p}\right)^{k-i}\right)\right) - \left(\frac{p}{n+p}\right)^{k}\log_2\left(\frac{p}{n+p}\right)^{k}$$

After a large number of rounds ($k \to \infty$) the above expression converges towards[4]

$$\lim_{k \to \infty} E_k = \left(1 + \frac{p}{n}\right)\log_2(n+p) - \frac{p}{n}\log_2 p$$

[3]Note that in taking the limit we are assuming an infinite set of users.
[4]This expression was derived by Danezis, [Dan04].

n	p	Anonymity	Average Delay (rounds)	Average Delay (message arrivals)
100	0	6.6439	1	50
100	1	6.7248	1.01	51
100	10	7.1273	1.1	60
100	100	8.6439	2	150
200	0	7.6439	1	100
200	1	7.6893	1.005	101
200	10	7.9339	1.05	110
200	100	9.0213	1.5	200

Table 3.1: Some threshold and pool mixes together with their anonymity and average delay.

Note that the additional anonymity that the pool mix provides is not "for free", of course, since the average latency of the messages increases from 1 round to $1 + \frac{p}{n}$ rounds, or, in terms of the average number of message arrivals, from $\frac{1}{2}n$ to $\frac{1}{2}n + p$. The delay and anonymity of various threshold and pool mixes are summarised in Table 3.4. Note that the mixes with pool of zero messages, are, of course, threshold mixes, which have anonymity $\log_2(n)$.

3.5 What is Anonymity?

At this stage we are ready to return to the key question "What is anonymity?". It turns out that the question is ill-posed. The anonymity of a message which has passed through a mix system is fundamentally a very different thing from the anonymity of a mix system.

We have quantified the sender/receiver anonymity of a message which passed through the system from the point of view of the attacker as the entropy of the sender/receiver anonymity probability distribution of this message which the attacker builds given his observation of the system. There are two key questions: "How much does the attacker know?" and "How does he compute the probability distribution?"[5]. Clearly, if the attacker knows a lot about the system (e.g. the keys of all the mixes the message went through), the anonymity of our chosen message will be zero. Also, if the attacker invests huge amounts of computational power and finds the private keys of all the mixes, the anonymity will be zero.

[5]One example of how this is done is presented in Chapter 6. It is interesting to note that the attacker may come up with an algorithm for computing the anonymity probability distribution which is incorrect! A trivial example of this is "Always blame sender A".

The entropy of sender/receiver probability distribution is useful for comparing different attacker models, or different ways of computing the anonymity probability distributions. It is less useful for comparing systems. The reason for this is that anonymity systems are very often non-deterministic, e.g. if a user sends message A through the same mix network at two different times t and t', he will get a different behaviour, resulting in that message having different anonymity as seen by the same attacker who uses the same algorithm to compute the anonymity probability distribution. What causes this non-determinism? In mix systems, anonymity is created by mixing messages together, hence different numbers of messages in the mix system at times t and t' will have caused A to have different anonymity. Even if there were the same number of messages, they would most likely travel via different routes, resulting in different anonymity. Finally, even if all the parameters of user messages at t and t' were the same, the non-determinism in the network can still cause a discrepancy in the anonymity of A at t and t'.

How can we compute the anonymity of an anonymity system? After all, we seemed to perform such an analysis for the pool mix. It turns out that the pool mix has several properties which makes its analysis relatively easy. Firstly, there is the fact that all the messages coming out of the pool mix had the same anonymity[6], so analysing the anonymity of one message passing through the pool mix is meaningful for all the messages. Indeed, we will use this method in analysing other mixes in just the same way. Secondly, we must remember that in the pool mix the user was very constrained – he could only do one thing, i.e. send a message to the pool mix. In other systems, users can perform different actions, some of which are much more difficult to anonymize than others.

Thus, if we take a fixed action A of a particular user, we can compute the anonymity of this action (e.g. the receiver anonymity of the message that our chosen user sent via remailers $[N, M]$) in each of the possible runs of the anonymity system. Given assumptions about (or probability distribution of) the actions of all the other users of the system, we can build a probability distribution of the possible values of anonymity action A will gain if this system is used. This is very useful to the user – he knows the action he is going to perform, and can therefore compare different anonymity systems by comparing these distributions.

At this point it is interesting to note that the requirements of the user can be quite complex too: they may require that against a certain attacker, their message, having passed through the anonymity system, under no circumstances has sender anonymity less than 10. On the other hand, they may be content with a guarantee that on average, the system would provide their message with sender anonymity of 20.

We are now in a position to compare the anonymity of a certain action provided by

[6] If we assume that the mix started working at the beginning of time!

different systems against a particular adversary. We could abstract this to account for all possible actions, and thus try to compare anonymity systems in general. This is entirely possible in principle, but likely to be very computationally expensive as we would need to build probability distributions over all the possible things that could have happened inside the anonymity system. We do not discuss this any further. It may well be the case that such analyses are performed in the future. It is worth mentioning that a very similar approach was taken by Shmatikov in his analysis of Crowds [Shm03]. There, he uses a model checker to enumerate all the possible states the anonymity system can reach given certain inputs. (Though, even that rather complicated analysis did not have to account for the case of users being able to perform different actions – the only thing a Crowds user can do is to initiate a connection.)

We hope that the reader will have gathered that analysis of anonymity systems in the general case is hard and computationally complex. Hence, we must try to find properties of the system we are trying to analyse which allow the analysis to be simplified. If such properties cannot be found, analysis is likely to be constrained to simple systems.

It is worth saying that although this is the general method of analysing anonymity systems, many current designs are not mature enough to be analysed in this way. Some break so easily that much simpler constructions can be used to show this fact (see the two analyses of Chapter 7).

3.6 Related Work

We will see many more uses of the entropy metric throughout the thesis. For now, let us have a more detailed discussion about some of the other metrics of anonymity, not restricting our attention to those employed in discussions of mix systems.

We have already discussed the anonymity set metric originally introduced by Chaum in Section 3.2.

A metric very similar to ours was developed independently by Diaz, Seys, Claessens and Preneel and (also) published in PET 2002 [DSCP02]. They have the same idea of using an anonymity probability distribution, but then compute the entropy divided by the size of the set. This measures precisely (from [PK00], Section 3.1.3) how evenly distributed the probabilities within the anonymity set are. However, it does not take into account the size of the set itself! Hence, it is fair to use this metric only in combination with the size of the set, but this is precisely what is achieved by our proposal above[7].

[7] In some later papers, Diaz uses the metric outlined here.

Another metric was proposed by Reiter and Rubin; and employed in their analysis of the Crowds anonymity system [RR97]. They proposed to measure anonymity as the probability of the attacker determining the sender of a particular message. The main *degrees of anonymity* are defined as follows. The sender is *beyond suspicion* if he is as likely to have sent the message as any other sender in the system. The sender is guaranteed *probable innocence* if he is no more likely to have sent the message as not (i.e. the probability of him sending the message is less than 0.5). The sender maintains *possible innocence* if there is a non-trivial probability of someone else to have sent the message.

There are several problems with this definition. First of all, it does not take into the *number* of senders in the set. "Beyond suspicion" is achieved just as well by a single mix with a threshold of 2 as by one with a threshold of 1000. This is somewhat similar to the knowledge vulnerability problem pointed out in Section 3.2.2. Secondly, the notion of possible innocence contains a rather vague term "non-trivial probability". Finally, the term probable innocence is also defined rather arbitrarily – why should we care about a user of having sent the message with a probability of 0.5 more than, say 0.55.

A different framework for expressing anonymity properties was recently proposed by Hughes and Shmatikov in [HS04]. Let us use an example of a set S of senders potentially communicating to a set R of receivers. Suppose the communication graph is represented by a function $f : S \to R$ (hence each sender sends a message to only one receiver). An attacker's *view* of this function f consists of *graph knowledge*, a binary relation F ($F \subseteq S \times R$ s.t. $f \subseteq F$); *image knowledge*, a subset of R representing the receiver who were communicated to and *kernel knowledge*, an equivalence relation representing the senders who communicated to the same receivers. Using these, we can represent the (by now) familiar concepts of anonymity set, sender untraceability, etc. The authors go on to define the concept of privacy (in their view, privacy is maintained when the type of communication is hidden, for instance whether Alice communicates to Bob in his capacity as a doctor or a counsellor, even if both the endpoints are known to the adversary). In our view, the term privacy in its broadest sense is rather too vague to be defined formally. Hence, although the authors formally define a variety of information hiding properties, calling one of them "privacy" is rather misleading.

A number of different formal definitions of anonymity were recently proposed by Halpern and O'Neill [HO03]. They present definitions very similar to those of anonymity set. In particular, they define an action as being totally anonymous if *all* the participants of the protocol are in the set and minimally anonymous if *more than one* participant is in the set. They also define a probabilistic notion of anonymity which essentially says that the probability of the participant having performed a particular action is less than α. Yet another probabilistic definition states that all members of the anonymity set ought to have equal probabilities of having performed the action. Overall, the authors present a good set of formal definitions broadly along the lines of those presented in this chapter.

The two PET 2002 papers on anonymity metrics [SD02, DSCP02] have lead to a probabilistic definition of unlinkability by Steinbrecher and Köpsell [SK03]. There they define more abstract notions of unlinkability of items within a set as well as unlinkability of items belonging to different sets. This notion is useful in research on anonymity systems as it allows us to measure, for example, the degree to which two messages might have been sent by the same sender, without considering who that sender might be.

3.7 Summary

We have now seen how anonymity can be quantified taking into account the different probabilities with which various parties have performed the particular action. In this case we examined the probability with which a particular sender sent the target message.

Having now equipped ourselves with a way of measuring the anonymity, we return to the subject of mix systems. Since we can analyse threshold pool mixes using the new metric, we embark on a more detailed investigation of the properties of various other mixes. In the next chapter, we examine their properties under active attack scenarios.

Chapter 4

Active Attack Properties of Single Mixes

"When an Ojebway is asked his name, he will look at some bystander and ask him to answer."
— Sir James George Frazer, The Golden Bough, Chapter 22.

Chaum first introduced the concept of a mix in 1981 [Cha81], and since then researchers and developers have described many mix variations, e.g. [GT96, Jak99, KEB98]. These have different aims and approaches, yet the performance and anonymity tradeoffs between them are still not well-understood. In fact, some of the mixes used in well-known fielded systems such as Mixmaster [Cot94, MCPS03] are mentioned only briefly or not at all in the literature. We start closing this gap by enumerating and exploring a variety of mix flushing algorithms. Here we consider *batching* mixes – i.e. ones that collect a batch of messages before processing them. This is in contrast with *continuous* or *synchronous* mixes which process messages individually and typically rely on a particular distribution of traffic (e.g. [KEB98]).

In the past, many anonymity systems have been concerned with protecting their users against passive adversaries, either global or local, usually citing the $n-1$ attack which we introduced in Section 2.3.5 as a vulnerability, with (quoting Berthold et al. [BPS00]) "no general applicable method to prevent this attack". In this chapter we discuss ways of reducing this vulnerability. In particular, we consider the extent to which the mixes are vulnerable to active attacks such as the $n-1$ attack.

An active attack on a batching mix is usually mounted in the following way: the adversary targeting a specific message going into a mix manipulates the messages entering that mix so the only message unknown to him in the batch is the target message [Cot94, GT96]. This manipulation may involve delaying or dropping most or all other incoming messages (a *trickle* attack [GT96]), or flooding the batch with attacker messages (a *flooding* attack). We call these attacks or combinations of them *blending* attacks.

We provide a rigorous analysis and comparison of several properties of each mix variant, including anonymity, latency, and resistance to blending attacks. We also give intuition and guidelines about which environments and circumstances are most suitable for each mix variant.

4.1 Blending Attack Taxonomy

Here we consider a global *active* adversary (see Section 2.3.2) who is not only able to see the traffic on all the links, but also to delay or insert arbitrarily many messages in a short time. These are reasonable assumptions – methods of logging per-packet data on high bandwidth links exist, as does the possibility of building fast hardware to insert or delay messages. We further assume our attacker can send messages from many different source addresses[1]. Any attempt to counter this – source authentication – would trivially defeat the very anonymity that free-route mix systems are intended to protect. The active attacker's threat comes from his power to log and manipulate messages on the links.

Note that the global active attacker can be viewed as a combination of two separate attackers: one who can only insert messages (global inserting attacker) and one who can only delay messages (global delaying attacker). The latter is obviously easier to implement.

It is well known that the ability to insert or delay messages may allow the attacker to determine, for instance, the recipient of a particular message originating from a sender. We illustrate this in the case of a single threshold n mix. The attack, commonly referred to as the $n-1$ attack, proceeds as follows: The attacker observes the target message leaving the sender heading towards the mix and delays it. He now starts sending messages into the mix along with anyone else's messages that happen along until it fires. As soon as the mix fires, he stops all other messages from entering the mix and sends in the target message along with $n-1$ of his own messages. After the mix fires, he can recognise each of the $n-1$ messages of his own when they leave the mix and can therefore determine the destination of the target message. The same attack can be used against a mix cascade by mounting it against the first mix, or against a mix network by repeating the attack for each of the mixes on the path of the message in turn.

We now consider the properties of this attack before going on to examine how a similar arrangement could work on other mixes.

[1] Indeed, infrastructures to ensure sender authentication have proved difficult to build.

First of all, we note that the attack is *exact* – that is, it provides the adversary with the ability to determine the receiver of a particular message with certainty 1 (the anonymity of a message passing through the mix is 0). We also note that this attack does not depend on the rest of the mix network; that is, the attacker has enough power to always isolate and trace a particular message. We call such an attack *certain*.

We classify mixes into the following categories of vulnerability to blending attacks.

- If no blending attack can reduce the anonymity of any message at all, the mix is *strongly resistant to blending attacks.*
- If no blending attack can reduce the anonymity of any message below a constant k, then the mix has *blending attack anonymity* k.
- If the attacker can always reduce the anonymity of a message arbitrarily, but never to 0, the mix is vulnerable to *non-exact, uncertain blending attacks.*
- If the attacker can always reduce the anonymity of a message to 0, but may need to spend an arbitrary amount of resources (time/messages) to do so, the mix is vulnerable to *exact, uncertain attacks.*
- If the attacker is always able to reduce the anonymity of a message to 0 in a bounded amount of resources, it is vulnerable to *exact certain attacks.*

Although it may appear that the "vulnerability" of the mixes goes up as we go down the list, this is not necessarily the case – it is important to consider the cost of the attack as well. For example, suppose the anonymity of a message going through Mix 1 under active attack is proportional to the inverse of the cube of the number of messages expended in the attack. This mix can be seen as "more vulnerable" than Mix 2 which is always compromised by 10^6 attacker messages. Note that Mix 1 is vulnerable only to non-exact blending attacks, while the Mix 2 is vulnerable to exact certain attacks.

We will proceed to analyse and categorize several mixes. For each of them, we will suggest their blending attack cost (both of time and number of messages it takes to carry out) where necessary. Before doing so, there is a further interesting point to consider – what aspects of the cost of the blending attack are important?

The cost in terms of attacker messages clearly correlates with the amount of processing power the adversary has to expend to generate the attacker messages (each message requires at least one public key encryption![2]) and bandwidth in order to send these messages.

The time taken to execute the attack is in some sense more important – it is also the

[2]Though messages can be precomputed.

time for which all the other messages need to be prevented from entering the mix which under attack. If this time is significant (i.e. not accountable for by network delays), the attack becomes visible to the rest of the mix network. If such an attack is detected, anonymity can be preserved by, for instance, manual intervention from the mix operator to delete the messages inside the mix which is under attack. Although in this case the target message does not reach its destination, its anonymity is certainly not compromised and thus the active attack has failed. In summary, we want to find mixes which require as many attacker messages and as much time to attack as possible.

4.2 Simple Mixes

In this section we describe various mixes divided according to their flushing algorithms which are sometimes known as batching strategies. In particular we set out, for each mix type: the mix parameters, the flushing algorithm, the delay on messages in normal operations (i.e. when not under attack), the minimum and maximum anonymity provided against a purely passive adversary, analysis of the blending attacks on the mix, and finally provide a discussion of the adversaries capable of performing blending attacks. This section describes simple mixes, in which all of the messages in the mix are sent each time it fires. The more complex pool mixes are discussed in Section 4.3 below.

In describing the different mix flushing algorithms and working out their properties, we make several assumptions:

- The mixes take a constant time to fire (send messages out).
- The mixes have limited physical memory and so can only contain a finite number of messages. Further, they have finite bandwidth, so can only receive a certain number of messages in a given amount of time.
- Mixes prevent message replays.
- Messages may or may not arrive at a uniform rate.
- In calculating the minimum and maximum anonymity we assume the global passive adversary. The vulnerability to the active attacker is described separately.
- When we talk about anonymity, we mean sender anonymity of any of the messages at the current round. Similar ideas can be applied to receiver anonymity as well.
- A mix can store a maximum of c messages.

- Just as in Section 3.2.1, we assume that all the messages in the batch are from different senders (and go to different receivers). This is done merely to be able to fairly compare the mixes against each other.
- A message takes ϵ time to arrive or leave the mix.
- A mix takes a constant (μ) time to process each message (decrypt, reorder, check for duplicates, etc).
- A mix only starts receiving messages only after it has sent on all the messages from the previous round.

Note that in practice, these can be ignored as the time it takes a mix to process the message is small compared to the inter-arrival time between messages.

4.2.1 Threshold Mix

Parameters: n, the threshold.

Flushing Algorithm: When the mix collects n messages, it fires (flushing all n).

Message Delay: The minimum delay is $\epsilon + \mu$ (the target message arrives when there are $n-1$ messages already in the mix and leaves first). The maximum delay can be infinite (the target arrives at a mix and no more ever arrive). Assuming a rate of arrival of messages r, the expected delay of a message is: $\frac{n-1}{2r}$.

Anonymity: For completeness, let is derive the anonymity of the threshold mix. Consider one round where the messages arrived to the mix directly from n different senders $\{s, s', s'', \ldots\}$. Take a message M and consider the probability it arrived from the various senders. Clearly, $\forall s. P(s = M) = \frac{1}{n}$.

Hence the anonymity of the mix is:

$$-\sum_{n} \frac{1}{n} \log_2 \frac{1}{n} = -\log_2 \frac{1}{n} = \log_2 n$$

Blending Attack Behaviour: The attack is exact and certain; it proceeds as outlined in Section 4.1 and is usually referred to as the $n-1$ or the flooding attack. It takes a maximum of $(4n-1)\epsilon + 2n\mu$ time to insert the required messages and for the two firings of the mix to happen. The attack takes a minimum of $n-1$ messages (the attacker waits until the mix is empty, forwards the target message to the mix along with $n-1$ attacker messages to flush it out) and a maximum of $2n-2$ messages. (the attacker does not wait for the mix to become empty).

Adversaries: The above attack seemingly requires both the global delaying attacker capabilities and the global inserting attacker capabilities. However, this is not necessarily so. If the global inserting attacker is able to insert $n-1$ of his own messages between each of the "good" messages going into a mix, he effectively mounts an $n-1$ attack on each of them.

Another possible attack scenario is when an attacker owns a mix in a free route mix network. He has the capability to delay all the messages going through this mix until the conditions are "just right", i.e. the next mix (on the path of a message) contains $n-1$ of the attacker's messages and will fire as soon as the target message reaches it. Thus, he has the capability to attack the next mix in the route of each of the messages going through the mix he controls. As a result, if the attacker owns all the mixes but one, he is able to compromise the anonymity of any message.

4.2.2 Timed Mix

Parameters: t, the period.

Flushing Algorithm: The mix fires (flushing all the messages) every t seconds.

Message Delay: The minimum delay is $\epsilon + \mu$, which occurs, for instance, if the message is the only one in the mix and arrives just before the mix is due to fire. The maximum delay is $t + c(\epsilon + \mu)$, which is the case when the message arrives just after the firing, then the mix is filled to capacity by other messages, and the target message is the last to leave. The mean delay, assuming a constant rate of arrivals of messages, is $\frac{t}{2} + rt(\epsilon + \mu)$, but is likely to be dominated by $\frac{t}{2}$.

Anonymity: The minimum anonymity is 0 – one message arrives during the entire time period. The maximum anonymity is theoretically infinite, but in practice is limited by the capacity of the mix. The mean anonymity set size (assuming a rate of arrival of r messages/s) is rt.

Note that the threshold and timed mixes are in some sense dual. If the goal of the anonymity system is to guarantee anonymity at the expense of fast message delivery, threshold mixes are good. (This is the scenario of a spy in a hostile country – if the anonymity is compromised, the spy is caught.) On the other hand, if timeliness of message delivery is crucial and anonymity is a bonus, the timed mix is ideal. Notice that if the messages are assumed to arrive at a constant rate, the properties of these mixes are exactly equivalent.

Blending Attack Behaviour: The attack is exact and certain and proceeds as follows: The adversary delays the target message for a maximum of t until the mix fires. He then delivers the target message and blocks all other incoming messages. After another t seconds the mix fires again producing the target message on its own. This takes a maximum[3] of $2t + c\mu + \epsilon + \mu$ and a minimum of $2\epsilon + \mu$ seconds (when the mix was empty and about to fire), and 0 messages. The attack is usually referred to as the trickle attack.

Adversaries: This attack does not require any insertion of messages, so the global delaying attacker is sufficient. We also note that this "attack" can happen naturally in low traffic conditions, so a dishonest mix may be able to attack the next hop of a message simply by holding it until the next hop is empty and about to fire.

4.2.3 Threshold-Or-Timed Mix

Parameters: n, the threshold; t, the period.

Flushing Algorithm: The mix fires (flushing all messages) every t seconds or when n messages accumulate in the mix, whichever comes sooner.

Message Delay: The maximum message delay is $t + n(\epsilon + \mu)$, the minimum is $\epsilon + \mu$.

Anonymity: The minimum anonymity is 0 – one message arrives during the entire time period. The maximum anonymity set size is n.

Blending Attack Behaviour: This design exhibits the worst case behaviour of both the threshold and the timed mixes (and hence is exact and certain). The adversary can choose to perform a trickle attack, a flood attack, or a mixture of the two depending on whether he prefers to wait or send messages. The attack can be

[3]This happens when the mix was full to capacity at the start of the attack, $c\epsilon$ seconds after the last firing.

performed in a minimum of 0 messages in somewhere between $2\epsilon + \mu$ and $2(t + \epsilon + \mu)$ seconds or with a maximum of $2n - 2$ messages, in a maximum of $n(2\epsilon + \mu)$ seconds.

Adversaries: This attack can be performed by either the global inserting or the global delaying attacker.

4.2.4 Threshold-And-Timed Mix

Parameters: n, the threshold; t, the period.

Flushing Algorithm: A mix fires (flushing all messages) every t seconds but only when at least n messages have accumulated in the mix.

Message Delay: The minimum delay is $n(\epsilon + \mu)$, and there is no maximum delay.

Anonymity: The minimum anonymity of this mix is n. The maximum anonymity is in theory infinite, but is limited in practice by the number of messages the mix can hold.

Blending Attack Behaviour: The $n - 1$ attack is still exact and certain and uses a combination of the attacks on the threshold and the timed mixes. It takes a maximum of $c\mu + 2t + n(\epsilon + \mu)$ and a minimum of $n(2\epsilon + \mu)$ seconds; and a maximum of $2(n - 1)$ and a minimum of $n - 1$ messages.

Adversaries: It is clear that to mount an attack on this mix, the attacker has to have the capability both to delay and to insert messages.

4.3 Pool Mixes

The active attacks on all of the mixes described in the previous section are not only exact and certain, but also low-cost. We now examine pool mixes, which give the adversary only an uncertain attack, and substantially increase the cost of it.

4.3.1 Threshold Pool Mix

Parameters: n, the threshold; p, the pool size.

Flushing Algorithm: The mix fires when $n+p$ messages accumulate in the mix. A pool of p messages, chosen uniformly at random from all the messages, is retained in the mix. (Consider these messages as feedback into the mix.) The other n are forwarded on. Note that the threshold is the threshold of messages that must be received to fire again during ongoing operation. For pool mixes this is distinct from the 'threshold' of messages to fire when the mix is completely empty, i.e. $n+p$ for the threshold pool mix being described.

Message Delay: The minimum delay is $\epsilon + \mu$, the maximum delay is infinite since until n new messages arrive, the mix does not fire. Note, however, that even if there is a constant flow of messages and the mix is firing periodically, there is still a small but non-zero probability of a message remaining in the mix for an arbitrarily long time. The mean delay is $1 + \frac{p}{n}$ rounds. If the messages arrive at a rate of r per second, then the mean delay is $(1 + \frac{p}{n})\frac{n}{r}$ seconds.

Anonymity: Here we have to resort to the information theoretic definition of anonymity described in Chapter 3 rather than the standard set-based definition, since the probabilities of the senders (receivers) sending (receiving) the message are unequal. Note also that the anonymity of the message going through a mix depends on the entire history of events that have happened in this mix. Thus, we achieve the maximum anonymity E_{max} when all of the messages that have ever passed through the mix come from different senders. We have already seen this analysis in Chapter 3, the result is given here merely for completeness.

$$E_{max} = \left(1 + \frac{p}{n}\right) \log_2 (n+p) - \frac{p}{n} \log_2 p$$

Of course, we can compare the anonymity above to that of the other mixes by recalling that the effective anonymity set size is given by $2^{E_{max}}$.

The concept of minimum anonymity in pool mixes is slightly more elusive. In the threshold mix case we assumed that all the messages in the batch come from different senders. Under this assumption, regardless of previous history, the minimum sender anonymity of a threshold pool mix is at least $\log_2 n$, and therefore no worse than that of a corresponding threshold mix. Indeed, unless the attacker is able to deduce that the pool contains no honest messages at the start of the current round (and all the messages in the current round come from different senders), the sender anonymity of any message in this round is necessarily greater than $\log_2 n$. A similar statement holds for receiver anonymity: unless the attacker is able to deduce that the pool contains no honest messages at the end of the current round (and all the messages in the current round go to different receivers), the sender anonymity of any message in this round is necessarily greater than $\log_2 n$. Thus the minimum anonymity of a threshold n pool mix is at least as high as that of a simple threshold n mix.

Blending Attack Behaviour: In general, the blending attack has two phases: flushing the mix so that no good messages remain inside it, then submitting the target message and flushing it out onto the network. With simple mixes, one flush was sufficient for each phase. With pool mixes, this is no longer the case. Furthermore, there is now a small but non-zero probability that a given message (e.g. the target message) remains in the mix for an arbitrary length of time. The attack ceases to be certain. It proceeds as follows:

The attacker delays the target message, fills up the pool mix and flushes it. If j of the attacker messages do not come out, he knows that $p - j$ good messages remain inside the pool mix. He now delays all the incoming messages and tries to flush the remaining good messages out of the mix (he can distinguish them as they come out). Of course, this is likely to take more messages than to flush out a threshold mix (see below for details), but the attack is still exact (because if/when the attacker succeeds in getting all the messages to leave the mix, he knows he has done so). When all good messages have been flushed, he just sends in the target message, and flushes the mix as before until the target message comes out. Because the attacker may need to spend an arbitrary amount of resource to flush out the mix completely or to flush out the target message, the attack is uncertain.

Analysis: The threshold pool mix cannot always be flushed of good messages in one round. In particular, if at the time of flushing there are fewer attacker messages than there are good messages inside the mix, then at least one good message will remain inside it. We assume that when the adversary starts the attack, all the messages in the pool are good messages. Hence, any threshold pool mix with $p > n$ is *impossible* to empty of good messages in one round.

For our analysis of blending attacks we consider a pool mix with parameters n, p with G good messages in it being attacked with N bad messages and calculate the probability of k good messages coming out ($k \leq G$) in one round:

$$\frac{\binom{G}{k}\binom{N}{n-k}}{\binom{n+p}{n}}$$

An important special case is when $k = G$, i.e. all the good messages leave the mix in one round.

The probability of this happening is:

$$\frac{\begin{pmatrix} N \\ n-k \end{pmatrix}}{\begin{pmatrix} n+p \\ n \end{pmatrix}} = \frac{N!n!}{(n-G)!(n+p)!}$$

Example 1 *Consider a pool mix with a threshold of 80 and a pool of 20 messages. The average delay of a message through this mix is 1.25 rounds. Assuming the attacker targets a mix with 20 good messages in it, the probability of flushing the mix in one round is 0.0066.*

We wish to calculate the probability of the attacker emptying the mix in ρ rounds. Clearly, given the opportunity to try to empty the mix with p good messages in it for ρ rounds, the attacker will send n of his own messages to the mix during each round. If the attacker succeeds, then there will be no good messages in the mix after round ρ. However, this may happen in many ways: all the good messages might leave during the first round, or stay in the mix until round k and leave then, or leave a few per round. The probabilities of each of these happening are different. Hence, we need to calculate all the possible ways the good messages will leave the mix, then calculate the probability of each of these scenarios happening, and then add up the probabilities. We illustrate this with a simple example.

Suppose the threshold pool mix with parameters n, p has one good message inside it and we wish to find the probability of the attacker getting it to leave the mix in three rounds. There are three possible ways this can happen; these can be represented by $[1,0,0], [0,1,0], [0,0,1]$ where the list $[1,0,0]$ represents one good message leaving during the first round and none during the second and the third. The formula above allows us to calculate the probability (taking the first example) that one good message leaves the mix which contains only that one good message (and $n+p-1$ bad ones). Hence, to calculate the probability of a scenario like $[1,0,0]$ happening, we calculate the probability of each of the rounds happening using the formula above and multiply the probabilities together. We now add the probabilities of each of the scenarios happening to get the probability of a mix being emptied in ρ rounds. Naturally, there are far too many scenarios to do so by hand, so we wrote a program in Haskell [PJH+99] to help us. The program enables us to analyse scenarios such as the following:

Example 2 *Consider a threshold pool mix with a threshold of 20 and a pool of 10 messages. The mean delay of this mix is 1.5 rounds and the anonymity (entropy) is 5.70. When the adversary begins his attack, there are 10 good messages in it. The probability of the attacker emptying this mix in 5 rounds is 0.96. This attack requires 100 attacker messages and* $100(\epsilon + \mu)$ *time.*

A threshold mix with the same mean delay (assuming a uniform rate of arrivals) has $n = 30$, and therefore anonymity of 4.91. Naturally, it can be emptied of good messages with no more than 29 attacker messages. Note that both the extra anonymity and the extra blending attack resistance come from the fact that the variance on the delay of the pool mix is greater than that of the threshold mix – after all, all the messages of a threshold mix have the delay of one round.

Thus, the use of a threshold pool mix in an anonymity system not only makes the $n - 1$ attack uncertain, but also much more expensive (than the threshold mix) in terms of the number of attacker messages which need to be inserted to perform the attack successfully. Note, however, that the attack can still be carried out quickly – it is merely the time it takes for the mix to process the messages employed in the attack.

A variety of results about how the threshold, the pool and the number of rounds affects the probability of the attacker emptying the mix can be calculated similarly. We do not present these here, as we believe that these are not important enough to warrant a detailed investigation. Similarly, the techniques outlined here enable us to perform a number of comparisons between the various mixes. We do not carry out such a study because it will inevitably draw far reaching conclusions while relying on particular assumptions about the distribution of message arrivals. Such a study can only be carried out in the context of a real anonymity system, for which message arrival distributions are known. Instead, we go on to highlight the blending attack properties of other mixes.

4.3.2 Timed Pool Mix

Parameters: t, the period; p, the pool.

Flushing Algorithm: The mix fires every t seconds. A pool of p messages chosen uniformly at random is retained in the mix. The others are forwarded on. If p or fewer messages are in the mix, it does not fire. (Thus, strictly speaking, this is a threshold and timed pool mix, for which the threshold is zero.)

Message Delay: The minimum message delay is ϵ, and there is no maximum delay (if no messages arrive, the messages that are in the pool will never leave the mix). As in the case of the threshold pool mix, there is again a small but non-zero probability that any given message could remain in the mix for an arbitrarily long time even if there are messages flowing through the mix.

Blending Attack Behaviour: Two flavours of blending attack are possible on this mix. The adversary has no control over when the mix fires, but he can choose to add many or just a few messages to each batch. (He prevents all other messages from reaching the mix).

By adding as many messages as possible in one round, he maximizes the probability that after one flush only the attacker messages will be left in the pool. He knows that he has succeeded if p of his messages do not come out with the flush. This approach is very inefficient in terms of the number of messages added by the attacker, N, since the probability of flushing the mix by this method is [4]

$$\frac{N!N!}{(N-G)!(N+p)!}$$

However, this aims to flush the good messages out of the mix in one round, and therefore a maximum of $t + N(\epsilon + \mu)$ seconds.

Example 3 *Consider a timed pool mix with a pool of 5 messages. The probability of flushing the mix in a single round by adding 90 messages is 0.76.*

Alternatively, the attacker can add just one message to each round for many rounds. This is very efficient in the number of messages N (indeed, this attack is much more efficient than the one on the threshold pool mix in terms of messages, but clearly not in terms of time). However, this approach delays all the messages to the mix by at least tN seconds, which is highly observable to the users of the anonymity system. The probability of flushing the mix using this scheme can be calculated using the same method we used for the threshold pool mix (and the same Haskell program).

Example 4 *The timed mix with a pool of 5 messages can be attacked by adding a single message at each round. After 10 rounds the probability of having flushed all the messages out is 0.36.*

The attacker can choose either or a combination of these approaches, based on whether he can send out many messages or delay messages to a mix for a long time. As argued previously, the attacker will probably choose the former attack in order to remain unobservable.

Minimum and Maximum Anonymity: The timed nature of this mix allows an arbitrarily large number (only limited by the memory and bandwidth capacity of the mix) of messages to be mixed together. If we assume a constant rate r of message arrivals, the anonymity provided by this mix must be calculated in just the same

[4] As previously mentioned, in practice there is an upper limit on N due to the finite memory capacity of the mix.

way as for the threshold case (if all the messages come from different senders). To be more precise, the anonymity of the timed pool mix is the same as the anonymity for the threshold pool mix with the same pool, and taking $n = rt$.

The minimum anonymity of a timed pool p mix (recall that in our analysis of the threshold pool mix we argued that the lowest possible anonymity coming just from the last batch of messages can be seen as the minimum anonymity of the mix) is 0, and therefore clearly smaller than that of a threshold pool p mix. The maximum anonymity is, in theory, infinite, in practice it is limited by the capacity of the mix. If the capacity of the mix is c, then the maximum anonymity of it is (at each round c messages arrive from different senders):

$$E_{max} = \left(1 + \frac{p}{c-p}\right) \log_2 c - \frac{p}{c-p} \log_2 p$$

If we do not assume a constant rate of message arrivals, then the anonymity of a message going through this mix depends on the messages which have arrived at this mix previously. Similarly to the threshold pool mix case, the total number (and the senders) of messages which have passed through it is important. However, here the number of messages which have arrived during each round is also important – there is a difference between all the messages having arrived during one round vs the same number of messages having arrived during different rounds. Hence, if we have a *history* of the mix (a record of its operation ever since it was started), we can calculate the anonymity of the timed pool mix for that particular case. The history of a mix includes the senders of each message and the number of messages that got mixed together in each batch.

Anonymity of the Timed Pool Mix – the General Case First of all, assume that all messages arrive at the pool mix directly from senders. Furthermore, for the purposes of comparison, assume that the senders of all the messages are distinct.

We will proceed as follows: first, we consider a message inside the mix at round k (we do not care whether this message leaves the mix or not) and calculate the probabilities that it had been sent by each of the senders who sent a message at round j, $j < k$. We will then have a probability distribution of senders who could have sent the message. Taking the entropy, $\sum p \log_2 p$, of this probability distribution, will give the sender anonymity.

We have already used this method to analyse the threshold pool mix in Chapter 3. However, the difference between the threshold pool mix and the timed pool mix is that in the former case the number of messages arriving at every round is the same (the threshold, n), while in the latter case it varies.

Given the mix at round k and a history of the numbers of messages which arrived at the mix during each of the rounds $[N_1, \ldots, N_k]$, let us calculate the probability of a message from rounds $1 \ldots k$ still being in the mix.

The probability that a particular message that is in the mix at round k has entered the mix at round k is:

$$p_k = \frac{N_k}{N_k + p}$$

Thus, a sender who sent a message at round k (there were N_k of them) was the sender of this message with probability:

$$p_{(s,k)} = \frac{1}{N_k + p}$$

Similarly, the probability that a message that is in the mix at round k has entered the mix at round $k - 1$ is:

$$p_{k-1} = \frac{p}{N_k + p} \left(\frac{N_{k-1}}{N_{k-1} + p} \right)$$

and the probability that a particular sender at round $k - 1$ sent this message is:

$$p_{k-1} = \frac{p}{N_k + p} \left(\frac{1}{N_{k-1} + p} \right)$$

If the message was in the mix before the first flush (round zero), the probability of it staying until round k is:

$$p_0 = \prod_{i=1}^{k} \left(\frac{n}{N_i + p} \right)$$

Of course, if the message came from round zero, then it must have been "sent" by the mix itself (recall this is the way a pool mix starts operating).

Hence, the sender anonymity probability distribution for a message which exited the mix at round k is:

$$p_{(s,0)} = \prod_{i=1}^{k} \frac{p}{N_i + p}$$

$$p_{(s,x)} = \frac{1}{N_x + p} \prod_{i=x+1}^{k} \frac{p}{N_i + p}$$

$$p_{(s,k)} = \frac{1}{N_k + p}$$

where $0 < x < k$.

Now we can calculate the entropy of the probability distribution (note that there is only one sender at round zero and N_i senders at round i).

$$E_k = -\frac{N_k}{N_k + p} \log_2 \frac{1}{N_k + p} - \left(\prod_{i=1}^{k} \frac{p}{N_i + p} \right) \log_2 \left(\prod_{i=1}^{k} \frac{p}{N_i + p} \right) + \\ - \sum_{x=1}^{k-1} \left(\frac{N_x}{N_x + p} \left(\prod_{i=x+1}^{k} \frac{p}{N_i + p} \right) \log_2 \left(\frac{1}{N_x + p} \prod_{i=x+1}^{k} \frac{p}{N_i + p} \right) \right)$$

At this point it may be helpful to refer back to Section 3.4 and observe how the above expression resembles the one for anonymity of the threshold pool mix.

An Alternative Method We can also derive the anonymity of a timed pool mix in a different way. This has largely been the work of other people, and is presented here briefly merely to show that the method above has been verified by other means and for completeness. It is based on the formula for composition of mixes introduced by George Danezis in Section 3.2 of [SD02], and has been suggested by Richard Newman as a different way of analysing the threshold pool mix.

When Richard Newman proposed this alternative method for analysing the threshold pool mix, it was not clear that the results obtained using this method and the method presented in Section 3.4 were the same. I proceeded to show that they were, and apply this method to the timed pool mix (as presented below). Unfortunately, we have not been able to prove analytically that the two methods give the same results for the timed pool mix, but the results coming from implementing the two methods suggest that they are consistent. The method proposed by Newman is roughly as follows (the reader will need to refer back to [SD02]):

In Section 3.2 of [SD02] the formula for composition of mixes was presented:

$$S_{total} = S_{sec} + \sum_{0<i<l} p_i S_i$$

Intuitively, this says that the anonymity of a message in the mix is the inherent entropy of the mix (E_{mix}) plus the mean anonymity of all the messages inside the mix.

We can use this formula in the following way. Consider the mix at round i. It has, essentially, 2 inputs: messages from round $i-1$ and messages from the senders. There are p messages from round $i-1$ and they have entropy E_{i-1}, ($E_0 = 0$). There are N_i messages from the senders, each having entropy zero. We now separate out the mixing of the N_i user messages together into a separate mix. Consider the pool mix at round k as two mixes: a threshold mix which mixes the user messages together and then another threshold mix which mixes the p messages from the previous round with the N_i messages from first mix. It is trivial to see that the first threshold mix produces N_i outputs, each with an entropy of $log_2(N_i)$.

A message coming out of the second mix could have come from two places: from the previous round (call the probability of this p_i) or from the first mix. So,

$$p_i = \tfrac{p}{N_i+p} \quad \text{and} \quad 1 - p_i = \tfrac{N_i}{N_i+p}$$

Then, the inherent entropy of the second mix at round i is

$$E_{i_{mix}} = -p_i \log_2 p_i - (1 - p_i)\log_2(1 - p_i)$$

Using the above formula we can now obtain the anonymity of a timed pool mix after r rounds (it is the entropy of the second mix plus the mean of the entropy of messages from the previous round and the entropy of the N_i messages from the first mix):

$$E_k = E_{k_{mix}} + p_k E_{k-1} - (1 - p_k)\log_2 N_k$$

This can be expanded as:

$$E_k = E_{k_{mix}} - (1 - p_k)\log_2 N_k + \sum_{x=1}^{k-1} E_{x_{mix}} \prod_{i=x+1}^{k} p_i - \sum_{x=1}^{k}(1 - p_x)\log_2 N_x \prod_{i=x+1}^{k} p_i$$

It is important to notice that to calculate the anonymity of a timed pool mix (using either method), we need to know the number of messages that had arrived at the mix during each of the previous rounds.

Now suppose we wish to analyse the anonymity of a timed pool mix at round K. It is clear that messages which have passed through the mix a long time ago will not contribute much to the current anonymity.[5] Thus, we can approximate the total anonymity by supposing that only the last k' of the K rounds contribute to the anonymity.

Suppose that message inter-arrival times during time period T follow some probability distribution (an exponential distribution, for example) and that N messages arrive in total. Furthermore, take T to be some multiple of t, the time parameter of the mix, so that the N messages arrive over $k' = \frac{T}{t}$ rounds. Now we can simply enumerate all the possible ways (we call these histories) that N messages can arrive in k' rounds, and calculate the probability of each one. Now all that is left to do is to calculate the anonymity of each one of the histories, the probability of each history occurring given the exponential (or in fact any other) distribution and compute the mean anonymity.

Implementation and Results This calculation is far too tedious to do by hand, so a short program was written in Haskell [PJH+99] to enumerate all the possible histories, calculate their probabilities using an exponential distribution and determine the anonymity of the pool mix. The program calculates anonymity using the two methods presented earlier and gives the same results in each of our test cases. This leads us to believe that the two methods are indeed consistent with each other.

Note that this approach is not as precise as the one used for the threshold pool mix – we were unable to come up with a closed form for the anonymity of a timed pool mix as the number of rounds grows large. However, we consider the method used above satisfactory and practical.

The program enables a comparison with a threshold pool mix, for which we take the same scenario and calculate its anonymity after r rounds.

We performed the comparison in the following settings: we took a timed pool mix with a period of 1 minute and a pool of 2 messages over 5 rounds with 10 messages arriving to it during these 5 rounds. The mean inter-arrival time for the Poisson process was taken to be 1 message/min.

We then compared it to a threshold pool mix with the same volume of traffic flowing through it over 5 rounds. We took the threshold to be 2 messages. Note that this corresponds to the anonymity of the timed mix when the history of the timed mix is $[2, 2, 2, 2, 2]$. The anonymity of the threshold pool mix was 2.91 bits, whilst the anonymity of the timed pool mix was 3.012 bits. If we model the arrival of messages using a zipf distribution, the anonymity of the timed pool mix is 3.07 bits.

[5] Implementors of mixes have also suggested that including a timeout clause such as "message should not be delayed for more than 10 rounds" would make them feel more comfortable.

We do not currently wish to make any general claims about the anonymity of timed mixes versus threshold mixes. Such a statement would be much more a product of the various assumptions we have made than the batching strategies themselves. In particular, as we have seen above, the probability distribution of message arrivals to the mix is an important factor. Also, we have measured the anonymity of the timed pool mix as the anonymity of the message passing through it after the particular history has happened. Perhaps if we measure it as the average anonymity of each message over the whole history, the results will favour the threshold pool mix. Finally, a rigorous comparison of the two mixes would also have to take account of the average delay of the messages through the two mixes, and possibly other factors. Here we merely illustrate that the method which we use to work out the anonymity of a timed mix is powerful enough to enable such a comparison.

4.3.3 Timed Dynamic-Pool Mix (Cottrell Mix)

Parameters: t, the period; p_{min} the minimum pool; $frac$, the fraction of messages to be sent; n, the threshold

Flushing Algorithm: The mix fires every t seconds, provided there are $n + p_{min}$ messages in the mix. We call the actual number of messages in the mix at the time of firing m,$(m - p_{min} \geq n)$. Instead of sending $m - p_{min}$ messages (as in a timed-and-threshold constant-pool mix), the mix sends the greater of 1 and $\lfloor (m - p_{min}) * frac \rfloor$ messages, and retains the rest in the pool.

If $n = 1$, this is the mix that has been used in the Mixmaster remailer system for years [Cot02, Cot94].

When messages arrive at a constant rate of one per t, Cottrell mixes are equivalent to both timed pool mixes and threshold-1 constant-pool mixes. Specifically, if the rate r of message arrival is $1/t$, the mix will forward 1 message in every period and retain p_{min} in the pool. For a general Cottrell mix, if the messages arrive at a constant rate of $n * frac/t$ and $\lceil n * frac \rceil = n * frac$, then this is equivalent to a constant-pool mix with a pool of $p_{min} + n(1 - frac)$ (and threshold $n * frac$).

Message Delay: As with the other pool mixes, the minimum delay is ϵ, and there is no upper limit on the delay. The mean delay depends on the future rate of arrival of messages into the mix; it is at least as big as that of a timed constant-pool mix with the same pool size.

Anonymity: The dynamic-pool mix has identical minimum and maximum anonymity properties to a timed constant-pool mix. We could similarly use a record of the mix's activity to calculate the anonymity of a message passing through it (although the calculation would be slightly different). Qualitatively, we may note that the anonymity provided by the mix would be higher than that provided by a timed constant-pool mix: As the number of messages in the mix goes up, *frac* keeps the chance of the message remaining in the mix constant, whereas it decreases in the case of the timed constant-pool mix. Naturally, the extra anonymity comes from the extra message delay.

Blending Attack Behaviour: The introduction of the dynamic parameter *frac* has several new consequences compared to the timed constant-pool mix.

Firstly, the maximum probability of a particular message exiting the mix in a single round does not asymptotically approach 1 as the number of messages goes up, instead it approaches *frac*. Hence, even if the attacker sends in a large number of his messages to the mix, the probability of flushing G good messages out of it in one round is $frac^G$.

Therefore there is no possibility of flushing the mix with high probability in one flush: the first of the two blending attacks on timed constant-pool mixes is blocked. As already noted, it is similarly more resistant to flooding than a constant-pool threshold mix.

Secondly, the attacker has to find out how many messages are in the mix. Of course, the number of good messages in the mix is easy to calculate from the number of messages that come out.

Finally, the number of messages inside the pool mix may be arbitrarily high and cannot be reduced to below p_{min} in one round. Therefore, if we wish to send a message that is harder for the active attacker to track, we should send it at a time of higher traffic – thereby increasing the cost (in terms of messages or time) of attempted attacks on it.[6]

Thus timed dynamic-pool mixes require the attacker to delay all the traffic to the mix for a substantial number of rounds, and therefore for a substantial time. Again, our Haskell program allows us to compute the probability of the attacker emptying the mix in a variety of scenarios:

This example shows that the protection this mix provides against blending attacks is still not very strong. Indeed, as the number of messages inserted by the attacker at each round goes up, the probability of a successful attack tends to $((1 - frac)^k)^G$ where k is the number of rounds. Ideally we would like a mix for which the anonymity

[6]Unfortunately, an attacker capable of arbitrary message insertions, as we have been assuming, will make it hard to determine times of higher legitimate traffic.

Messages	Rounds	Probability of Success
30	3	0.14
60	3	0.28
300	3	0.46
600	3	0.48
30	5	0.474
60	5	0.66
300	5	0.82
600	5	0.84
30	7	0.73
60	7	0.87
300	7	0.95
600	7	0.95

Table 4.1: Emptying the Cottrell Mix

does not go to 0 even as the cost of the attack goes up. We would also be happier with a mix where the probability of a successful attack is proportional to $\frac{1}{P(k)}$ where P is a polynomial of a small degree.

4.4 Related Work

There are relatively few attempts to deal with active attacks in the literature. First and foremost, Kesdogan proposes two schemes for dealing with such attacks in [KEB98]. We look at these in more detail.

4.4.1 Limitations of Stop-and-Go Mixes

The first of the proposals outlined in [KEB98] is a scheme for an anonymity system based on a mix cascade that requires authentication. This is based on a stronger assumption – that the attacker cannot mount a distributed $n-1$ attack where the hostile messages come from different users. Unfortunately, it seems very hard to re-use this idea in free route networks: since there is no centralized input location at which we can do authentication, compromised mixes could claim to have done authentication on traffic they instead generate themselves.

The second scheme, Stop-and-Go mixes, involves the sender estimating a time window during which the message is allowed to arrive at each of the mixes making up the route of the message. If the message arrives during that time period it will be forwarded on; otherwise it will be discarded. Therefore the attacker's ability to delay messages is much more limited, so active attacks are harder. The attack is uncertain, and the authors argue that the probability of executing the attack successfully is very low.

However, the scheme relies on the users being able to accurately calculate a security parameter that depends on their estimate of the rate of arrival of messages to the mixes that the target message will go through *at the time it will travel through them.* In other words, the users need to be able to predict the traffic levels in the system in the future to remain anonymous. This is likely to be exploited by the attacker (the exact details depend, of course, on how the security parameter is calculated). Furthermore, an active attacker is able to arbitrarily affect the levels of traffic in the mixes which the target message goes through.

Thus, we defer the evaluation of Stop-and-Go mixes to future work as the precise details of parts of the protocol, crucial to the security of the system, have not yet been specified.

4.4.2 Other Work on Batching Mixes

Mixes have been studied fairly extensively in the literature. Chaum proposed the threshold mix. The pool mixes were described by Cottrell in [Cot94]; in particular, he describes the threshold pool, the timed pool and the Cottrell mixes. The last was the mix used in the Mixmaster system for a long time. There is an interesting discussion of the anonymity provided by two types of mixes in the Babel paper [GT96]. They do not use an information theoretic metric of anonymity, nor do they analyse the same mixes (their mix is a variant of a pool mix where a message is never delayed for more than one round), but this work is one of the earliest attempts at quantitatively analysing anonymity of different mixes.

A substantial account of the functionality of batching mixes is given in Jerichow's PhD thesis [Jer00]. She outlines very strong requirements on the mixes, essentially adopting the active attacker with many compromised mixes model. She then proceeds to analyse the extent to which we can protect against the $n-1$ attack, investigating a broad range of attacks and prevention measures. However, the study is broad and very often open-ended with little in the way of concrete results (e.g. on page 121 the author claims that in pool mixes "the size of the anonymity set becomes bigger").

There is another strand of research devoted to mixes. This is a collection of papers (see [Jak99, OA00, DK00, FS01, Nef01, NSN03, Möl03, Abe98]) which focuses on designing mixes which provide strong properties like verifiability. Such properties are useful in applications like electronic voting where it is extremely important that every message gets to its "destination".

One new strand of research shows that bitwise unlinkability does not have to be achieved by public key decryption as in the original scheme by Chaum. Instead, re-encryption can be used (using, for example the El-Gamal public key encryption [ElG85]). Traditional re-encryption requires the mixes to possess a public key for the message in order to re-encrypt it. A more recent scheme, universal re-encryption [GJJS, Dan03a, GJ03] can be used to allow mixes to provide bitwise unlinkability without knowledge of a public key. Unfortunately, mixes must provide other properties, e.g. resistance to tagging attacks and replay prevention. Universal re-encryption does not support these naturally, and this currently renders anonymity system designs based on them impractical or simply insecure. Nevertheless, we feel that this is an important research area which is likely to provide interesting results in the future.

4.4.3 Deployed Mixes

The analysis we have presented in this chapter has been very much from a theoretical point of view, but the mixes which we have described are used in practice.

In particular, the Mixmaster system uses the Cottrell mix. The period of the mix is set by the remailer operator and so many mixes operate with different parameters. Typically, the period is between 5 and 30 minutes. Similarly, the number of messages arriving at a mix per round varies between zero and around a hundred messages. This is part of the reason for including the minimum and the maximum anonymity properties of mixes in this chapter and not relying on assumptions about distribution of traffic. Rigorous analysis of the performance of the various mixes under realistic traffic is future work.

4.5 Summary

The main contribution of this chapter is the comparison of the blending attack behaviour of the different mixes. The results are summarised in Table 4.5.

Several things ought to be noted. Firstly, two different blending attacks are possible on the Threshold or Timed mix and on the Timed Pool mix; we presented the results for the attack which takes less time and is therefore less observable, although more expensive in the number of attacker messages. Secondly, the probability of success is equal to $((1 - \mathit{frac})^k)^G$ only when $N = \infty$, i.e. an infinite number of attacker messages per round are sent to the mix. When a finite number of messages is sent, the probability of success is lower; see Section 4.3.3. Also, the attacks on the timed

		Cost of Blending attack		Prob. of success
		Time	Messages	
Simple	Threshold	$(4n-1)\epsilon + 2n\mu$	n	1
	Timed	$2t + (c+1)\mu + \epsilon$	0	1
	Thresh. or Time	$n(2\epsilon + \mu)$	n	1
	Thresh. & Time	$c\mu + 2t + n(\epsilon + \mu)$	n	1
Pool	Threshold	$\lambda(\epsilon + \mu)$	$> n$	See 4.3.1
	Timed	$2t + \lambda'(\epsilon + \mu)$	N	$\frac{N!N!}{(N-G)!(N+p)!}$
	Cottrell	$(k+1)t + \lambda''(\epsilon + \mu)$	$(k+1)N$	$((1-frac)^k)^G \leq$

n	threshold
t	period
p	pool
N	number of attacker messages
G	number of good messages
k	number of rounds the attack lasts
λ,λ',λ'',	constant factors

Table 4.2: Summary of blending attack properties of various mixes.

pool mixes are calculated for the case when $N \gg G$, if this is not true the attack might take an extra round, t seconds (+ constant factor for message processing). Finally, we did not include the precise numbers for the time needed for processing of messages as the duration of the attack is dominated by the number of periods t it takes.

In addition to providing the blending attack properties of the mixes, we provided descriptions of them and considered their anonymity and delay. We have also proposed a new method of working out the anonymity of timed pool mixes along with an implementation of this method. We do not provide a full comparison as the anonymity of the timed mixes depends heavily on the patterns of message arrivals to them.

We concluded that the timed dynamic pool mix provides a reasonable resistance to blending attacks, both in terms of the number of attacker messages required and the time the mix has to be isolated from the rest of the network in order to get a reasonable probability of a successful attack. In addition, we provided commentary on the popular continuous mix, the Stop-and-Go mix, and on its blending attack properties.

Chapter 5

Generalising Mixes and the Binomial Mix

"*When one cannot invent, one must at least improve.*"
— fortune cookie

In this chapter we continue looking at batching mixes, proposing a common framework for expressing them. This leads us to a design of yet another mix with an interesting property – it hides the number of messages inside it from the attacker. This chapter concludes our study of single mixes. In Chapter 6 we go on to examine mix networks as a single entity and try to formalise some of their properties.

5.1 Expressing Mixes as Functions

In the process of analysing individual batching mixes in the previous chapter it became clear that they all contain some common features. Their flushing algorithm, which determines the time *when* flushing happens and *how many* messages are sent out, depends on two factors: the time that has passed since the last flushing and the number of messages inside the mix. Naturally, this influences both the anonymity and the message delay properties of the mix. In existing literature the batching strategy is often expressed by giving the algorithm for collecting messages together and forwarding them on to the next hop. This is cumbersome and, although it corresponds well to the implementor's view of the mix, it does not isolate the details important for the analysis of the mix well at all. Instead, we propose a simple formalism for expressing mixes which also allows quick (qualitative) comparison.

We express the batching strategy of mixes as functions $P : \mathbb{N} \to [0, 1]$ from the number of messages inside the mix (m) to the fraction of messages to be sent to their next hop. Examining the graph of such a function is a very nice way to gain the intuition behind the operations of a mix. We also need to specify when the mix flushes, hence we include a parameter t which specifies how often the batching strategy is executed. For timed mixes, this is the time between flushes (known as the period, t). For threshold mixes we want the flushing to happen as soon as the appropriate threshold is achieved, hence the batching strategy to be executed constantly; $t = 0$.

Existing mixes can be expressed in this framework as follows. The function of each mix $P(m)$ is illustrated in Figures 5.1, 5.2, 5.3, 5.4, 5.5.

- Threshold mix, threshold n: This mix flushes *as soon as* n messages are accumulated ($t = 0$) and sends out all the messages ($P(n) = 1 \wedge \forall n'.(n' \neq n) \Rightarrow P(n') = 0$). Hence it is represented on a graph by a single point.
- Timed mix, period t: This mix flushes all the messages which are inside it at the time of flushing. Hence, $\forall m.P(m) = 1$.
- Timed pool mix, period t and pool p: This mix flushes when the time t expires and sends out $m - p$ messages. Hence $\forall m.P(m) = 1 - p/m$.
- Threshold pool mix, threshold n, pool p: This mix flushes as soon as the threshold n is reached ($t = 0$), and sends out n messages. Hence, $\forall m.((m < n + p) \Rightarrow P(m) = 0) \wedge (m = n + p) \Rightarrow P(m) = n/(n + p)$.
- Timed dynamic pool mix (Cottrell mix), minimum pool p_{min}, period t, fraction $frac$[1]: This mix flushes every time period t, sending out $\lfloor (m - p_{min}) frac \rfloor$ messages. Hence, $\forall m.P(m) = \frac{(m-p_{min})frac}{m}$.

At this point it is worth noting that all these flushing algorithms are stateless, i.e. the fraction (and therefore the number) of messages forwarded on by the mix at each round depends only on the number of messages inside it during *this* round and not the number of messages in the mix during the previous round.

5.2 Extensions Arising Out of the Framework

The natural way to proceed is to say that a mix is an arbitrary function from the number of messages inside the mix to the percentage of messages to be flushed. What does this gain us?

Throughout the mix literature, a tradeoff between message delay and anonymity can clearly be seen. Indeed, as shown first in [SD02] (and in Chapter 3), the pool

[1]For clarity of exposition, we use a timed dynamic pool mix with a threshold of zero.

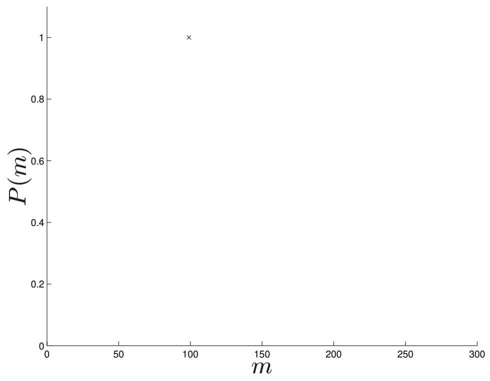


Figure 5.1: Threshold mix, $n = 100$

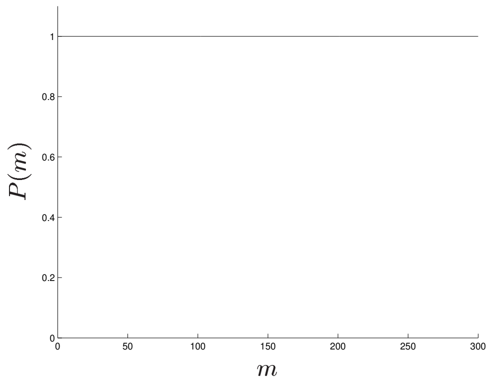


Figure 5.2: Timed mix

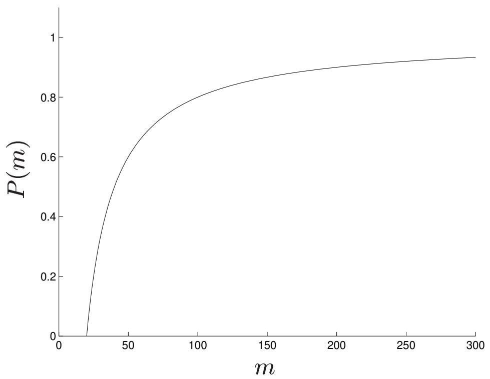


Figure 5.3: Timed pool mix, $p = 20$

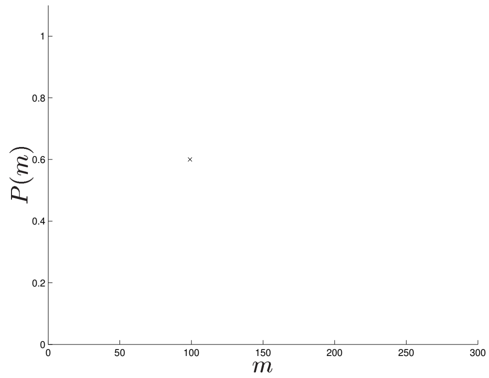


Figure 5.4: Threshold pool mix, $n = 100, p = 67$

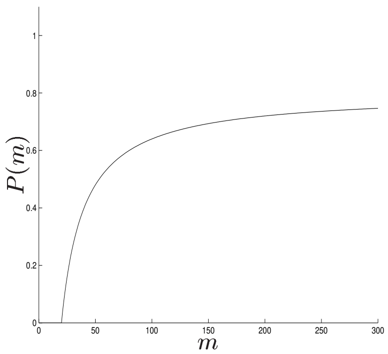


Figure 5.5: Timed dynamic pool mix (Cottrell mix), $p_{min} = 20, frac = 0.8$

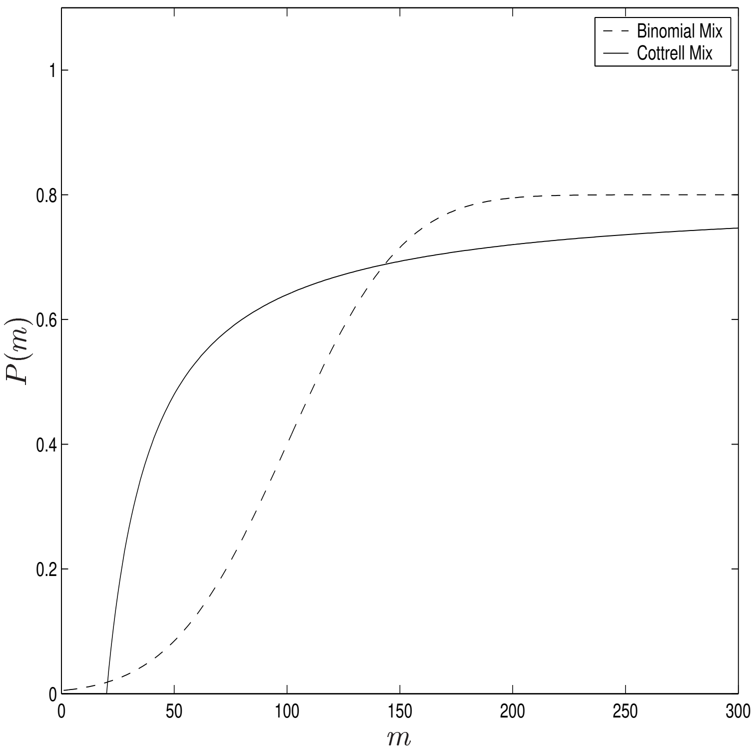


Figure 5.6: P(m) of a Cottrell mix with $frac = 0.8$ vs a normal cumulative distribution function with $\mu = 100, \sigma = 40$.

mix gains more anonymity from higher average delay as compared to the threshold mix. Expressing the mix batching strategy as a function allows us to define an arbitrary tradeoff between anonymity and message delay. We now go on to examine a particular mix function.

Suppose that we would like to develop a mix which has same the properties in low and high traffic conditions as a particular timed dynamic pool mix, but which gains more anonymity for a longer delay in low traffic conditions. This is easily possible – all one needs to do is to invent a suitable function.

In Figure 5.6 we show a comparison between the Cottrell mix and our new mix, which is defined by a suitable function (normal cumulative distribution function[2]).

The normal cumulative distribution function has desirable properties. It grows smoothly at low n, providing a larger anonymity when the mix contains few messages. This is achieved at the cost of increasing the delay in low traffic conditions. On

[2]This function has no closed form, hence is computed numerically.

the other hand, when the number of mixed messages is large enough, the cumulative function improves the delay over the Cottrell function.

5.3 Adding Randomization: The Binomial Mix

In this section we add randomness to the batching strategy of the mix. Suppose we treat the result of the function $P(m)$ not as a fraction, but as a probability. We can then use it as the bias of a coin, which we toss for each message inside the mix. A head indicates that this message should be sent out during this round, a tail – that it should remain in the mix. It is clear that the expected number of messages sent out at a particular round is $mP(m)$; the actual number is picked from a binomial distribution with bias $P(m)$. Due to this property, we call this mix the *binomial mix*.

Adding randomness has the effect of hiding the number of messages which are in the mix at the time it flushes. This has two desirable consequences:

- It makes it more difficult to decide whether a message was a dummy message or a real message. We have so far not touched on the issue of dummy traffic in a mix network. If dummy traffic is added in a mix network, the attacker performing a blending attack will see several messages leaving a mix – the target message along with some dummy messages. Now, at some point the dummies will get discarded by the mixes in the mix network. (None of the cover traffic schemes propose to send the dummies all the way to the receivers. This is because such traffic would have the plaintext exposed to the attacker, who is likely to be able to differentiate real messages from the automatically generated dummies. Furthermore, such traffic will annoy the receivers.) If the attacker knows how many messages are in the mix, he can trivially deduce whether the mix has discarded a certain message (because it was a dummy) or kept it (because it was the real message). In a threshold mix, he arranges for there to be $n - 1$ real (his own) messages in the mix and sends in the target message. If the mix flushes, the message was real, if it does not, it was a dummy message. Similar decision procedures can be executed for the timed, threshold pool, timed pool and Cottrell mixes.
- It makes the blending attack on the mix probabilistic rather than exact. This is explained in more detail below.

Before we proceed to examine the blending attack properties of this mix, we summarize its batching algorithm for completeness.

Binomial mix: t, $P(m)$ = cumulative distribution function of the normal distribution.

$$P(m|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{m} e^{\frac{-(t-\mu)^2}{2\sigma^2}} dt$$

The mix flushes every t seconds; at each flush the value of $P(m)$ is determined from the function, and is used as the bias of a coin tossed for each message. A head indicates that the message should be forwarded on to the next hop, a tail that it should stay in the mix.

The analysis of the anonymity of this mix is rather more difficult than that of the timed mixes because the number of messages it sends out is not a deterministic function of the state of the mix. Thus, to compute the average anonymity of this mix, we not only need to average over all the possible histories of the mix, but also over all mix "behaviours". (Recall that at each round the mix can potentially send out anywhere between zero and all the messages in the mix).

5.3.1 Blending Attack on the Binomial Mix

The flooding strategy.

The goal of the attacker is to trace a particular message (the target message) that is sent by a user to the mix. The actions of the attacker can be divided into two phases: the *emptying* phase and the *flushing* phase.

The emptying phase. During this stage of the attack, the goal of the attacker is to remove all unknown messages contained in the pool, while preventing new unknown messages from going into the mix. In order to flush the mix, the attacker sends some number of messages to the mix at every round. In the analysis that follows, we will assume this value, N_T, to be such that it forces $P(m)$ to be equal to its asymptotic value, p_{asym}, which gives us an upper bound on the effectiveness of the attack. The attacker can compute the probability of success of his actual attack by using the value of $P(N_T)$ instead of p_{asym} in the formulae below.

If the attacker wants to empty the mix with probability $1-\epsilon$, then he will have to flood the mix for some number of rounds. We call this k.

The formula that can be used to estimate the number of rounds needed to flush all unknown messages with probability $1-\epsilon$ is:

$$(1-(1-p_{asym})^k)^G \geq 1-\epsilon \tag{5.1}$$

where G is the number of good messages in the mix initially. If the attacker does not have any information about n he will have to assume $G = N_{asym}$, the maximum number of messages a mix may contain (worst case scenario for the attacker).

We can rewrite this as:

$$k = \frac{ln(1-(1-\epsilon)^{1/n})}{ln(1-p_{asym})}$$

which for small ϵ simplifies to:

$$k = \frac{ln(\epsilon/n)}{ln(1-p_{asym})}$$

Cost of emptying the mix. We compute the cost of this phase of the attack taking into account the following:

- Number of messages the attacker has to send to the mix.
- Time needed to complete the operation.

Number of messages the attacker has to send to the mix. In the first round the attacker has to send N_T messages, to ensure that the function $P(m)$ takes a value close to p_{asym}, and therefore the probability of each message leaving is close to maximum. In the following rounds, it is enough to send as many messages as the mix outputs. Note that if $G + N_T$ is bigger than N_{asym}, then some messages may be dropped (depending on the implementation) and the mix will contain N_{asym} messages. We do not specify this, and it does not affect our analysis.

Thus, for the first round the attacker sends N_T messages, and the following rounds he sends $(N_T + G)p_{asym}$ messages on average. The total number of messages sent during this process is:

$$\text{Number of messages sent} = N_T + (k-1)(N_T + G)p_{asym} \tag{5.2}$$

Time needed to complete the operation. This is a timed mix, so the attacker has to wait t units of time for each round. This is highly likely to dominate the duration of the attack, so we ignore the time it takes for the messages to arrive and be processed. Therefore, the total time needed is around kt time units.

The flushing phase Once the mix has been emptied of unknown messages, the attacker sends the target message to the mix. Now, he has to keep on delaying other incoming unknown messages and also send messages to make the mix flush the target.

The number of rounds needed to flush the message is, on average, $k' = \frac{1}{p_{asym}}$. The cost of this phase is computed according to the previous parameters.

Number of messages the attacker has to send to the mix Assuming that the attacker carries out this phase of the attack immediately after the emptying phase, the number of messages needed in the first round is $(N_T + G - 1)p_{asym}$, and in the following ones $(N_T + G)p_{asym}$. The total number of messages is:

$$p_{asym}(N_T + G - 1 + (k' - 1)(N_T + G)) = p_{asym}(k'N_T + k'G - 1) \tag{5.3}$$

The other two parameters are computed in the same way as in the emptying phase, taking into account the new value of k'.

Total cost of the attack Clearly, if the attacker has chosen to empty the mix in k rounds and flush the message in k' rounds, then the total time the attack lasts is $(k + k')t$, the total number of messages is:

$$N_T + (k - 1)(N_T + G)p_{asym} + p_{asym}(k'N_T + k'G - 1)$$

and the probability of success is:

$$((1 - (1 - p_{asym})^k)^G)(1 - p_{asym})^{k'}$$

Guessing the number of messages within the mix with an active attack The attacker can use the flooding strategy (emptying phase only) in order to determine the number of messages contained in the pool of the mix. However, it is rather expensive and is not a good use of the resources of the attacker.

Probabilistic success Note that, due to the probabilistic nature of the binomial mix, the attacker only succeeds with probability $1 - \epsilon$. Therefore, with probability ϵ there is at least one unknown message in the mix. In this particular case, the attacker can detect his failure if during the flushing phase more than one unknown message leaves the mix in the same round (and there is no dummy traffic policy), which happens with probability p_{asym}^2 for the case of one unknown message staying during the emptying phase (the most probable case). With probability $p_{asym}(1 - p_{asym})$ the target message leaves the mix alone, and the attack is successful. Also with probability $p_{asym}(1 - p_{asym})$, the other unknown message leaves the mix first, and the attacker follows a message that is not the target without noticing. Finally, with probability $(1-p_{asym})^2$, both messages stay in the pool and the situation is repeated in the next round.

5.4 Related Work

There has not been a huge amount of work on analysing single mixes, as mentioned in Section 4.4. This chapter is based on the paper "Generalizing Mixes" [DS03b, SN03] (coauthored with Diaz) where the new framework of expressing mixes and the analysis of timed pool mix was presented. Since then, more work has been done by Diaz [DP04a, DP04b].

We also mentioned that the idea of picking the number of messages to be sent out from a probability distribution was proposed in the description of the Babel system [GT96], though the metric they used for the analysis of the anonymity of the mixes they considered was misleading. It effectively amounted to using the largest value in the anonymity probability distribution to measure the anonymity.

5.5 Summary

In this chapter we proposed a framework for expressing batching mixes as functions from the number of messages in the mix to the probability of a message being sent on. This turned out to be useful and has lead to to a design of a new mix – the binomial mix.

We now leave the issue of anonymity of single mixes and go on to examine mix networks.

Chapter 6

From Mixes to Mix Networks

> *"Privacy is part philosophy, some semantics, and much pure passion"*
>
> — Alan Westin, 1967

In the two previous chapters we examined properties of various single mixes in the context of the global passive and global active adversaries. In this chapter we give a definition of the anonymity of mix networks. In particular, we approach the issue of calculating the anonymity probability distribution for a message which has travelled through a free route mix network composed of threshold mixes as viewed by the global passive adversary.

First of all, we present a formal framework for analysing the anonymity of a run of a network of threshold mixes (given an observation made by the attacker). We go on to present a method for defining the anonymity probability distribution and hence the entropy-based metric which we saw in Chapter 3. Finally, we briefly use the model to illustrate our analysis with a simple example.

Our main aim is to develop a principled way of defining the anonymity of mix networks. It is computationally infeasible to do calculations on real anonymity networks, but our definition should provide a good basis for future work on approximate calculations.

6.1 The Mix Network

First of all, let us outline the overall characteristics of the anonymity system which we are modelling.

We consider an anonymity system which consists of senders which send messages to receivers via a mix network made up of the simplest type of mix – threshold mix. The choice of sequence of mixes through which each message passes through is left to the sender (in contrast to, say, MorphMix [RP02]). Once this choice has been made, the senders construct onions in the usual fashion. Our model is flexible enough to accommodate any "not quite free route" mix network – routes defined by sequences of mixes in which one mix occurs twice consecutively in the sequence are not allowed. More restrictive mix networks (and in particular mix cascades) are also easily modelled within our framework; a fully free route mix network requires a slight extension. We discuss this later in the chapter.

We analyse the anonymity this system provides against the global passive adversary who is able to observe the network very close to each of the senders, receivers and mixes (rather than only in the core of the network). Thus, each message (i.e. sender to mix, mix to mix and mix to receiver, not sender to receiver) is observed twice – at its source and destination. Furthermore, we assume that the adversary cannot break the encryption used by the mixes and does not possess any of the mixes' private keys.

We do not model several well-understood aspects of mix networks. These include replay prevention by mixes, retrieval of the mixes' public keys by the senders, padding to ensure that all the messages are the same size, etc. The focus of model is precisely to be able to calculate the anonymity of a message passing through the mix network, and we aim to abstract away other, perhaps tangentially relevant, details. One of these is encryption of messages – our model requires certain properties from the encryption scheme used in the mix network. First of all, if the sender encrypts the message for the receiver before constructing the onion, we require that the encryption scheme satisfies semantic security (broadly, seeing the ciphertext does not give any information about the plaintext). Secondly, the (public key) encryption scheme used in each layer of the onion structure should be semantically secure encryption against chosen ciphertext attacks. For us, this has two important consequences: firstly, observing the outer layer of the onion gives the attacker no observation about the inner layer; and secondly the attacker flipping any bits of a message will cause many bits to be flipped in the decryption (and thus the message is very likely to be discarded by the mix). We note that public key cryptosystems satisfying this property exist, e.g. [Sho98].

We have now specified the overall architecture of the anonymity system, but not necessarily the specific parameters. As before, the threshold of the mixes will be denoted by n. Of course, the threshold may be different for each mix. Since this is an easy extension of the model we present below, hence we omit it.

Another important parameter which we have so far ignored is the maximum number of mixes that a message can pass through. Why is there a maximum number of mixes? Recall that all the messages in a mix network have to be the same size (see Section 2.3.3). Furthermore, each mix that a message passes through "peels off" a layer of the onion, thus reducing the size of the message (and replacing the layer by random padding to make the true size of the message unobservable to the attacker looking at the network). Hence, there is a limit on the maximum number of mixes messages pass through. We denote this by rl for route length. Naturally, the attacker is aware of the values of n and rl.

Some attention has to be paid to the contents of the messages which travel through the network. In the formal model, we assume that all message contents are different from each other. If desired, this can be easily implemented by including, for instance, 128 bit nonces together with the actual content.

Now that we have some idea of the system which is being modelled, let us see how we can express this system formally.

6.2 Formal Representation of the Mix Network

A mix network is described by a 5-tuple: $(S, R, M, n, \mathrm{rl})$ of (senders, receivers, mixes, threshold of the mixes, maximum route length of a message), where S, R, M are pairwise disjoint sets, ranged over by s, r, m, and $\mathrm{rl}, n \in \mathbb{N}$. User message contents are taken from a set C, ranged over by c.

Then, messages which travel through the network (onions) can be represented as elements of

$$A = M^* \times R \times C$$

denoting a sequence of mixes through which the onion is yet to travel, the address of the receiver of the message, and the message content. The encryption layers are left implicit.

We write ε for the empty sequence and $\overrightarrow{x}$ for a sequence of zero or more x's.

Example 1. Thus, the onion $\{\{c\}_{k_r}, r\}_{k_n}, n\}_{k_m}$ (where c represents the message content itself) would be represented by (mn, r, c) and $\{c\}_{k_r}$ by (ε, r, c).

We call a message a *sender message* if this message travels from the sender to the first mix in its sequence. Correspondingly, a *receiver message* is a message which travels from the last mix to the receiver.

6.3 Mix Network State

If we look at the mix network at any point in time, some of the messages will be in transit, i.e. in the routers or in the wires, while others will be inside mixes. While a message is in transit, the attacker knows where it is travelling from and to – this information is included on every IP packet comprising the message[1]. The network (in other words everything but the mixes, senders and receivers) can therefore be represented as a set of messages.

Messages inside mixes are handled in a more interesting fashion. In all our description of the batching mixes so far, we have never explicitly outlined how the messages are received, stored or sent out. We do this now. A mix has a buffer called the input buffer in which messages accumulate. It may also, as we have seen in the previous chapters, have a separate pool of messages. As soon as a mix fires, the incoming messages are redirected to a new buffer and the batching algorithm (which in our case is the threshold one) is executed on the old one. We have previously said that the mix now "sends out" messages selected for sending by the batching strategy. Let us be more specific. Strictly speaking, the mix should put the selected messages into a separate "output" buffer (having reordered them, of course), and now tries to send these out to the appropriate mixes or receivers. Now, it may be the case that the mix is unable to send out all the messages in the output buffer before the mix fires again. In this case, the new messages are put into a separate output buffer and none of them are sent out before the old output buffer is empty. In summary, the mix works like a FIFO queue (with each element of the queue being a batch of messages), with new messages coming in, being processed by the batching algorithm and then sent out. Thus, we model a mix as a non-empty sequence of buffers (there is always at least one buffer into which the messages accumulate). Messages in a buffer are sent out in a random order (and all messages are distinct), so each buffer is modelled by a set.

Jumping ahead slightly, it is worth pointing out that a message travelling from one mix to another will be transferred from the buffer inside the first mix to the network and then from the network into a buffer of the second mix. This allows us to easily model the arbitrary delay and reordering which may be introduced by the network.

6.4 Formal Representation of the Mix Network State

The state of the mix network is then (σ, ν) where

$$\sigma : (M \to (\mathcal{P}(A))^+) \qquad \text{and} \qquad \nu : \mathcal{P}((S \cup M) \times A)$$

where $\mathcal{P}_n(A)$ denotes the set of subsets of A, each of which is of size n. Here σ represents the state of all the mixes, i.e. the messages inside their buffers, and ν is

[1]IP spoofing, although effective against the receiver, does not provide any extra security against our threat model, the global passive attacker.

the set of messages in transit in the underlying communication network, each tagged with its (most recent) origin. We let o (for 'origin') range over $S \cup M$. The destination is included in the message itself – it is the first mix in the sequence of mixes, or the receiver if that sequence is empty.

Let α range over $\mathcal{P}(A)$. Sometimes we abuse notation and write $(\overrightarrow{m}, r, c) \in \sigma(m)$ to mean a message is in one of the buffers of the mix m.

6.5 Mix Network Dynamics

If we take the state of the mix system at one particular instance, we see where all the messages are. What might happen next? We can describe the things that happen in a mix system by a small set of formal rules – reduction rules. What are the different types of events that happen in a mix network and changes its state?

1. The network might receive a message from a sender (note that we only model the mix network, not the sender).

2. One of the mixes might send a message which amounts to transferring it from the buffer inside the mix into the network.

3. A message might arrive at a mix, in which case it is taken out of the network and placed in the incoming buffer of the mix.

4. A message might leave the network to be received by the receiver.

Every time a state changes, we would like to be able to record this fact. Thus, a label summarising the event that has occurred is attached to each of the reduction rules. Sometimes we do not wish to record anything, so the reduction rule has no label attached to it. We use the word label and event interchangeably.

There are two points of notation about the labels. The labels which denote a sending action (items 2 and 4 above) have an overbar in CCS style [Mil80]. The labels of state transitions which happen inside the network (items 2 and 3), i.e. do not involve senders or receivers, have a τ attached to them.

6.6 Formal Representation of the Mix Network Dynamics

Mix networks have labelled transitions of the form $(\sigma;\nu) \xrightarrow{l} (\sigma';\nu')$ where labels l are taken from

$$
\begin{array}{rcll}
L & = & S \times A & \text{external receive} \\
& \cup & \{\tau\} \times \{\overline{(m,a)} \mid m \in M \wedge a \in A\} & \text{internal send} \\
& \cup & \{\tau\} \times ((S \cup M) \times A) & \text{internal receive} \\
& \cup & \{\overline{(m,a)} \mid m \in M \wedge a \in A\} & \text{external send}
\end{array}
$$

In each case, the two main components of the label define where the message is from and what the message is. Where the message is going to is part of the message (the first element of the mix sequence or, if it is empty, the receiver).

Thus, an external send label actually represents a receiver message arriving at the receiver, and an external receive label represents a sender message leaving the sender.

The transitions are defined by a number of axioms. For the network receiving a message from a sender (ext-recv-1):

$$(\sigma;\nu) \xrightarrow{(s,a)} (\sigma;\nu \cup \{(s,a)\})$$

For the network sending a message to a receiver (ext-send-1):

$$(\sigma;\nu \uplus \{(m,(\varepsilon,r,c))\}) \xrightarrow{\overline{(m,(\varepsilon,r,c))}} (\sigma;\nu)$$

For a mix receiving a message from the network and storing it in the current buffer (int-recv-1):

$$
\begin{array}{l}
(\sigma, m \mapsto \alpha\overrightarrow{\alpha};\nu \uplus \{(o,(m\overrightarrow{m},r,c))\}) \quad \xrightarrow{\tau,(o,(m\overrightarrow{m},r,c))} \\
\qquad\qquad (\sigma, m \mapsto (\alpha \cup \{(\overrightarrow{m},r,c)\})\overrightarrow{\alpha};\nu)
\end{array}
$$

where $|\alpha| < n-1$.

If the current buffer is nearly full and the last message arrives (int-recv-2):

$$
\begin{array}{l}
(\sigma, m \mapsto \alpha\overrightarrow{\alpha};\nu \uplus \{(o,(m\overrightarrow{m},r,c))\}) \quad \xrightarrow{\tau,(o,(m\overrightarrow{m},r,c))} \\
\qquad\qquad (\sigma, m \mapsto \emptyset(\alpha \cup \{(\overrightarrow{m},r,c)\})\overrightarrow{\alpha};\nu)
\end{array}
$$

where $|\alpha| = n-1$.

Note that $\emptyset$ denotes the empty set (buffer) while ε denotes the empty sequence.

For a mix sending messages from the last buffer (int-send-1):

$$(\sigma, m \mapsto \overrightarrow{\alpha}(\alpha \uplus \{(\overrightarrow{m}, r, c)\}); \nu) \xrightarrow{\tau, \overline{(m,(\overrightarrow{m},r,c))}} (\sigma, m \mapsto \overrightarrow{\alpha}\alpha; \nu \cup \{(m, (\overrightarrow{m}, r, c))\})$$

where $|\alpha| \neq 0$.

When the last message from the current output buffer is sent, the next output buffer in the FIFO queue becomes the output buffer from which the messages are sent out (int-send-2):

$$(\sigma, m \mapsto \overrightarrow{\alpha}(\{(\overrightarrow{m}, r, c)\}); \nu) \xrightarrow{\tau, \overline{(m,(\overrightarrow{m},r,c))}} (\sigma, m \mapsto \overrightarrow{\alpha}; \nu \cup \{(m, (\overrightarrow{m}, r, c))\})$$

Let us consider a small example to illustrate the reduction rules. The mix system consists of one mix m (threshold two) and starts out empty. Then, sender s sends in a message (rule ext-recv-1).

$$(\lambda m.\emptyset; \emptyset) \xrightarrow{(s,(m,r,c))} (\lambda m.\emptyset; \{(s, (m, r, c))\})$$

Now this message arrives at mix m (rule int-recv-1).

$$(\lambda m.\emptyset; \{(s, (m, r, c))\}) \xrightarrow{\tau,(s,(m,r,c))} (\lambda m.\{(\varepsilon, r, c)\}; \emptyset)$$

Similarly, another message is sent into the mix network from s' and arrives at mix m (rules ext-recv-1 and int-recv-2).

$$(\lambda m.\{(\varepsilon, r, c)\}; \emptyset) \xrightarrow{(s',(m,r',c'))} (\lambda m.\{(\varepsilon, r, c)\}; \{(s', (m, r', c'))\})$$

$$(\lambda m.\{(\varepsilon, r, c)\}; \{(s', (m, r', c'))\}) \xrightarrow{\tau,(s',(m,r',c'))} (\lambda m.\emptyset\{(\varepsilon, r, c), (\varepsilon, r', c')\}; \emptyset)$$

Now one of the messages comes out of the mix and then gets delivered to the receiver (rules int-send-1 and ext-send-1).

$$(\lambda m.\emptyset\{(\varepsilon, r, c), (\varepsilon, r', c')\}; \emptyset) \xrightarrow{\overline{(m,\tau,(\varepsilon,r',c'))}} (\lambda m.\emptyset\{(\varepsilon, r, c)\}; \{(m, (\varepsilon, r', c'))\})$$

$$(\lambda m.\emptyset\{(\varepsilon, r, c))\}; \{(m, (\varepsilon, r', c'))\} \xrightarrow{\overline{(m,(\varepsilon,r',c'))}} (\lambda m.\emptyset\{(\varepsilon, r, c)\}; \emptyset)$$

The other message will reach its receiver by two more transitions (rules int-send-2 and ext-send-1). Of course, many other interleavings of these events are also possible.

6.7 Attacker Observations of the Mix Networks

So far we have represented the change of state of the mix network with labels. Thus, if we know the initial state of the network and the sequence of labels, we know everything about what happened in the network.

Our aim is to use the model to compute the anonymity probability distribution of a message which travels through the network. How many state changes do we need to consider to perform such a computation? In other words, how long do we need to "run" the model for to be able to calculate the anonymity probability distribution? We do not answer this question directly, instead we take a safe over-approximation. After all, if the run we consider is too small, then we may be missing some of the messages our target messages was mixed with, while if our run was too big (assuming our algorithm for calculating anonymity was correct) we will not get anything wrong, though we might perform far more computation than necessary. In our calculation of anonymity we consider runs of the system from the time the system was completely empty of messages (both the network and the mixes) to the time it was completely empty again. If we have such a run of the system (we call such runs *complete traces*), we can compute the anonymity of any of the messages which were received by any of the receivers during this run.

Another assumption of our model is that senders only send a message to the anonymity system once. This, at first, seems totally unrealistic, but the situation can be easily rectified if we take our sender in the model as the real identity of the sender of a message tagged with the time the message was sent. Thus, we are able to neatly separate our analysis from the entirely orthogonal issue of intersection attacks [Dan03c, WALS02, WALS03].

Before we proceed with the definition of anonymity, we need to consider what the global passive attacker can see when the network transitions take place.

If a message (m', r, c) leaves mix m (the label would be $\tau, \overline{(m, (m', r, c))}$), the global passive adversary is not able to see it as the message (m', r, c) which left mix m – what he sees is a message (suppose he calls it M_1) going from m to m'. He cannot see the internal structure of this message. On the other hand, when he observes the message (m', r, c) being received by mix m' (again, merely as a message M_2 from m to m'), he can deduce that it is really the same message as M_1. In other words, although he cannot see the true contents of the message, he is able to see that the bits comprising the message have not changed and can thus correlate the "send" and the "receive" events of a particular message. In other words, the network does not do any mixing – only the mixes do.

We need to capture this in our formalism. In particular, if we are relying on the attacker to be able to "match up" send and receive events, we need to make sure that all send events can be matched up to a unique receive event and vice versa. Indeed, we need to prove that our formalism has these properties. Once this has been done, we can define a function which converts a sequence of labels $\overrightarrow{l}$ into a sequence of attacker-observed labels.

6.8 Formal Representation of the Attacker Observations

6.8.1 Defining Labels Observable by the Attacker

Let us first of all consider what the attacker who can observe the network is able to see. For every transition with a label l, he would be able to see the corresponding label l_{att} from the set L_{att} below. Note that the third line of the grammar represents the attacker's observations of receiver messages[2].

$$\begin{array}{rcl}
L_{\text{att}} & = & \{(o,m) | o \in (S \cup M) \wedge m \in M\} \\
& \cup & \{\overline{(o,m)} | o \in (S \cup M) \wedge m \in M\} \\
& \cup & \{\overline{(m,r,c)} | m \in M \wedge r \in R \wedge c \in C\}
\end{array}$$

We can now define a function strip: $L \to L_{\text{att}}$ which extracts the information which the attacker can observe from a single label directly.

$$\begin{array}{lcl}
\text{strip}(s,(m\overrightarrow{m},r,c)) & = & (s,m) \\
\text{strip}\overline{(m,(\varepsilon,r,c))} & = & \overline{(m,r,c)} \\
\text{strip}(\tau,(o,(m\overrightarrow{m},r,c))) & = & (o,m) \\
\text{strip}(\tau,\overline{(o,(m\overrightarrow{m},r,c))}) & = & \overline{(o,m)} \\
\text{strip}(\tau,\overline{(m,(\varepsilon,r,c))}) & = & \overline{(m,r,c)}
\end{array}$$

As noted above, we want to make explicit the fact that the attacker can match up the sending and receiving events of one message by looking at the bits which comprise the encrypted message.

To do so, we proceed to define tags which specify which outgoing message corresponds to which incoming message. First, however, we need some more definitions.

[2]Whether the attacker actually observes the content of a receiver message depends on whether the communication was end-to-end encrypted. Here we suppose that it was not and the content was visible.

6.8.2 Defining Traces

A *complete trace* $\overrightarrow{l}$ is a sequence of labels $l_1 \ldots l_n$ generated by the transition system which also satisfies some conditions.

$$(\sigma_0; \nu_0) \xrightarrow{l_1} \ldots \xrightarrow{l_n} (\sigma_0; \nu_0)$$

where (σ_0, ν_0) represents the empty system – no messages in mixes and no messages in the network. Expressed formally, $\sigma_0 = \lambda m \in M.\emptyset$ and $\nu_0 = \emptyset$.

The conditions which traces have to satisfy are listed below (some of these have been mentioned previously).

1. All external receive labels have to have different content:

$$\overrightarrow{l} = (\overrightarrow{l_1}\,(s, (\overrightarrow{m}, r, c))\,\overrightarrow{l_2}\,(s', (\overrightarrow{m'}, r', c'))\,\overrightarrow{l_3}\,) \Rightarrow c \neq c'$$

2. All external receive labels cannot have the same mix more than once consecutively.

$$((s, (\overrightarrow{m} m m' \overrightarrow{m'}, r, c)) \in \overrightarrow{l}\,) \Rightarrow m \neq m'$$

   Omitting this restriction would not be faithful to the real network as our model would allow the possibility of messages going from a host to itself taking arbitrary amounts of time to do so, whilst this does not happen in a real network.

   The model could be augmented with further rules which would model the fact that when messages go from the host to itself, they do so quickly, but this would obscure the focus of the work. Therefore we choose to forbid this.

3. All external receive labels have to come from different senders (i.e. each sender sends just one message).

$$\overrightarrow{l} = (\overrightarrow{l_1}\,(s, (\overrightarrow{m}, r, c))\,\overrightarrow{l_2}\,(s', (\overrightarrow{m'}, r', c'))\,\overrightarrow{l_3}\,) \Rightarrow s \neq s'$$

   As we mentioned above, if we feel unhappy about this, we can just define a new network where each sender S is represented by several senders S_i, such that S_i sends only the ith message that S sent in the original network.

4. The length of the route a message is to pass through cannot be more than the maximum route length. $((s, (\overrightarrow{m}, r, c)) \in \overrightarrow{l}\,) \Rightarrow |\overrightarrow{m}| < \mathrm{rl}$

Combining these, the trace property trace($\overrightarrow{l}$) is:

$$
\begin{array}{rll}
\text{trace}\,(\overrightarrow{l}) = & & (\forall \overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3}, s, s', \overrightarrow{m}, \overrightarrow{m'}, r, r', c, c'. \\
& & ((\overrightarrow{l} = (\overrightarrow{l_1}\,(s, (\overrightarrow{m}, r, c))\,\overrightarrow{l_2}\,(s', (\overrightarrow{m'}, r', c'))\,\overrightarrow{l_3})) \Rightarrow c \neq c')) \\
& \wedge & (\forall s, \overrightarrow{m}, m, m', \overrightarrow{m'}, r, c. \\
& & ((s, (\overrightarrow{m} m m' \overrightarrow{m'}, r, c)) \in \overrightarrow{l} \Rightarrow m \neq m')) \\
& \wedge & (\forall \overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3}, s, s', \overrightarrow{m}, \overrightarrow{m'}, r, r', c, c'. \\
& & ((\overrightarrow{l} = \overrightarrow{l_1}\,(s, (\overrightarrow{m}, r, c))\,\overrightarrow{l_2}\,(s', (\overrightarrow{m'}, r', c'))\,\overrightarrow{l_3}) \Rightarrow s \neq s')) \\
& \wedge & (\forall s, \overrightarrow{m}, r, c.((s, (\overrightarrow{m}, r, c)) \in \overrightarrow{l} \Rightarrow |\overrightarrow{m}| < \text{rl}))
\end{array}
$$

$$
\text{complete\_trace}(\overrightarrow{l}) = (\sigma_0; \nu_0) \xrightarrow{\overrightarrow{l}} (\sigma_0; \nu_0) \;\wedge\; \text{trace}(l)
$$

We also define a function typ for extracting the type of a label as follows:

$$
\begin{array}{l}
\text{typ}(s, (\overrightarrow{m}, r, c)) = \text{ext-recv} \\
\text{typ}(\overline{(m, (\varepsilon, r, c))}) = \text{ext-send} \\
\text{typ}(\tau, (o, (\overrightarrow{m}, r, c))) = \text{mix-recv} \\
\text{typ}(\tau, \overline{(m, (\overrightarrow{m}, r, c))}) = \text{mix-send}
\end{array}
$$

6.8.3 Attacker Observations

As mentioned before, we need to establish the correspondence between the send and receive labels which correspond to sending and receiving of one message. We illustrate this idea with an example. Suppose a message leaves mix m for mix m', and gets delayed in the network for a long time. Then, (after mix m has fired several times), another message leaves mix m for mix m' and arrives quickly at mix m'. Subsequently, the first message arrives at mix m'.

The following part of a real trace corresponds to this scenario:

$$
\begin{array}{l}
[\ldots (\tau, \overline{(m, ([m'], r_1, c_1))}); (\tau, \overline{(m, ([m'], r_2, c_2))}); \\
(\tau, (m, [m'], r_2, c_2)); (\tau, (m, [m'], r_1, c_1)) \ldots]
\end{array}
$$

Clearly, the sequence of attacker labels that the adversary would be observing is the following (this should be clear from the range of the function strip):

$$
[\ldots \overline{(m, m')}; \overline{(m, m')}; (m, m'); (m, m') \ldots]
$$

Thus, it is not clear from this sequence of labels that the first message arrived after the second. On the other hand, the real attacker who watches the network can deduce this fact easily – the bits comprising the two messages are different and he can compare the bits of messages coming out of mix m to the bits of messages coming into mix m'. The network does not transform messages in any way (if it does, the message will be dropped by the mix as it will not decrypt to anything sensible), so it is easy for the attacker to work out which message is which. This is, of course not a problem – it is the mixes which are supposed to introduce anonymity in the system, not the network.

We express this correspondence by means of a relation $\hat{R}_{\overrightarrow{l}}$ on $\mathbb{N}\times\mathbb{N}$. If (i, i') is in $\hat{R}_{\overrightarrow{l}}$, this represents the fact that l_i is the sending of a message which was received with the label $l_{i'}$. By l_i, of course, we mean the ith label in the sequence, starting from zero. The situation in our example above could then be described by the attacker trace

$$[\ldots \overline{(m, m')}; \overline{(m, m')}; (m, m'); (m, m') \ldots]$$

together with the relation $\{(0, 3), (1, 2)\}$.

In our model, there are three ways different labels can represent sending and receiving of messages. First, an ext-recv label will necessarily correspond to a mix-recv label (a message sent in by a user gets received by a mix), a mix-send label can correspond to a mix-recv label (as in the example above) or a mix-send label can correspond to an ext-send label (representing a message sent out by a mix to a user and the user receiving it).

Given a trace $\overrightarrow{l}$, we represent the attacker observation of it as follows:

1. We construct the relation $\hat{R}_{\overrightarrow{l}}$ which establishes the correspondence between sending and receiving events of messages.
2. We use the relation to include the information about the correspondence of the different events together with the events themselves.
3. We construct a new trace with the information hidden by encryption removed (using the function strip defined above), but with the correspondence included.

There are a number of things we have to take care of. First of all, we would like to ensure that given a complete trace $\overrightarrow{l}$, we can construct a relation $\hat{R}_{\overrightarrow{l}}$ "properly".

First, let us make sure that messages in our system cannot be duplicated.

Lemma 1 *A trace of the system contains no label more than once.*

More formally,

$\forall \overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3}, l, l' \,.\, \mathrm{trace}(\overrightarrow{l_1}\, l\, \overrightarrow{l_2}\, l'\, \overrightarrow{l_3}) \Rightarrow l \neq l'$

Proof. By induction on the number of the mixes in the sequence of mixes, while observing that if that does not change then the type of events must be different.

Now we can define the relation $\hat{R}_{\overrightarrow{l}}$ that we are looking for. First, we define a family $\mathcal{R}_{\overrightarrow{l}}$ of relations $R_{\overrightarrow{l}}$ where each $R_{\overrightarrow{l}}$ represents a correct but partial "matching up" of send and receive labels of the trace $\overrightarrow{l}$. We define $\mathcal{R}_{\overrightarrow{l}}$ as the set of all relations R satisfying a predicate $P_{\overrightarrow{l}}$.

$$\begin{array}{rl} P_{\overrightarrow{l}}(R) = & \text{complete\_trace}(\overrightarrow{l}\,) \wedge \\ & (\forall (i, i') \in R \\ & \exists m, m', \overrightarrow{m}, r, c, s. \\ & \quad ((l_i = (\tau, \overline{(m', (m\overrightarrow{m}, r, c))}) \ \wedge \ l_{i'} = (\tau, (m', (m\overrightarrow{m}, r, c)))) \\ & \vee \quad (l_i = (s, \overline{(\overrightarrow{m}, r, c)}) \ \wedge \ l_{i'} = (\tau, \overline{(s, (\overrightarrow{m}, r, c))})) \\ & \vee \quad (l_i = (\tau, \overline{(m, (\varepsilon, r, c))}) \ \wedge \ l_{i'} = \overline{(m, (\varepsilon, r, c))})) \end{array}$$

Now we want to find the biggest relation in $\mathcal{R}_{\overrightarrow{l}}$. We can compare these relations using the inclusion ordering. The biggest relation is the union of all others, as we can show that P satisfies the following property.

Lemma 2 *For a trace $\overrightarrow{l}$, the union of a set of relations satisfying $P_{\overrightarrow{l}}$ is itself a relation which satisfies $P_{\overrightarrow{l}}$. More formally, $P(\bigcup_{R \in \mathcal{R}_{\overrightarrow{l}}} R)$.*

Proof. Consider an arbitrary member of the union of the sets of relations, and check that it is consistent with P.

Now we need to make sure that no label is in the relation more than once, e.g. it is impossible that our model shows that some message was sent once and received twice (or, sent twice and received once!).

Lemma 3 *Every label index occurring in a relation satisfying P appears at most once.*

$$\forall (i, i') \in R. \ \nexists i''.(i, i'') \in R \wedge \nexists i'''.(i''', i') \in R$$

Proof. By case analysis of an arbitrary label l_i.

We can also show that a label occurs at least once in a relation satisfying $P_{\overrightarrow{l}}$.

Lemma 4 *Every label index occurring in a relation satisfying P appears at least once.*

$$\forall l_i \in \overrightarrow{l}. \exists l'_j, R.(i, j) \in R$$

Proof. By case analysis of an arbitrary label l_i – in a complete trace there is always a corresponding label which "causes" l_i or "is caused" by it.

Clearly, the largest relation satisfying $P_{\overrightarrow{l}}(R)$ is the union of the set of all the relations satisfying $P_{\overrightarrow{l}}(R)$, and by Lemma 2 it satisfies $P_{\overrightarrow{l}}(R)$. Hence, we can find the largest relation $\hat{R}_{\overrightarrow{l}} = \bigcup_{R \in \mathcal{R}_{\overrightarrow{l}}} R$ which will contain all the label indices exactly once.

Now that we are sure that we can generate a "proper" $\hat{R}_{\overrightarrow{l}}$, we are ready to use it and obtain a trace representing the attacker's observation.

6.8.4 Erasing the Trace

Now we are ready to define the erasure function. It takes a complete trace as input, and outputs an attacker trace tagged with a pair of integers according to the relation $\hat{R}_{\overrightarrow{l}}$. The pair of integers are the sequence number of this label and the sequence number of the label it is related to by the relation.

The erasure function $\text{erase}_{\overrightarrow{l}} : L^* \rightarrow (L_{\text{att}} \times \mathbb{N} \times \mathbb{N})^*$ can now be defined as follows:

$$\text{erase}(\overrightarrow{l}) = \text{map } (\lambda i.(\text{strip}(l_i))_{id_{\overrightarrow{l}}(i)})\ [0, \ldots, |\overrightarrow{l}| - 1]$$

where $id_{\overrightarrow{l}}(i)$ is the unique pair (i, i_1) such that $(i, i_1) \in \mathcal{R}_{\overrightarrow{l}} \vee (i_1, i) \in \mathcal{R}_{\overrightarrow{l}}$ and map is the function which applies its first argument (which is itself a function) to each element of its second argument (which is a list).

Hence, in our example at the beginning of Section 6.8.3, the attacker observes:

$$[\ldots \overline{(m, m')_{(0,3)}}; \overline{(m, m')_{(1,2)}}; (m, m')_{(2,1)}; (m, m')_{(3,0)} \ldots]$$

Having obtained from $\overrightarrow{l}$ a sequence of integer tagged attacker labels which we call $Obs_{\overrightarrow{l}}$, we proceed with the definitions of anonymity.

6.9 Calculating the Anonymity Probability Distribution

We have seen that we can model the behaviour of the system and that from any trace which records one particular behaviour of the system we can obtain a representation of what the attacker sees. We now proceed to define the process of calculating the anonymity probability distribution of a message which has passed through the network, and thus the anonymity of it.

We have previously said that the senders determine various parameters of the messages they send through the anonymity system, e.g. the route the message will take. We assert that it is best to remove this burden from them and leave the choice of route to the anonymity client. The arguments for this are well-known, and need not be presented here (e.g. usability, removing the burden of security-critical decisions from the user, etc).

At this point we make a rather strong assumption (in favour of the attacker). We assert that the routes are chosen by the sender from a probability distribution which is also known to the attacker[3]. This is clearly true if the route length and the route are picked by the anonymity client from a fixed probability distribution. Unfortunately the popular Mixmaster client QuickSilver [Qui] currently leaves the choice of route and route length up to the user.

However, there are several parameters which cannot be determined by the anonymity client. One of these is the receiver of the message! Thus, we assume that the attacker has some idea (i.e. a probability distribution) of who the sender is likely to send messages to. Naturally, if he does not, a uniform distribution is the best he can assume. Another such parameter is the content c of the message. Again, we assume the attacker has a set of probability distributions (one per sender) from which each of the contents was picked. Obviously, if all the content which the attacker sees is encrypted (with the public key of the receiver), then the probability distribution of a sender constructing any of these is uniform. Note that the choice of content of a message and the choice of receiver are not independent. For example, an NSA cryptographer is likely to send a message about his specialist subject, but is unlikely to send this message to someone at the KGB.

Now, for each sender, we multiply the probability of each of the routes being chosen by the probability of each of the combinations of receiver and content being chosen and thus get the probability of that particular sender sending content c to receiver r via the route $\overrightarrow{m}$. In other words, for each sender we have a probability distribution over all the possible messages he could send.

We can then construct probability distributions over sets of sender messages being sent out by senders (and thus entering our anonymity system) by merely multiplying together the probability distributions for each sender. This, of course, requires the senders to construct their messages independently from each other. Why is this the case? Recall that each sender has "anonymous email software" which helps them choose the route through the mix network randomly and therefore, independently of one another. Even if the anonymity client sends dummy traffic to the network, a good dummy policy is likely to be randomized and ensure that the messages generated by

[3]Naturally, if the probability distribution is not known to the attacker, the sender is better off.

the clients are not dependent on each other[4].

Each set of sender messages can give rise to a number of traces $\overrightarrow{l}$ due to network reordering. Suppose we have one such set of sender messages, and its corresponding set of traces.

Now, a global passive attacker can take an observation which he has made and check if any of the traces resulting from the set of sender messages matches it. If there is a match, that means that this set of sender messages could have resulted in the actual observation.

For each of the possible set of sender messages we have a probability of it being received by the network and a binary decision of whether it was compatible with the observation the adversary has made. Now we renormalise the distribution, throwing away all the sets of sender messages which could not have generated traces consistent with the observation and obtain something we call a *scenario distribution*. From this it is easy to calculate an anonymity probability distribution for each message.

6.10 Calculating the Anonymity Probability Distribution Formally

6.10.1 External Receive Events

First of all, we introduce a notation for probability distributions. A function f of type $X \to [0,1]$ where the sum of the range is equal to 1 is written as $f : X \rightsquigarrow [0,1]$.

For each sender, the adversary makes an assumption about the probability distribution from which that sender chooses routes for his messages travelling through the network. We call this probability distribution SM, for sequences of mixes.

$$\mathrm{SM} : S \to \left(\left(\sum_{i \in \{0 \ldots \mathrm{rl}\}} M^i \right) \rightsquigarrow [0,1] \right)$$

He also assumes how likely each sender is to send all the different contents of messages to each receiver. We call this probability distribution RC, for receivers and contents.

$$\mathrm{RC} : S \to ((R \times C) \rightsquigarrow [0,1])$$

[4]More concretely, a dummy policy like "send a message every hour on the hour" means that the sender messages are not independent, but will not add much anonymity.

Note a couple of interesting cases.

Example 2. If the attacker is not able to determine the content of messages (i.e. has never seen the plaintexts of messages coming out of the anonymity network), then $\forall s, r, c, c'.\mathrm{RC}(s)(r, c) = \mathrm{RC}(s)(r, c')$.

Example 3. If the system we are modelling is a cascade with a sequence of mixes $\overrightarrow{m'}$, then

$$\begin{array}{rl} \mathrm{SM}\ s\ \overrightarrow{m} = & 0 \text{ if } \overrightarrow{m} \neq \overrightarrow{m'} \\ & 1 \text{ otherwise} \end{array}$$

Now for each sender we construct a probability distribution ERV over external receive events. This essentially amounts to multiplying the two probability distributions we have (SM and RC) together. This is justified as the route is independent of the choice of content and sender, as the choice of route is performed by the anonymity client. In some systems the route might be chosen in such a way that the last mix is close to the client (to minimize delay), but this is hardly justified in the context of message-based systems.

$$\begin{array}{l} \mathrm{ERV} : S \to \left(\left(\sum\limits_{i \in \{0 \ldots \mathrm{rl}\}} M^i \right) \times R \times C \right) \rightsquigarrow [0, 1] \\ \mathrm{ERV}(s) = \lambda(\overrightarrow{m}, r, c).(\mathrm{SM}\ s\ \overrightarrow{m}) \times (\mathrm{RC}\ s\ (r, c)) \end{array}$$

Now we build a distribution SERE over sets of external receive events (for a particular number of senders). We assume that each of the senders chooses the content and receivers of their messages independently of the others, hence the probability distributions for each of the senders can simply be multiplied together.

$$\begin{array}{l} X \subseteq S \times (M^* \times R \times C) \\ \mathrm{SERE} : \mathcal{P}_n(S \times (M^* \times R \times C)) \rightsquigarrow [0, 1] \\ \mathrm{SERE}(X) = \prod_{x \in X} \mathrm{ERV}((\pi_1 x)(\pi_1 \pi_2 x, \pi_2 \pi_2 x, \pi_3 \pi_2 x)) \end{array}$$

The attacker now has an *a priori* probability distribution over sets of external receive events of size $|X|$. Note that we have not yet examined the trace observed by the adversary at all. We do this next.

6.10.2 Scenario Anonymity

Each set of external receive events which we have obtained can give rise to several different traces $\overrightarrow{l}$. This is caused by both network reordering and mix reordering. An example is appropriate at this point.

Example 4. Take a network with two senders, two receivers, a maximum route length of 1 and one mix with a threshold of two. Hence, the network is defined as a five-tuple $(\{a, b\}, \{p, q\}, \{m\}, 2, 1)$.

Now consider two messages injected into this network. These are represented by the following set of sender messages (external receive events):

$$\{(a, (m, p, c)), (b, (m, q, c'))\}$$

The possible traces resulting from it include:

$$\begin{aligned}&[(a, (m, p, c)); (b, (m, q, c')); (\tau, (a, (m, p, c))); (\tau, (b, (m, q, c')));\\ &(\tau, \overline{(m, (\varepsilon, p, c))}); (\tau, \overline{(m, (\varepsilon, q, c'))}); \overline{(m, (\varepsilon, p, c))}; \overline{(m, (\varepsilon, q, c'))}]\end{aligned} \tag{6.1}$$

$$\begin{aligned}&[(a, (m, p, c)); (b, (m, q, c')); (\tau, (b, (m, q, c'))); (\tau, (a, (m, p, c)));\\ &(\tau, \overline{(m, (\varepsilon, p, c))}); (\tau, \overline{(m, (\varepsilon, q, c'))}); \overline{(m, (\varepsilon, p, c))}; \overline{(m, (\varepsilon, q, c'))}]\end{aligned} \tag{6.2}$$

(Difference from 6.1 caused by network reordering)

$$\begin{aligned}&[(a, (m, p, c)); (b, (m, q, c')); (\tau, (a, (m, p, c))); (\tau, (b, (m, q, c')));\\ &(\tau, \overline{(m, (\varepsilon, q, c'))}); (\tau, \overline{(m, (\varepsilon, p, c))}); \overline{(m, (\varepsilon, p, c))}; \overline{(m, (\varepsilon, q, c'))}]\end{aligned} \tag{6.3}$$

(Difference from 6.1 caused by mix reordering)

Now suppose the attacker has made an observation. He does not know what trace this observation corresponds to (after all, this is the entire point of the anonymity system!). Hence, we call the observation of the attacker $Obs : (L_{\mathrm{att}} \times \mathbb{N} \times \mathbb{N})^*$. We compute the set of traces compatible with Obs.

$$Scen_{Obs} = \{\overrightarrow{l} \mid \text{complete\_trace}(\overrightarrow{l}) \ \wedge \ \text{erase}(\overrightarrow{l}) = Obs\}$$

Let us define the function rcv which extracts the set of receive events from a trace.

$$\mathrm{rcv}(\overrightarrow{l}) = \{l | l \in \overrightarrow{l} \wedge \mathrm{typ}(l) = \text{ext-recv}\}$$

Now extract the set of sets of receive events which could have caused all the traces.

$$ERV_{Obs} = \{X | X = \mathrm{rcv}(\overrightarrow{l}) \wedge \overrightarrow{l} \in Scen_{Obs}\}$$

Clearly, all the traces in $Scen_{Obs}$ have the same number of receive events, so the cardinality of all the members of ERV_{Obs} is the same. Each element of ERV_{Obs} has a probability associated with it (as calculated above). Hence, to obtain a probability distribution over ERV_{Obs} we renormalise the probability distribution SERE over sets of external receive events on those sets which correspond to elements of ERV_{Obs}.

First, we calculate the probability that any of the sets of external receive events in ERV_{Obs} occurred:

$$Tot_{Obs} = \sum_{X \in \mathrm{ERV}_{Obs}} \mathrm{SERE}(X)$$

$$Y : \mathrm{ERV}_{Obs} \rightsquigarrow [0, 1]$$
$$Y = \lambda X. \tfrac{\mathrm{SERE}(X)}{Tot_{Obs}}$$

From Y we can compute the sender anonymity of a message arriving at a particular receiver. To do this, we define $\mathcal{U}_{M,Obs}$ – the anonymity probability distribution discussed in Chapter 3. Note that here the anonymity probability distribution is parameterised with the observation of the attacker, Obs. Given an external send event $\overline{(m, (\varepsilon, r, c))}$, the sender anonymity probability distribution of this message is:

$$\begin{aligned} &\mathcal{U}_{M,Obs} : S \rightsquigarrow [0, 1] \\ &\mathcal{U}_{\overline{(m,(\varepsilon,r,c))},Obs} = \lambda s. \sum_{\substack{l=(s,(\overrightarrow{m},r,c)) \wedge \\ l \in X \ \wedge X \in \mathrm{ERV}_{Obs}}} Y(x) \end{aligned}$$

From this distribution we can compute the information theoretic metric, as described in Chapter 3.

$$Anon_{\overline{(m,(\varepsilon,r,c))},Obs} = \sum_{s \in S} \mathcal{U}_{\overline{(m,(\varepsilon,r,c))},Obs}(s) \log_2 \mathcal{U}_{\overline{(m,(\varepsilon,r,c))},Obs}(s)$$

We now proceed to illustrate the whole process with a simple example.

6.11 Calculating Anonymity – a Simple Example

Suppose our mix network consists of two mixes $M = \{m, m'\}$. Suppose further that we have three senders $S = \{a_0, a_1, a_2\}$ and three recipients $R = \{r_0, r_1, r_2\}$. Furthermore, our mix threshold $n = 2$ and the senders pick their routes though through no more than two mixes from a uniform distribution. There are four different routes for the messages to travel through: $[m, m']; [m', m]; [m]; [m']$. Hence, $\mathrm{SM} = \lambda s.\lambda\overrightarrow{m} = 0.25$.

Suppose further that the anonymity client used by all the senders of the anonymity system forces them to encrypt the message content with the senders' public key and they have obtained this key out of band (e.g. by meeting up in the street). Furthermore, the adversary knows very little about each sender in the network, and hence assigns a uniform probability distribution to the potential recipients each sender might want to communicate to. Hence,

$$\forall s, r, c, s', r', c'.\mathrm{RC}(s\ (r, c)) = \mathrm{RC}(s'\ (r', c'))$$

Suppose further that the attacker has made the following observation of the network and would like to find the anonymity of the message arriving at r_2.

$$\begin{array}{ll} Obs = [& (a_0, m)_{(0,2)}; (a_1, m)_{(1,3)}; (a_0, m)_{(2,0)}; (a_1, m)_{(3,1)}; (m, m')_{(4,8)}; \\ & (m, r_0, c_0)_{(5,6)}; (m, r_0, c_0)_{(6,5)}; (a_2, m')_{(7,9)}; (m, m')_{(8,4)}; \\ & (a_2, m')_{(9,7)}; (m', r_1, c_1)_{(10,12)}; (m', r_2, c_2)_{(11,13)}; (m', r_1, c_1)_{(12,10)}; \\ & (m', r_2, c_2)_{(13,11)}] \end{array}$$

This observation can be represented by the mix network diagram shown in Figure 6.1[5].

Now, there are four traces which erase to Obs.

$$\begin{array}{ll} \overrightarrow{l_0} = [& (a_0, ([m], r_0, c_0)); (a_1, ([m, m'], r_1, c_1)); (\tau, (a_0, ([m], r_0, c_0))); \\ & (\tau, (a_1, ([m, m'], r_1, c_1))); (\tau, (m, ([m'], r_1, c_1)); (\tau, (m, (\varepsilon, r_0, c_0))); \\ & (m, (\varepsilon, r_0, c_0)); (a_2, ([m'], r_2, c_2)); (\tau, (m, [m'], r_1, c_1)); \\ & (\tau, (a_2, ([m'], r_2, c_2))); (\tau, (m', (\varepsilon, r_1, c_1))); (\tau, (m', (\varepsilon, r_2, c_2))); \\ & (m', (\varepsilon, r_1, c_1)); (m', (\varepsilon, r_2, c_2))] \end{array}$$

[5]The careful reader will note that there is a one to many correspondence between mix network diagrams and attacker observations (as above).

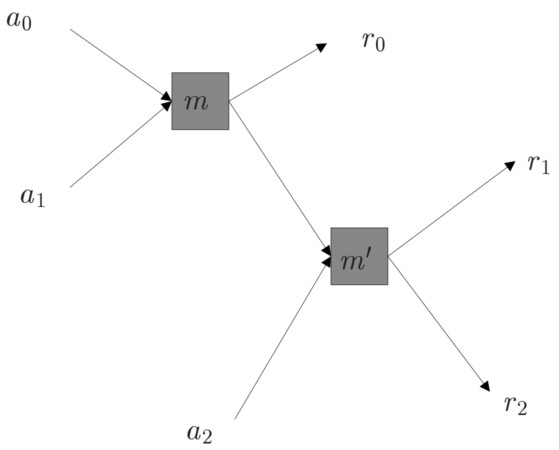


Figure 6.1: The example attacker observation of the system is represented in this mix network diagram.

$$
\begin{aligned}
\overrightarrow{l_1} = [\quad & (a_0,([m],r_0,c_0)); (a_1,([m,m'],r_2,c_2)); (\tau,(a_0,([m],r_0,c_0)));\\
& (\tau,(a_1,([m,m'],r_2,c_2))); (\tau,\overline{(m,([m'],r_2,c_2))}); (\tau,\overline{(m,(\varepsilon,r_0,c_0))});\\
& \overline{(m,(\varepsilon,r_0,c_0))}; (a_2,([m'],r_1,c_1)); (\tau,(m,[m'],r_2,c_2));\\
& (\tau,(a_2,([m'],r_1,c_1))); (\tau,\overline{(m',(\varepsilon,r_1,c_1))}); (\tau,\overline{(m',(\varepsilon,r_2,c_2))});\\
& \overline{(m',(\varepsilon,r_1,c_1))}; \overline{(m',(\varepsilon,r_2,c_2))}]
\end{aligned}
$$

$$
\begin{aligned}
\overrightarrow{l_2} = [\quad & (a_0,([m,m'],r_1,c_1)); (a_1,([m],r_0,c_0)); (\tau,(a_0,([m,m'],r_1,c_1)));\\
& (\tau,(a_1,([m],r_0,c_0))); (\tau,\overline{(m,([m'],r_1,c_1))}); (\tau,\overline{(m,(\varepsilon,r_0,c_0))});\\
& \overline{(m,(\varepsilon,r_0,c_0))}; (a_2,([m'],r_2,c_2)); (\tau,(m,[m'],r_1,c_1));\\
& (\tau,(a_2,([m'],r_2,c_2))); (\tau,\overline{(m',(\varepsilon,r_1,c_1))}); (\tau,\overline{(m',(\varepsilon,r_2,c_2))});\\
& \overline{(m',(\varepsilon,r_1,c_1))}; \overline{(m',(\varepsilon,r_2,c_2))}]
\end{aligned}
$$

$$
\begin{aligned}
\overrightarrow{l_3} = [\quad & (a_0,([m,m'],r_2,c_2)); (a_1,([m],r_0,c_0)); (\tau,(a_0,([m,m'],r_2,c_2)));\\
& (\tau,(a_1,([m],r_0,c_0))); (\tau,\overline{(m,([m'],r_2,c_2))}); (\tau,\overline{(m,(\varepsilon,r_0,c_0))});\\
& \overline{(m,(\varepsilon,r_0,c_0))}; (a_2,([m'],r_1,c_1)); (\tau,(m,[m'],r_2,c_2));\\
& (\tau,(a_2,([m'],r_1,c_1))); (\tau,\overline{(m',(\varepsilon,r_1,c_1))}); (\tau,\overline{(m',(\varepsilon,r_2,c_2))});\\
& \overline{(m',(\varepsilon,r_1,c_1))}; \overline{(m',(\varepsilon,r_2,c_2))}]
\end{aligned}
$$

Thus,

$Scen_{Obs} = \{\overrightarrow{l_0}, \overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3}\}$

We now extract the set of sets of receive events from $Scen_{Obs}$:

$$
\begin{aligned}
\mathrm{ERV}_{Obs} = \quad \{ & \\
& \{(a_0,([m],r_0,c_0)),(a_1,([m,m'],r_1,c_1)),(a_2,([m'],r_2,c_2))\}, \\
& \{(a_0,([m],r_0,c_0)),(a_1,([m,m'],r_2,c_2)),(a_2,([m'],r_1,c_1))\}, \\
& \{(a_0,([m,m'],r_1,c_1)),(a_1,([m],r_0,c_0)),(a_2,([m'],r_2,c_2))\}, \\
& \{(a_0,([m,m'],r_2,c_2)),(a_1,([m],r_0,c_0)),(a_2,([m'],r_1,c_1))\} \\
\} &
\end{aligned}
$$

It is clear from the distributions SM and RC that the probability of each of these sets of receive events occurring is the same. Hence, we are able to calculate the anonymity probability distribution for any of the receiver messages directly. For example, the sender anonymity probability distribution of the message $\overline{(m,(\varepsilon,r_0,c_0))}$ (expressed in the set notation) is $\{(a_0,0.5),(a_1,0.5)\}$ while that of $\overline{(m',(\varepsilon,r_2,c_2))}$ is $\{(a_0,0.25),(a_1,0.25),(a_2,0.5)\}$. The corresponding sender anonymity is 1 and 1.5 bits respectively.

6.12 Commentary and Model Design Choices

We have shown how to use the model to analyse the anonymity of a message which has travelled through the system given the observation of the system by the global passive attacker. This analysis was done by hand and has not been excessively painful. Clearly, with more complex systems and larger traces, such methods rapidly become infeasible. The next step was to write an analysis tool which implements our semantics.

The tool, written in Haskell [PJH+99] can perform several tasks: Firstly, given a number of sender messages, it can act as a mix network and produce a set of resulting real traces and the final state in each case. Secondly, given a real trace, it can erase it to give the trace observed by the attacker. Finally (and most importantly) we can use the tool to work out the anonymity of a message given a particular observation – this involves generating the set $Scen_{Obs}$. The algorithm for generating $Scen_{Obs}$ from an erased trace is non-trivial, and much has been gained from implementing it in a high-level functional language. Unfortunately, the size of $Scen_{Obs}$ is $O(n!)$ if the trace has no more than one message between two mixes during any round, and even worse otherwise. So, even having implemented the semantics we are still limited to fairly small examples (around 5 messages). This is clearly unsatisfactory (hence we do not describe our algorithm here) and we need to look for a more efficient algorithm which computes an approximation. It is, however, unclear, that a good algorithm for doing so exists. One such algorithm was proposed in our PET paper [SD02]; finding a proof or even an argument for its correctness has turned out to be very difficult. We leave this for exciting future work. On the other hand, one can always go back to non-probabilistic analysis, i.e. compute the anonymity set (much easier to do approximately) and declare that the probability distribution over all the senders in the sender anonymity set is uniform. Instead, we go on to discuss the design choices of the semantic model we have constructed followed by a survey of formal models for anonymity systems and a speculation about the future.

In creating our model we made a number of design choices. Here we document the more important ones and discuss alternative approaches.

First of all, we decided to include the reordering produced by the network in the model. This is done using non-determinism in the reduction rules of the semantics. Following on from this, we assumed that the attacker observes the traffic at the senders, receivers and mixes and showed that the attacker is able to factor out the network reordering. This is consistent with the observation that anonymity is produced by the mixes, not the network. If the attacker is not observing the traffic very close to the mixes[6], he may not be able to deduce correctly which message belongs to which batch. Thus, it may appear to him that messages from different batches got mixed together, hence yielding increased anonymity. We are not aware of this simple observation having been made before.

Secondly, we have chosen to abstract from the issue of intersection attacks. This was a valuable design decision as our work is completely orthogonal to (and could therefore be easily combined with) previous research on statistical disclosure: [Dan03c, AKP03, KAP02].

We have also chosen to model a "not quite free route" mix network. This merely saved us several reduction rules in the model, though one must note that additional technicalities would be required for the construction of the attacker's observation. The one difference between a free route mix network and the one we modelled is that in the latter messages are not allowed to have the same mix occurring consecutively in the sequence of mixes of any message. What happens if a message does have two mixes consecutively in its sequence? The answer is simple – it does not get sent out onto the network and thus is not seen by the attacker. However, because the mixes we are dealing with in this model are threshold mixes, the attacker is able to *deduce* that a message remained in the mix and infer the correct "observation" from the lack of one. This inference would have to be included in the definition of the attackers observation. Note that a similar inference could be made in the case of timed, timed pool and threshold pool mixes, but not in the case of binomial mixes or in the presence of a randomised dummy traffic policy. Modelling this formally is challenging future work, which may turn out to be infeasible for some mixes! A related, (and much

[6]The technical condition is the attacker sees the messages in the same order as they leave/arrive at the mix.

easier) inference would have to be performed to account for message drops. Take the case of a threshold mix network, as above. From an observation Obs the attacker can deduce when a mix drops messages – in this case this mix would have more arrival events than send events. A dropped message, of course, is equivalent to a message leaving the network. Hence, once a message drop has been identified, we can augment Obs with two events in an appropriate place which represent the message leaving the network to a receiver *drop*. From here on, the analysis proceeds mostly as before (though now R must also include this new "receiver" and the mix must be made not to fire until $n + 1$ messages arrive at the appropriate time). Although we have not formalised this, it is clear that it can be added relatively easily to the framework presented above.

Finally, we have chosen to abstract away the details of encryption, message transfer protocols, retrieving the necessary public keys by the users, etc. This is the subject of other work, e.g. [Dan04], and is mostly orthogonal to our efforts.

6.13 Related and Future Work

The anonymity properties of mix networks, however simple, have not been analysed previously. In this section we list papers which have merely touched on the subject.

First, there is some fairly speculative debate in [BPS00] on the advantages of mix cascades over mix networks. The authors present some observations: first they argue that in the context of a threat from the global active attacker with many compromised mixes, an attack which partitions the anonymity set of a mix network is possible. Further arguments made in favour of mix cascades are made on the basis that the probability of choosing at least one trustworthy mix is higher in a mix cascade. There is an informal description, though no algorithm, of how one might go about working out the anonymity set size of a run of a mix network. Naturally, there is no probabilistic analysis, and from the descriptions it is apparent that the method is not general enough to analyse any trace, rather it computes the upper bound on the anonymity of a run (recall the formula for the composition of mixes in [SD02] does this taking into account probabilities).

The anonymity of some of the other systems have been analysed formally. The best example of this is, perhaps, Shmatikov's analysis of the Crowds system [Shm03]. There, a model checker was used to help explore the state space of the Crowds system with a small number of nodes. New and interesting results were derived. It may be possible to explore such a mechanised approach to look at the anonymity of mix networks as well.

One direction of future work which we have mentioned previously is finding an efficient algorithm for calculating the anonymity of the mix network and validating it against the definitions presented above.

Another promising direction is extending the model to cope with pool mixes. The difficulty here is two-fold. Firstly, a network of pool mixes always has some messages in the mixes, so our definition of a trace, i.e. $\overrightarrow{l}$ s.t. $(\sigma_0, \nu_0) \xrightarrow{\overrightarrow{l}} (\sigma_0, \nu_0)$ is no longer appropriate (the network is never in a state σ_0). Thus, we have to find an alternative definition of traces. Secondly, the problem is that the probability of a message leaving the network merely tends to 1, so computing a receiver anonymity probability distribution requires an infinitely long observation. Alternatively, computing the sender anonymity probability distribution requires an observation of the entire history of the mix. Finally, the complexity of working out the anonymity of a message travelling through a network of pool mixes is going to be even worse!

An easier (yet still fruitful) piece of work is to find a different definition of traces which still allows us to compute anonymity. This is likely to be necessary if we are to build tools for the analysis of the anonymity of mix networks.

6.14 Summary

In this chapter we have looked at the anonymity of a message going through a network made up of threshold mixes. We have designed a model of such a network, derived the information that can be seen by the global passive adversary and thus computed how much anonymity each message going through it has.

In the last few chapters we have looked at the anonymity of message-based mix systems. Our results broadly support the hypothesis that these systems provide a substantial amount of anonymity against the global passive attacker, and some anonymity against the global active attacker. Furthermore, there are techniques (which we have not investigated in this thesis) for gaining even more anonymity; dummy traffic is a good example. Of course this will be the subject of further investigations.

We now leave the subject of message-based systems altogether and embark on an investigation of the anonymity of connection-based systems. As mentioned in Chapter 2, these are subject to stricter delay requirements and therefore we expect their anonymity to be much worse than that of message-based systems. In the next chapter we show that this is indeed the case.

Chapter 7

On the (non)-Anonymity of Connection-based Systems

"Shut up and code"
— Notice on the office wall of an industrial software manager

7.1 Introduction

In this chapter we examine the anonymity of connection-based systems. These commonly aim to provide low-latency bidirectional communication for a variety of applications (e.g. SSH connections, web browsing, etc). There are several implemented designs [DMS04, RP02, FM02]. Although these are also sometimes called mix systems, current designs do not do any mixing as such.

The anonymity properties of connection-based systems is potentially a big area for future research; here we take the first steps towards their comprehensive analysis. We take two simple attacks, identify the conditions under which the attacker succeeds in linking the endpoints of a connection, and try to reduce the probability of those conditions occurring in the anonymity system.

Here we examine the protection such systems can provide against a passive attacker (i.e. one who watches the network rather than tries to compromise nodes). Some systems, MorphMix [RP02] for example, explicitly state that they are vulnerable to such an adversary; our work may provide insight into ways of strengthening them. Others, notably Tarzan [FM02], employ an expensive dummy traffic policy in an effort to protect against this adversary. We show that it is possible to avoid this.

It is important to note that in this chapter we consider connection-based anonymity systems which are running on top of standard Internet protocols, i.e. packet-switched networks. We do not consider schemes over circuit-switched networks such as ISDN mixes [PPW91].

7.2 Systems and Usage

We begin by outlining the application scenario, high-level anonymity goals, and system architecture that we consider in our analysis. The architecture presented below is a distillation of the key choices of Onion Routing, Tarzan, and MorphMix [DMS04, FM02, RP02]. None of these have been widely deployed, although the first generation of Onion Routing was a prototype with 5 nodes.

Scenario We are primarily considering systems for anonymous web browsing. A number of *users*, running anonymity clients, connect through the system to some *web servers* (not running special software). HTTP requests and responses both travel through the system.

System goals Such a system should:

1. provide usable web browsing, with no more than a few seconds additional latency; and
2. make it hard for an attacker to determine what website any given user is browsing

The system need not protect against the attacker determining *that* the user is browsing, or which web servers are being accessed through the anonymity system. The system should, of course, also protect against the webserver determining who is browsing the website, unless it is compromised by an application-level feature (e.g. cookies).

In particular, as we discuss below, the system should protect a user against an attacker who can observe all traffic on the path of their connection. Note that to do this, the adversary does not need to observe all traffic in the anonymity system.

The goals clearly involve a trade-off: the more delay, the higher the (potential) anonymity.

Architecture We make some basic assumptions about the system structure:

- The system consists of a number of *nodes*. Some designs have a ‘classic’ architecture, with relatively few nodes, whereas others have a ‘Peer-to-Peer’ architecture, with nodes run by each user. Each node has logical *links* to some (not necessarily all) other nodes, along which it forwards traffic. Links are implemented above inter-node TCP connections between IP/port addresses, link-encrypted. To protect against node compromise, each connection passes through several nodes. Nodes also accept connections from end-user clients.

- To protect against the simple passive observer, who can bitwise compare traffic entering and leaving a node, traffic is onion-encrypted (as first suggested by Chaum in [Cha81]). This also provides bitwise unlinkability against collaborating nodes (see Section 2.3).

- The length of messages remains observable, so the data is divided into fixed-length *cells*. Typically these are small for efficiency reasons (in the original Onion Routing design, each cell carries 128 bytes of data).

- “Onion connection” setup is expensive. Hence all the cells for a particular client/server communication are routed via the same sequence of nodes. (Application proxying may reduce the number of communications, e.g. fetching all objects of a webpage in one communication.)

- Routes may be chosen either by the end-user or by the network.

This architecture broadly follows the design of second generation Onion Routing (Tor) [DMS04], Tarzan [FM02], and MorphMix [RP02]. Our results are also applicable to JAP [JAP] and the Freedom Network [BGS00].

Adding *dummy traffic* is a standard technique used in message-based anonymity systems e.g. Mixmaster. For connection-based systems, however, practical experience shows the bandwidth requirements of nodes are large, and so the additional cost of dummies must be minimised. Accordingly, in this chapter we assume that inter-node links do not involve dummies (though it turns out to be beneficial to apply some padding to the links between the client and the first node). We leave for future work the question of how a given quantity of dummy traffic can be most effectively used.

7.3 Threat Models

Prior work on threat models for connection-based systems has focused on the threat of malicious nodes, looking at how anonymity is affected by the fraction of attacker-controlled nodes [Shm03, STRL00].

In this chapter we focus on the threat of traffic analysis by a *passive* observer. Earlier notions of "global passive" attacker, as used in analysis of message-based systems, are too vague for connection-based systems. The threats must be stated more precisely: the quality (time accuracy) of the traffic data available to different global passive attackers may vary considerably, making different traffic analyses possible.

There are several different low-level mechanisms an attacker might use to obtain traffic data, differing in the quality of data they make available, and in the effort required.

- Attacker-controlled nodes. Not considered here.
- By applying legal (or sub-legal) pressure to an ISP, a high-resolution traffic monitor can be installed on a machine within the same collision domain as a node. This could capture all IP packets travelling to and from other nodes, with precise (sub-millisecond) timestamps; that data could be forwarded online to an analysis machine. Note that if nodes are distributed among judicial domains, it is hard to attack a substantial proportion of them.
- By compromising a machine in the same collision domain as a node the same data could be captured, though here there may be difficulties in surreptitiously forwarding it to the analyser.
- By installing non-intrusive fibre taps 'in the field', on the fibres that carry traffic between nodes [Hod91], one can capture similar data, but here, as there are typically routers between a node and an external fibre, some timing accuracy will be lost (several router delay variances). How many such attackers are required to intercept all node-to-node communications depends on the topology, but typically examining just backbone fibres will not suffice.
- Traffic data can also be obtained by compromising a router on each of the inter-node links and placing traffic monitoring code there. However, here the attacker is more likely to get per-link packet counts (over large fractions of a second) rather than per-packet data with timestamps. These can be retrieved via the standard SNMP protocol. More accuracy can be obtained by compromising routers closer to each node.

Broadly, all these attackers gain access to the same class of data – the number of packets that travel between pairs of nodes (on anonymity-system logical links) during particular time intervals. The packet counting interval determines what kinds of traffic analysis the attacker can perform: taking long intervals amounts to low-pass filtering of the data, erasing informative high-frequency components.

A further distinction is between *per-interval* and *waveform* analysis. In the former, each packet-counting interval is treated separately – the attacker can forget the data for each interval after it has been analysed – whereas in the latter a substantial

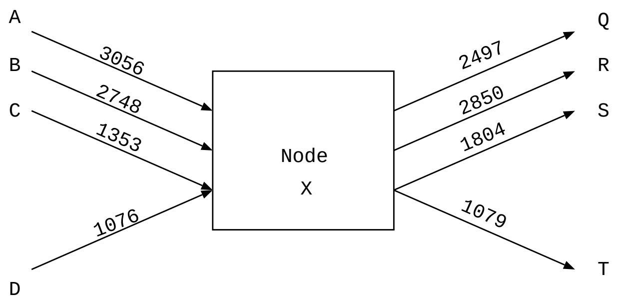


Figure 7.1: Each arrow represents an incoming or an outgoing link. The number on the arrow represents the number of packets observed by an attacker on the link over one time period.

region of the traffic waveform must be considered. The latter attacks may obviously be more expensive to mount.

7.4 Analysis: Lone Connection Tracking

Our first attack is based on *packet counting*. We recall that traffic travels down a connection in small cells. Consider a node in the system. During a particular time interval the number of packets on each of the connections travelling through it is highly likely to be different. The attack is based around the idea that if the attacker could see the numbers of packets of each connection coming into the node during this time interval and the number of packets of each connection coming out of the node, then he should see which incoming connections corresponds to which outgoing ones. It is worthwhile to note that the links of a connection-based anonymity system are typically implemented above TCP. The attacker watching the link can easily distinguish packets which are carrying encrypted data and should be counted because they carry data from the anonymity system from the others (such as ACKs) which should not. Such a packet counting attack also requires the delay introduced by the node to be small compared to the size of the time interval, so the number of incoming and outgoing packets of the node on each connection will be very similar. They will not be identical as some packets will have been in the node before the interval started and some will remain after the interval ends.

However, a passive attacker can observe the number of packets of a connection which arrive at the node and leave the node only if this connection is *lone* – it is the only one travelling down its incoming link and its outgoing link during the time interval. This scenario is illustrated on Figure 7.1.

It is clear that the numbers of packets on links from D to the node X and from X

to T are very similar, so the attacker can be confident that the connection(s) from D have been forwarded to T. Naturally, there is a possibility that he is mistaken: one of the connections from A, B or C could have carried 1079 packets and been forwarded to T, while the connection(s) from D were forwarded to Q, R or S. We now go on to examine this in more detail.

Consider a node which has d incoming and outgoing links, and c connections during the current time period. Now take a target connection which has arrived on a link lone, and had its number of packets, *pack*, observed by the attacker. Now, what is the probability of this connection exiting on a link lone and thus being compromised?

$$p_{\text{compromise}} = \left(\frac{d-1}{d}\right)^{c-1}$$

Now consider the probability of the attacker making a mistaken inference from his observation of similar packet counts on an incoming and an outgoing link. For simplicity, let us say that the attacker only declares the correspondence between an incoming link and an outgoing link if the packet counts on them are *the same*. Clearly, we must calculate the probability of *pack* packets (which do not belong to the target connection) appearing on exactly one of the outgoing links. Suppose that the attacker has some knowledge of the probability distribution V (volume) of the number of packets on a connection going through the anonymity system (this is easy to obtain, he can simply watch the unpadded links between the anonymity system and the webservers). It is important to note that *pack* packets on a single link can be caused either by a single lone connection other than our target connection or by several connections on the same outgoing link with a total packet count of *pack*. The probability of the latter is likely to be small compared to the former[1], hence we ignore it. Naturally, when we are able to obtain an estimate of V from a deployed anonymity system, we will be able to compute the error we introduced by this simplification precisely. Similarly, we only consider the case where one of the incoming connections has *pack* packets on it (and is, hence, linear in $V(pack)$ which is small[2]). Now we can calculate the probability of there being one incoming connection with *pack* packets and it being forwarded to a link alone.

$$p_{\text{error}} = (c-1)V(pack)(1-V(pack))^{c-2}\frac{d-1}{d}\left(\left(\frac{d-1}{d}\right)^{c-2} - \left(\frac{d-2}{d}\right)^{c-2}\right)$$

[1]As an analogy, the reader is invited to consider the probability of getting a 3 from a roll one die, compared to getting a 3 from a roll of 2 dice.

[2]At this point it is worth considering how small $V(pack)$ might be. If each cell carries 128 bytes of data, then the rate of flow of cells through a connection could be as high as several hundred per second. Naturally, V is not uniform, but it is still reasonable to expect that the highest value of $V(pack)$ is no more than 0.1.

Naturally, the more links and the fewer connections on them, the lower the probability of error and the higher the probability of a compromise. Note also that, as we argued above, $V(pack)$ is small, hence the overall probability of error is small. The attacker can further reduce this probability by doing the same observations during a different time interval and checking the results are consistent.

Note that this attack does not require a global packet counting attacker, merely one who observes all the links a connection travels on.

This attack is based on other assumptions, some of which we have touched on already:

- The packet counting interval is much larger than the mix delay. This is necessary as otherwise the packets inside the mix at the starts and ends of packet counting intervals will make the incoming and outgoing packet counts dissimilar.

- The packet counting interval is much smaller than the mean time between new connections being set up on this pair of links. The longer the counting interval, the more likely there is to be a new connection initiated which will traverse an incoming or an outgoing link, thereby ruining the packet counts and thus the attack. Note that if the adversary is unable to obtain packet counts for short enough time periods (e.g. if he is extracting packet counts via SNMP), he loses some opportunities for attacks.

It may seem that the attacker can just as easily count packets coming in from the users to the first node of the connection and try to correlate this with the packet counts coming out of the anonymity system to the webservers. This is the extreme version of packet counting across several nodes. Such an attack is less likely to succeed (and easier to protect against) for the following reasons:

- In the description of the attack on a single node, it was necessary for the packet counting interval to be much larger than the node delay. Thus, in the case of mounting the attack on the anonymity system as a whole, the packet counting interval will have to be made much larger than the delay on all the nodes of a connection together plus all the link delays. This increases the chances of the user initiating another connection to the same first node, thereby confusing the packet counts. Implementors of connection based systems should note that this can be a basis of a good and cheap defence strategy (though it relies on the first node being uncompromised) — all the connections the user makes should be established through the same first node.

- A small amount of padding between the user and the first node protects against the attack. This requires much less bandwidth than padding each link in the anonymity system as suggested by [Ren03]. Of course, it would be desirable to also pad the link from the last node to the webserver, but this is impossible as the webserver is not running anonymity software.

- It may be possible to arrange for the first link not to be observable by the attacker (e.g. run a node at the edge of a corporate network and ensure that everyone inside the company uses it as their first node).

Having shown that lone connections allow the attacker to compromise anonymity, we now calculate how many such connections a system is likely to have under different conditions (and thus how likely a user's connection is to be compromised). First, we derive an approximation, and then examine the subject in more detail using a simulator. Finally we suggest ways of defending against this attack.

7.4.1 Mean-based analysis

We assume the users initiate on average c connections per second, each forwarded along ℓ links (inside the network) and that there are n nodes in the anonymity system. Furthermore assume each connection has duration dur.

Thus on average at any instant there are $c \times dur$ connections. Each connection exists on l links, so on average there are $c \times dur \times l$ link-occupancies. Links are, of course, unidirectional as the attacker can distinguish packets going from P to Q from those going from Q to P. If there is a link between each pair of nodes, there are roughly $n \times n$ links[3]. On average there are $c \times dur \times l/(n \times n)$ connections per link. It is clear that the absolute lower bound of the number of connections per link is 1, and for a good anonymity system this number should be much greater.

Let us illustrate this with an example. Suppose we have a system with $n = 30$ nodes, the users initiate connections through $l = 3$ network links (i.e. 4 nodes), each lasting $dur = 2$ seconds. If each node can talk to every other, then around 150 connection initiations per second are necessary for this system not to be susceptible to this attack.

7.4.2 Definitions

It is clear that the approximations calculated in the previous section are rather crude. We now proceed to define lone connections formally and show how to work out the fraction of lone connections of a particular system.

First, define the anonymity system graph as a set of nodes G with $|G| = n$ and a set of directed edges (links) E (ranged over by e), with each edge being a pair of nodes. A

[3]It is debatable whether routes with the same node occurring twice consecutively should be allowed.

path (of a connection), then, is a sequence of adjacent edges. Take all the connections c_i (of length l_i) which are open during a particular packet counting interval in some particular order and let $g = [[e_{1,1} \ldots e_{1,l_1}]; [e_{2,1} \ldots e_{2,l_2}]; \ldots]$ be the sequence of paths of these connections. We can easily express the number of connections on each link resulting from such a configuration.

$$f_g(e) = \sum_{p \in g} \text{occurrences of } e \text{ in } p$$

A connection is lone when it is lone on all the links it is going through. Now calculating the set (and therefore the number) of lone connections in a configuration is straightforward:

$$\text{lone} = |\{\!|p | p \in g \wedge \forall e \in p.f_g(e) = 1|\!\}|$$

We can also find the fraction of lone connections: $\frac{\text{lone}}{|g|}$.

We now go on to calculate the probability of a connection going through the anonymity system being lone.

First, let us assume some parameters of the anonymity system.

- $\Phi(c)$, the probability that c connections go through the anonymity system during the same packet counting interval. An attacker can easily obtain this probability distribution by observing the anonymity system for a while and building up the distribution. Indeed, the operators of the JAP system [JAP] are currently providing their users with c in real time as a very crude indication of the anonymity of their connections.
- $\Psi(j)$, the probability that a connection route is chosen to have length j. This distribution is encoded into the anonymity client, e.g. Tor, and hence can be easily obtained by an attacker.
- The graph of the anonymity system is given by G, E. This is clearly visible to a global passive attacker.
- The maximum number of connections which can go through a system is $\mathsf{max\_c}$ and the maximum route length is $\mathsf{max\_rt}$. The latter is an inherent parameter of the anonymity system determined by the size of the onions, the former can be obtained from observing the system for some time.

Now define $\mathcal{G}([l_1, \ldots, l_c])$, to be the set of all sequences of paths of lengths $[l_1, \ldots, l_c]$ in the graph G, E.

$$\mathcal{G}([l_1,\ldots,l_c]) = \{m| \quad \begin{aligned}[t] & m = [[e_{1,1}\ldots e_{1,l_1}];[e_{2,1}\ldots e_{2,l_2}];\ldots;[e_{c,1}\ldots e_{c,l_c}]] \\ & \wedge\, e_{x,y} \in m \Rightarrow e_{x,y} \in E \\ & \wedge\, ((i,j)_{x,y} \in m \wedge (k,l)_{x,y+1} \in m) \Rightarrow j = k\} \end{aligned}$$

Now, the probability P of a randomly chosen connection being lone is:

$$P = \sum_{c \in 0 \ldots \mathsf{max\_c}} \Phi(c) \times \sum_{\substack{L=[l_1,\ldots,l_c] \\ \forall i.l_i \leq \mathsf{max\_rt}}} \prod_{l \in L} \Psi(l) \times \sum_{g \in \mathcal{G}(L)} \frac{|\{\!|p|p \in g \wedge \forall e \in p.f_g(e) = 1|\!\}|}{|g|}$$

There are several assumptions which we have made implicitly in the above formula. Firstly, the probability distribution for path lengths of all connections is the same, $\Psi(l)$. Secondly, paths are chosen uniformly at random from all the possible paths (of length l) through the graph G, E[4]. Finally, all the path lengths and all the paths themselves are chosen by the connection initiators independently.

Although the above formula defines the probability of a connection going through an anonymity system unmixed, it is hard to see the quantitative implications of it directly. We therefore make a simulation of the anonymity system.

7.4.3 Simulator Results

We have constructed a simulator which uses the definitions above to calculate the fraction of lone connections. Given a graph of nodes connected by links, and the number of connections we wish to simulate, it picks a route length for each connection ($\Psi(j)$ is assumed to be a uniform distribution between a minimum and a maximum value) and then generates the routes themselves. Then it calculates the fraction of lone connections using the definitions above. Clearly, the fraction of lone connections going through the network is also the probability that a particular user's connection is going to be observed by the global packet counting attacker.

For example, let us take a Peer-to-Peer anonymity system with 100 nodes (each user running a node) all connected to each other. Suppose each of the 100 users has a connection ($\Phi(100) = 1$ during the packet counting interval we are considering) through a minimum of 2 and a maximum of 4 network links[5]. This system provides very low anonymity – around 92% of the connections going through it are lone.

[4]Hence the number of connections a node will handle is determined by the number of links it has to other nodes.

[5]Such a route length distribution is typical for Tor.

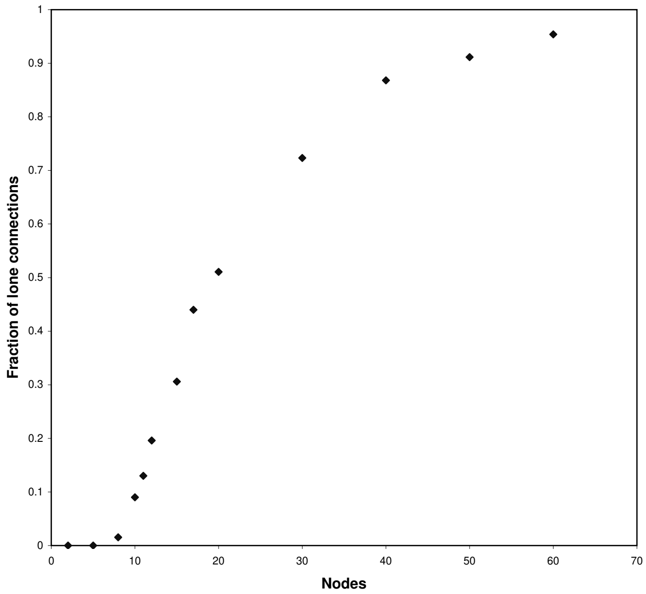


Figure 7.2: Fraction of lone connections (as viewed by a global packet counting attacker) vs the number of nodes in the anonymity system. There are 60 connections, each going through two network links.

A graph of the number of nodes vs the probability of connection compromise is shown in Figure 7.2. There are 60 connections going through the network and each connection is going through 2 network links.

We note that the fraction of lone connections is not the only measure of anonymity we could have used. Indeed, although it conveys a very clear message to the user (the probability of the connection they are about to establish being observable), it also suffers from some disadvantages. First, it does not indicate how many other connections a particular connections has been mixed with, as an anonymity set (or the information theoretic metric of Chapter 3) does. It is worth pointing out that if a connection is lone on some, but not all of its links, its anonymity set is very much reduced (which is not reflected in the probability of it being compromised). Secondly, the designers of the anonymity system might like to know the probability of at least one connection being compromised – a much stronger property. We leave calculating these metrics and analysing the attack in more detail to (rather tedious) future work.

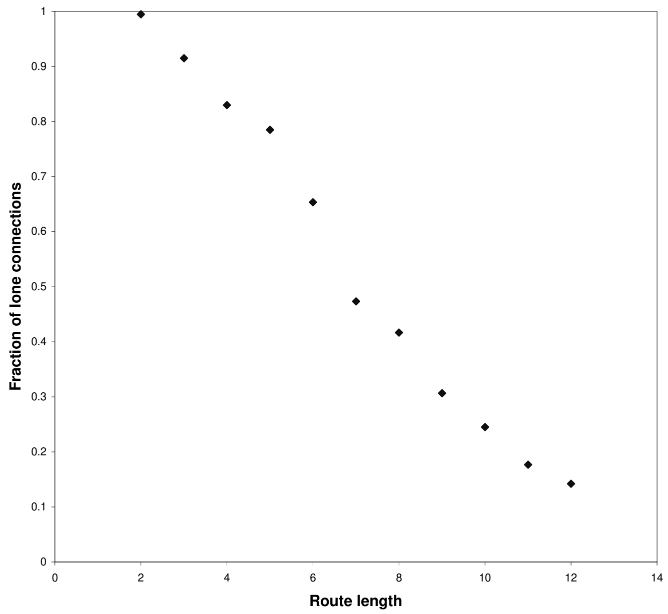


Figure 7.3: Fraction of lone connections (as viewed by a global packet counting attacker) vs route length. There are 50 connections and 50 nodes.

7.4.4 Protection

As we saw in the previous section, the packet counting attack on the lone connections is quite powerful against some systems. Here we examine ways of protecting against it.

Firstly, more traffic (and/or fewer nodes) makes the system much less vulnerable to the attack. Modifying the system from the example above to one with 20 nodes with 200 connections going through it (and keeping the route length the same at between 2 and 4 links) reduces the fraction of compromised connections from 92% to 2.5%.

Secondly, increasing the route length helps increase the total volume of traffic in the network, but also has the undesirable effect of increasing latency. For example, doubling the route length in our example above (100 nodes, 100 connections, route length of 4 to 8 network links) reduces the probability of a connection being compromised from around 92% to 72%. The graph showing how route length affects the fraction of lone connections is shown in Figure 7.3.

Thirdly, and most importantly, we can design the architecture of the system to suit the amount of traffic we expect to flow through it. If there is very little traffic, a cascade ought to be used. If there is slightly more, a restricted route architecture (see [Dan03b]) can be employed to dramatically decrease the fraction of lone connections.

For instance, for an anonymity system of 100 nodes, each able to forward traffic to 5 others (with route length of between 2 and 4 links and 100 connections), the fraction of lone connections is reduced to around 17%. This is still unacceptable, however, and suggests that making every client run a node is not a good choice for a strong anonymity system.

The fact that Peer-to-Peer anonymity systems do not provide much anonymity against the passive attacker who can observe all the links is an important conclusion which may seem counter-intuitive. However, it can be easily explained when we observe the fact that the connections have to "fill up" the network graph at least twice. If we run a node at every client, the number of connections will be within a constant factor of number of nodes. It is likely that each user does not have many active open connections with data flowing through them all the time, so this constant factor is typically less than one. The number of links in a fully connected network is roughly n^2, so the anonymity of such a network will be low. This is clear even from the crude calculations in Section 7.4.1.

As well as designing the system in a way which suits the expected level of traffic, we need to be able to handle daily or weekly variations in the number of connections established through the system. This may be possible by dynamically reconfiguring the network topology from cascade to restricted routes to full network. Of course, short term (intra-day) variations in traffic could not be handled in this way; this should be handled by suggesting to the users the number of nodes their connections should go through to stay anonymous.

7.5 Analysis: Connection-start Tracking

Our second attack is based on tracking the increase in the volume of traffic from an incoming to an outgoing link of a node which results from data starting to flow on a new connection. This increase happens when the webserver is starting to send the webpage data in response to a request made by a client. We call such an increase a "connection start". We note that the propagation of such a connection start through a node is observable to a packet counting attacker, even if the connection is not lone. If the node does not delay traffic significantly (as current systems do not), the attacker will observe a node in a steady state; then a start of connection arriving followed by a start of connection leaving, and will deduce where the new connection has come from and been forwarded to.

Hence, nodes must delay traffic (but still provide low latency communications). The most appropriate mixing strategy here is the Stop-and-Go Mix of Kesdogan et al. [KEB98]. It is easy to analyse, handles each packet separately and does not rely on batching. We proceed to describe this mix and examine how it can help us protect against the above attack.

The Stop-and-Go mix treats each packet independently. When a cell arrives, the mix draws a random value from an exponential distribution with parameter μ and delays the cell by that time. The mean delay of the mix is thus $1/\mu$.

Assume the users initiate (on average) c connections per second, each going through ℓ nodes. The system consists of n nodes. Write λ for the mean rate of arrival of starts of connections (per second) to a particular node. We have $\lambda = c\ell/n$.

Assume further that the arrivals of the starts of connections to the node are Poisson distributed (with parameter λ).

This is essentially the $n-1$ attack scenario described in [KEB98], though performed here for starts of connections instead of individual asynchronous messages. We want to choose the parameters λ and μ such that the probability of the attacker tracking a start of connections through all the mixes is small.

Now, the attacker tracks a connection through a mix if and only if:

1. When the start of the connection arrives, the mix is "empty of starts of connections". This means that there has not been an incoming start of connection that has not been followed by an outgoing one. From queueing theory, the probability of this is: $e^{-\lambda/\mu}$.
2. Having arrived on an incoming link, the start of the connection leaves the mix whilst no other start of a connection has arrived. The probability of this is equal to the probability of the delay of the message being smaller than the time of the next message arrival, $\frac{1}{1+\lambda/\mu}$.

Now to get the probability that a connection is tracked through one node, we just multiply the two together.

$$\frac{e^{-\frac{\lambda}{\mu}}}{1+\frac{\lambda}{\mu}}$$

The probability of the attacker tracking a particular connection which is going through ℓ mixes is:

$$\left(\frac{e^{-\frac{\lambda}{\mu}}}{1+\frac{\lambda}{\mu}}\right)^{\ell}$$

substituting in the expression for λ from above gives:

$$\left(\frac{e^{-\frac{c\ell}{n\mu}}}{1+\frac{c\ell}{n\mu}}\right)^{\ell}$$

A user of this system would incur a delay of roughly $2\ell/\mu$ seconds for onion connection setup, ℓ/μ for a request and ℓ/μ for a response, or a total *connection delay* of: $4\ell/\mu$.

We will now do some order-of-magnitude calculations to see how much traffic needs to go through the anonymity system to get acceptable delay and anonymity values and then show graphs of the probability of having the connection compromised in a variety of different scenarios.

Clearly, as long as $\ell \geq 3$ and $c\ell/n\mu \geq 1$, the probability of tracking a connection is low (< 0.006). Hence, $c\ell/n\mu \geq 1$ or $c\ell \geq n\mu$.

Suppose $\ell = 3$ and the maximum acceptable delay is 2 seconds, hence $4\ell/\mu = 2$, hence $\mu = 2\ell = 6$. Substituting in, we get $c \geq 2n$. This implies that the users of the system have to initiate twice as many connections per second as there are nodes.

Suppose we have U users browsing every day. If each browses 100 pages a day, $c = U \times 100/(3600 \times 24)$.

Now suppose we have an anonymity system of 30 nodes. We find the number of users U needed for the system to provide anonymity. $U \times 100/(3600 \times 24) \geq 2 \times 30$, or $U \geq 2 \times 30 \times 36 \times 24 = 51000$. This is a realistic target for a small to medium size anonymity system.

Naturally, the calculation above is rather crude as it involves the mean amount of traffic (and suppose that the traffic is evenly distributed throughout the day). More traffic is required to to protect against traffic troughs (e.g. at night).

We now show how the probability of a connection being tracked through the anonymity system is affected by the various parameters. In Figures 7.4, 7.5, 7.6, 7.7 the probability of a connection being compromised is plotted against route length, the delay parameter of the mix, the rate of incoming connections and the number of nodes in the system (respectively).

It is worth considering the quality of traffic data the adversary has to have access to to mount such an attack. If a timestamp for every packet is available, the attack can be mounted with maximum effectiveness. However, if the adversary can only do packet counting over some time intervals, then the time interval must be much longer than the node delay and much smaller than the interarrival times of starts of connections *to a node*. Note that this is more precision than was required for the lone connections attack (the time interval there had to be much less than the interarrival times of connections *on a single link*).

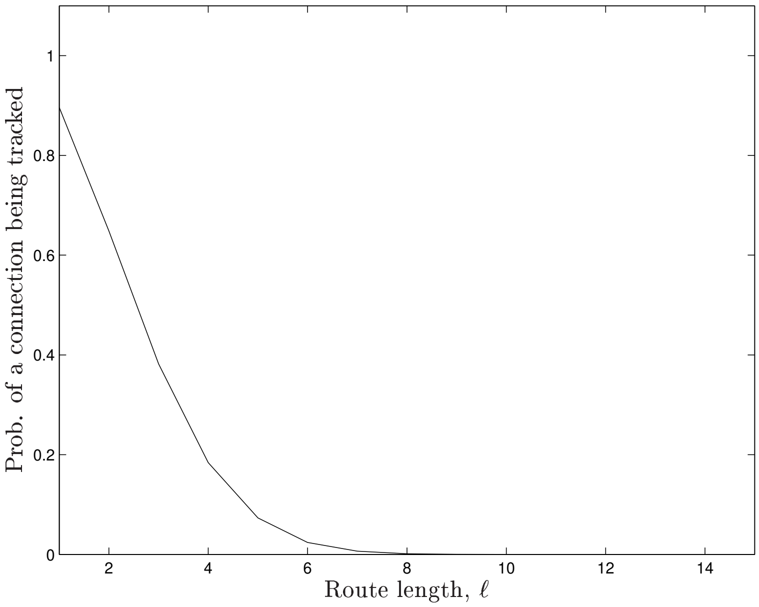


Figure 7.4: Probability of a connection being tracked vs route length. $\mu = 6$, $c = 10$, $n = 30$.

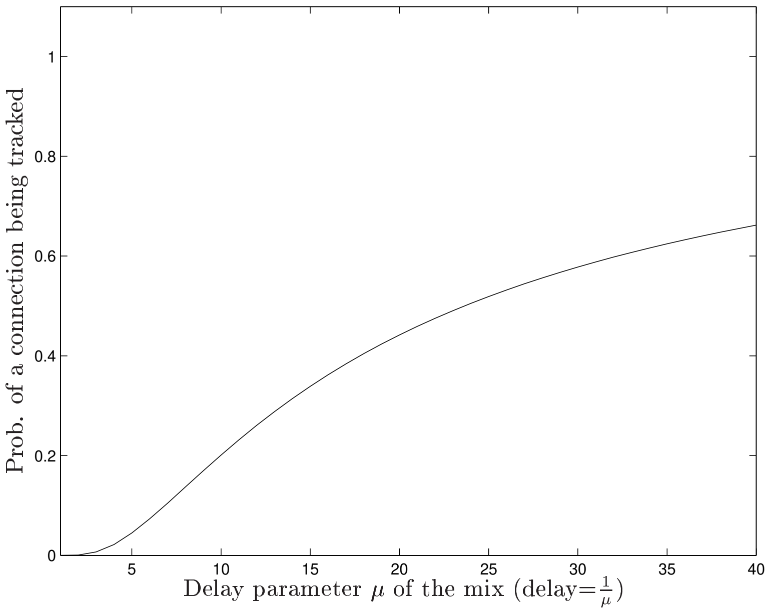


Figure 7.5: Probability of a connection being tracked vs delay parameter. $\ell = 5$, $c = 10$, $n = 30$.

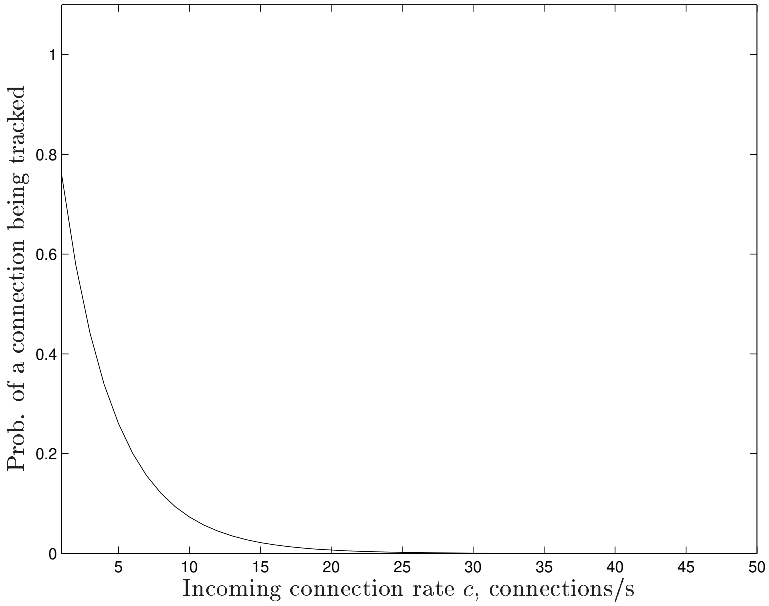


Figure 7.6: Probability of a connection being tracked vs connection rate length. $\mu = 6$, $\ell = 5$, $n = 30$.

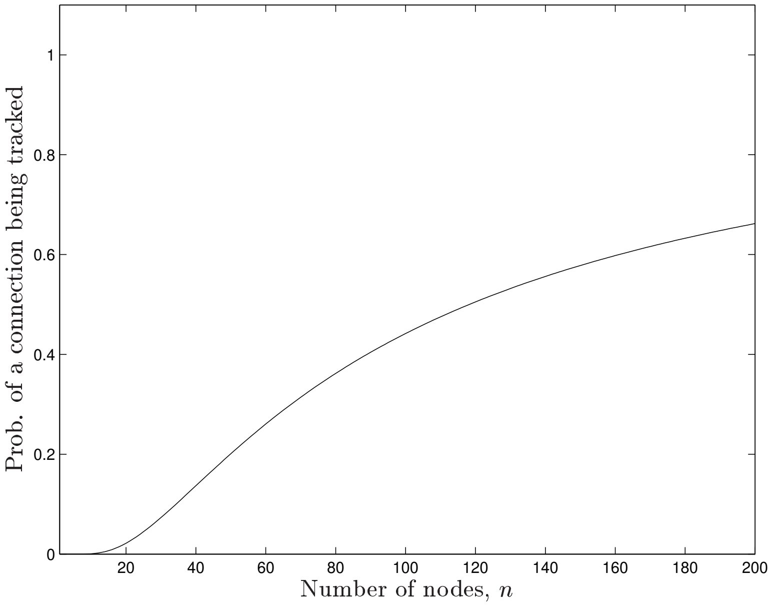


Figure 7.7: Probability of a connection being tracked vs number of nodes. $\mu = 6$, $\ell = 5$, $c = 10$.

7.5.1 Working with Richer Traffic Features

We have so far considered starts of connections and showed that if these are allowed to propagate through the network then a certain level of traffic is required to maintain anonymity. Now we consider how the attacker could use more general traffic features (e.g. spikes) to track individual connections. This is an example of a waveform analysis – data from several intervals will be required.

Let us consider a simple case of a node with 2 incoming and 2 outgoing links. The adversary sees a spike on one of the incoming links (say from A) and one of the outgoing links (to Q) some time later. For clarity of exposition, we suppose he knows that both the links which exhibited spikes have lone connections on them[6], but the other links (from B and to R) contain many connections, so some spikes may be hidden. The smart adversary does not jump to conclusions about a correlation between the links with spikes, but instead calculates the probability of it.

There are two possibilities: Either the attacker is correct and the spike from A really went to Q, or the attacker is mistaken and there was a hidden spike which came in from B and went to Q, while the spike from A got forwarded to R and hidden in the traffic.

The probability of the former is $1/2$ (assuming the connection from A was equally likely to be forwarded to Q and R). The probability of the latter is $P(\text{spike}) \times 1/2$. Hence, the attacker is correct with probability $\frac{1}{1+P(\text{spike})}$. The probability of a spike occurring on a link is low (spikes do not occur often, and a similar argument to the one we presented in Section 7.4 applies), so the probability of the attacker being correct is high.

A complete analysis is not presented here – we would need to examine real connection traffic to determine the probability of spikes occurring and determine what other kinds of traffic features we might make use of. It is notable that interactive applications like SSH are much more vulnerable to this kind of attack than web browsing as the traffic is much less uniform. In general one would expect to use signal-processing techniques – filtering the signal to frequency ranges known to include identifiable features and/or calculating running correlations between signals and common feature patterns. We leave this for future work.

7.6 Discussion and Solutions

In the previous sections we looked at two powerful attacks which can be mounted on connection-based anonymity systems by passive attackers, evaluated their effective-

[6] This constraint is easily relaxed.

ness and assessed potential protection measures.

Unlike previous analyses in the literature, we stayed clear of using vague and costly proposals of adding dummy traffic to the system, instead calculating the number of user connections required to maintain anonymity. This approach is crucial for building efficient, fast, and therefore deployable connection-based anonymity systems, whilst still providing anonymity to the users.

However, we have not examined all the attacks which the adversaries can potentially mount against connection-based anonymity systems. In particular, in this chapter we have not considered the "first and last node" attack or any active attacks. We comment upon them briefly here.

The "first and last node" attack involves the attacker compromising two nodes and then using them to identify the connections which start and end at these nodes. He can filter padding from the client to the first node (if there was any) and modify traffic travelling in both directions. In particular, he can insert a signal into the inter-arrival times of cells of a particular connection and then look for it (low-pass filtered to account for the variances in network and mix delays) on the other side. As the packets are small, the signal is likely to carry a substantial amount of information and help the attacker succeed. Note that an active attacker who can modify traffic on links (but has not compromised any nodes) has the same capability.

There are several potential countermeasures which will help make this attack less powerful. First, longer routes will help reduce the amount of signal which propagates from the first to the last node. Secondly, increasing the packet size (and thus decreasing the number of packets) will help reduce the size of the signal which can be inserted into the connection. In the limit, if all webpages fit into one packet, active attacks become ineffective (though this comes with a massive efficiency loss).

We have not yet commented upon the potential use of traffic shaping as a countermeasure to both the attacks presented here and future, more sophisticated ones. It is worth noting that such a traffic shaping policy would have to make all the connections in the anonymity system have the same profile, which is likely to be expensive in terms of introducing delays or bandwidth (dummy traffic). We have not investigated this mostly because protection against the attacks outlined could be achieved by cheaper means.

One of the implications of the results presented here is that anonymity systems which involve all the users running nodes are impractical simply because there is not enough traffic to fill all the links. It is evident that adding nodes provides *less* anonymity (contrary to popular belief) against the global passive attacker. Whether this statement is true for the case of partial attackers remains the subject of future work. Of course, increasing the system size makes it harder to be a global attacker or compromise a certain fraction of the system.

Another interesting line of research is to see whether an anonymity system could be built using an architecture which elevates a subset of the Peer-to-Peer nodes to a "supernode" status. The role of such supernodes would be to perform mixing, while the other, "lesser" nodes would simply forward traffic. This may yield a good compromise between the performance and scalability of current Peer-to-Peer architectures and resistance against passive attackers.

7.7 Related Work

As mentioned before, there is not a great deal of work on the analysis of connection-based anonymity systems. Qualitative analyses include [BMS01, Ray00]. Quantitative analysis has so far been limited to evaluating the impact of compromised nodes on the anonymity provided. Wright and coauthors examine this issue in many different anonymity systems [WALS02], while [STRL00] gives a detailed account of the security of the first generation of the Onion Routing system against compromised nodes.

To the best of our knowledge, the first work which describes the packet counting attack is the analysis by Back, Möller and Stiglic [BMS01], however, they fail to specify the attack precisely. Clearly, simply from observing similar packet counts on an incoming and outgoing link, the inference that a connection is forwarded from one to the other cannot be made. Indeed, in this work we showed how to calculate the probability of such an inference being correct and erroneous in Section 7.4. Neither do they investigate the parameters of the anonymity systems which make them more or less vulnerable to this attack, or how to protect against it.

Another recent work [Ren03] looks at packet counting attacks of the end-to-end variety, arguing that over time an intersection attack combining all these will succeed. However, the author does not calculate the probability of success, arguing simply that protection against these using dummy traffic is necessary. He then proposes a constant dummy traffic policy which is likely to be too costly to be acceptable to volunteers who might run nodes in a connection-based anonymity system (though the dummy traffic policy would also protect against some attacks which we have not considered).

The connection-start tracking problem was observed in [PPW91]; there it is solved by dividing the connection into a number of "time slice channels". It is interesting to consider the ISDN mixes scheme in the context of the current infrastructure (each user having a DSL connection to the Internet). It requires two independent channels to be used all the time, which imposes large overhead on the ISP of the users. ISPs currently implement bandwidth caps to discourage "heavy bandwidth use", hence a solution which involves constant padding is unlikely to be acceptable to them[7]. This scheme requires extra overhead and its effectiveness remains to be evaluated.

There are also systems which provide anonymous connections for web browsing [SBS02] which do not follow the "mix" architecture of Chaum, but they also lack quantitative analyses of the anonymity provided.

7.8 Summary

We examined in some detail two attacks which can be mounted by passive adversaries on connection-based anonymity systems. These compromise current (experimental) anonymity systems completely as they do not delay packets at nodes and have little traffic. Similarly, we do not feel it is appropriate to compare system X against system Y at this moment as parameters of these systems crucial to the analysis change dramatically over time and will hence invalidate any results we can come up with. We feel that having provided easy methods of analysis, the system designers can use them easily enough to ensure their system is not vulnerable to attacks described here. In particular, we showed that the threats can be protected against without resorting to dummy traffic and keeping the delay to users' connections acceptable.

Finally, the work in this chapter shows that analysis of connection-based anonymity systems is just as feasible as that of message-based ones. Furthermore, such analysis is required to develop and evaluate methods of protection against real threats. As a promising direction for future work, we suggest that mounting real attacks on implemented (and deployed) anonymity systems will provide further insight into the measures necessary to keep anonymity systems anonymous.

We now finish our rather brief excursion into the analysis of connection-based systems and go on to look at an application which can be built on top of strong anonymity systems.

[7] A scheme where each ISP runs a node may be more acceptable as it will cause no additional external traffic, but it does require the co-operation of the ISPs.

Chapter 8

Anonymity and Censorship Resistance

"Any problem in Computer Science can be solved by another layer of indirection"

— David Wheeler

In the previous chapters we presented various analyses of both message-based and connection-based anonymity systems. In this chapter, we assume a strong anonymity system and build a censorship resistant scheme with strong properties on top of it. This provides a novel application for the anonymity system and also strengthens previous work on censorship resistant systems.

8.1 Introduction

Back in Chapter 1 we remarked that censorship resistance is a property which essentially amounts to long-term availability in an adversarial environment. More specifically, censorship resistant systems have to cope not only with byzantine failures, but also "real world" attacks aiming to remove content from the system.

Many censorship resistant systems have been proposed recently, yet most still lack one crucial feature – protection of the servers hosting the content.

In the past this was not considered an issue. For instance, in Anderson's Eternity Service [And96], it was deemed sufficient to guarantee that a document was always available through the system. However, many examples indicate that servers hosting content are vulnerable to censorship, due to "Rubber Hose Cryptanalysis" – various kinds of pressure applied by attackers to shut down servers or remove files. Examples of documents subjected to censorship on the Internet include DeCSS [Tou], the paper detailing the attack on SDMI [$CWL^{+}01$], and documents (or quotes from documents) which the Church of Scientology described as their secrets. In cases like this, the server administrators receive "cease and desist" letters when the censor finds the offending document on their server.

Most modern censorship resistant systems, for example Publius [WRC00] and Freenet [CSWH00], have not addressed this problem in a satisfactory way. It is possible for a malicious reader to find out which servers some particular piece of content is stored on, and subsequently try to pressure the server administrators to remove it. With Publius in particular, the situation is slightly more complicated as each document is encrypted and the key is split into n shares, any k of which are recombinable to form the document back. In this case, the attacker needs to remove content from $n - k + 1$ servers, but can easily locate all of them[1]. With the number of servers reasonably small and static, the job of censoring documents becomes easier than one might expect.

An alternative approach to dealing with censorship resistance is taken by systems such as Dagster [SW01] and Tangler [WM01] which prevent removal of any single document from the system by entangling documents together. However, in our view, these are not effective enough at dealing with the problem either: if the offending document was entangled with the Declaration of Independence, Das Kapital and the Little Red Book, censoring it (thereby removing all of the above from the system) would not be a major problem as all the other documents are readily available from other sources. Even if the other documents were *not* available elsewhere, the censor is unlikely to be discerning enough to want to keep them. Furthermore, if the system is run on a single server (e.g. Dagster), then the censor may simply try to shut down the entire server.

The rationale behind the design of our system can be seen by considering a simple attack which works against most other censorship resistant systems. First of all, we need to consider how the attacker locate certain piece of content, or even a part (share) of it. In Publius, for instance, the name of the machine on which each share is stored is included in the address of the document. Of course, the storer can deny the fact that he is storing a particular share, but, if watched by the attacker, he will be faced with a dilemma of whether to answer requests (and thus reveal the fact that he is storing the share) or drop the request, effectively censoring the document. Once the attacker has obtained proof that a storer has, and has been answering requests for a particular share, legal pressure can be applied. We will return to this attack later in this chapter to see how the various features of our proposed design help protect against it.

[1]Indeed, if one server has been pressured into removal, the other server administrators may simply follow the precedent and remove the offending content themselves.

Broadly, in our system, we consider the storers of the files to be valuable entities, and protect them against "Rubber Hose Cryptanalysis". Furthermore, we protect the documents they are storing by providing *active-server document anonymity* (as first introduced in Free Haven [DFM00a]). This is the property that the storer should not be able to determine (parts of) which document it is storing, not even by retrieving the document from the system. We now proceed with a description of the system and then give an analysis and critique of it.

8.2 System Description

Our system consists of many identical peers, each of which can fulfil five different roles:

- Publisher p. The node which has a document and wishes to make it available and censorship resistant.
- Storer s. A node which stores part of a document.
- Forwarder a. A node which has an anonymous pointer to a node storing part of a document.
- Client c. A node which retrieves a document.
- Decrypter l. A node which decrypts part of a document and sends it off to the client.

The system is built on top of an existing Peer-to-Peer document storage service such as PAST [DR01]. PAST itself is built on top of Pastry [RD01], a Peer-to-Peer routing scheme. Pastry can be viewed as a network of machines (peers), each with a unique identifier. The only thing required to send a message to a machine is its id, furthermore, Pastry guarantees that the message takes around $\log N$ hops, where N is the number of nodes in the system. Using an existing Peer-to-Peer architecture allows us to abstract away routing, clients leaving and joining the network, and other low level issues.

On top of Pastry, PAST also provides robustness: neighbouring machines (machines within a certain distance of each other within a logical namespace) share state. This is further discussed below. We also assume a public key infrastructure, so any peer is able to learn any other peer's public key. This is further discussed in Section 8.4. Finally, we make use of an anonymous connection system such as Mixminion [DDM03] which is capable of handling replies.

As usual in censorship resistant systems, the operations available to a node are publishing and retrieval. There is no search facility, therefore we rely on a broadcast

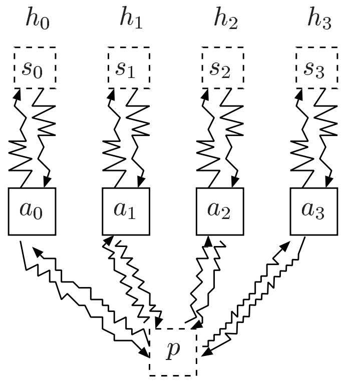


Figure 8.1: Publishing. Zig-zag lines indicate anonymous connections.

mechanism such as an anonymous newsgroup to transmit retrieval information to potential readers. We do not support content deletion or modification.

8.2.1 Publishing

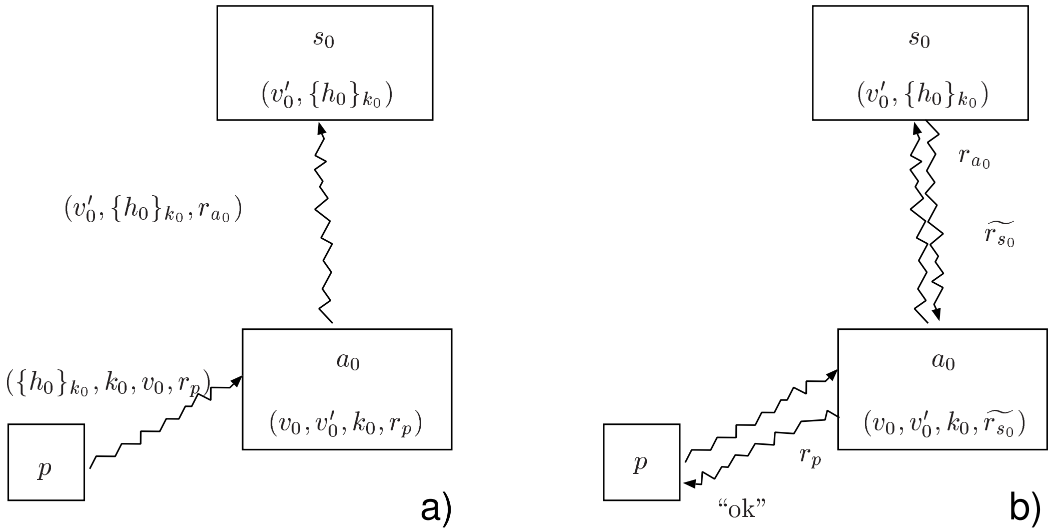
The overall publishing process is illustrated in Figure 8.1. The main idea is to split the documents into many parts or shares h_i, and store them (encrypted) on machines s_i, while making them accessible through machines a_i which forward requests for the appropriate shares anonymously.

Publisher:

To publish a document (see Figure 8.2), the publisher p splits it into $n+1$ shares h_i, any $k+1$ of which can be combined to form the whole document again. This can be done using one of the standard algorithms such as Shamir's secret sharing [Sha79]. He then generates $n+1$ keys k_i and encrypts each share with the corresponding key. He now picks $n+1$ peers $a_0 \dots a_n$ at random to act as forwarders and constructs onions to send (via the anonymous connections layer) each of them the encrypted share $\{h_i\}_{k_i}$, the corresponding key k_i[2] and a (large) random integer v_i together with a return address (reply onion)[3] r_p. The publisher can now wait for a confirmation to come back from each of the a_i's (via the reply onion) saying whether the publishing has been successful or not. If the operation failed, the publisher should try different a_i's.

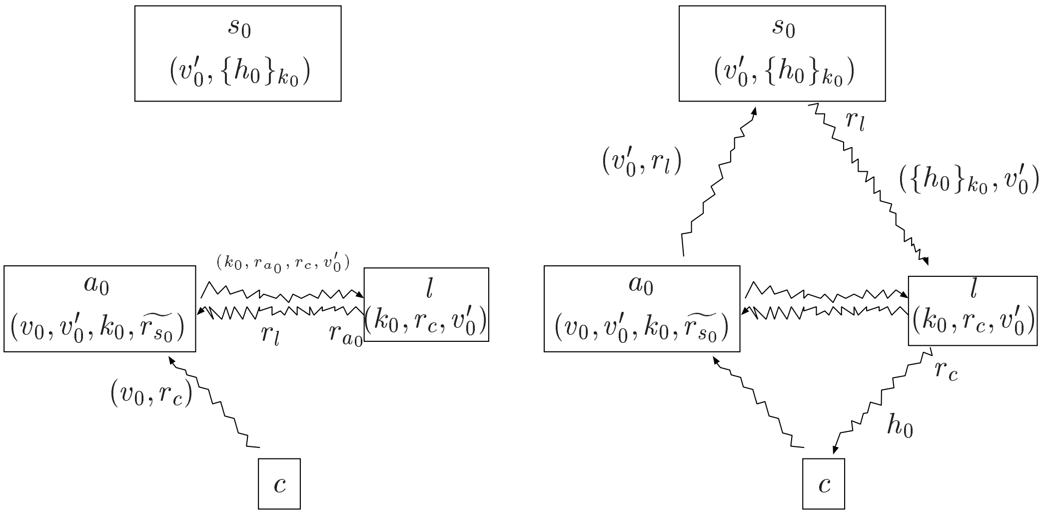
[2]Both $\{h_i\}_{k_i}$ and k_i are sent to minimize the work which has to be done by the forwarder.

[3]A return address is a kind of onion which, if included in an anonymous message, can be used to reply to that message without revealing the original sender (see [Cha81] for details).



Share	h_0
Key	k_0
Storer	s_0
Random numbers	v_0, v_0'
Return addresses	r_p, $\widetilde{r_{s_0}}, r_{a_0}$

Figure 8.2: Inserting share h_0. All communication is done via the anonymous connection system using randomly constructed onions. If the message is sent using a return address, it is displayed at the base of the arrow. Anonymous return addresses are denoted by r, e.g. r_{a_0}



Share	h_0
Key	k_0
Storer	s_0
Client	c
Decrypter	l
Random numbers	v_0, v'_0
Return addresses	r_{a_0}, r_{s_0}, r_l, $\widetilde{r_{s_0}}$, r_c

Figure 8.3: Retrieval

Forwarder:

The forwarder (all of them perform the same operation, here we use a_0 as an example) receives the message, finding an encrypted share $\{h_0\}_{k_0}$, a key k_0 and a random number v_0 and the publisher's return address. He then picks a storer s_0 to store the share and a number v'_0 which the storer should associate the share with. He constructs an onion for delivering these to the storer. Thus, he puts the encrypted share, v'_0, as well as his own anonymous return address r_{a_0} into the onion as the message and sends it off (see Figure 8.2a). He remembers v_0, v'_0, k_0 and r_p. When the onion is received by s_0, it stores the share and issues a number of different return addresses $\widetilde{r_{s_0}}$ (to be used for retrieval), sending them back to a_0 via the return address r_{a_0}. Now a_0 associates v_0, v'_0 and k_0 with the return addresses $\widetilde{r_{s_0}}$, forgets s_0, and replies "ok" to the publisher via r_p. Once all the shares have been stored, the publisher destroys them and announces the name of the file, together with the $n+1$ pairs (a_i, v_i) to potential users.

8.2.2 Retrieval

To retrieve a document (see Figure 8.3), the client c asks the forwarder a_0 (and each a_i in the same way) to retrieve the share h_0 by sending them an anonymous message with v_0 and their anonymous return address r_c. The forwarder a_0 then picks a random server l to act as a decrypter and sends it k_0, the key it is storing which decrypts the stored share, v'_0, r_c, and a return address r_{a_0}, getting back a return address for l. Now a_0 forwards r_l and v'_0, which identifies the share, to s_0 via one of the $\widetilde{r_{s_0}}$ (r_{s_0}). Now s_0 looks up the encrypted share corresponding to v'_0 and forwards it and v'_0 to l, which decrypts the share and sends it to the client via r_c. The process continues until c has accumulated enough shares to reconstruct the document.

We note that when the forwarder starts running out of return addresses for the storer (you can use each one only once), all the forwarder needs to do is request some more via one of the return addresses it still has.

The other important detail which we have so far left out of the description of the system is that the Peer-to-Peer storage layer (PAST) replicates state among neighbouring nodes. This enables requests to be routed to any of the nodes which contain the replicated state. In particular, the forwarder shares $(v_0, v'_0, k_0, \widetilde{r_{s_0}})$ with neighbouring nodes which can therefore also answer requests. Similarly, the storer shares $(v'_0, \{h_0\}_{k_0})$. The decrypter does not need to share anything as he will only get one request to decrypt the share and will then give up this role.

8.3 Commentary on the Protocol

In the last section we presented the protocol. Here we will try to explain some of our design decisions and show how they relate to the properties we want our system to satisfy.

There are several novel aspects in the design of our censorship resistant system as compared to existing architectures:

- Replication. The use of a Peer-to-Peer layer such as PAST to replicate state in forwarders and storers.
- Forwarders. The use of forwarders to provide an extra layer of indirection and prevent the storers from being visible by clients.
- Encryption of shares. Storing the shares in an encrypted form and keeping the keys at the forwarders.
- Decrypters. The use of separate nodes (as opposed to, for instance, forwarders) to decrypt shares.

8.3.1 Replication

Replication of state in the system provides fault tolerance, efficiency and prevents several kinds of attacks.

Firstly, it makes denial of service attacks and simple efforts to take down individual forwarders ineffective because there are always a number of other hosts ready to forward requests. Furthermore, even if the attacker succeeds in taking down a particular forwarder, state will be replicated onto a new node which will also start forwarding requests.

More subtly, it reduces the link between any particular forwarder and the share which has been retrieved. This is because the address for a particular forwarder (for instance, a_0) which is published in the anonymous newsgroup denotes a dynamic set of physical hosts rather than a single machine. This is due to the behaviour of the underlying layer (PAST): if asked to route a request to a node (a_0), it does not necessarily forward it to that specific node, but instead to any node which shares state with a_0. Therefore, it is not easy to establish precisely which physical machines the address a_0 represents; indeed, this set changes as machines go down and come back up.

However, this introduces a slight complication. Although the forwarders share state, they cannot share private keys (it would be impossible to keep these keys secret because the set of forwarders constantly changes as nodes go up and down). Therefore requests addressed to them must be delivered in plaintext[4]. This turns out not to be problem here as the attacker who watches traffic arriving at the forwarders sees v_0 and r_c. The former is public anyway, and the latter gives away no information about c itself (but enables him to send a fake share to c[5]).

8.3.2 Forwarders

The use of forwarders serves several purposes. First of all, they help protect the storers against "Rubber Hose Cryptanalysis" by hiding them from the clients. Secondly, they can help provide active-server document anonymity by randomly introducing new dummy requests into the system and dropping some of the valid ones. Thus, it will make it hard for the storer to find out (part of) which document it is storing, even by acting as a client in the system. Finally, we use the forwarders to store keys which decrypt the shares and forward them to the decrypters.

[4]This means that the last layer of the onion is not encrypted. Therefore, the message is still anonymous but not secret.

[5]This does not constitute an attack as the adversary would have to perform this active and therefore expensive operation for every share every client requests.

8.3.3 Encryption of Shares

We have argued that the storers should not be able to see the content they are storing, to prevent the possibility of them being pressured into censorship. Therefore, we must make them store the shares in an encrypted form and stop them from getting hold of the keys which would decrypt the shares. Thus these keys cannot be published as this would enable the storers to retrieve all of them and see which one decrypts each of the shares. Hence they are stored by the forwarders.

8.3.4 Decrypters

Some would argue that the use of decrypters is superfluous. The storer could just send the shares back to the forwarder who would send them back to the client. However, this would expose the forwarder to the risk of being caught red-handed with the share. Furthermore, they might be pressured into installing a filter to censor shares which the attacker does not like. As the forwarder is the publicly visible part of the system (and therefore the most vulnerable), we decided to delegate the task of decrypting the share to a completely different entity who does not have any information about what it is decrypting.

Having looked at the various features of the system, let us see how they combine to protect against the attack we described earlier. First of all, the use of forwarders means that the storers do not have to be included in the address of the document. Furthermore, the use of a strong anonymity system does not allow the storers to be identified even by a global passive adversary who tries to analyse where the messages from the forwarders travel to. Finally, even if the attacker picks a particular storer for reasons other than analysing the communication patterns, he can deny knowledge of the content on his machine – the shares are encrypted and he does not possess the keys. Having established the fact that we protect the storers against this attack, let us consider the forwarders. Certainly, it is possible to determine some of the forwarders for a particular share of a certain document are. However, it is possible for a forwarder to deny that a request for a particular share has been forwarded. Firstly, due to replication of the forwarder state, over time different machines act as forwarders for a singe share, so simply sending a request for a share does not identify the machine which forwarded it. Secondly, even if a record of a forwarder is identified, his link to the document is extremely weak – he forwards what essentially amounts to random bits which may (or may not) cause a part of a document to arrive from a completely different part of the network (the decrypter) later. Certainly, in our view this provides much better protection than, say, Publius.

8.4 Discussion

In this section we discuss the limitations of our system.

First, one should question the validity of assuming a public key infrastructure on a Peer-to-Peer network. We need each peer to be able to retrieve the public key of any other peer and verify that the key belongs to that particular peer. The simple solution is to use a global repository. However, such a scheme has well-known problems: it would limit the scalability and the resilience of the whole system. This is a much more general problem which is likely to receive a fair amount of attention in the coming years. A related issue is the fact that working on top of a Peer-to-Peer system may result in attacks. For instance, if the attacker is able to arrange requests not to reach a particular set of nodes corresponding to a forwarder at this level (by, for instance, modifying routing tables in some of the nodes of the Peer-to-Peer system), he will effectively censor the corresponding share. We do not address these problems here, but merely point out that this is an active research area and one which we need to pay close attention to. A nice summary of the problems in security of Peer-to-Peer systems and some solutions to them can be found in [SM02, Dou02].

Secondly, we should consider how likely the forwarders are to suffer from "Rubber Hose Cryptanalysis" – they are certainly visible to the attacker and contain information which is necessary for share retrieval. However, we argue that they are much less likely to be subjected to such pressure than, for example, Publius servers for the following reasons:

- They are not storing the offending document, not even in an encrypted form, so their connection with it is somewhat indirect.
- They do not store the identities of the s_i entities, so an attack to try to get it out of them will not succeed.
- The share does not actually go through the forwarders a_i after publication is completed.
- The requests addressed to the forwarder a_i are likely to end up being handled by a number of different physical hosts.

A slight modification to the protocol (which we do not describe here) can be introduced to further reduce the role the a_i play in the protocol, and therefore reduce the potential for them to be attacked. It is essentially based around the idea of passing the received message to another forwarder as well as possibly handling it yourself. This should provide better receiver untraceability.

We must also consider the number of compromised peers it takes to remove a document from the system. A possible attack is as follows: each forwarder a_i remembers (rather than forgetting) the corresponding storer s_i at the time of publication in the hope of exposing them later, if the content turns out to be offending. Once the document has been successfully published, a_i notes the correspondence between the random integer (v_i) published with the document and one in its lookup table, and works out the fact that s_i is storing a share of a particular document. It can now pressure s_i into removing the share. However, the chances of the $n - k + 1$ peers picked as a_i being compromised are small and the peers have to be compromised at the time of publication, otherwise the attack fails.

Having stated that we provide active server anonymity, we must pay attention to the amount of information the storer can gain by repeatedly requesting the document and noticing the requests coming in for the share it is storing. This is essentially an intersection attack (which, as various papers [WALS02, Dan03c] point out, will ultimately be successful), though here it is easier to protect against as it is possible to introduce end-to-end dummy traffic *as well as* tolerate some percentage of packet loss. The precise details of this, along with a quantitative analysis (in the spirit of the previous chapters) of the anonymity of a storer is left for future work.

We also note that we are presenting just one part of a design of a system. Many questions are left unanswered and a few attacks are not addressed. For example, the attacker can simply flood the censorship resistant system with random data so that there is no room for "real" documents to be inserted. This design does not include protection against such an attack.

Neither do we deal with accountability in any systematic way. Consider a scenario where the attacker is powerful enough to insert many nodes into the system. Each of these, when asked to act as a forwarder, replies "ok", but drops the share and fails to answer subsequent requests. In this design, we are relying on the inability of the attacker to insert enough malicious nodes to censor documents in this way. We felt that although standard methods which are summarised in [DFM00b] can be used in this system, they are inappropriate in this context or do not provide adequate protection. Therefore, we leave this for future work.

8.5 Related and Future Work

A variety of censorship resistant systems have been designed, some of which have also been implemented. We have already discussed Publius, Dagster and Tangler but perhaps the system closest to ours, in terms of the aims it tries to achieve, is Free Haven [DFM00a].

Free Haven is built on top of an underlying network of anonymous remailers and deals with reader anonymity, server anonymity and censorship resistance. However, it uses a Gnutella-like search for retrieval of shares of the document. More specifically, a user request to retrieve a particular document gets broadcast from the user node to all the neighbouring nodes, and so on. When a request arrives at a peer which has a matching share, it gets sent off to the requester via a chain of remailers. This scheme for locating files is rather inefficient, and, as the Gnutella experience has shown, does not scale for large numbers of peers. Furthermore, we note that peers frequently exchange shares with each other. This is costly in terms of network bandwidth and makes it hard to provide guarantees that a document will be located. Our system aims to be more efficient both in terms of bandwidth and share retrieval. We note, however, Free Haven authors suggest an interesting technique for increasing censorship resistance – moving the individual shares around in the system – which may be incorporated into our system. For example, the shares may be moved periodically, and the state in the forwarders updated.

We have two main future objectives. Firstly, having described an anonymity system and claimed that it satisfies some properties, we consider it worthwhile to formalise those and prove them rigorously. Finally, we would like to build a prototype implementation on top of a Peer-to-Peer storage layer like PAST and a suitable anonymity system such as Mixminion.

8.6 Summary

In this chapter we have presented a design of a system which deals with censorship resistance and satisfies strong anonymity requirements. We have also made a point of reusing existing designs which provide efficient routing and replication rather than reinvent this ourselves. This may, in principle, also make implementation easier as a free implementation of Pastry (though not PAST) is available[6].

We have argued for building an anonymous censorship resistant system on top of a peer to peer architecture and demonstrated the feasibility of doing so to provide strong anonymity and robustness guarantees.

This chapter concludes the main part of the thesis. We now go on to summarise our findings and ponder the things left to be discovered.

[6] `http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry/`

Chapter 9

Conclusion and Future Work

"*You have zero privacy anyway, get over it.*"
— Scott McNealy, CEO Sun Microsystems, 1999

"*Some problems are better evaded than solved.*"
— C.A.R. Hoare

A variety of anonymity systems have been developed, but relatively little attention has been paid to the analysis of their anonymity properties. In this work we have shown that *analysis of anonymity systems is feasible, necessary and somewhat practical.* A secondary thesis that this work does a great deal to support is that *systems providing different desirable degrees of anonymity on the Internet, each with an associated cost, are a practical option rather than a theoretical possibility.*

All the analyses presented here use relatively simple mathematical techniques and most of our simulations are not overly costly. Yet the results we have obtained are relevant to the anonymity systems which are in the process of being designed, implemented and deployed today and in the future. We have also seen clear benefits naturally arising out of our analyses. First of all, where the analysis has identified a new way of looking at the problem, an improvement to the existing system was quickly found (e.g. Chapters 4 and 5). Secondly, we were able to compare different systems and identify their properties more clearly, which lead us to their better understanding. Finally, in the case of Chapter 6, we laid the foundations for future analysis of the anonymity of mix networks by providing the definition of anonymity of a run of a mix network given an observation of the attacker. Although the work of this chapter cannot provide results which are applicable to deployed mix networks, we feel it is a significant step towards this goal. Besides, our program for calculating anonymity can be used by researchers to gain an intuition as to which configurations (e.g. uniform route length) provide more anonymity on small mix networks.

In this dissertation we have explored a broad range of topics related to mix systems. Some were explored in more depth than others: for instance, our investigation of the properties of various single mixes has left relatively few important questions unanswered. Indeed, here our analysis has driven the design of a new mix (the binomial mix) which has lead to its implementation and deployment within the Mixmaster system. More specifically, our analysis of Chapter 4 has confirmed the intuition of Cottrell which he expressed in [Cot94] and implemented in Mixmaster. The work which followed (presented in Chapter 5) proposed a generalisation and two alterations to the mix design which resulted in the binomial mix.

Our analysis of mix networks has given us some understanding of the anonymity provided by mix networks with simple mixes. It is clear that more complex models are needed before the results from such analyses can be used to drive design decisions in real systems. However, this is an important first step towards an analysis of mix networks. It also proposes an important level of abstraction at which the anonymity properties of mix networks can be analysed. We note that although in the course of this analysis we have made some interesting and important observations, there were no "nasty surprises". In other words, the level of anonymity we found matched our expectations. This suggests that the anonymity of the Mixmaster system (which uses more sophisticated mixes) is no worse than that of the system we modelled[1].

Our analysis of packet-switched connection-based systems told a different story. Having taken the global passive adversary as a threat model, we found several powerful attacks on many of the systems. Although we described such attacks more precisely than before, quantified them and suggested some solutions, much work in this area remains. Indeed, while some have been content with analysing such systems against a weaker threat model, we hope that work on securing them against the global passive adversary continues.

In Chapter 8, we suggested a novel use of anonymity systems to improve properties of censorship resistant systems. Although it requires rather strong properties from the underlying anonymity system, it shows the need for deriving such properties from anonymity system. It also serves to illustrate a different type of application of such systems (other than just email or web browsing).

Before performing any of the analyses presented in this dissertation, we had to devise a suitable metric for anonymity. Furthermore, the difference between the anonymity of a message and the anonymity of a system had to be clarified. Thus, we claim to have provided some insight into how anonymity properties can and should be expressed. Perhaps this is the most valuable part of the thesis for those who take up this field in the future.

[1] We stress that this is merely informed intuition rather than scientific evidence.

9.1 Future Directions

The more you know, the more you don't know. Analysis of anonymity systems is a big field and there is only so much one can do in a dissertation. Here we look at the areas for future research we have identified in the course of our work.

First of all, let us consider the problem of defining and measuring anonymity. There are many interesting properties left to be defined formally. Recall, for instance, our informal notions of sender and receiver untraceability presented back in Chapter 2. A promising direction for future work is to develop a formal framework in which one can express a protocol which satisfies one or more of these properties and prove them. This could lead to the development of new protocols or attacks on existing ones. We have already started exploring this direction.

In the area of analysis of the anonymity of single mixes, one might like to perform a rigorous comparison between timed and threshold pool mixes along the lines we have described and investigate the effect of dummy traffic on mixes. Overall, we feel that further work in this area will not bring much more insight into the remailer community and one should concentrate on analysis of whole systems rather than single mixes.

Perhaps the most interesting technical problem we have looked at is the anonymity provided by a mix network. In Chapter 6 we considered the anonymity of a message passing through a network made up of simple mixes. Several extensions of this work are possible; they are outlined in Section 6.13. The most promising of these include extending the model to account for more sophisticated mixes such as the binomial mix and developing more efficient algorithms to compute the anonymity of a message passing through a mix network. Such research would also have direct impact on the community running systems such as Mixmaster and Mixminion.

Yet another high-impact area of research is that of connection-based anonymity systems. As we have seen in this thesis, current systems do not provide much anonymity and significant modifications are required to ensure that even the minimum levels of anonymity are attained. The two most promising areas for investigation in this field are to work out a scheme for maintaining some level of anonymity against the global passive adversary with a few compromised mixes and to come up with a threat model which enables a useful comparison between the various existing anonymity systems: Onion Routing, Tarzan, MorphMix and Crowds.

9.2 Concluding Remarks

Despite the future work remaining, we feel that our two theses are broadly justified. We presented numerous analyses of the anonymity of anonymity systems and derived benefits from each one, whether in the ability to compare systems or in gaining insight into the design of better ones. We also showed that there are systems (or at least designs of systems) which can provide significant amounts of anonymity to Internet users. Clearly, there is more anonymity to be gained in the case of message-based systems than connection-based ones. We feel that further work on the analysis of anonymity systems would be beneficial to the anonymity community, as would contributions towards the debate on the pros and cons of the availability of anonymity to a wide variety of users.

Appendix A

Variable Conventions

In the thesis we adopt the following variable conventions:

Chapter 3

$\mathcal{R}$	the set of roles – {sender, recipient}
n	threshold
p	pool
$\mathcal{U}_{(M,r)}$	r anonymity probability distribution of a message M
A_k	sender anonymity set of a message coming out of the pool mix at round k
$E_{(M,r)}$	Anonymity (entropy of the anonymity probability distribution)
E_k	entropy of the anonymity probability distribution of a message coming out of the pool mix at round k

Chapter 4

n	threshold
t	period
c	maximum number of messages a mix can contain
p	pool
r	rate of arrival of messages to a mix
N	number of attacker messages
G	number of good messages in the mix
m	total number of messages in the mix
p_{min}	the minimum number of messages in the pool
$frac$	fraction of messages above the minimum pool which are to be flushed
k	number of rounds of a particular blending attack

Chapter 5

P	function expressing the batching strategy of a mix
m	number of messages in a mix at the time of flushing
t	period
n	threshold
p	pool
G	number of good messages in the mix
N_T	number of messages the attacker sends in to achieve a certain probability of each message being flushed
k, k'	numbers of rounds

Chapter 6

s, s'	senders
r, r'	receivers
m, n	mixes
ε	empty sequence
a	onion encrypted message (comprising of a sequence of mixes, a receiver and message content)
o	a sender or a mix
$\overrightarrow{l}$	a trace
$\mathcal{P}(X)$	powerset of X

Chapter 7

n	number of nodes
d	number of links of a node
c	number of connections
V	distribution of the number of packets on a connection
l	route length, number of links
ℓ	route length, number of nodes
G, E	graph represengting the topology of the network
e	edge
λ	parameter of the distribution of message arrivals
μ	delay parameter of the mix

Bibliography

[Abe98] M. Abe. Universally verifiable MIX with verification work independent of the number of MIX servers. In K. Nyberg, editor, *Proceedings of EUROCRYPT 1998*, pages 437–447. Springer-Verlag, LNCS 1403, 1998.

[ADS03] A. Acquisti, R. Dingledine, and P. Syverson. On the Economics of Anonymity. In J. Camp and R. Wright, editors, *Proceedings of Financial Cryptography (FC '03)*. Springer-Verlag, LNCS (forthcoming), 2003.

[AES01] Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001.

[AKP03] D. Agrawal, D. Kesdogan, and S. Penz. Probabilistic Treatment of MIXes to Hamper Traffic Analysis. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 16–27. 2003.

[AMCK+02] J. Al-Muhtadi, R. Campbell, A. Kapadia, M. D. Mickunas, and S. Yi. Routing through the mist: Privacy preserving communications in ubiquitous computing environments. In *Proceedings of International Conference of Distributed Computing Systems (ICDCS 2002)*, pages 74–83. 2002.

[And] R. Anderson. Personal Communication.

[And96] R. Anderson. The eternity service. In *1st International Conference on the Theory and Applications of Cryptology (Pragocrypt '96)*, pages 242–252. Czech Technical University Publishing House, 1996.

[And01] R. Anderson. *Security Engineering: A guide to building dependable distributed systems*. Wiley, 2001. ISBN 0-471-38922-6.

[BD03] A. Beimel and S. Dolev. Buses for anonymous message delivery. *Journal of Cryptology*, 16(1):25–39, 2003.

[BG03] K. Bennett and C. Grothoff. GAP – practical anonymous networking. In R. Dingledine, editor, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2003)*, pages 141–160. Springer-Verlag, LNCS 2760, 2003.

[BGS00] P. Boucher, I. Goldberg, and A. Shostack. Freedom system 2.0 architecture. `http://www.freedom.net/info/whitepapers/`, 2000. Zero-Knowledge Sytems, Inc.

[BMS01] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, 2001.

[BPS00] O. Berthold, A. Pfitzmann, and R. Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, 2000.

[BS03] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 3(1):46–55, 2003.

[CDG$^+$] J. Claessens, C. Diaz, C. Goemans, B. Preneel, and J. Vandewalle. Revocable anonymous access to the Internet. To appear in the *Journal of Internet Research*.

[CDK01] R. Clayton, G. Danezis, and M. G. Kuhn. Real world patterns of failure in anonymity systems. In I. S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 230–244. Springer-Verlag, LNCS 2137, 2001.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2):84–90, 1981.

[Cha88] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[Cla99] R. Clarke. Introduction to dataveillance and information privacy, and definitions of terms, 1997-1999. `http://www.anu.edu.au/people/Roger.Clarke/DV/Intro.html`.

[Cla03] R. Clayton. Improving onion notation. In R. Dingledine, editor, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2003)*, pages 81–87. Springer-Verlag, LNCS 2760, 2003.

[Cla04] R. Clayton. *Anonymity and Traceability in Cyberspace*. Ph.D. thesis, University of Cambridge, 2004. Forthcoming.

[Cot94] L. Cottrell. Mixmaster & remailer attacks, 1994. `http://riot.eu.org/anon/doc/remailer-essay.html`.

[Cot02] L. Cottrell, 2002. Personal communication.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66. Springer-Verlag, LNCS 2009, 2000.

[CWL+01] S. A. Craver, M. Wu, B. Liu, A. Stubblefield, B. Swartzlander, D. S. Wallach, D. Dean, and E. W. Felten. Reading between the lines: Lessons from the SDMI challenge. In *Proceedings of the 10th USENIX Security Symposium*, pages 13–17. 2001. Did **not** appear in the proceedings of the Information Hiding Workshop 2001 where it was submitted and accepted.

[Dan03a] G. Danezis. Meteor mixing, 2003. Unpublished.

[Dan03b] G. Danezis. Mix-networks with restricted routes. In R. Dingledine, editor, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2003)*, pages 1–17. Springer-Verlag, LNCS 2760, 2003.

[Dan03c] G. Danezis. Statistical disclosure attacks. In Gritzalis, Vimercati, Samarati, and Katsikas, editors, *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*, pages 421–426. IFIP TC11, Kluwer, 2003.

[Dan04] G. Danezis. *Better Anonymous Communications*. Ph.D. thesis, University of Cambridge, 2004. Forthcoming.

[DDM03] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15. 2003.

[DFM00a] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95. Springer-Verlag, LNCS 2009, 2000.

[DFM00b] R. Dingledine, M. J. Freedman, and D. Molnar. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter 16, pages 271–340. O'Reilly, 2000. ISBN 0-596-00110-X.

[DK00] Y. Desmedt and K. Kurosawa. How to break a practical MIX and design a new one. In *Proceedings of EUROCRYPT 2000*, pages 557–572. Springer-Verlag, LNCS 1807, 2000.

[DMS04] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*. 2004.

[DO00] S. Dolev and R. Ostrobsky. Xor-trees for efficient anonymous multicast and reception. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):63–84, 2000. ISSN 1094-9224.

[Dou02] J. Douceur. The Sybil Attack. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, pages 251–260. Springer-Verlag, LNCS 2429, 2002.

[DP04a] C. Díaz and B. Preneel. Reasoning about the anonymity provided by pool mixes that generate dummy traffic. In *Proceedings of 6th Information Hiding Workshop (IH 2004)*, LNCS. Toronto, 2004.

[DP04b] C. Díaz and B. Preneel. Taxonomy of mixes and dummy traffic. In *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*. Toulouse, France, 2004.

[DR01] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *The 8th Workshop on Hot Topics in Operating Systems*. 2001.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002.

[DS03a] G. Danezis and L. Sassaman. Heartbeat traffic to counter $(n-1)$ attacks. In P. Samarati and P. Syverson, editors, *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*. ACM, 2003.

[DS03b] C. Diaz and A. Serjantov. Generalising mixes. In R. Dingledine, editor, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2003)*, pages 18–31. Springer-Verlag, LNCS 2760, 2003.

[DSCP02] C. Diaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*, pages 54–68. Springer-Verlag, LNCS 2482, 2002.

[ElG85] T. ElGamal. A public key cryptosystems and a signature schems based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[FM02] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pages 193–206. 2002.

[FS01] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In J. Kilian, editor, *Proceedings of CRYPTO 2001*, pages 368–387. Springer-Verlag, LNCS 2139, 2001.

[GJ03] P. Golle and M. Jakobsson. Reusable anonymous return channels. In P. Samarati and P. Syverson, editors, *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*. ACM, 2003.

[GJJS] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. Universal re-encryption for mixnets. Preprint. Available from: `http://crypto.stanford.edu/~pgolle/papers/univrenc.ps`.

[GRPS03] S. Goel, M. Robson, M. Polte, and E. G. Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Technical Report 2003-1890, Cornell University, 2003.

[GRS96] D. Goldschlag, M. Reed, and P. Syverson. Hiding routing information. In R. J. Anderson, editor, *Information Hiding*, pages 137–150. Springer LNCS 1174, 1996.

[GT96] C. Gülcü and G. Tsudik. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium (NDSS '96)*, pages 2–16. IEEE, 1996.

[HO03] J. Halpern and K. O'Neill. Anonymity and information hiding in multiagent systems. In *Computer Security Foundations Workshop, CSFW 16*. 2003.

[Hod91] H. Hodara. Secure fiberoptic communications. In *Symposium on Electromagnetic Security for Information Protection*, pages 21–22. Rome, Italy, 1991.

[HS04] D. Hughes and V. Shmatikov. Information hiding, anonymity and privacy: A modular approach. *Journal of Computer Security, special issue on WITS '02 (ed. Joshua Guttman)*, 12(1):3–36, 2004.

[Jak99] M. Jakobsson. Flash Mixing. In *Proceedings of Principles of Distributed Computing – PODC '99*, pages 83–89. ACM Press, 1999.

[JAP] The JAP project. `http://anon.inf.tu-dresden.de/index_en.html`.

[Jer00] A. Jerichow. *Generalisation and Security Improvement of Mix-mediated Anonymous Communication.* Ph.D. thesis, Technischen Universitat Dresden, 2000.

[KAP02] D. Kesdogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In F. Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*, pages 53–69. Springer-Verlag, LNCS 2578, 2002.

[KBS02] D. Kesdogan, M. Borning, and M. Schmeink. Unobservable surfing on the world wide web: Is private information retrieval an alternative to the mix based approach? In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*, pages 224–238. Springer-Verlag, LNCS 2482, 2002.

[KEB98] D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of Information Hiding Workshop (IH 1998)*, pages 83–98. Springer-Verlag, LNCS 1525, 1998.

[KSRW03] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting system. Technical Report TR-2003-19, Johns Hopkins Information Security Institute, 2003.

[LS02] B. Levine and C. Shields. Hordes – a multicast based protocol for anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.

[MCPS03] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol – Version 2. Draft, 2003.

[Mer03] R. Mercuri. Electronic voting, 2003. `http://www.notablesoftware.com/evote.html`.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.

[Mix] Mixmaster. `http://mixmaster.sourceforge.net/`.

[Möl03] B. Möller. Provably secure public-key encryption for length-preserving chaumian mixes. In *Proceedings of CT-RSA 2003*, pages 244–262. Springer-Verlag, LNCS 2612, 2003.

[Nef01] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In P. Samarati, editor, *Proceedings of 8th ACM Conference on Computer and Communications Security (CCS-8)*, pages 116–125. ACM Press, 2001.

[NMSS03] R. E. Newman, I. S. Moskowitz, P. Syverson, and A. Serjantov. Metrics for traffic analysis prevention. In R. Dingledine, editor, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2003)*, pages 48–65. Springer-Verlag, LNCS 2760, 2003.

[NSN03] L. Nguyen and R. Safavi-Naini. Breaking and mending resilient mix-nets. In R. Dingledine, editor, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2003)*, pages 66–80. Springer-Verlag, LNCS 2760, 2003.

[OA00] M. Ohkubo and M. Abe. A Length-Invariant Hybrid MIX. In *Proceedings of ASIACRYPT 2000*, pages 178–191. Springer-Verlag, LNCS 1976, 2000.

[OR] Onion routing. `http://www.onion-router.net/`.

[PGP] The international PGP home page. `http://www.pgpi.org/`.

[PJH+99] S. Peyton Jones, J. Hughes, et al. Report on the programming language Haskell 98, a non-strict, purely functional language, 1999. `http://www.haskell.org/`.

[PK00] A. Pfitzmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity: A proposal for terminology. Draft, version 0.14, 2000.

[PP90] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of MIXes. In J.-J. Quisquater and J. Vandewalle, editors, *Proceedings of EUROCRYPT 1989*, pages 373–381. Springer-Verlag, LNCS 434, 1990.

[PPW91] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems*, pages 451–463. 1991.

[Qui] Quicksilver home page. `http://www.quicksilvermail.net/`.

[Ray00] J.-F. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In H. Federrath, editor, *Proceedings of Designing*

Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability, pages 10–29. Springer-Verlag, LNCS 2009, 2000.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*. 2001.

[Ren03] M. Rennhard. Practical anonymity for the masses with mix-networks. Technical Report 157, ETH Zurich, Switzerland, 2003.

[RP02] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In P. Samarati, editor, *Proceedings of the Workshop on Privacy in the Electronic Society(WPES 2002)*. 2002.

[RR97] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *DIMACS Technical Report*, 97(15), 1997.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RSA02] PKCS #1 v2.1: RSA Cryptography Standard, 2002. V 2.1, available from `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html`.

[SBS02] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. $\mathcal{P}^5$: A protocol for scalable anonymous communication. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 58–72. 2002.

[SD02] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*, pages 41–53. Springer-Verlag, LNCS 2482, 2002.

[SDS02] A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. In F. Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*, pages 36–52. Springer-Verlag, LNCS 2578, 2002.

[Ser02] A. Serjantov. Anonymizing censorship resistant systems. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, pages 111–120. Springer-Verlag, LNCS 2429, 2002.

[Sha48] C. Shannon. The mathematical theory of communication. *Bell Systems Technical Journal*, 30:50–64, 1948.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[Shm03] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security, special issue on 15th IEEE Computer Security Foundations Workshop (ed. Steve Schneider)*, 2003. To appear.

[Sho98] V. Shoup. Why chosen ciphertext security matters. Technical report, IBM Research, 1998.

[SK03] S. Steinbrecher and S. Köpsell. Modelling unlinkability. In R. Dingledine, editor, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2003)*. Springer-Verlag, LNCS 2760, 2003.

[SL03] A. Serjantov and S. Lewis. Puzzles in P2P systems. CaberNet Radicals Workshop, 2003. Corsica.

[SM02] E. Sit and R. T. Morris. Security considerations for peer-to-peer distributed hash tables. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 261–269. Springer-Verlag, LNCS 2429, Cambridge, MA, 2002.

[SN03] A. Serjantov and R. E. Newman. On the anonymity of timed pool mixes. In *Proceedings of the Workshop on Privacy and Anonymity Issues in Networked and Distributed Systems*, pages 427–434. Kluwer, 2003.

[SS03] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In E. Snekkenes and D. Gollman, editors, *Proceedings of ESORICS 2003*, pages 116–131. Springer-Verlag, LNCS 2808, 2003.

[SSW01a] A. Serjantov, P. Sewell, and K. Wansbrough. The UDP calculus: Rigorous semantics for real networking. In N. Kobayashi and B. C. Pierce, editors, *Theoretical Aspects of Computer Software, 4th International Symposium (TACS 2001)*, pages 535–559. Springer-Verlag, LNCS 2215, 2001. Full version [SSW01b].

[SSW01b] A. Serjantov, P. Sewell, and K. Wansbrough. The UDP calculus: Rigorous semantics for real networking. Technical report, University of Cambridge, England, 2001. Conference version [SSW01a].

[STRL00] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, 2000.

[SW01] A. Stubblefield and D. Wallach. Dagster: Censorship-resistant publishing without replication. Technical Report TR01-380, Rice University, 2001.

[Tou] D. S. Touretzky. Gallery of CSS descramblers. `http://www-2.cs.cmu.edu/~dst/DeCSS/Gallery/`.

[WALS02] M. Wright, M. Adler, B. N. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. In *Proceedings of the Network and Distributed Security Symposium (NDSS '02)*, pages 28–43. IEEE, 2002.

[WALS03] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending anonymous communication against passive logging attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. 2003.

[WM01] M. Waldman and D. Mazières. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 126–135. 2001.

[WNSS02] K. Wansbrough, M. Norrish, P. Sewell, and A. Serjantov. Timing UDP: Mechanized semantics for sockets, threads and failures. In D. L. Mtayer, editor, *European Symposium on Programming (ESOP 2002)*, pages 278–294. Springer-Verlag, LNCS 2305, 2002.

[WP90] M. Waidner and B. Pfitzmann. The dining cryptographers in the disco: unconditional sender and recipient untraceability with computationally secure servicability. In J.-J. Quisquater and J. Vandewalle, editors, *Proceedings of EUROCRYPT 1989*, pages 302–319. Springer-Verlag, LNCS 434, 1990. Abstract. Full version available from: `http://www.semper.org/sirene/publ/WaPf1_89DiscoEngl.ps.gz`.

[WRC00] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72. 2000.