THE DESIGN AND IMPLEMENTATION OF A PRIVATE MESSAGE SERVICE FOR MOBILE COMPUTERS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

David Anthony Cooper

August 1995

THE DESIGN AND IMPLEMENTATION OF A PRIVATE MESSAGE SERVICE FOR MOBILE COMPUTERS

David Anthony Cooper, Ph.D.

Cornell University 1995

Even as wireless networks create the potential for access to information from mobile platforms, they pose a problem for privacy. In order to retrieve messages, users must periodically poll the network. The information that the user must give to the network could potentially be used to track that user. However, the movements of the user can also be used to hide the user's location if the protocols for sending and retrieving messages are carefully designed.

In this thesis, we will present a protocol for a replicated memory service which allows users to read from memory without revealing which memory locations they are reading. Unlike previous protocols, this protocol is efficient in its use of computation and bandwidth. We will then show how this protocol can be used in conjunction with existing privacy preserving protocols to allow a user of a mobile computer to maintain privacy despite active attacks.

Allowing users to retrieve messages anonymously introduces a new problem. In order to limit memory usage, it is necessary to remove old messages from the system. However, since users may become disconnected from the network for

periods of time, it is important that the system hold onto messages until they have been retrieved by their intended recipients. The result is a conflict between the system's need for information and users' desire for privacy. We will present the design of a vacation service which we have developed which stores messages for users which are disconnected which does not require users to reveal any private information.

Finally, we will describe the implementation of the private message service and discuss the performance estimates that we derived for the system based on experimental results. As we will show, the potential throughput of the private message service is reasonable.

Biographical Sketch

David Cooper was born on March 8, 1968 in Boston, Massachusetts where he lived until he was 5. From there he moved to Greenville, South Carolina where he lived for 3 years. Next, he moved to Carmel, Indiana where he lived for 4 years. At age 12, he moved to Pittsburgh, Pennsylvania where he lived until coming to Cornell in 1986. He received his B.S. degree in Computer Science in May 1990, his M.Eng. degree in May 1991, and his M.S. degree in January 1994.

Acknowledgements

More than anyone else, Ken Birman was responsible for guiding me through my final years at Cornell. His advice and encouragement as I struggled through my research were invaluable. I would also like to thank Dexter Kozen, Miriam Leeser, and Sam Toueg who served on my committee and provided much support.

I was also fortunate to have the support of many friends and colleagues. Mike Reiter, through his own research, developed many concepts which proved useful to my own research. Guerney Hunt gave me a lot of advice which helped me to make my way through the Ph.D. program. I also received a lot of help from many other members of the Horus group.

Anindya Basu, Mike Cox, and Ashvin Dsouza were always around when I needed to take a break from work and were always willing eat my food whenever I had the urge to cook. Tom Olsen provided me with the original "Carrot Cake Fan Club" and Michellé Mock provided me with the opportunity to prepare "Death by Chocolate".

I would also like to thank the many people who had to put up with me either as an officemate or a housemate over the years and who had to deal with my periodic bouts of frustration: Vipin Bansal, Anindya Basu, Mike Cox, Ashvin Dsouza, Xiaolin Ge, Paula Gorden, Pei-Hsin Ho, Joon Ho Lee, Jerry Liu, Karen

Middaugh, Ranjana Murthy, Mark Ollis, Tom Olsen, Glenda Van Oort, Sam Paik, Chris Staffa, Kristen Summers, King Tan, Bruce Watler, and Bing Zhang.

Finally, I would like to thank my family who patiently supported me through my many years in school.

This work was supported by ARPA/ONR grant N00014-92-J-1866 and a grant by Siemens Corp. The views expressed herein are those of the authors and do not represent the opinions of ARPA/ONR or Siemens Corp.

Table of Contents

List of Figures

Chapter 1

Introduction

> Privacy in one's associations... may in many circumstances be indispensable to freedom of association, particularly where a group espouses dissident beliefs. – JUSTICE JOHN HARLAN, DELIVERING THE DECISION OF THE SUPREME COURT, *NAACP* V. *ALABAMA* (1958) [Hen92]

Most people, as a general rule, have both a desire and a need for privacy. While there are many laws in place to guarantee people's right to privacy, the existence of these laws is not always sufficient to prevent the unauthorized collection, by both legal and illegal means, of large amounts of private information about individuals [Bur83]. In some cases, branches of the federal government itself have been involved in widespread interception of telephone conversations [Bur83] and telegrams [GII76].

While the protection of private information has always been a concern, the increased use of computers and electronic communications to transmit, collect, and collate private information has significantly increased the amount of private information that may be collected. An "eavesdropper" has the potential to acquire private information about someone whenever that person uses the telephone, pur-

chases something with a credit card, buys an airline ticket, checks into a hotel, connects to an on-line service, etc. With the introduction of mobile computing devices, the threat will only increase.

Since legal protections have not sufficiently protected, and will not sufficiently protect, the privacy needs of individuals, technical solutions to the problem must be made available. While a strictly technical solution to the problem of privacy is not possible, it may, when coupled with improved legal protections, greatly reduce the availability of private information about individuals.

1.1 What is Mobile Computing?

Mobile computers are portable computers that have wireless communications abilities. When used in combination with base stations connected to a static network, mobile computers provide a powerful means of communication for people who are on the move. There are currently several types of mobile computers ranging from general purpose devices such as laptop and palmtop computers to special purpose devices such as cellular phones, pocket pagers, and active badges (see figure 1.1). While mobile computing has much in common with static (or non-mobile) computing, there are several essential differences which lead to novel problems [BAI93, BIV92,IB93a,IB93b,IB93c,DFM91,SD91,Kat94,FZ94].

Since size and weight are of major concern in mobile devices and the devices must run off of batteries, power consumption is of major concern. This affects all aspects of the device: the CPU, the display, the disk drive, the memory, etc. One result of the power consumption problem is that CPUs on mobile computers tend to be slower than on static computers since power consumption tends to increase with

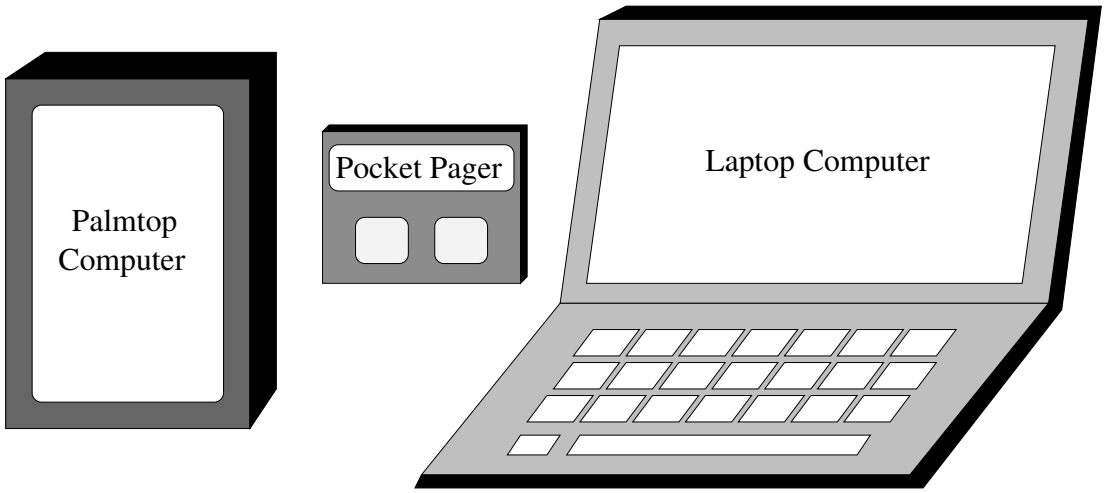

Figure 1.1: Examples of Mobile Computing Devices

clock rate. Also, in order to save both space and power consumption, memories on mobile computers tend to be relatively small. As a result of these concerns, protocols involving mobile computers try to push as much of the computation and storage onto the static computers as possible.

The use of wireless communications between mobile computers and base stations on the static network also poses many problems. First, both sending and receiving messages consume battery power. For this reason, the communication overhead of protocols must be carefully considered. This problem is further complicated by the fact that sending messages consumes more power than receiving messages. With wireless communications, there is also the need to allocate bandwidth usage [ZY89,ZY91,YW93,PSS95].

Mobility imposes problems since wireless communications have limited range. As users move, their computers become disconnected from the base stations with which they are communicating and eventually reconnect to other base stations once they move within range of these stations. This leads to several problems.

First, mobile computers may become completely disconnected from the network for periods of time. Whenever possible, information that users may want should be cached on their mobile computers to minimize the visible effects of disconnection. Second, as mobile computers switch from base station to base station, there will be a need to reallocate communications channels. Finally, any messages being sent to a mobile computer will need some way to find the computer despite its unpredictable movements through the network [BP,Joh93,Per,Rek,TT93].

1.2 Privacy

In some cases, the sole purpose for carrying a mobile computer is to allow others to quickly locate the carrier (e.g. an active badge location system [WHFaG92]). In most cases, however, people carry mobile computers in order to send and receive information. While this may include messages from other users who are trying to contact them, it is not necessary, in general, to locate someone in order to contact that person. However, without proper care the messages that a mobile computer sends and receives in order to collect and disseminate information may inadvertently leak private information about the computer's owner to the person with whom the owner is communicating as well as to eavesdroppers.

In designing a system which preserves privacy for mobile users, we found it useful to partition privacy into three semi-independent components. The first is content privacy which is preserved if an attacker [1] is unable to extract the plain-text of the data sent from one computer to another. Content privacy can be maintained through the proper use of message encryption and signatures.

[1] Throughout the thesis, when we refer to an attacker, we mean any entity which attempts to acquire information that it is not intended to receive. In all cases, an attacker will be assumed to be limited to polynomial time computations.

Even if the data portion of a message is encrypted, an attacker may be able to obtain useful information. By observing the addressing information attached to messages, the attacker may be able to determine who is communicating with whom. Since the network needs to have some means of getting messages to their intended recipients, addressing information can not simply be encrypted along with the message data. One way to solve this problem is to send messages through intermediary computers which secretly pass messages from one computer to another (an example of this is a MIX-network [Cha81,PP89] which will be discussed later in the thesis). However, in a mobile network, it is possible to take advantage of the computers' mobility to design a more efficient protocol to hide this information from an attacker than is possible in a static network.

The third type of privacy is location privacy. Just as with cellular telephones, many people will soon begin to carry mobile computers with them wherever they go. While the users of these computers will wish to be able to receive messages from others at any time, they may not want others to be able to locate them. In addition to determining who is communicating with whom, an attacker may attempt to use traffic analysis to electronically "stalk" users. As we will show later, a MIX-network can be used by a computer that wishes to send a message while hiding its location.

While a MIX-network can also be used to allow a computer to receive a message while hiding its location, it is not very efficient. We will present a technique which will allow a computer to read from a shared memory in such a way that an attacker will be unable to determine which piece of information is being read. We will also

show how this protocol can be used to create a message service that will allow mobile computers to read messages without revealing their location.

1.3 Related work

Message encryption and signatures are essential to all security and privacy schemes. There are two basic types of encryption schemes, symmetric and asymmetric. In a symmetric (secret key) scheme, the same key is used for both encryption and decryption. In most cases, the secret key is known to a pair (or group) of communicating parties and kept hidden from all outsiders. While there are many different secret key encryption schemes, the most well known is DES [DES77]. In an asymmetric (public key) scheme, the encryption and decryption keys are distinct. In addition, it is infeasible for someone who only knows the encryption key to determine the value of the decryption key. In most cases, the encryption (public) key is made widely available while the decryption (private) key is known only to a single user (its owner). An example of a public key scheme is RSA [RSA78]. As with most public key encryption schemes, RSA can also be used to sign messages. Messages are signed using the decryption key and verified using the encryption key.

There are several papers which describe protocols for maintaining the unlinkability of message senders and recipients. The concept of a MIX-network was introduced by David Chaum in [Cha81]. A MIX-network takes in a batch of messages and scrambles them so that an attacker can not match incoming messages with outgoing messages. There are several other papers describing variations of the original scheme [PIK93,PPW91,PW87,RS93]. The protocols in [Cha81,PIK93] have security problems which were corrected in [Pfi94,PP89].

In [Cha88], David Chaum presents an information theoretically secure technique for preserving the unlinkability of the sender and the recipient of a message. This paper describes a protocol for creating a virtual network in which computers can send messages anonymously. Every computer can read every message (although they may be encrypted), but no computer is able to determine the sender of any message. Since messages are broadcast to every computer, recipient anonymity is also guaranteed. While this technique is secure, it requires that every computer send and receive a large volume of data as well as share a large amount of secret data. This technique is also not well suited for mobile computers which may frequently disconnect from the network.

In [BCR87], Brassard, Crepeau, and Robert present a technique which allows a computer to read from a database without revealing which piece of information it is reading. In addition, it guarantees that the reading computer will only be able to read one piece of information. In the protocol, the entire content of the database is transferred to the reader in an encrypted form. The reader and the database then engage in a zero-knowledge protocol to enable the reader to decrypt one of the database entries. Since our protocol does not limit the amount of information that a reader can acquire, our memory service could be implemented by simply sending the entire contents of memory unencrypted. In section 3.5, we will present a protocol which satisfies our more limited requirements and has a small bandwidth overhead.

There has been some work in the area of privacy for mobile computers. In [AD94, BCY93,Car94], protocols are presented which encrypt messages that are sent along wireless links thus preventing an attacker from using the contents of these mes-

sages to locate users. The main goal of the protocols in these papers is to limit the computational overhead of the mobile computers. While these protocols will maintain the unlinkability of message senders and recipients as well as the location privacy of mobile computers, they assume that the static network is secure.

Uwe Wilhelm addresses some of these problems in [Wil95] by compartmentalizing information and dividing the separate pieces of information among distinct administrative domains. An attacker which is unable to obtain information from each of the administrative domains will gain no useful information. While the design has a smaller overhead than our system, it is not as resilient to corruption.

1.4 Thesis Contributions

This thesis presents a set of protocols which implement the core of a private message service for mobile computers. Our main interest in designing the private message service was to support secure e-mail exchanges between pairs of users of mobile computers. However, these protocols can also be used for any interaction between mobile computers or between static and mobile computers as long as there are no latency or bandwidth requirements (i.e. our protocols, in their current form, will not work for real-time audio or video). As an example, our protocols could be used to allow a user of a mobile computer to interact with an on-line database without revealing his/her identity or location. It could also be used to interact with a bank computer without revealing the user's location to the bank or revealing the user's identity to an attacker.

While most of the previous solutions to the problems of security and privacy in a network of mobile computers assume that the static network is secure, we

assume that all communications links (both wired and wireless) can be read by an attacker. In addition, we consider the possibility that an attacker may be able to corrupt some of the servers in the system, where the purpose of the attack is either to eavesdrop or to actively interfere with the system.

Disconnection of users from the network can pose problems for a message service. Users may not be available to read messages that are sent for them at the time that the messages are sent. In order to deal with this, the system should hold onto messages for users until they have been read. However, this is complicated by the anonymity that is built into our system to preserve users' privacy. In this thesis, we address the conflict between users' privacy and the need of the system to acquire information in order to properly dispose of messages which have already been read by their intended recipients.

The thesis discusses issues related to the implementation of the private message service. We also discuss the implementation of the authentication and time servers in the system and our reasons for choosing the particular protocols that were used to implement these services. Finally, we discuss the performance of our system as well as speculate on the performance achievable by the system with proper hardware support.

1.5 Thesis Organization

Many of the ideas presented in this thesis have been published elsewhere [CB95a, CB95b]. Chapter 2 describes the system model in which system is intended to operate. In chapter 3 we describe the protocols for the core services of the private message service. In order to simplify the presentation of the protocols, chapter 3

does not deal with malicious failures. The handling of these types of failures is discussed in chapter 4. Chapter 5 describes techniques for dealing with the disconnection of users. Chapter 6 describes the implementation of the private message service and discusses the performance of the system. Chapter 7, concludes the thesis.

Chapter 2

System Model

A wireless network for mobile computers consists of two distinct parts: the static network and the mobile computers. The static network consists of a collection of static (non-mobile) computers which are connected by wired communications channels. Among the static computers are servers, network routers, and base stations. Base stations are computers that are part of the static network and that are also capable of wireless communication. The base stations constitute the link between the static network and the mobile computers.

2.1 Mobile Computers

There are many different types of mobile computers. In this thesis, however, we are only interested in computers which are capable of both sending and receiving messages. This could include receive-only devices such a pocket pagers as long as the devices are capable of sending request messages to the network in order to retrieve messages for their users.

Mobile computers interact with each other and with the static network by send-

ing messages to and receiving messages from base stations. Since wireless communications signals have limited range, a mobile computer and a base station must be sufficiently close in order to be able to communicate with each other. Since users move over time, their mobile computers will move in and out of communication range with different base stations at different times. At times, a mobile computer may be too far away to communicate with any base station. If a mobile computer can not communicate with any base station, then it is said to be *disconnected* from the network.

It is a basic assumption of our system that the wireless network will service a large number of users (i.e. the owners of the mobile computers). It is further assumed that these users will move around in ways that are unpredictable to an attacker (it is not possible for our protocols to hide a user's location from an attacker which has a priori knowledge of the user's movements). The basic premise behind these assumptions is that an attacker should be incapable of determining the location of a particular user unless that user is spotted, by the attacker, sending or receiving a message which allows the attacker to identify the user. In other words, users are assumed to be *anonymous by default*. Starting from this assumption, a user's privacy can be maintained as long as the protocols for the private message service do not force the user to send (receive) any messages which might reveal the identity of the sender (recipient).

2.2 Base Stations

A base station is a network router that has a wireless connection. Due to the limited range of wireless communications, only mobile computers which are within

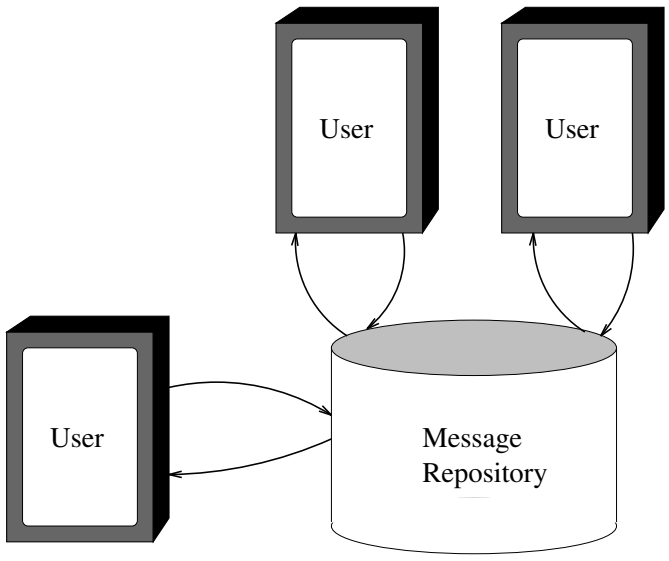

Figure 2.1: The Message Repository

a certain distance from a base station will be able to communicate with that base station. Any mobile computer which is within communications range of a base station is said to be in the *area* of that base station. At any given time, if a mobile computer is within range of at least one base station, it will be connected to exactly one base station. In the future, when we refer to the location of a user, we will mean the area which is within range of the base station with which the user's mobile computer is connected.

A mobile computer sends a message by attaching a header to the message with the destination address and then forwarding the message to the nearest base station which routes the message towards its destination. In our system, the destination of a message from a mobile computer will always be a server on the static network. If the message is intended for another mobile computer, then it will be sent to a message repository from which the intended recipient will retrieve the message (see figure 2.1).

When a mobile computer wishes to retrieve a message from the message repository, it sends a request message to the repository and the repository responds with the requested message. Whenever a mobile computer wishes to send a query for which it expects a response, it must use an RPC. The request will be sent to the appropriate server and the server will respond by sending its reply message to the base station from which the query originated. If the mobile computer moves to another base station before the reply arrives then it must re-send its request. If mobile computers move from base station to base station quickly relative to the round-trip time of an RPC, the server can send its reply to all of the base stations whose areas are neighbors of the area from which the request originated in addition to the originating base station.

2.3 The Network

In our system, we assume that all of the communication links (both wireless and wired) can be read by an attacker. In addition to eavesdropping, an attacker may attempt to disrupt the system by deleting or modifying users' messages or by creating bogus messages. While it is not possible to prevent an attacker from modifying or creating messages, it is possible to detect this behavior. All messages can be "signed" by their senders. The recipients of these messages can then check the signatures to verify that the messages actually came from the proper sources and that the messages have not been modified. If the signatures on the messages can not be verified, then the messages can be discarded and treated in the same way as lost messages are treated. If an attacker is able to modify or delete a significant number of messages within the network, then it may be able to prevent users from

exchanging messages. This will not, however, allow the attacker to obtain any private information or trick users into accepting bogus messages. Techniques for preventing this type of denial of service attack are outside the scope of this thesis. In this thesis, we will assume that attackers are unable to modify or delete messages. We will further assume that timestamps and nonces are used as appropriate to prevent message replay attacks.

Unlike communication links, servers are considered to be secure. An attacker can read messages that go into or come out of a server but does not have access to the contents of the server's memory. In some cases, however, the security of a server may be compromised. This can happen in one of two ways. If an attacker is able to obtain the private decryption key of a server, then it will be able to decrypt any messages that are intended for that server. This would give the attacker access to any secret information to which the server is privy.

The second way in which an attacker could corrupt a server would be by gaining full access to the server. This would allow the attacker to obtain the server's private decryption key (along with any other information in the server's memory). In addition to using its access to eavesdrop, an attacker may use its access to a server to change the protocol which the server executes. By forcing the server to behave maliciously, the attacker may be able to feed incorrect information to users as well as to other servers.

Whenever the security of a server has been compromised, we will say that that server has been corrupted. In chapter 3, we will describe ways to deal with servers which have been corrupted by attackers which only eavesdrop on corrupted

servers. In chapter 4, we will discuss techniques for dealing with servers that behave maliciously as a result of corruption by an attacker.

2.4 Messages

Since the messages that a user sends may contain information which could identify it to the intended recipients of those messages, users must have some way to send messages while hiding the locations from which they send the messages. Similarly, there needs to be a way for users to retrieve messages without revealing their locations. Since an attacker can observe all of the communications links in the network, it is only possible to hide the source (or destination) of a message from an attacker if it is possible to "confuse" the attacker through the use of other messages in the network. In order to accomplish this, users (as a whole) must produce and consume a large number of messages.

In order to guarantee that there will be enough messages in the network at any given time to ensure that the origins of users' messages can be hidden, there are certain servers within the network which buffer messages and do not forward the messages until their buffers are filled. In order to limit an attackers ability to pinpoint the origin of a message to one of a thousand possible locations, there must be at least one thousand messages in the network at the same time as the message whose origin is to be hidden. In order to guarantee this, each of the server's buffers will hold at least one thousand messages. So, while a temporary reduction in message traffic may not threaten the privacy of the system, it will increase the amount of time necessary to fill the servers' buffers and thus increase

the amount of time between when a user sends a message and when that message is available to be retrieved.

Just as there must be a large volume of message traffic in order to hide the origins of messages, it must be impossible for an attacker to distinguish one message from another. In particular, an attacker should not be able to match the encrypted version of a message with the plain-text version of that message unless it has access to the key used to encrypt the message. Since the lengths of a message and the encrypted version of that message are similar in many encryption schemes, it is important that all messages sent or received by mobile computers be of the same length. Whenever a user wishes to send a message of a length that does not conform to the standard length, that message must be segmented into packets and/or padded so that the resulting messages sent through the private message service are of the proper length.

Chapter 3

The Private Message Service

3.1 Introduction

In this chapter, we present the protocols for the core of the private message service. The private message service actually consists of several services which work together to provide privacy to users of mobile computers. These services are the message service, the MIX-network, and the vacation service. In addition, there is a time service and an authentication service which help to support the other services. The message service and the MIX-network comprise the core of the private message service and will be discussed in this chapter. The vacation service, which provides a vital service to users of the private message service which may become disconnected from the network for extended periods of time, will be discussed in chapter 5. Finally, the time and authentication services will be discussed in chapter 6.

As was described in section 1.2, there are three basic types of privacy. The first, content privacy, involves preventing an attacker from determining the contents of

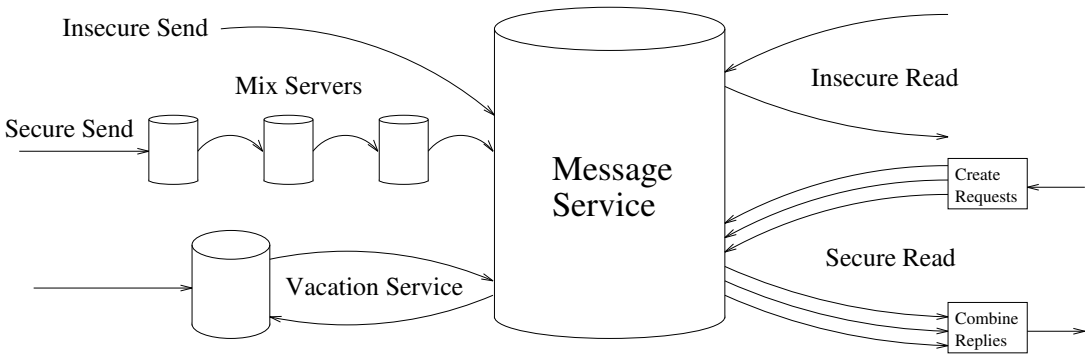

Figure 3.1: The Private Message Service

secret messages. Content privacy is protected by using end-to-end encryption and is described in section 3.2.

The basic structure of the private message is depicted in figure 3.1. The message service acts as a message repository for all messages intended for mobile computers. A message is sent to a mobile computer by attaching a label [1] to the message and sending it to the message service. Once the intended recipient of the message discovers that there is a message in the message service with a label in which it is interested, it can retrieve the message from the message service (along with anyone else). As will be shown later, the message service has been designed to allow users to read from the message service in such a way that no one can determine which message the user is reading.

Since message labels are used by users to determine which messages to read from the message service, it is important to avoid having the same label used in two different conversations at the same time. While such a collision will not lead to a breach in the security or privacy of the system, it will affect performance. As will be shown later, for privacy reasons, many of the labels that are used in the

[1] A label in our system is equivalent to a visible implicit address in [PW87].

system are chosen by users at random. Since there is no centralized mechanism to prevent collisions, message labels should be large enough to make the chances of two users choosing the same label at approximately the same time acceptably small. As will be shown in section 3.3, the inability of an attacker to determine who is communicating with whom is maintained by the careful choice of message labels.

The final type of privacy, location privacy, is maintained through the joint effort of the message service and the MIX-network. The message service, by allowing users to read messages without revealing which messages they are reading, allows users to remain anonymous when they are reading messages, thus effectively keeping their location hidden. The MIX-network, on the other hand, takes in batches of messages and scrambles them together. The result is that it is impossible for an attacker to determine the origin of any message in the batch and thus an attacker is unable to determine the location of any of the message senders. The MIX-network will be described in section 3.4.

As will be shown later, depending on the perceived threat at the moment, a user may be able to save some computation and communication by avoiding the use of some of the privacy enhancing mechanisms. For example, the MIX-network is not necessary when sending a message to a user which is trusted by the message sender. In many cases, a user can also save some time when retrieving a message if the user trusts the message's sender.

3.2 Content Privacy

The primary concern in designing a system which provides privacy for users is to ensure that users can communicate with each other while preventing others from determining what they are saying. This is accomplished by using a two step process. First, the contents of messages are obscured (encrypted) to prevent eavesdroppers from reading the contents of the messages sent between the users. Second, the identities of the users are verified (authentication) to prevent an attacker from impersonating one of the users. Encryption and authentication will be described in some detail below.

3.2.1 Encryption

The contents of messages sent over insecure communications channels are hidden using encryption (see [Bra88,Sim92]). Say that one user, p, wishes to send a message, m, to another user, q, in such a way that an eavesdropper will be unable to determine the value of m. First (in a way to be described later), p and q will agree on a pair of functions, E and D, which satisfy the property that $D(E(m)) = m$ for all $m \in \mathcal{M}$ (where $\mathcal{M}$ is the set of all possible messages). In addition, it must be impossible (or at least computationally infeasible) to compute m from $E(m)$ without knowledge of D.

There are two basic types of encryption schemes. The first is secret key, or symmetric, encryption (e.g. DES [DES77]). In such a scheme, the participants in a conversation choose a key, K_s, which is known only to them. The message sender uses the function E_{K_s} to encrypt messages and the receiver uses D_{K_s} to decrypt

them. Secret key encryption schemes are generally much faster than public key encryption schemes. As a result, they tend to be used whenever possible.

The major problem with secret key encryption schemes is the need to agree on the key. Since the value of the key must remain secret, it can not be transmitted, unencrypted, over an insecure communications channel. This problem is solved by public key, or asymmetric, encryption schemes (e.g. RSA [RSA78]). In a public key encryption scheme, a user, q, that wishes to receive private messages chooses two keys, K_q and K_q^{-1} (K_q is the encryption function, E, and K_q^{-1} is the decryption function, D). The encryption key, K_q, is called the public key and can be made widely available. The decryption key, K_q^{-1}, is called the private key and is known only to q. A user, p, wishing to send a message, m, to q, computes $K_q(m)$ and sends to the result to q. Upon receipt, q can use its private key, K_q^{-1}, to compute $K_q^{-1}(K_q(m)) = m$.

Using a combination of public and secret key encryption schemes, it is possible to develop a protocol which has most of the efficiency advantages of using secret key encryption while also avoiding the key exchange problem that goes along with such schemes. Say that a user, p, wishes to communicate with another user, q. First, p chooses a secret key, K_s, which it encrypts using q's public key, K_q. Next, p encrypts the initial message, m_1, that it wishes to send using K_s and sends $K_q(K_s), K_s(m_1)$ to q. Upon receipt, q uses K_q^{-1} to extract K_s from $K_q(K_s)$ and uses K_s to extract m_1 from $K_s(m_1)$. Subsequent messages sent from p to q or from q to p can be encrypted using K_s.

The scheme shown above has two basic shortcomings. First, there needs to be some way for p to get q's public key. While secrecy is not an issue (as it is with

the secret keys), public key distribution must still be handled very carefully. An attacker, a, wishing to read messages intended for some other user, u, may attempt to impersonate u by convincing users in the system that its public key is u's public key. If a is successful, users wishing to send messages to u will use a's public key instead of u's public key and a will be able to read the messages instead of u.

In most systems, users get the public keys of other users by consulting a directory service which is responsible to maintaining and distributing the public keys of users within the system. There are two basic ways in which an attacker could cause a user to receive an incorrect key. First, the attacker could gain control of the directory service and force the service to distribute incorrect information. Second, it could attempt to impersonate the directory service. Later in the thesis, we will describe a technique for designing an authentication service which can be used to distribute keys which makes such attacks difficult.

3.2.2 Authentication

As long as the authentication service is set up properly, p can begin a conversation with q and be confident that q will be the only user receiving the messages that p sends to it. However, using the protocol shown above, q has no way of knowing that the messages it is receiving actually came from p. In order for q to be able to verify the origins of the messages that it receives, p must have some way to "sign" messages that can be verified by q.

In order for a digital signature scheme to be effective it must be able to prevent an attacker, a, from creating a message along with a signature for that message such that the message appears to have been signed by another user, u. The digital signature scheme must also prevent a from modifying a valid message which was

signed by u in such a way that it appears that u signed the modified message. More formally, a digital signature scheme consists of a signature function, S_K, and a verification predicate V_K such that $V_K(m, s)$ holds iff $s = S_K(m)$. In addition, it must be computationally infeasible for an attacker to compute $S_K(m)$ from m without knowledge of K even if the attacker has access to several $< m, S_K(m) >$ pairs.

Just as with encryption schemes, there are both secret key and public key signature schemes. Both types center around the use of one-way hash functions [2]. In some cases, the hash function itself can serve as the signature function. If the hash function, h, can take two arguments, m and K, such that it is infeasible to compute $h(m, K)$ without knowledge of K, then we can make the signature function $S_K(m) = h(m, K)$ (for a secret key signature function). The verification predicate $V_K(m, s)$ would be "$s = S_K(m)$". The inability to compute $h(m, K)$ without knowledge of K would prevent an attacker from creating signatures for messages and the one-way property of the hash function would prevent an attacker from creating new messages which have the same signatures as properly signed messages.

An alternative is to use an encryption scheme in combination with a one-way hash function [3]. In the case of secret key encryption, the signature would be $E_K(h(m))$ which could be verified by the recipient by re-computing the signature for m and comparing the result to the signature attached to the message.

Public key signature schemes work in a similar fashion. In many cases (such as RSA), the encryption and decryption functions in a public key encryption scheme

[2] A hash function, h, is one-way if, given $h(m)$ for some m, it is infeasible to find an $m' \neq m$ such that $h(m) = h(m')$.

[3] Not all encryption functions are suitable for use in digital signature schemes.

can be reversed (e.g. $K_q(K_q^{-1}(m)) = K_q^{-1}(K_q(m)) = m$). Thus, q can sign a message m by computing $K_q^{-1}(h(m))$. While it is infeasible for anyone but m to compute $K_q^{-1}(h(m))$, anyone with access to q's public key, K_q, can verify the signature s of m by verifying that $K_q(s) = K_q(K_q^{-1}(h(m))) = h(m)$.

Using a combination of public and secret key signature schemes, it is possible to efficiently authenticate messages sent between users. In general, messages will be signed and verified using a secret key scheme (just as is the case with encryption). In order to start the process, the users will need to agree on a key (which can be accomplished in the same way as was done for encryption). A public key signature scheme, however, will need to be used to sign the secret keys that are sent at the initiation of a conversation.

3.3 Unlinkability of Sender and Recipient

The section above describes how two users can establish a conversation and communicate in such a way that no one else can determine what they are saying to each other. In many circumstances, however, the users may also wish to prevent other people from determining that they are communicating at all. In order to prevent an attacker from acquiring this information, it must be possible to hide the identity of at least one participant in each conversation that is taking place in the system.

A solution to this problem was first proposed by David Chaum in [Cha81] and was later refined in [Pfi94,PIK93,PP89,PPW91,PW87,RS93]. The basic idea behind these schemes is to use servers (called MIXes) which are responsible for reading in batches of messages and reordering them in such a way that it is impossible

for an attacker to determine the correspondence between incoming and outgoing messages. Since the technique involves using public key encryption/decryption for each message, it computationally expensive.

The papers cited above all deal with preventing an attacker from linking the senders and recipients of messages in a static network. In a static network, since users' locations within the network can not be hidden from an attacker, it is impossible for a user to send or receive a message anonymously [4]. As a result, in order to prevent an attacker from determining the identities of both the sender and recipient of a message, it is necessary to hide the path of the message through the network. This is the responsibility of the MIXes.

With mobile computers, if users can keep their locations hidden [5], then it is possible to set up a system in which they send and receive messages anonymously. In order to do this, there must be a way to send a message within the network which does not contain any information (in plain-text) which might allow an attacker to determine the identity of either the sender or the intended recipient of the message.

The key to sending messages anonymously is based on the routing mechanism used in the system. A message is sent to a mobile computer by attaching the recipients "address" to the message and sending the message to the message service. Once the message service has the message, the intended recipient sends a request to the service for messages with that "address" and the service sends any such messages in response. Unlike in a static network, the "address" is not used by the system to locate the intended recipient. (Since the "address" does not need to

[4] It may be possible for a user to hide the destination (or source) of a message being sent (or received) along with the contents of the message, but the user can not hide the fact that it is sending (or receiving) a message.

[5] A technique for accomplishing this will be shown later in this chapter.

convey any information about intended recipient, in the future we will simply refer to it as a *label* to emphasize that it has no special meaning).

In order to successfully transfer a message from sender to recipient, it is only necessary that the sender attach a label to the message which the intended recipient will eventually use to request messages from the message service. Since the value of the label is meaningless to the network, it is possible to have private labels (similar to secret keys). However, just as with encryption, the use of public labels is necessary to initiate conversations [6].

In order to allow for the initiation of conversations between users, each user has a public label which is stored in an on-line directory [7]. The directory service is similar to the authentication service of the last section (in fact the authentication service can be used to distribute public labels as well as public keys). However, since the corruption of the directory service will not lead to a breach of security or privacy, the directory service does not need to be as secure as the authentication service [8].

Let's say that one user, p, wishes to initiate a conversation with another user, q. First, p will look up q's public label, l_q (along with q's public key, K_q). Next, p will attach l_q to the initiation message and send it to the message service. Since q is always on the lookout for initiation messages, q will eventually make a request to the message service for messages with label l_q and will receive p's initiation

[6] It is possible to avoid the use of public labels by using a technique such as Diffie-Hellman key agreement [DH76] to establish private labels to initiate conversations, however, it is not practical for systems with a large number of users.

[7] Some users may choose not to have a public label (this would be equivalent to having an unlisted phone number). Alternatively, some users may have multiple public labels under multiple aliases.

[8] An attacker, by corrupting the directory service, may be able to prevent users from beginning new conversations, but will not be able to compromise the security or privacy of the system.

message. Since l_q is a public label, anyone observing network traffic will be able to determine that some user is attempting to begin a conversation with q, but since p can send the message anonymously, no one (except possibly q) will know that p sent the message.

Since q's identity is revealed in the initiation message, it is necessary, in order to prevent an attacker from determining that p and q are communicating with each other, to keep p's identity hidden. This can be done by using private labels. In the initiation message, p will include a random number, r_1, which can be used by q to send a reply message. If q wishes to send a reply to the initiation message, it can send the message to the message service using the label r_1. Since p created r_1, p will know to request messages with that label from the message service and will, therefore, receive the reply message. In all future messages, sent by either p or q, the sender can include a new random number to be used as the label for any reply message sent by the other user.

While it is not necessary to use a new random value for each message sent in a conversation, doing so will reduce the amount of information that an attacker will be able to infer about a conversation (such as the number of messages exchanged between q and some anonymous user). For this reason, the random labels used in this scheme should, if possible, be generated using a source of truly random numbers. If no source of truly random numbers is available, then a cryptographically strong pseudo-random number generator should be used [ILL89,BM84,BM82, VV83,BBS86,Sha81,BBS82].

3.4 Location Privacy while Sending a Message

In the previous section, we described how to prevent an attacker from determining the identities of both parties in a conversation by using private message labels to help make conversation participants anonymous. While the use of private labels helps to hide users' locations, it is not sufficient. In this section, we will deal with the problem of sending messages without the risk of revealing one's location and in the next section, we will deal with receiving messages.

While a user will never include information about its own identity, in plaintext, in a message that it sends to another user, its identity may be known to the intended recipient of the message. This will be the case, in a conversation, if the user was not the conversation initiator or if the user was the initiator but revealed its identity to the other participant. Since communications links are not secure, it is possible to determine the location from which any message was sent. While this information will not be useful to most eavesdroppers (since they will not know the identity of the sender), it may be useful to the intended recipient of the message. Therefore, if the sender is concerned about hiding its location from the intended recipient of a message that it is sending, it must work to prevent the recipient from determining the location from which it sends the message.

One solution to this problem is to use a MIX-network [Cha81] (as was described briefly in the previous section). The purpose of a MIX is to make it impossible for an attacker to determine the correspondence between the sources and destinations of the messages that pass through the MIX. This is accomplished using encryption and message batching.

A user, wishing to protect its location from another user with which it is having

a conversation, prepares any messages that it wishes to send to that user by encrypting the messages with the public key of a MIX, K_M. The user will then send the result, $K_M(l, m, S)$, to the MIX (in the formula, l is the label for the message, m is the message, and S is the address of the message repository). Upon receipt, the MIX decrypts the message. Once the MIX has received enough messages to fill a batch (say 1000 messages), it reorders the messages and sends each message to its destination.

As written above, an attacker could determine the correspondence between incoming and outgoing messages by simply re-encrypting each of the messages. In order to prevent this, some form of nondeterminism must be introduced into the encryption function. This can be accomplished by encrypting some random data along with the message which the MIX will remove after decryption (as is shown in [PP89], the way in which the random data is incorporated is important to the security of the system).

In order to prevent another type of attack, the MIX must be responsible for preventing message replays. Since the same input to a MIX produces the same output (i.e. decryption is deterministic), an attacker could locate a specific message from an input batch in an output batch by re-sending the message. Since the input message will produce the same output both times, it can be located in the output by taking the intersection of the outputs of the two batches in which the message was sent (assuming no other attacker is attempting a replay attack from the same batch at the same time). By preventing the attacker from re-sending the message through the MIX, this type of attack is blocked.

The above scheme for hiding the location a message sender from the message's

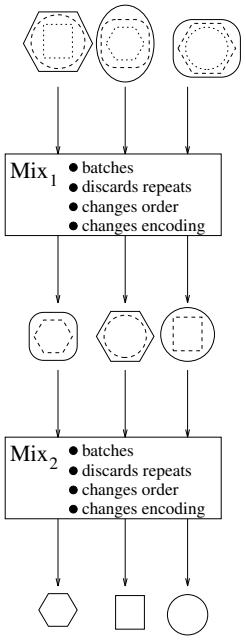

Figure 3.2: A MIX-Network

intended recipient will work as long as the MIX is not corrupt. If the intended recipient is able to read the contents of the MIX's memory or knows the MIX's private key, then the correspondence between messages coming into and going out of the MIX will not be hidden from that user. In order to make it more difficult for an attacker to gain useful information, a MIX-network can be created using a set of MIXes in such a way that an attacker would not gain any information unless it corrupts all of the MIXes in the network (see figure 3.2).

A MIX-network consists of a chain of MIXes, each operating as described above. As an example, if there are three MIXes, M_1, M_2, and M_3, then the user would send $K_{M_1}(K_{M_2}(K_{M_3}(l, m, S), M_3), M_2)$ to M_1. M_1 would decrypt the message and send $K_{M_2}(K_{M_3}(l, m, S), M_3)$ to M_2 which would decrypt that and send $K_{M_3}(l, m, S)$ to M_3. Finally, M_3 would send (l, m) to S. As long as at least one of the MIXes is not corrupted, the location of the sender will remain hidden.

3.5 A Memory Service with a Blinded Read Operation

Just as the protocol of the previous section is necessary to protect the location of a message sender from the message's intended recipient, a technique is needed to protect a user's location when it is attempting to retrieve a message from the message service. Such a technique is needed in two cases. First, when a user makes a request to the message service for an initiation message. Since the label on the message is publically known, anyone observing network traffic will be able to identify the requesting user and will also be able to determine that user's location. In addition, just as in the previous section, a user's attempt to request a message with a private label can be used by the sender of that message to locate the recipient.

One way to hide a users location would be to use a MIX-network as in the previous section. A user would send a request message through the MIX-network to the message service. This would get the request message to the message service without anyone being able to determine from where the request originated. In order to be able to receive the response from the message service, the request message would need to include an *anonymous return address* [Cha81,PPW91,PP89] to be used by the message service to send the response (which also goes through the MIX-network). Since both the request and the response go through the MIX-network, the cost of retrieving a message, using this technique, would be twice the cost of sending a message.

The basic problem with the above technique is that the user must inform the message service of the label in which it is interested. Since providing this label

to the message service may reveal the user's identity, the user must use the MIX-network to hide its location. In order to avoid the need for the user to hide its location, the user must be able to hide its identity. This can be accomplished by using a protocol which allows the user to retrieve messages of interest from the message service without revealing which messages it is retrieving.

In this section, we will describe a protocol for a replicated shared memory which will allow a computer (whether mobile or not) to perform a blinded read operation (one in which an attacker is unable to determine which position in memory is being read). In the next section, we will show how a modified version of this memory can be used to create the message service. Unlike previous protocols, ours is efficient in its use of both bandwidth and computation.

A memory service (the replicated shared memory) consists of a set of n memory servers each of which has an array of m cells labeled $M[0], M[1], \ldots, M[m-1]$. As in the previous section, we will assume that an attacker may be able to corrupt some, but not all, of the servers. For this section, we will assume that an attacker is able to corrupt at most $t < n$ of the servers.

3.5.1 Reading from Memory

The technique for reading from memory is similar in nature to secret sharing (see [Sim92]). In a (k, n) secret sharing scheme, a user generates n shares of a data value and distributes one share to each of n servers. The system is designed so that any k shares can be used to regenerate the original value while fewer than k shares reveals no (or very little) information about the original value. Thus, secret sharing schemes allow users to store backups of private information (to help

prevent accidental loss) while minimizing the risk of revealing that information to others.

In our scheme for a blinded read operation, there are n servers which hold copies of the memory. A user wishing to read the value of one of the cells in the memory constructs a set of $t+1$ formulas, $f_1, f_2, \ldots, f_{t+1}$, and sends each formula to different server. Each server evaluates the formula that it receives and sends the response back to the user. The user then combines the responses from all of the servers to determine the value of the cell that it wished to read. Just as in the secret sharing schemes, an attacker which is able to obtain at most t of the formulas/responses will gain no (or very little) information about which cell the user read.

For this section, we will assume that only read operations are performed and that the contents of the servers' memories are the same. Say that a user wishes to perform a blinded read operation to read the value in cell p of the memory. The user begins by creating a set of $t+1$ bit-vectors of length m, $V_1, V_2, \ldots, V_{t+1}$. Bit-vectors $V_1, V_2, \ldots, V_t$ should be created at random. Using these bit-vectors, the user computes V_{t+1} as:

$$V_{t+1} = I_p \oplus \bigoplus_{i=1}^{t} V_i$$

where

$$I_p[j] = \begin{cases} 0 & \text{if } j \neq p \\ 1 & \text{if } j = p \end{cases}$$

The result is a set of $t+1$ bit-vectors which satisfy $V_1 \oplus V_2 \oplus \ldots \oplus V_{t+1} = I_p$ (an example of such a set is shown in figure 3.3).

vector	bit position							
	0	1	2	3	4	5		$m-1$
V_1	0	1	1	0	1	0	$\cdots$	0
V_2	1	1	0	1	1	0	$\cdots$	1
V_3	1	1	1	1	0	0	$\cdots$	1
$V_1 \oplus V_2 \oplus V_3$	0	1	0	0	0	0	$\cdots$	0

Figure 3.3: Sample Bit-Vectors for $t = 2, p = 1$

Using these bit-vectors, the formulas $f_1, f_2, \ldots, f_{t+1}$ are defined as:

$$f_j = \bigoplus_{V_j[i]=1} M[i]$$

In order to prevent an attacker from learning the values of the bit-vectors, the user creates a set of $t+1$ secret keys, encrypts each key with the public key of one of the memory servers, encrypts each bit-vector with one of the secret keys, and sends the results to the appropriate servers. Each server decrypts the message, computes a response by evaluating the formula defined by the bit-vector it receives, encrypts the response with the secret key that was used to encrypt the bit-vector that it received, and sends the response to the user. The user then computes the value in cell p as $r_1 \oplus r_2 \oplus \ldots \oplus r_{t+1}$ (see figure 3.4).

Security of Blinded Read

Lemma 1 *If each of the bits in the t random bit-vectors are set to 1 with probability $\frac{1}{2}$ then an attacker which has access to at most t of the requests/responses associated with the bit-vectors will gain no information about which cell the client is reading.*

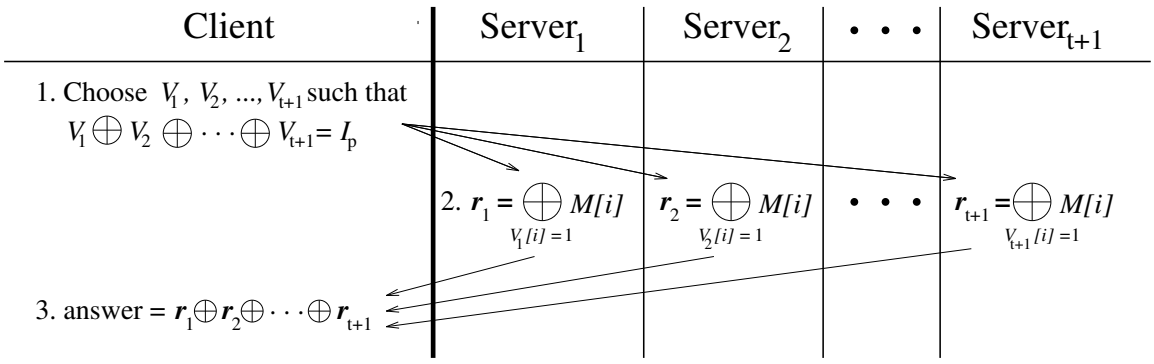

Figure 3.4: Bit-Vector Protocol

Proof: Since the first t bit-vectors are chosen independently of the cell being read, an attacker will gain no information unless it has access to the $t+1^{st}$ bit-vector. We will, therefore, assume that the attacker has the $t+1^{st}$ bit-vector along with $t-1$ of the t random bit-vectors. Let's call the bit-vectors that the attacker knows $V'_1, V'_2, \ldots, V'_t$ and the bit-vector that it doesn't know V''.

Say that the client is reading the value of cell p.

- case 1: $i = p$

  Since this is the cell being read, we know that $V'_1[p] \oplus V'_2[p] \oplus \ldots \oplus V'_t[p] \oplus V''[p] = 1$. Since $V''[i]$ is equally likely to be 0 or 1 and $V'_1[i] \oplus V'_2[i] \oplus \ldots \oplus V'_t[i] = \neg V''[i]$, $V'_1[i] \oplus V'_2[i] \oplus \ldots \oplus V'_t[i]$ is also equally likely to be 0 or 1.

- case 2: $i \neq p$

  Since this is not the cell being read, we know that $V'_1[i] \oplus V'_2[i] \oplus \ldots \oplus V'_t[i] \oplus V''[i] = 0$. Since $V''[i]$ is equally likely to be 0 or 1 and $V'_1[i] \oplus V'_2[i] \oplus \ldots \oplus V'_t[i] = V''[i]$, $V'_1[i] \oplus V'_2[i] \oplus \ldots \oplus V'_t[i]$ is also equally likely to be 0 or 1.

Since, for each position, the value of $V'_1[i] \oplus V'_2[i] \oplus \ldots \oplus V'_t[i]$ is equally likely to be 0

or 1 whether it is the position being read or not, the attacker gains no information about which cell is being read.

Sparse Bit-vectors

As was shown in lemma 1, if the bits in the random bit-vectors are truly chosen at random (i.e. each bit is equally likely to be either a 0 or a 1), then an attacker which sees at most t of the bit-vectors will gain no information about the position being queried. However, using such bit-vectors, each memory server will, on average, have to exclusive-or together $\frac{m}{2}$ values. This can be computationally expensive for the memory servers if the memories have a large number of cells or if the cells are large (in the next section, each cell will contain a message).

One way to reduce the amount of work necessary to compute a response is to create random bit-vectors with fewer 1's. Instead of setting each bit to 1 with probability $\frac{1}{2}$, each bit could be set to 1 with probability $\varphi < \frac{1}{2}$. This will decrease the computation time but will increase the amount of information that an attacker can infer. The value for φ must, therefore, be chosen carefully.

The specific value of φ to use in any given system will be based on the available computational power and the security needs of the system. However, in order to choose a proper value for φ, it is important to know the amount of information that may leak to an attacker when any given value is chosen for φ. In order to do this, we must look at the set of bit-vectors from an attackers point of view [9].

As was mentioned in the proof of lemma 1, an attacker will gain no information unless it has access to the bit-vector V_{t+1} since the all of the other bit-vectors are created independently of the cell to be read. We will, therefore, assume that the

[9] We would like to thank Anindya Basu, Sayandev Mukherjee, and Michael Turmon for their help in deriving the probability distributions which are presented in the formulas below.

attacker has access to V_{t+1} along with $t-1$ of of the remaining bit-vectors. We will call the t bit-vectors that the attacker knows $V'_1, V'_2, \ldots, V'_t$ and the remaining bit-vector V''.

In order to simplify the derivation, we will need the following definitions:

$$C = V'_1 \oplus V'_2 \oplus \ldots \oplus V'_t$$

$$S_0 = \{i \mid C[i] = 0\}$$

$$S_1 = \{i \mid C[i] = 1\}$$

$$S'_0 = \{i \mid V''[i] = 0\}$$

$$S'_1 = \{i \mid V''[i] = 1\}$$

$$I_p[j] = \begin{cases} 0 & \text{if } j \neq p \\ 1 & \text{if } j = p \end{cases}$$

Due to the way that bit-vectors are created, we know that $V'' = C \oplus I_p$. Given knowledge of C, the attacker knows that the only possible values for V'' are $C \oplus I_0, C \oplus I_1, \ldots, C \oplus I_{m-1}$ and that the user is reading position j iff $V'' = C \oplus I_j$. Since the attacker knows the way in which V'' was chosen (i.e. knows the value of φ), it can compute the a priori probabilities that each of the possible values of V'' was chosen as the actual value for V'' and use these probabilities to compute the a posteriori probabilities for each of the possible values using the formula [10]:

$$P(V'' = C \oplus I_i) = \frac{\text{a priori probability that } V'' = C \oplus I_i}{\sum_{j=0}^{m-1} \text{a priori probability that } V'' = C \oplus I_j}$$

The a priori probability that $V'' = C \oplus I_i$, for each i, can be computed as follows:

[10] We are assuming, for simplicity, that, from the attackers point of view, the a priori probability of each position being read by the user is the same. Consideration of other probability distributions is straightforward and is left as an exercise to the reader.

- case 1: $i \in S_0$

  In this case, $|S_0'| = |S_0| - 1$ and $|S_1'| = |S_1| + 1$. From this we can conclude that the a priori probability that $V'' = C \oplus I_i$ is $\varphi^{|S_1|+1}(1-\varphi)^{|S_0|-1}$.

- case 2: $i \in S_1$

  In this case, $|S_0'| = |S_0| + 1$ and $|S_1'| = |S_1| - 1$. From this we can conclude that the a priori probability that $V'' = C \oplus I_i$ is $\varphi^{|S_1|-1}(1-\varphi)^{|S_0|+1}$.

Using the above formulas, we get:

$$P(V'' = C \oplus I_i) = \begin{cases} \frac{\varphi^{|S_1|+1}(1-\varphi)^{|S_0|-1}}{|S_0|\varphi^{|S_1|+1}(1-\varphi)^{|S_0|-1}+|S_1|\varphi^{|S_1|-1}(1-\varphi)^{|S_0|+1}} & \text{if } i \in S_0 \\ \\ \frac{\varphi^{|S_1|-1}(1-\varphi)^{|S_0|+1}}{|S_0|\varphi^{|S_1|+1}(1-\varphi)^{|S_0|-1}+|S_1|\varphi^{|S_1|-1}(1-\varphi)^{|S_0|+1}} & \text{if } i \in S_1 \end{cases}$$

$$= \begin{cases} \frac{\varphi^2}{|S_0|\varphi^2+|S_1|(1-\varphi)^2} & \text{if } i \in S_0 \\ \\ \frac{(1-\varphi)^2}{|S_0|\varphi^2+|S_1|(1-\varphi)^2} & \text{if } i \in S_1 \end{cases}$$

Since the values for $P(V'' = C \oplus I_i)$ depend on $|S_0|$ and $|S_1|$, the actual probability distribution that an attacker will be able to infer can not be calculated a priori (except in the special cases of $\varphi = 0, \varphi = \frac{1}{2}$, and $\varphi = 1$). However, using φ, we can estimate the values for $|S_0|$ and $|S_1|$ and use these values to estimate $P(V'' = C \oplus I_i)$. Since, $V'' = C \oplus I_p$ (for some p), we know that $C[i] = V''[i]$ for $m-1$ values of i ($i \neq p$) and $C[i] = \neg V''[i]$ for 1 value of i ($i = p$). We know that $V''[i]$ will be set to 1 with probability φ and to 0 with probability $1-\varphi$. Therefore, we can expect that of the $m-1$ positions for which $C[i] = V''[i]$, $\varphi(m-1)$ will be in S_1 and $(1-\varphi)(m-1)$ will be in S_0. The remaining position, p, will be in S_0

φ	S_0		S_1	
	P_0	$\lvert S_0\rvert$	P_1	$\lvert S_1\rvert$
0	0	1023	1	1
$\frac{1}{100}$	$\frac{0.0092}{1024}$	1012.78	$\frac{90.43}{1024}$	11.22
$\frac{1}{10}$	$\frac{0.1103}{1024}$	920.8	$\frac{8.938}{1024}$	103.2
$\frac{1}{5}$	$\frac{0.2495}{1024}$	818.6	$\frac{3.991}{1024}$	205.4
$\frac{3}{10}$	$\frac{0.4283}{1024}$	716.4	$\frac{2.332}{1024}$	307.6
$\frac{2}{5}$	$\frac{0.6666}{1024}$	614.2	$\frac{1.500}{1024}$	409.8
$\frac{1}{2}$	$\frac{1}{1024}$	512	$\frac{1}{1024}$	512

Figure 3.5: Probabilities for $m = 1024$

with probability φ and in S_1 with probability $1 - \varphi$. Combining these, we get can estimate $|S_0| = \varphi + (1 - \varphi)(m - 1)$ and $|S_1| = (1 - \varphi) + \varphi(m - 1)$.

In figure 3.5, we show some sample values for $P(V'' = C \oplus I_i), |S_0|$, and $|S_1|$ for a memory with 1024 cells (in the figure, $P_j = P(V'' = C \oplus I_i)$ for any $i \in S_j$). The values for $P_0, |S_0|, P_1$, and $|S_1|$ were computed using the formulas which were derived above to estimate $|S_0|$ and $|S_1|$. In the case of $\varphi = \frac{1}{100}$, there is a 99% chance that the position being read is in S_1 and we can expect that $|S_1| \approx 11.22$. While there is a chance that the user is reading one of the approximately 1012.78 positions in S_0, it is very unlikely. So, while seeing t of the $t + 1$ bit-vectors does not allow the attacker to rule out any of the positions entirely (for $0 < \varphi < 1$), if φ is relatively small (or large), the attacker will be able to extract a relatively small group of positions such that the cell being read is highly likely to be in that group.

3.5.2 Writing to Memory

Given the protocol for performing blinded read operations, the protocol for writing to memory is relatively straightforward. The main concern in designing a protocol

for writing to memory is to ensure that the contents of the memories of the different servers remain consistent. The protocol in figure 3.4 assumes that the contents of each server's memory will be the same. If the responses to the bit-vectors are computed using cell values that differ from server to server, the computed value for the desired cell will be incorrect. As an example, consider the bit-vectors in figure 3.3. The responses from the 3 servers will be

$$\begin{aligned} r_1 &= M_1[1] \oplus M_1[2] \oplus M_1[4] \\ r_2 &= M_2[0] \oplus M_2[1] \oplus M_2[3] \oplus M_2[4] \oplus M_2[m-1] \\ r_3 &= M_3[0] \oplus M_3[1] \oplus M_3[2] \oplus M_3[3] \oplus M_3[m-1] \end{aligned}$$

and the computed answer will be

$$\text{answer} = \begin{array}{l} M_2[0] \oplus M_3[0] \oplus \\ M_1[1] \oplus M_2[1] \oplus M_3[1] \oplus \\ M_1[2] \oplus M_3[2] \oplus \\ M_2[3] \oplus M_3[3] \oplus \\ M_1[4] \oplus M_2[4] \oplus \\ M_2[m-1] \oplus M_3[m-1] \end{array}$$

If $M_1 = M_2 = M_3$ then the above equation will reduce to answer $= M[1]$. However, if $M_2[3] \neq M_3[3]$ for some reason (perhaps a write operation is in progress), then answer $= M[1] \oplus M_2[3] \oplus M_3[3] \neq M[1]$.

There are two ways that the above situation could occur. The first is if two write operations are performed concurrently. Say that there are two operations, $w_1 \equiv M[3] := 2$ and $w_2 \equiv M[3] := 5$, that server 2 performs w_1 before w_2, and that server 3 performs w_2 before w_1. If, after these write operations have completed, the read operation depicted in figure 3.3 is performed, the result will be:

$$\begin{aligned} \text{answer} &= M[1] \oplus M_2[3] \oplus M_3[3] \\ &= M[1] \oplus 5 \oplus 2 \\ &= M[1] \oplus 7 \end{aligned}$$

(i.e. the 3 least significant bits of $M[1]$ will be flipped). In order to prevent this type of problem, we must ensure that all write operations are performed by every server in the same order. This can be accomplished by using a multicast protocol which guarantees totally ordered delivery to distribute write requests [BSS91,CASD85, CM84].

In some cases, it may be necessary to protect against the possibility of malicious users. If this is the case, the multicast protocol must protect against the possibility of a user either sending different values to different servers or of a user causing write operations to be performed in different orders at different machines. This can be most easily accomplished by having users send update requests to a single server which will multicast the requests to the other servers. (Servers must also sign their messages in order to prevent a user from impersonating a server). The use of an appropriate multicast protocol can prevent a malicious user from corrupting memory by causing the values in the memories of the servers to differ. However, a user may still be able to corrupt memory by simply writing bad values into the cells. While this can not be entirely prevented, the problem can be mitigated by using some form of access control mechanism [11].

The second way in which a user could receive a bad value in response to a blinded read operation would be if a read operation was performed concurrently

[11] As this type of attack will not be relevant in our use of the memory service, we will not discuss the use of access control mechanisms.

with a write operation. There are two ways around this problem. The first is to bundle the request messages and use a totally ordered multicast to send them to the servers [12]. The second option is to prevent read and write operations from executing concurrently. In our system, read operations on a memory are delayed until all write operations have been completed. As a result, read requests can be sent as unordered unicast messages to a subset of the servers thus completely avoiding the cost of multicasting a bundled set of requests to all of the servers.

3.6 Retrieving a Message

The message service acts as intermediate storage for messages intended for mobile computers (as shown in figure 3.1). Messages are sent to the service either directly or through a MIX-network and are eventually retrieved by the intended recipient. In this section, we will show how to use the memory service of section 3.5 to implement a message service which will enable users to read messages without revealing their identities.

3.6.1 Sending a Message to the Message Service

Messages are sent to the servers using a totally ordered multicast (as was described in section 3.5.2). At each server, arriving messages are placed in a list in the order in which they are delivered. This list is stored in a series of tables each of which holds m messages (i.e. the i^{th} message delivered is stored in table $(i-1)$ **div** m in cell $(i-1)$ **mod** m). Figure 3.6 shows the first 4 tables for a message service in which $m = 5$. There is a tradeoff that must be considered when choosing

[12] Notice that since read requests do not change the states of the memories, the delivery of read requests need only be totally ordered with respect to write requests and not with respect to each other.

table 0	table 1	table 2	table 3
l_0, m_0	l_5, m_5	l_{10}, m_{10}	l_{15}, m_{15}
l_1, m_1	l_6, m_6	l_{11}, m_{11}	l_{16}, m_{16}
l_2, m_2	l_7, m_7	l_{12}, m_{12}	l_{17}, m_{17}
l_3, m_3	l_8, m_8	l_{13}, m_{13}	l_{18}, m_{18}
l_4, m_4	l_9, m_9	l_{14}, m_{14}	l_{19}, m_{19}

Figure 3.6: A Sequence of Message Tables

a table size. On the one hand, the amount of effort needed to read a cell from a table is proportional to the size of the table (the client must create and encrypt bit-vectors of length m and the servers must decrypt the bit-vectors and exclusive-or together φm messages). In addition, as will be described later, users must wait until a table has been filled before reading the messages in that table. Therefore, as m increases, the time between when a message arrives at the message service and when it can be read from the service increases. On the other hand, as will be described later, as m increases, the amount of privacy increases for users who are reading messages.

Once a table has been filled, users may read messages from that table. In order to enable message reading, a digest of the table's contents is created and sent to all of the mobile computers. The digest of a table is $h(l_0), h(l_1), \ldots, h(l_{m-1})$ where $l_0, l_1, \ldots, l_{m-1}$ are the labels attached to the messages in each position of the table and h is a hash function (figure 3.7 shows the table digests for the 4 tables shown in figure 3.6).

Since every user will need to see the digest for every table, table digests are

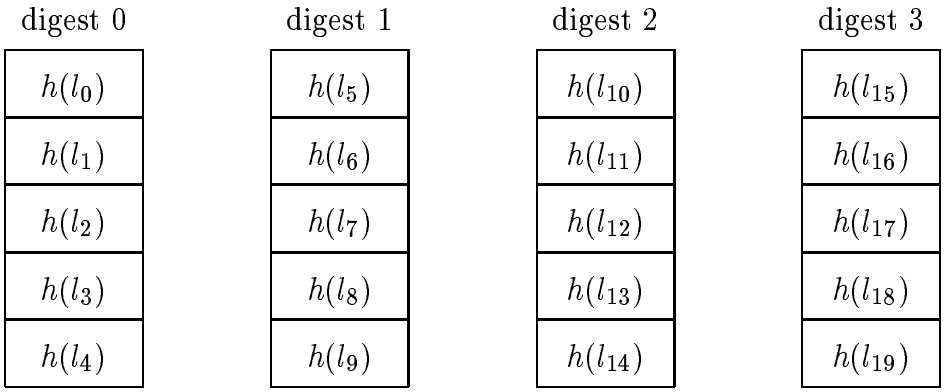

Figure 3.7: A Sequence of Table Digests

broadcast to mobile computers. Once a table is filled and its digest computed, the digest is sent to all of the base stations (using a multicast protocol for the static network). Upon receipt, each base station broadcasts the digest over its wireless link. Some of the mobile computers will not receive the broadcast (for example, those that are disconnected from the network). Therefore, the base stations will also maintain a local copy of the digest and resend it as necessary to ensure that every mobile computer receives the digest (see [AB93,DFM91] for more information on multicasting in mobile networks).

3.6.2 Reading from a Table

Each user will have a list of message labels in which it is interested, $(l'_0, l'_1, \ldots, l'_k)$. This list includes the labels for reply messages that the user is expecting from ongoing conversations as well as the user's public label. When the user receives a digest, it will look for $h(l'_0), h(l'_1), \ldots, h(l'_k)$ in the list $h(l_0), h(l_1), \ldots, h(l_{m-1})$. If the user finds some i and j for which $h(l'_i) = h(l_j)$ then it will read the message from cell j of the table.

Users can read messages from the message service in one of two ways. If l'_i is a private label and the user trusts the message sender (or the message sender does not know the identity of the recipient), then it can send a request to one of the servers containing the pair (k, j) where k is the number of the table to be read and j is the number of the cell within that table which contains the message (since the user is not attempting to hide which cell it is reading, there is no need to encrypt either the request or the response in this case). If the label to be read is public or if the user does not trust the message sender, then it must use the blinded read operation from section 3.5. The mobile computer will create $t + 1$ bit-vectors of length m and send each bit-vector, along with the number of the table to read, to a different message server. Since mobile computers can not read from tables until after they are filled (i.e. after the last write operation has completed), the request messages do not need to be bundled and the totally ordered multicast protocol is not needed.

Once the user has read in the message, it must check that the message it has received has the correct label. Before reading in the message, the user was only able to see $h(l_j)$ and it may be) that $h(l'_i) = h(l_j)$ but $l'_i \neq l_j$. If the labels do match, then the user will attempt the decrypt the message using the key that is appropriate for the message. Since users choose private labels independently, there is a small chance that the message was not intended for the user despite having the correct label. If this is the case, then the user will detect this when attempting to decrypt the message and will discard the message without delivering it.

3.6.3 Choosing a Hash Function

Unlike the scheme in section 3.5, the message service can not be set up in such a way that an attacker gains no information about which message a user is attempting to read. Since, by corrupting just one message server, an attacker can determine from which table a user is reading, the uncertainty about which message the user is reading is limited by the size of the table, m. However, since users may read messages that are not intended for them, an attackers uncertainty about the identity of a user reading a message may be greater that its uncertainty about which message is being read.

If the hash function, h, used in creating message digests is the identity function then $h(l) = l$ for all l and there will be no case where $h(l') = h(l)$ for $l' \neq l$. This means that

ave messages intended for them in a table will read
hoose the same return "address" label at about the
d in a message that was not intended for it, but this
since an attacker can determine the locations from
vill have a list of (at most) m locations from which
if the attacker knows the identity of the intended
abel l_j, it will know that that user is in one of m
hen the attacker will have gained useful information
can not be made large enough to sufficiently confuse
ion must be chosen which will force some users to
nat were not intended for them.
the total number of message labels in which every
we choose a hash function, h, which maps message

labels to values between 0 and 31,999, then there will be, on average, two message labels which hash to each value. If $m = 1024$, then there will be approximately 2048 requests to the message service. Of these, 1024 will be from the intended recipients of the messages in the table and 1024 will be from randomly chosen users. Thus, an attacker seeing a user read a message from the table will know that there is only a 50% chance that the user is the intended recipient of one of the messages in the table.

3.6.4 Garbage Collection

In an infinite run of the system, an infinite number of messages will be sent to the message service. It is, therefore, essential to have some mechanism for removing old messages from the system. Ideally, a message should be deleted after it has been read by its intended recipient. However, since the message servers will, in general, not know whether or not a message has been read and since messages may be read by users other than the intended recipient, the message servers will not know when a message has been read by the intended recipient.

An approximate solution is to delete messages after some period of time, Δ, has passed. In our system, a table is left intact until its newest message has been in the system for time Δ at which point the entire table is deleted. If Δ is chosen properly, then every computer will have sufficient time to retrieve all messages intended for it while the number of tables stored in the system at any one time is manageable.

In some cases, a user will be disconnected from the network for a long period of time. This can happen if the user moves outside of the range of all of the base stations or if the user's computer is turned off to conserve battery power. In either

case, if the user is disconnected for too long, it may miss some of the messages that were sent to it. In order to avoid this, we have developed a vacation service.

If a user is expecting a message and is afraid that it may miss the message as a result of being disconnected for some time greater than Δ, then the user will register the label associated with that message with the vacation service. If necessary, the vacation service will store a copy of the message and will hold onto that message until the user has reconnected and will resend the message to the message service so that the user can retrieve it. Protocols for implementing the vacation service will be described in detail in chapter 5.

3.7 Ending a Conversation

Until this point, we have treated a conversation as a sequence of messages with a beginning but with no end. In practice, most conversations will only last for a short period of time. Many other conversations will be sporadic in nature, with periods of high message traffic followed by long periods with no traffic. Since a mobile computer, for each conversation, must store label and key information and check every table digest for a message, it is inefficient to have a large number of conversations when most of them are inactive.

There are two basic ways in which a conversation can be ended. If one participant in a conversation, when sending a message, does not wish to receive a reply message, then that user can replace the return "address" label in the message with an "end of conversation" marker. After sending this message, the sender can erase from its memory any information about the conversation and the recipient can do the same upon receipt of the message.

In many cases, however, the sender of a message will not know whether or not that message will be the last message in the conversation. In this case, the sender can add an expiration time to the return "address" label. The basic idea is that if the recipient has not sent a reply message by the time the return "address" label expires, the sender will consider the conversation to have ended and will erase the return "address" label and the session key for the conversation from its memory.

There are two complications which must be dealt with in using return "address" labels with expiration times. The first is that the clocks of the sender and recipient will not be perfectly synchronized and the second is that there will be a delay between when a user sends a message and when that message can be retrieved from the message service. While it is impossible to have perfectly synchronized clocks, there are several protocols which achieve partial synchronization [SWL90, Rei93,RBvR94]. The basic idea behind all of the clock synchronization protocols is that the clients (those wishing to have their clocks synchronized with each other) periodically communicate either with each other or with a clock synchronization service. At these times, the clients receive estimates of the current time which they use to reset their own clocks. The result of the synchronization step for each client will be that its local clock is approximately synchronized either with the other clients or with the clock service (since message delivery times may vary, the degree to which the local clocks differ can only be limited by the uncertainty in message delivery times). After clients have reset their clocks, as a result of variances in the rates of the local clocks of the different machines, the differences in the local clocks may increase over time necessitating another round clock synchronization.

In [RBvR94,Rei93], every client's clock periodically synchronizes with a single

server whose local clock is treated as a source of real time. Clients periodically send a request to the server for the current time and use this value, along with estimates of the minimum and maximum possible message delivery delays and the actual round-trip time to get a response, to compute a range of clock values (the range of values is the smallest possible range which is guaranteed to include the real time). After synchronization, the range is determined by using the range at the time of synchronization, the amount of time that has passed since synchronization according to the local clock, and an estimate of the maximum possible clock drift.

Using the protocol of [RBvR94,Rei93], expiration times for return "address" labels can be used as follows. When a user, p, sends a message to another user, q, it includes an expiration time, e, for the return "address" label. Using the expiration time, q will know that p will only be guaranteed to receive its message if it sends the reply by time $H_q(t_q) \leq e$ (where $H_q(t_q)$ is q's upper bound of the range of possible values of the master clock when q's local local clock reads t_q). If we assume a maximum delivery delay of δ time units, then any reply message sent by time $H_q(t_q)$ will arrive at the message service by time $H_q(t_q) + \delta$.

Whenever a table in the message service becomes filled, one of the message servers will broadcast the digest for the table along with the time that the last message arrived in the table (the time used will be $L_{ms}(t_{ms}) \leq t$ where $L_{ms}(t_{ms})$ is the message server's lower bound of the range of possible values of the master clock when the message server's local clock read t_{ms}). Based on the properties of the clock synchronization algorithm, we know that at real time t, $H_q(t_q) \geq t$ and $L_{sm}(t_{sm}) \leq t$ (where t_p is the value of p's local clock at real time t and t_{sm} is the value of the message server's local clock at read time t). Therefore, we can

conclude that at any given time, $L_{sm}(t_{sm}) + \delta \leq H_q(t_q) + \delta$. As a result, once p has received a message digest with a timestamp greater than $e + \delta$, p will know that there will be no messages in any future tables from p which were sent before $H_q(t_q)$ read e and therefore p can treat the return "address" label as expired and erase the label and the session key for the conversation from its memory.

Chapter 4

Dealing with Failures

4.1 Introduction

In chapter 3, we made the assumption that servers do not crash. While operating under this assumption simplified the protocols, it is not a reasonable assumption for most systems. Even though crashes are relatively infrequent, any system which runs for very long periods of time will be likely to experience some server crashes over the lifetime of the system. Since it is not reasonable to expect users to accept either an extended period of time in which messages are unavailable (while the crashed servers are rebooted) or the loss of messages by the system, the private message service should be designed to remain available despite a limited number of server crashes.

In many cases, it may be deemed necessary to prepare for the possibility of malicious failures. An attacker which is able to corrupt some of the servers, while unable to compromise users' privacy, may be able to prevent users from receiving messages (denial of service attack) by causing the servers it has corrupted to behave

incorrectly. This incorrect behavior could take the form of lying to users when responding to request messages or attempting to disrupt other servers.

Many protocols have been developed to solve various problems in distributed computing in system models in which processes may crash or behave maliciously. In this chapter, we will briefly describe some of the relevant protocols and will show how these protocols can be used to design a private message service which tolerates failures.

4.2 Crash Failures

There are two basic things that need to be done in order to tolerate crash failures. The first is to ensure that operational sites agree about which sites are operational and which are not and the second is to ensure that there is sufficient redundancy to be able to maintain functionality despite the unavailability of the crashed sites.

4.2.1 Agreeing on Group Membership

Even if all of the servers which are responsible for providing a particular service are honest, it can still be difficult for the servers to agree on the status of each of the servers in the group. The basic problem is the inability to distinguish between a server which is temporarily overloaded and therefore nonresponsive and one that has crashed. Instead of actually determining whether or not a server has crashed, a server will usually guess that another server has crashed if a period of nonresponsiveness lasts longer than some predetermined timeout period. However, since these guesses can sometimes be inaccurate, it is possible that a server p will guess that another server q has crashed while all other servers still believe that q is operational.

Since disagreement on which servers have crashed may cause many protocols to behave incorrectly, it is important to have a mechanism available to ensure that there is a set of operational servers all of whom agree about which servers are operational and which are not. A solution to this problem can be found in [Ric92]. The basic idea behind this protocol is that when one server decides that another has crashed, it informs a coordinator which is responsible for informing all of the other servers of the crash and for coordinating the removal of that server from the group. In addition, there are mechanisms for handling the failure of the coordinator in such a way that a new coordinator takes over while still maintaining agreement of the group membership.

Since it is useful for both availability and efficiency to have as many operational servers as possible at any one time, it is important to have a mechanism to allow servers which have crashed (or which were erroneously believed to have crashed) to rejoin the group. There are things that must be done when a server wishes to rejoin the group. First, the other servers must be informed about the new member so that they can each add the new member to their view of the group membership. Second, the state of the new server must be set to be consistent with that of the other servers [1].

Since users (or clients) remain anonymous, it is not possible for the servers to maintain status information about them. However, none of the protocols in the private message service require this information for correctness. On the other hand, users will need to know which servers are operational in order to decide to which servers they should send requests. While it is not necessary that the beliefs of the

[1] In the case of the message servers, this would mean sending the new server a copy of every message that has been received but not yet deleted along with the organization of these messages into tables.

users agree with the beliefs of the servers (or with each other), it will, in general, be most convenient for the servers to broadcast status changes to the users.

4.2.2 The MIX-Network

The MIX-network, when used, sits between users and the message service (when the users are sending messages). It is, therefore, the responsibility of the MIX-network to ensure that any messages that it receives are delivered to the message service. In the scheme presented in chapter 3, at any point in time, each message was held by only one MIX server. Since the result of this server crashing would be the loss of all of the messages that the server was holding, this scheme needs to be enhanced in order to satisfactorily handle crash failures.

One solution would be to make users responsible for ensuring that their messages arrive at the message service. This could be accomplished in one of two ways. The first would be to use acknowledgements. A message sent by a user would propagate through the chosen chain of MIXes until it reached the message service. Using the same chain, in reverse, an acknowledgement would be sent back to the user. If the user receives a message declaring that one of the MIXes in the chain that it was using has crashed and has not received an acknowledgement, then it will choose a new set of MIXes to form a chain and resend the message. The major problem with this technique is that the user may have moved or become disconnected between the time that it sent the message and the time that the acknowledgement arrives.

Since messages in the message service can be read by anyone, users could determine whether or not their messages have arrived at the message service by looking for the message in the message service directly instead of waiting for an acknowl-

edgement. While this avoids some of the problems of the first technique, it still has several problems of its own. First, if a user becomes disconnected from the network soon after sending a message which is lost by the MIX-network as a result of a crash, the message will not arrive at the message service (and be available to the intended recipient) until the sender becomes reconnected to the network. Second, since users will need to read every message that they send to the message service, the message servers will have to perform extra work, thus reducing the number of messages that the service can handle.

In order to avoid these problems, the MIX-network should take responsibility for ensuring that messages are not lost even when some of the MIX servers crash. Since it is impossible to design a system which can handle any number of crashes, one must first determine how many simultaneous crashes the system should be able to handle without losing any messages. In order to handle up to t simultaneous crashes, each server should have t backups. Each backup should have a copy of the server's private key (or access to it) and should receive copies of every message sent to the server. If the server crashes, then one of the backups (having all of the necessary information) can take over the job of the server until it has been rebooted. Another solution is to design the MIX-network so that servers which have crashed can be bypassed [PW87].

4.2.3 The Message Service - Receiving Messages from Users

The message service has three responsibilities towards incoming messages. It must ensure that none of the messages are lost, that they are delivered to all of the

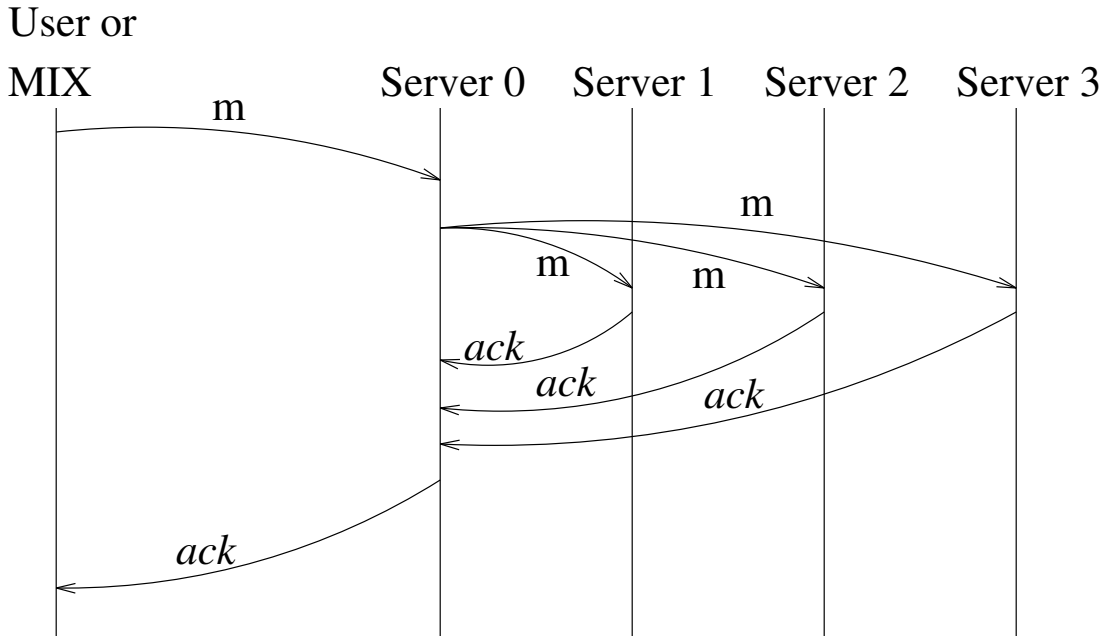

Figure 4.1: Coordinator Based Totally Ordered Multicast

operational message servers in the same order, and the every user receives a copy of every table digest so that they will know about the messages.

Messages are sent to the message servers using a totally ordered multicast protocol [CASD85,Bir93]. One way to avoid the problem of lost messages is to wait until all of the message servers have received the multicast before sending an acknowledgement to the message sender (as in figure 4.1). In the protocol depicted in figure 4.1, server 0 acts as a coordinator which is responsible for forwarding messages to the other servers and for acknowledging the receipt of messages on behalf of the servers. Totally ordered message delivery is guaranteed by having the coordinator forward messages in the order in which it delivers the messages using a multicast which guarantees FIFO delivery. If the coordinator ever fails, a new coordinator takes over. This new coordinator first checks for any messages

that were sent to some servers but not others and ensures that these messages are received by all of the servers in the correct order after which it can handle new messages as normal.

4.2.4 The Message Service - Retrieving

The protocol for retrieving a message from the message service does not need to change substantially in order to tolerate server crashes. Since request messages do not change the states of the servers, a user which sent a request to a server which failed can simply send the request to another server. If the user is attempting to perform a blinded read (using the bit-vector technique), then the user must be careful when choosing a new server to which to send the request. If we assume that up to t_1 servers may become corrupted, then it is important to ensure that no set of t_1 servers will see all of the bit-vectors used to make a request. Since, for efficiency reasons, the user will only create $t_1 + 1$ bit-vectors, the user must ensure that no server sees more than one of the bit-vectors. Therefore, if up to t_2 servers could be unavailable at any given time, there must be at least $t_1 + t_2 + 1$ message servers in order to guarantee that users will always be able to safely perform a blinded read.

4.3 Malicious Failures

The possibility of malicious failures adds several new complications to the design of the private message service. In addition to the need to deal with the possibility of crash failures, the protocols must contend with the possibility that some of the servers may send messages with incorrect information. This possibility has a large effect not only on the users which need to ensure that they receive the correct

information when they attempt to retrieve messages but also on the servers which need accurate group membership information and need to guarantee that the states of all of the (correct [2]) servers remain consistent.

The basic technique for handling malicious servers is to ensure that there will be enough correct servers available to "out vote" the malicious ones, thus allowing users (and other servers) to determine which information is correct and which is incorrect. In order ensure that in a system with a majority of correct servers that the correct servers will always out vote the malicious servers, we must prevent malicious servers from impersonating correct servers. Otherwise, the malicious servers could "steal" votes from correct servers in order to make it appear that a majority of the servers (and thus at least one correct server) agreed on some incorrect value causing a user (or server) to accept the value as correct.

In this section, we will describe the changes that need to be made to the message service in order to tolerate a limited number of malicious failures among the message servers. The message service uses three basic protocols each of which must be modified. In section 4.2, we described protocols for maintaining consistent group membership information and for providing totally ordered multicast in a system in which crash failures are possible. In 1994, Michael Reiter developed protocols for solving these problems in systems in which a limited number (less than one-third) of the servers fail. The details of these protocols are beyond the scope of the thesis and can be found in [Rei94a,Rei94b].

[2] In this context, a server is correct if it is both honest and operational.

4.3.1 Servers Lying to Users

Given the protocols for group membership and totally ordered multicast, the only remaining protocol which needs to be altered is the one used by users to retrieve messages from the message service. There are two basic ways to ensure that users will get correct responses. The first is to have each user make every request to several servers and take the majority response as the correct response. This will work as long as a majority of the servers which respond to any given request are correct.

The problem with the first technique is that it is very expensive. In a system which can tolerate at most one malicious server, every request message will have to be sent to three different servers. This will reduce the effective throughput of the message service by a factor of one-third and will require every user to encrypt every request using the keys of three different servers. In addition, all of this extra work will be required even if there are no malicious servers.

In order to avoid the use of a voting scheme, there must be some means by which a user can distinguish between a correct response and an incorrect response when looking at the response of a single server (or set of servers). (As we will show later, the system can be both safe and live as long as a user never mistakes an incorrect response for a correct one. Occasional mistakes in the other direction can be tolerated.) In our system, users are requesting messages from the message service. Since these messages are created by other users (and not by the message service), users sending their messages to the message service can sign their messages. If a user p receives a message from the message service which is supposed to be from

q but which has not been properly signed by q, then p will know that either the message did not come from q or that q's message has been altered.

Even before a user can verify the signature on a message that it receives, it must know that the message has arrived at the message service so that it will know to retrieve it. In our system, users rely on the table digests to inform them of the possibility of a message being available. Since table digests are created by the message service and there is no way for users to distinguish a correct digest from an incorrect digest, all digests must be broadcast using the voting technique. In order to tolerate up to t malicious failures, each digest must be broadcast by at least $2t + 1$ message servers. Each user will receive all of the copies and accept the majority response as the correct value. In order to prevent one server from impersonating another, all table digests must be signed by the servers.

In section 3.6, each entry of the table digest was of the form $h(l)$ where l was the label attached to the message and h was a hash function. If a user was looking for a message with label l_1 and $h(l_1) = h(l)$ then the user would read in the corresponding message, $< l, m >$, and check to see if $l_1 = l$. If $l_1 \neq l$, then the user would discard the message. If we assume that some servers may lie, then it is possible that the user will receive $< l', m' >$ from the servers where $l' \neq l$ but $h(l') = h(l)$. If this is the case, then the user will not be able to determine if $< l', m' >$ is a valid message for another user or if the server(s) which responded to its request lied. Whenever a user is unable to distinguish between these two scenarios, it will have to assume that the servers lied and revert to the voting technique to either get the correct response or very that the original response was correct.

In order to minimize the chances that a user will need to revert to the voting technique, we must prevent servers from undetectably sending messages with incorrect labels. There are two ways to do this. The first is to use the identity function for h and the second is to use a one-way hash-function. Since the use of a one-way function will not (substantially) reduce the size of table digests and will increase computational overhead, we recommend using the identity function. By using the identity function, since the user knows that the digests are correct, the user will know if the value that it reads from the message service, $< l', m' >$ has an incorrect label. If l' does not match the label from the digest, then at least one of the servers which the user contacted lied and the user will have to revert to the voting technique to retrieve the correct message. If l' is the correct label, then the user will check the signature on the message. If the signature is valid, then the user will know (with very high probability) that it has received the correct message. If the signature on a message is invalid, then the user will have to use the voting technique to read the message.

There are several reasons that a user, waiting for a message with label l from another user q, could receive a message of the form $< l, m >$ from the message service in which m is not properly signed by q. One possibility is that it did not receive the proper response from the message service (which is the reason for reverting to the voting technique in order to verify the response). However, it may be the case that the response from the message service was correct. This could happen in one of two ways. First, m could be a valid message intended for another user. Since it will not be possible for the user to determine whether m is an invalid message or a valid message intended for another user, the user will have to use

the voting technique to read in the message and verify that m was the correct response. Since this should only happen in cases in which two users choose the same return "address" label at approximately the same time, this should occur very infrequently and so should not impact performance.

Another reason that a user might receive a correct response from the message service which does not seem to carry a valid signature would be if another user sent a bogus message to the message service. This is most likely to happen if the label on the message is the user's public label. A user, p, wishing to "harass" another user, q, could send a large number of bogus messages to the message service with q's public label. Since q will not be able to distinguish between a bogus message sent to the message service and an incorrect response from the message service, it will have to read in the each message, determine that the signature is invalid, and then read each message again using the voting technique. Only after verifying that the responses from the message service were correct (using the voting technique) will q be able to determine that the messages were bogus and ignore them. Since there may be a valid initiation message (probably sent by some user other than p) interspersed among the bogus messages, q can not safely ignore the initiation messages from p. Also, since q may have to deal with some malicious message servers in addition to the bogus messages from p, it can not avoid using the voting technique to verify each of the responses it gets from the message service.

Chapter 5

The Vacation Service

5.1 Introduction

The purpose of the vacation service is to prevent users who become disconnected from the network for a long period of time from losing messages as a result of the garbage collection that takes place in the message service. There are two basic competing interests with which one must contend when designing such a service. The first is the need to prevent the vacation service from becoming a potential source of information which might compromise privacy. The second is the need for the vacation service be able to limit the number of messages that it must store at any given time while holding on to messages until they have been received by their intended recipients. In this chapter, we will discuss some of the issues related to the design of a vacation service and present several solutions to the problem.

5.2 The Problem of Privacy

In chapter 3, there was a distinction made between private and public message labels. If a label was private and in use in a conversation in which the user trusted the other participant then the process of sending or receiving a message using that label was greatly simplified. As a result, it is important that users not be required to provide any information to the vacation service which might reveal one of its private message labels in a way which could be useful to an attacker. On the other hand, the vacation service needs to know which messages to store and which to discard.

One solution would be for users to interact with the vacation service using an alias. Periodically the user would inform the vacation service of the list of labels in which it is interested so that the service would know which messages to save in case the user became disconnected. However, if the list of labels contained the user's public label then the vacation service would be able to determine the user's identity and thus would know that user's private labels. As a result, users would need to use the expensive techniques for sending and retrieving messages in order to prevent the vacation service (in addition to the people with whom they are communicating) from being able to locate them.

Even if the list of labels which the user provides to the vacation service does not contain any publically known labels the attack described above could be performed to a lesser extent. Say that a user, u, provides the vacation service with a list of labels, $(l_1, l_2, \ldots, l_k)$, such that l_i is for use in a conversation with user u_i. Suppose that u trusts user u_a and wishes to take advantage of this to retrieve messages from u_a efficiently. However, there may be another user, u_b, which u does not trust

which has access to the vacation service and which gets u's list by looking in the vacation service for a list of labels which contains the label l_b. Since u will be the only user to have l_b in its list, u_b will know that this list belongs to u and will know that u is also the intended recipient of the message with label l_a. As a result, in order for u to hide its location from u_b, it will have to use the bit-vector technique to retrieve all of its messages instead of just those from u_b.

5.3 A Simple Solution

One possible solution to the above problems would be for users to always read messages from the message service using the bit-vector technique. In this case, users could simply register their list of labels with the vacation service under their own identities (sending the registration messages through the MIX-network in order to hide their locations). Whenever a user sends a message it would add the return "address" label to the list that it had registered with the service and whenever it receives a message it would remove the corresponding label from the list (unless the message was an initiation message using the user's public label). In addition, the user would periodically send "I'm connected: (table j)" messages to the vacation service, telling the service that the user is connected and has received all messages in tables up through j.

If, after some timeout period, the vacation service had not heard from a user, the service would assume that the user had become disconnected from the network. Once the vacation service determines (based on the length of time since the user last sent an "I'm connected: (table j)" message) that the user is at risk of missing a message, the vacation service will begin to scan the tables in the message service,

beginning at table $j+1$, for messages with any of the labels in the users list. If the vacation service finds any such messages, then it will store them for later retrieval by the user.

When a user reconnects to the network after being disconnected for a long enough period of time that it was unable to read from some of the tables from the message service, it will send a message to the vacation service specifying which tables it missed. If the vacation service had stored any messages for the user from any of the missed tables, then the service will re-send those messages to the message service so that they can be retrieved by the user. After sending the messages to the message service, the vacation service can delete the messages from its memory [1].

5.4 A Solution which does not Prevent Efficient Message Reading

The reason that the previous solutions did not allow users to efficiently read messages which came from users which they trusted was that all of a user's labels were tied together under the same alias in the vacation service. One way to avoid this would be for users to register each label under a different alias thus preventing an attacker from using knowledge of the intended recipient of one label to determine the intended recipients of other labels.

This system would work in basically the same way as the previous system with the exception of a few small changes. First, when registering a private label which will be used to receive a message from a trusted user, there would be no need to send

[1] The labels associated with these messages would not be removed from the user's list of labels until the user had acknowledged receiving them. Therefore, in the unlikely case that the user immediately becomes disconnected from the network again and is unable to read the messages, the vacation service would simply re-read the messages from the message service upon detecting that the user is disconnected.

the registration message through the MIX-network since the registration message would not identify the user. Second, the user would need to send a separate "I'm connected: (table j)" message for each active alias that it had with the vacation service. The final difference is that each alias would never be used for more than one message label.

5.5 The Final Solution

There is one basic problem with the solution of section 5.4. In order for the vacation service to know whether or not to store a copy of a message (or when it can delete a message), the user must keep the network informed about when it is connected to the network and which messages it has received. There are two basic ways to accomplish this. The first is for the user to periodically send "I'm connected: (table j)" messages to the vacation service as long as it is connected. By using these messages, the vacation service will be able to avoid saving copies of most of the messages that are sent through the network. The second solution is for the vacation service to store a copy of every message and for users to inform the network whenever they have received a message so that the vacation service can delete the message.

While the protocols in sections 5.3 and 5.4 both solve this problem, the protocol in section 5.4 requires the user to perform extra work without giving the user an incentive to do this work. A user, wishing to minimize work for itself by ignoring the protocol, could save work by not sending "I'm connected: (table j)" messages and by not telling the vacation service when it has received a message. Since the vacation service would be unable to tell the difference between a connected user

which was refusing to send "I'm connected: (table j)" messages and a user which was disconnected, it would never be able to delete the user's messages. Since the user registers each of its message labels under a different alias, there would be no way for the vacation service to prevent this type of activity [2].

There are two basic solutions to the above problem. The first is to limit the amount of storage space for each user within the vacation service and the second is to charge for space. In the first case, users will have to monitor their use of memory since it is a limited resource and in the second case will choose to limit their use due to the costs. In both cases, the solution centers around the use of electronic cash [Cha85,Cha92]. Below, we will describe the solution based on strictly limiting the amount of space allotted to each user. The technique for charging money for space is similar and the necessary changes to the system are straightforward.

5.5.1 Limiting Storage Space

Using the first solution, each user will be allotted a certain number of storage spaces. In order to use a storage space, the user first sends a request to the vacation service to "withdraw" one of its space allotments. The request will be same as a withdrawal in David Chaum's electronic cash scheme [Cha85,Cha92]. Assuming the user has not already withdrawn all of its allotments, the result of the transaction will be that the user will have a certificate entitling it to the use of

[2] In the solution of section 5.3, the user could still refuse to send "I'm connected: (table j)" messages but, since the vacation service will know the identities of users registering message labels, it could discourage such behavior. One possible deterrence would be to send bills to users for the use of the vacation service in which the amount of the bill was proportional to the number of messages that the service had to hold for the user and the length of time that the service had to hold on to the messages. If the cost of refusing to send "I'm connected: (table j)" messages was greater than the benefits, then this would effectively discourage such behavior.

a unit of space (enough space to store a message). Just as with cash, there will be no way for the vacation service, or anyone else, to tie the certificate to the user.

When a user is ready to send a message, it will need to register the return "address" label in order to avoid missing any reply message. To do this, the user will send the return "address" label along with one of its certificates to the vacation service. The vacation service will check the validity of the certificate and, if it is valid, will accept the label and will check the message service for messages with that label. If the certificate had already been in use, then the old label will be deleted along with any message being stored under the certificate.

As in the previous protocols, if a user has been disconnected for a long enough period of time that it was unable to read from some of the tables in the message service, then it will send a request message to the vacation service, for each one of its active certificates, to see if the vacation service had found any messages under the registered label.

A user can use the same certificate to register all of the return "address" labels for a given conversation. However, in order to avoid the problems described in section 5.2, a user should not use the same certificate in two different conversations (which the user may wish to do if one conversation ends before another begins). On the other hand, since each user is only allotted a fixed number slots in the memory of the vacation service and each certificate represents a slot, there needs to be a way for users to refresh certificates. Once a user is done using a certificate, it can send the certificate to the vacation service along with a request for a new certificate. Just as with "withdrawing" a certificate, the new certificate will not be traceable to either the user or to the old certificate. Once the vacation service

receives the request, it will place the old certificate on a list of invalid certificates to prevent its further use.

In order to prevent the memory needed to hold the list of invalided certificates from continuously growing, there can be a set of valid signatures for certificates. Each certificate will have a limited lifetime and will become unusable once its signature becomes invalid unless it has been cashed in for a valid one before the expiration time. Once a signature becomes invalid, all certificates with that signature can be removed from the list of invalid certificates.

Chapter 6

The Implementation and Performance

6.1 Introduction

In chapter 3, we presented the basic protocols for the core of the private message service. In this chapter, we will describe some of the important details of the implementation of the system along with some performance figures to show the potential throughput of the system.

Due to our limited resources, it was not feasible for us to implement a mobile network with a large number of mobile computers. Instead, we implemented a prototype of the system in which the mobile computers were replaced by processes running on static computers which were directly connected to the network. This resulted in several minor changes to the system. First, since the user processes could communicate directly with the servers, there as no need for base stations. In addition, the lack of mobility meant that there was no need to deal with users

changing location over time or with disconnection. This simplified the protocol for broadcasting table digests to users and also eliminated the need for a vacation service. Other than these changes, however, our prototype of the private message service was designed to mimic the actions of a true implementation in a mobile network as closely as possible. Since the main concern in our prototype was to determine the achievable throughput of the core servers of the private message service, this did not pose a problem.

6.2 Horus and the Communications Infrastructure

There were several low level services that were necessary for the implementation of the private message service. These included such things as a directory service, a group membership service, as well as message sending and multicast primitives. In our system, these basic operations were provided by the Horus system [vRHB94]. Horus is a highly flexible system for implementing reliable distributed systems based on the group communication model of computation. It is also extensively layered, allowing applications to only pay for those services that they actually use.

All communication in Horus takes place in the context of process groups. For each group, processes declare a protocol stack which specifies the semantics of the group and of communication within the group. Communication within the private message service and between the private message service and the users makes extensive use of the group communication model. As shown in figure 6.1, our implementation makes use of three different process groups: the Message Service

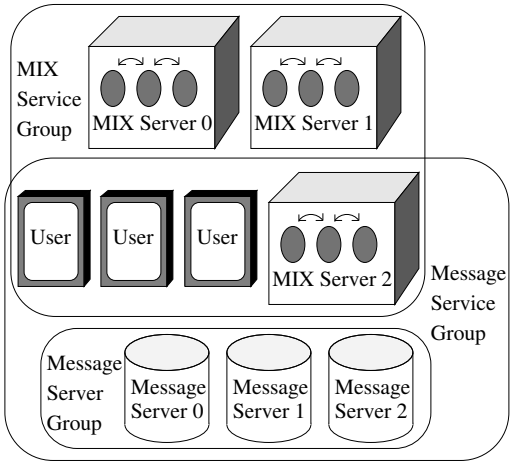

Figure 6.1: Group Structure

Group, the Message Server Group, and the MIX Service Group. The details of these groups and the processes within them will be explained later.

While there are a large number of protocol layers available in the Horus system, our system uses relatively few of them. Figure 6.2 shows the protocol stacks for each of the groups used within the system. At the bottom of every protocol stack is the COM layer. The semantics of this group are unreliable communication within groups whose views are maintained by the application. In all three of our groups, the NAK (negative acknowledgement) layer sits on top of the COM layer. The NAK layer simply adds reliable FIFO communication to the semantics of whatever protocol stack sits below it. On top of this sits the FRAG (fragmentation) layer which is responsible for breaking up messages into packets small enough for the underlying communications protocol (e.g. UDP, TCP/IP, ATM, etc.).

As was mentioned in chapter 4, in order to deal with server failures, it is necessary that the servers be able to maintain consistent views of the group membership. This is handled by the MBRSHIP (membership) layer (for a system model in which

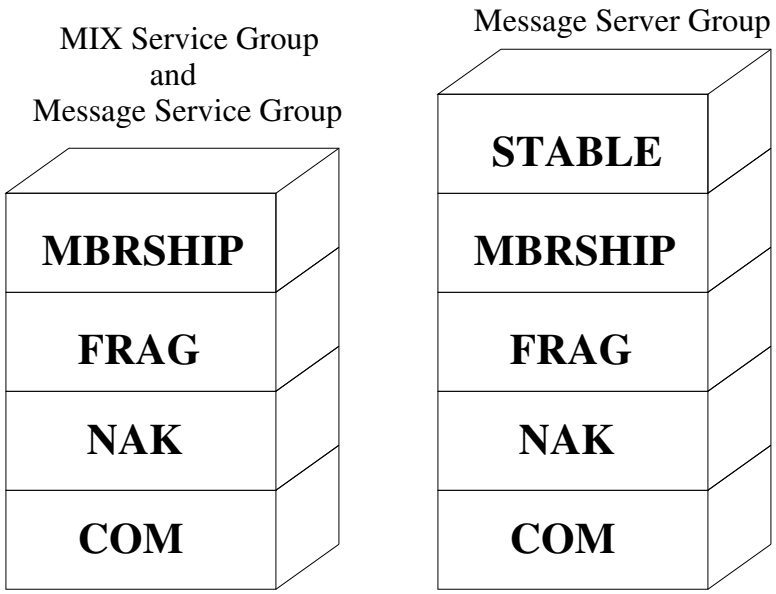

Figure 6.2: Horus Protocol Stacks

only crash failures can occur). The MBRSHIP layer implements the virtual synchrony model of computation. In addition to ensuring that all group members receive consistent group views, this layer also has a flush protocol which is run whenever a failure occurs or a new process joins the group. The flush protocol is designed to help ensure that the semantics of communications protocols are maintained despite changes in the group membership.

Unlike the other two groups, the Message Server Group uses an additional layer. The STABLE layer collects information about the stability of the messages that are sent within a group. This stability information is used for two reasons. First, as shown in figure 4.1, the user (or MIX) sending a message to the message service does not receive an acknowledgement for that message until it has been received (and acknowledged) by all of the message servers. Second, a table digest for a table in the message service is not multicast to the users until the message server

which multicasts the tables knows that all of the messages in that table have been received by all of the servers.

While not related to the communications infrastructure of the system, the final service which Horus provides is a machine independent threads package. In our system, user processes are implemented in layers, the message client layer and the applications layer (see figure 6.3). Currently, two applications layers have been implemented, one of which uses the X Window System. The main thread of execution runs the user interface (written in X) and allows users to create and send messages. Whenever an incoming message arrives (whether it is a table digest or a response to a request), a new thread is created which processes the message and then terminates.

6.3 The Message Servers

The message servers are responsible for both storing user messages and for handling requests from users wishing to retrieve messages. As a result, the message servers will, in general, be the bottleneck for system performance. With this in mind, we have carefully designed the system to minimize the costs of operations associated with the message servers.

There are three basic operations that are associated with the message servers. The first is receiving user messages. As was mentioned in section 3.6, in order for the message servers to maintain consistent states and for users to receive the correct answers to their requests, users' messages must be delivered at all of the message servers in the same order. Since all of the messages come from users (instead of from message servers), it is necessary, for efficiency purposes, to divide

the process of sending a user message to the message service into two steps. In the first step, the user (or the final MIX server in the MIX-network) sends the message to a single designated message server (Message Server 0) using the Message Service Group to send the message. This server, in turn, multicasts the message to the remaining message servers using the Message Server Group to multicast the message. Message Server 0 multicasts each of the messages that it receives in the same order in which it delivers the messages. Since the group in which it multicasts the messages includes the NAK layer in its protocol stack, the remaining message servers will deliver the messages in the same order as Message Server 0, thus ensuring consistency. As a result of this two step process, the cost of a general purpose totally ordered multicast is avoided and only the message servers will deliver the messages. (If the messages had been multicast within the Message Service Group, then all of the users would have delivered the messages in addition to the message servers.)

When a table has been filled, the digest for that table must be sent to all of the users. In our system, since users are actually static processes which are always connected to the network, this process is relatively simple. Once Message Server 0 has received information from the STABLE layer certifying that all of the messages for a table have been delivered by all of the message servers, it multicasts the digest within the Message Service Group. As was the case with user messages multicast within the Message Server Group, the NAK layer of the protocol stack ensures that every user receives exactly one copy of each digest and that the digests are received in order. In addition to the users, the table digests are also delivered by the other message servers. Each server keeps track of which digests have been multicast. If a

server ever receives notification (from the MBRSHIP layer) that Message Server 0 has crashed, then it will take over as Message Server 0 if all other lower numbered servers have also crashed. By keeping track of which digests have been broadcast, the new Message Server 0 can take over the responsibilities of the old Message Server 0 without causing any disruption for the users.

Once the users have received a table digest, they will read any messages from the table which may be intended for them. Messages are read using RPC style requests. A user wishing to read a cell in the table can do so in one of two ways. If the label attached to the message which the user believes it is reading is private and the user trusts the sender associated with that label, then the user chooses a single server at random and sends a request to that server for the message in the appropriate cell of the table. The user then waits for the response, checks the label on the message that it receives, and, if the label is correct, decrypts the message. If the message server to which the user sends the request crashes before sending its response, then the user simply chooses another server and resends its request.

If the user believes that it is reading a message with its public label or with a private label from a user that it does not trust, then the user will perform a blinded read operation. For this, it will choose $t + 1$ servers at random, create $t + 1$ bit-vectors, and encrypt one bit-vector for each server. The encryption for each server is performed by creating a new secret DES key [DES77], encrypting the DES key with the public RSA key [RSA78] of the server, and encrypting the bit-vector with the DES key [1]. Each encrypted request is sent to the appropriate server and the user waits for all of the responses. As in the case above, if one of the servers crashes

[1]The basic RSA and DES encryption, decryption and key generation functions were provided by the RSAREF(TM) software package from RSA Laboratories.

before responding, the bit-vector which was sent to that server will be re-encrypted for a new server and sent to that server. Once responses to all of the bit-vectors have arrived, they are decrypted (the message servers encrypt responses to blinded read requests using the DES key sent along with the bit-vector) and combined to get the final response. Once the response has been computed, the label is checked and, if it is correct, the message decrypted (just as above).

6.4 The MIX Servers

Messages intended for users who are not trusted by the message senders can be sent through the MIX-network. In the current implementation, the MIX servers form a single chain beginning with MIX Server 0 and ending with MIX Server $t+1$. A user wishing to send a message through the MIX service encrypts the message for each of the servers (as described in section 3.4) and then sends the message to MIX server 0.

As the first MIX in the chain, MIX Server 0 is responsible for batching messages. Messages, as they arrive, are placed in the first available batch with the restriction that in each batch there is at most one message from any given user [2]. This is accomplished by placing each incoming message (after being decrypted) onto a queue along with its sender's identity (or location). Then, once the message is in the queue, the queue is scanned for any messages which can be placed in the current batch. Once a batch has been filled, the messages in the batch are sorted [3] and then sent to MIX Server 1.

[2] In a real system, in which it may not be possible to determine whether or not two messages came from the same user, this could be approximated by limiting each batch to one message from each base station.

[3] Since the sorting is based on the decrypted form of the message, this has the same effect as reordering the messages randomly.

There are two reasons for restricting each batch to having at most one message from each user. First, allowing a single user to place a large number of messages in a single batch poses a privacy risk. Since the uncertainty in the origin of any message that exits the MIX-network is limited by the number of different origins of messages at the entrance to the MIX-network, allowing more than one message in a batch to originate from the same location would reduce the uncertainty in the origins of all of the messages from that batch once they exit the MIX-network. The second reason for the limitation is one of convenience. In order to simplify the implementation for the users, the private message service guarantees FIFO message delivery. Since messages within a single batch are reordered and since the destination of each message is unknown to MIX Server 0, the only way to maintain this guarantee is to limit each user to at most one message per batch.

Since the job of batching is handled by MIX Server 0, the remaining MIXes only need to read in each batch of messages, decrypt the messages in each batch, sort the messages, and then send them to the next MIX in the chain. MIX Server $t+1$ performs the same job as the other MIXes with the exception that it sends the messages to Message Server 0. This is the reason that MIX Server $t+1$ belongs to the Message Service Group in addition to the MIX Service Group.

As can be seen from figure 6.1, the MIX Service Group contains all of the MIX servers in addition to all of the users. Since the protocol stack for the MIX Service Group contains the MBRSHIP layer, all of the members in the group are aware of the current status of each of the MIX servers. Using this knowledge, MIX servers which have crashed can be removed from the chain in the MIX-network until they have recovered. Removing a MIX requires two steps. First, users will no longer

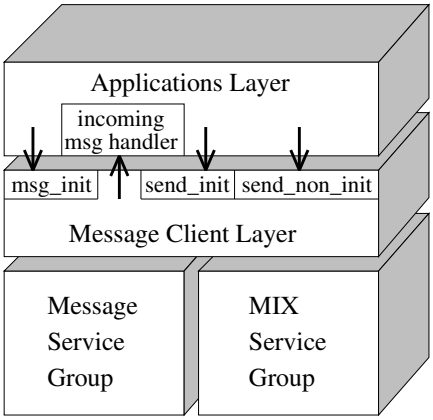

Figure 6.3: The user process

encrypt their messages for that MIX. If MIX Server 0 crashes, then the next lowest numbered MIX will take over responsibility for receiving messages from users and for batching the messages. Similarly, if the last MIX in the chain crashes, then the highest numbered MIX remaining will have to join the Message Service Group and start sending its messages to Message Server 0. Finally, if the crashed MIX is in the middle of the chain, then the preceding MIX will simply send its messages directly to the MIX that followed the crashed MIX in the chain.

6.5 The Users

The code for the user processes is divided into two layers which communicate through a simple interface (see figure 6.3). The two layers are the message client layer and the applications layer.

6.5.1 The Message Client Layer

In this section, we will describe the message client layer, including its interface with the applications layer. The message client layer implements the user's portion of

the protocols for the private message service. The code can be divided into three basic parts: initialization, sending messages, and retrieving messages.

A process begins by calling `msg_init` with the user's public label and private key and the address of a function to be called whenever an initiation message arrives. The message client layer joins the Message Service Group and the MIX Service Group. It also stores the user's public label and private key for use whenever a table digest arrives.

Once `msg_init` has been called, the user is free to send and retrieve messages. The user can send an initiation message by calling the function `send_init`. The parameters to this function are the intended recipient's public label and public key, the message to be sent, and the address of a function to call when a reply message arrives. The message client layer generates a new session key for the conversation as well as a return "address" label for the first reply message. It then encrypts the session key with the public key of the intended recipient, attaches the return "address" label to the front of the message, and encrypts the message (along with the return "address" label) with the session key. Next, it attaches the encrypted session key and the intended recipient's public label to the front of the message. Finally, the return "address" label and the session key are placed on a list of active conversations which is used to scan table digests and to read and decrypt incoming messages.

While the function `send_init` accepts arbitrary length messages, the private message service (for privacy reasons) requires that all messages be of the same length. Therefore, the message client layer is responsible for fragmenting and/or padding messages so that each message (or packet) sent to the message service is

the same length. In order to accomplish this, the message client layer breaks up messages into fixed length pieces and uses randomly generated labels to form a 0 terminated linked list of the packets which constitute the original message. If necessary, random padding is added to the final packet. In order to prevent an attacker from learning the lengths of messages, the labels which form the next packet "pointers" are encrypted with the session key.

The message client layer can send the final, encrypted packets in one of two ways. If the user sets the `secure` flag (one of the parameters to `send_init`), then each packet is encrypted for the MIX-network and sent (within the MIX Service Group) to MIX Server 0. If the `secure` flag is not set, then the packets are sent directly to Message Server 0.

When a table digest arrives, the message client layer checks each value in the table $(h(l_1), h(l_2), \ldots, h(l_{m-1}))$ with each label in its list of active conversations (plus the user's public label). If one of the values in the table matches one of the labels in the user's list (i.e. $h(l_i) = h(l'_j)$), then the message client layer reads in the appropriate message. Just as was the case with sending a message, there are two ways to retrieve a message. In order to decide whether to use a blinded read or an unblinded read, the message client layer must determine whether or not the user trusts the message sender. If $h(l_i) = h(l_{pub})$, where l_{pub} is the user's public label, then the sender is unknown and therefore not trusted. If $h(l_i) = h(l'_j)$ for some return "address" label, l'_j, then the message client layer examines the way in which the previous outgoing message was sent to this user. If the user set the `secure` flag when sending the message, then it will be determined that the sender of the reply message is not trusted. Otherwise, it will be assumed that the sender

of the reply message is trusted. If more than one label matches a single hash value (i.e. $h(l_i) = h(l'_j) = h(l'_k)$), then the blinded read is used if any of the matching labels are associated with an untrusted sender.

Once the message has been retrieved (using the appropriate technique), the actual label attached to the message is read to determine to which (if any) active conversation the message belongs. Just as the message client layer is responsible for fragmenting and/or padding messages when asked to send messages, it is also responsible for reassembling the packets and removing padding. Therefore, a received message may be a packet which belongs at the beginning, middle, or end of a list of packets which form a longer message. Once the packet is retrieved, it is decrypted using the appropriate session key (or the user's private key is used to extract a session key and that key is used to decrypt the rest of the packet). If the packet is the first of a message, then space is allocated for the message and the contents of the packet are placed at the beginning. If it is not the first packet, then its contents are appended to the portion which has already arrived. After the contents of the current packet have been handled, the linked list "pointer" is examined. If the "pointer" is 0, then this is the final packet: the padding is removed from the packet, the information associated with the conversation is removed from the list of active conversations, and the message along with the return "address" label for the message and the session key are passed up to the applications layer by calling the function specified at the time that the outgoing message was sent (or at initialization if this is an initiation message). If the "pointer" label is not 0, then the current label for this conversation in the list of active conversations

is replaced by the "pointer" label and the message client layer waits for the next packet to arrive.

Given the return "address" label and the session key that the message client layer passes up to the applications layer along with any incoming message, the user can send a reply message to the message's sender. This is done using the function `send_non_init`. This function works in basically the same way as `send_init` except that it does not need to generate or encrypt a session key.

6.5.2 The Applications Layer

The code for the user processes was designed in such a way to enable multiple applications layers to be easily built on top of the message client layer. Currently, we have implemented two versions of the applications layer. The first of which provides an X Windows interface for creating, sending, and reading messages and the second of which is used for performance testing. In this section, will be describe the X Windows version.

The X Windows version of the applications layer has three basic parts. The first is a directory of the current users in the system. Whenever a user joins the system, the user's name and public key are loaded and a public label is generated. These pieces of information are then sent off to the directory service so that they can be accessed by other users. Next, the applications layer reads in the names, public labels, and public keys of all of the users who are registered in the directory service. This information is then placed in a window for access by the user. Since other users may join the system later, the directory service is periodically scanned for new members. Figure 6.4 shows a sample window with two users.

Since the directory service is the source of public labels and keys for all of the

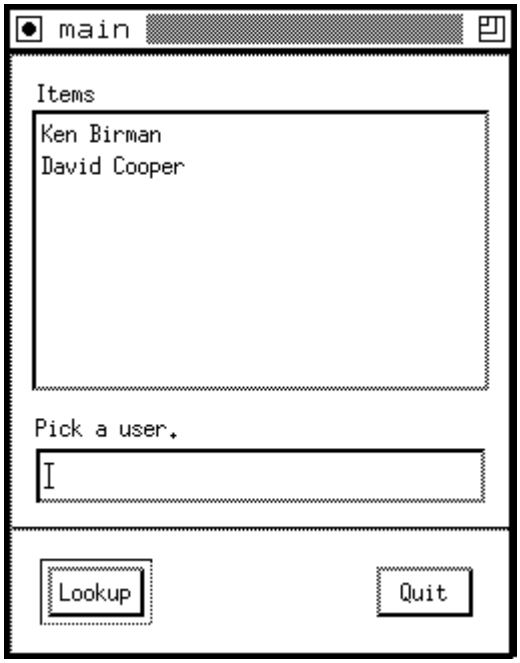

Figure 6.4: The Directory of Users

users in the system, the security of this service is vital to the security of the entire system. An attacker which is able to corrupt the directory service would be able to impersonate any user in the system thus allowing it to receive and read private mail intended for other users.

By choosing one of the names in the directory window and pushing the "Lookup" button, the user will cause an editing window to appear (as shown in figure 6.5). Using this window, the user can either type in the text of a message or load in the contents of a file to send. As is shown in figure 6.5, the name of the intended recipient of the message being created is displayed in the upper left corner of the window. When the window is created, the label and the key of the intended recipient are are included among the attributes of the window and so the window can only be used to send a message to that particular user.

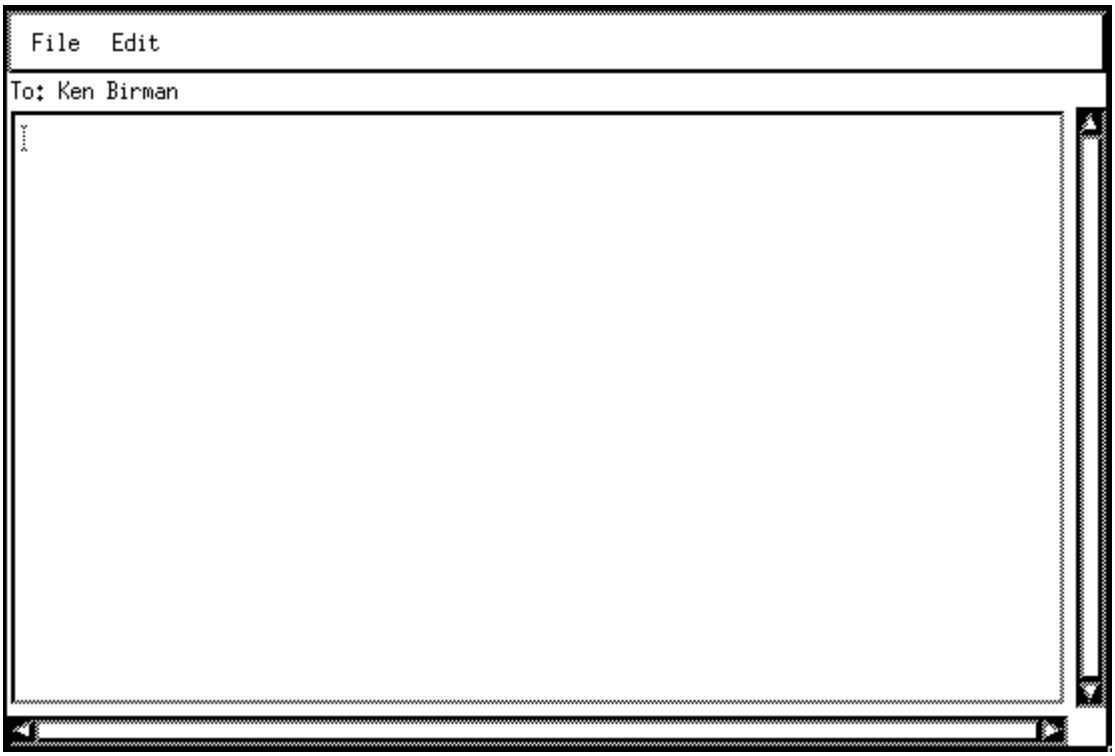

Figure 6.5: The Send Window

Once the message has been created, the user can send it to the message service. As was described in section 6.5.1, the message can either be sent directly or through the MIX-network by either setting or not setting the `secure` flag. In the X Windows interface, the user makes the choice by selecting the "Send Secure" or "Send Insecure" option in the "File" menu (see figure 6.6). As was described in section 6.5.1, the method by which the user chooses to send the message will also determine the method by which any reply message will be retrieved.

If, after opening a send window, the user decides not to send a message to this particular user, the send window can be closed by selecting the "Quit" option. The send window will also close after the user has sent the message.

There is one function that is used to handle all incoming messages (and thus

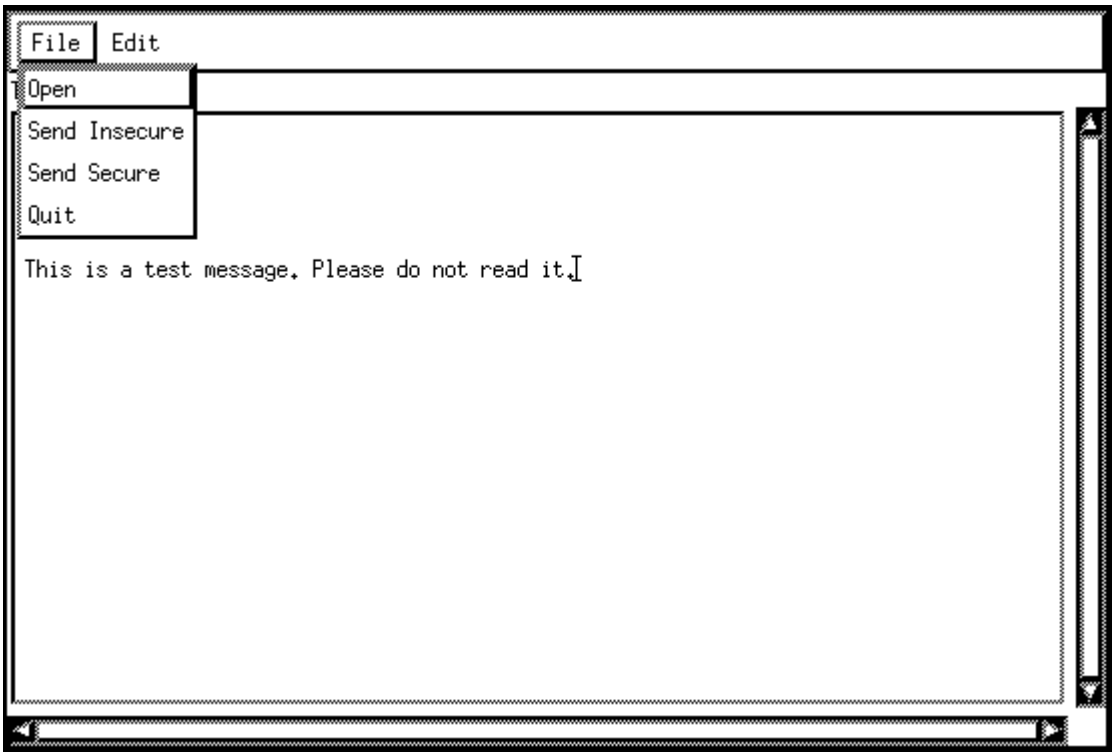

Figure 6.6: Sending a Message

the address of this function is passed in whenever a message is sent and when `msg_init` is called). This function creates a new window which is used to display incoming messages (see figure 6.7). The body of the window uses the same editing window (with scroll bars) as the send window (figure 6.5) except that a flag is set to prevent the user from modifying the contents of the window.

In addition to displaying the contents of the message and the identity of the message sender, the incoming message window has two buttons. The "Reply" button is used to send a reply message. When this button is pressed, a new send window is opened with the senders identity in the "To" field and the sender's return "address" label and session key stored as attributes. In addition, a copy of the original message is placed in the edit window with "> " placed at the

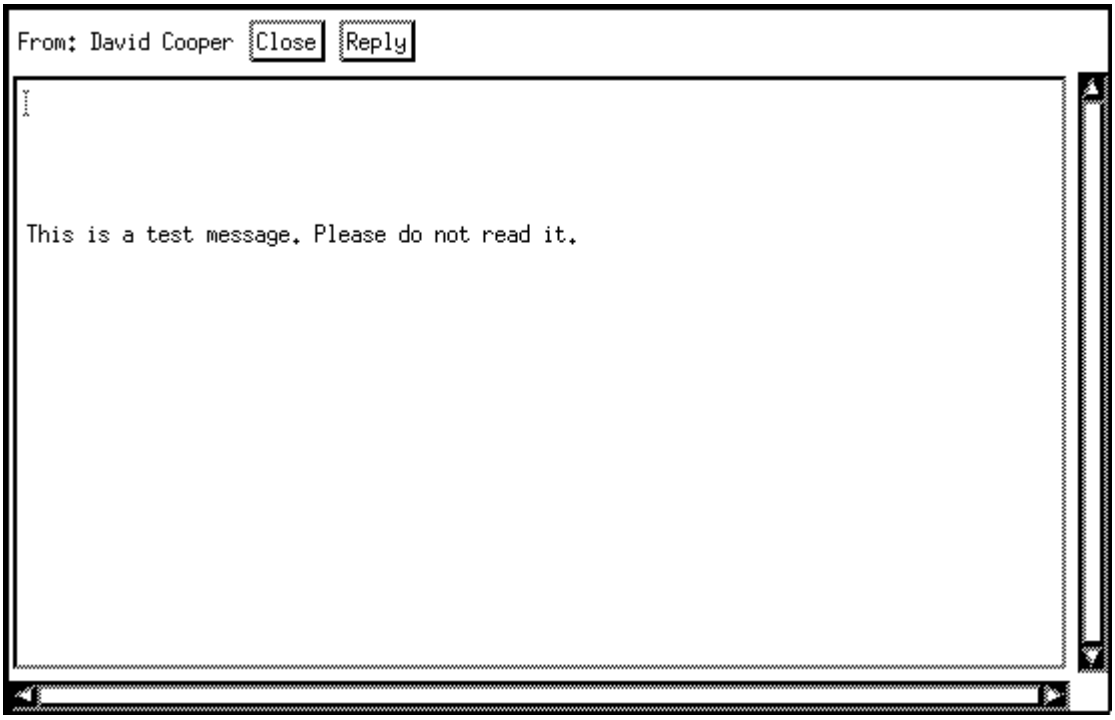

Figure 6.7: An Incoming Message

beginning of each line. Since only one reply message can be sent using any given return "address" label, the "Reply" button disappears from the incoming message window once it has been pressed. Once the user is done with the incoming message window, the window can be closed by pressing the "Quit" button.

6.6 The Authentication Service

The protection of the authentication service is one of the most important security concerns in the system. The authentication service is responsible for disseminating the public keys of all of the servers as well as the public keys and labels of all of the users. As a result, an attacker which is able to compromise the authentication service could disseminate false keys allowing it to impersonate any process in the system. Since the privacy guarantees of the system are based on the as-

sumption that at least some of the servers in the system are honest, an attacker, by impersonating all of the servers, would have access to any private information that it wanted. In addition to impersonating servers in order to determine who is talking to whom and where each user is located, the attacker could corrupt the conversations themselves. Using the appropriate means, the attacker could either impersonate one of the users and send false information to the other user in a conversation or could silently sit between the two users and undetectably eavesdrop on the conversation.

Authentication services are generally either based on public or secret key cryptography. When using public key cryptography, the authentication service creates certificates for users. A certificate for a user (or server) p contains p's identity and p's public key (along with any other information, such as a public label, which is to be certified by the authentication service) and is signed by the authentication service. Anyone knowing the public key of the authentication service can verify the signature on the certificate and will know that the information contained in the certificate has been certified by the authentication service.

In addition to creating certificates, the authentication service may occasionally need to revoke a certificate. In a large system with many users carrying mobile computers wherever they go, there will inevitably be users who will lose their computers or have them stolen. Since having access to a user's computer will give an attacker access to that user's private key and knowledge of a user's private key is considered to be proof of identity, the attacker will be able to impersonate the owner of the stolen computer. In order to prevent this there must be a way to inform others that knowledge of the private key should no longer be considered

proof of identity. This is accomplished by revoking the certificate which originally connected the user to the user's public key. (This is similar to the process used when someone reports a credit card as stolen).

One way to revoke a certificate is to explicitly send out a message declaring that the certificate is no longer valid. However, this method can be thwarted by an attacker by preventing the revocation message from being received by other users. An alternative to have users assume that a certificate has been revoked unless it receives a message from the authentication service "refreshing" the certificate (similar to using timeouts to detect process failures). In our system, the authentication service provides users with certificates which have expiration times where the expiration time is set to be δ time units after the generation of the certificate. If a user (or server) ever reports its private key as stolen, then the authentication server will immediately stop producing new certificates with that user/key pair. Given the expiration times of outstanding certificates, an attacker who has stolen a private key will be able to use it for at most δ time units after the key is reported as stolen.

Since every valid certificate must be periodically refreshed by the authentication service, it is important that the service be available at all times. Since the authentication service will run for an extended period of time and crash failures are possible, the service should be replicated so that the service will not become unavailable as a result of one (or a few) crash failures. While using a large number of servers increases availability, it can also increase the likelihood of a server (or servers) becoming corrupted by an attacker since it is harder to protect several machines than it is to protect just one.

A solution which allows for a compromise between these competing interests is a threshold scheme. In a (k, n)-threshold scheme, there are n servers and any group of k servers can create a certificate while no group of fewer than k servers can do so. Increasing the value of k increases the services protection from corruption, while decreasing k increases the services ability to remain available despite crash failures.

The authentication service in our system is based on the protocol designed by Reiter in [Rei93] which uses the threshold signature scheme of [DF90,FD92]. In this scheme, a certificate is signed using a single RSA private key. The private key is "divided" among n servers in such a way that k servers are needed to create a valid signature. The primary advantage of this scheme over other schemes is that there is a single RSA public key which is used to verify all certificates independent of which servers were used to create the signatures for the certificates.

6.7 The Time Service

There are several places in the system in which protocols are executed which make use of time. In the MIX-network, each MIX must ensure that all of the messages it accepts are fresh since message replay is a form of attack against the MIX-network. Return "address" labels may have expiration times which must be understood by both the potential sender and recipient of any messages which would use the labels. In addition, time is used to guarantee the freshness of certificates from the authentication service.

All of the situations above require that the clocks of two or more computers be at least partially synchronized. Therefore some form of clock synchronization

mechanism is necessary. Since the inability of computers to synchronize their clocks could lead to serious problems [Gon92], it is important that the mechanism be secure. As with the authentication service, the easiest way to ensure security is to have a single, centralized server. Clock synchronization across the system would be achieved by having computers periodically synchronize their clocks with the clock of the server. While the use of a single server may lead to periods of time in which the time service is unavailable, computers should be able to go for long periods of time without synchronizing their clocks while still maintaining a reasonable level of synchrony [Rei93]. As a result, the unavailability of the time service for the period of time necessary to reboot a crashed server should be acceptable.

In our system, we use the clock synchronization protocol of [Rei93]. In this system, as was described in section 3.7, each computer maintains a range of possible times. The range of times is designed to be the smallest possible range which is guaranteed to include the actual time (the time according to the time server). Computers use their low estimates of the current time to generate timestamps and use their high estimates to verify the freshness of messages. If the time server ever becomes unavailable for a period of time, these high and low estimates will slowly drift apart. In order to handle a very long period of unavailability, the computers can simply increase the lifetimes of "fresh" messages. Message lifetimes should, however, be kept as small as possible for security reasons.

6.8 Performance

In order to verify the feasibility of our protocols, we ran a series of tests to determine the costs associated with the different operations that the system must perform.

Since latency is not a primary concern in a message system, we concentrated on measuring the throughput of the servers. Each of the tests involved a single user sending a series of 10 messages to itself in which each message was large enough to fill 128 packets (which was the size of a table in the message servers). Each packet was 1 KByte long.

In our system, the encryption and decryption functions are performed by the RSAREF(TM) software package from RSA Laboratories. Using this system, the costs of the cryptographic operations dominate the overall cost of the system. However, in a large scale implementation, it is likely that cryptographic hardware will be used to improve performance (especially in the servers). In order to account for this, we will present our performance numbers in two ways. In the first section, we will present the actual performance figures from our system. In the second section, we will present estimates of the performance of a system using cryptographic hardware.

6.8.1 Actual Performance

Since the message service is responsible for both receiving packets and answering requests from users attempting to retrieve packets, it is the bottleneck in the system. In our experiments, the time spent retrieving packets was significantly greater than the time spent sending the packets to the message service. We will, therefore, concentrate on the different costs of the two methods in which packets can be retrieved from the message service.

The simplest test case involved sending a series of non-initiation messages directly to the message servers and receiving those messages using a regular read operation. In this case, the only encryption involved was the end-to-end, DES,

encryption of the messages. In this test, a throughput of about 29 packets per second was achieved.

The next test was the same as the first with the exception that the packets were read from the message service using the blinded read operation instead. This affected performance in two ways. First, in responded to a blinded read request, a server must compute the response (by exclusive-oring together several packets) instead of simply looking up the answer. Second, with the blinded read technique the requests and replies are encrypted whereas no encryption is used with the other technique. The encryption/decryption involves the creation of a DES key, its encryption with an RSA public key and decryption with an RSA private key, and DES encryption and decryption of the bit-vectors and responses.

Using the blinded read, the throughput of the system was reduced to 1 packet per second. The majority of the time to perform the blinded read operation was spent on the RSA decryption operation (0.7 seconds per decryption - 1 decryption per packet). Most of the remaining time, 0.17 seconds, was spent using DES to encrypt and decrypt the bit-vectors and responses. Since each table had 128 entries and about half the bits were set in each bit-vector, computing a response involved exclusive-oring together approximately 64 packets. Despite this, only about 0.02 seconds was spent by each server computing the response to each request. Since the message servers compute responses to requests in parallel with no interprocess communication necessary, adding extra message servers and increasing the value of t (the maximum number of corrupted servers, or one less than the number of bit-vectors used to retrieve a message) did not significantly effect the performance of the system.

In addition to testing the throughput of the message servers, we also examined the effect of the MIX-network on the overall throughput of the system. Since each MIX server must decrypt each incoming packet and this involves performing an RSA decryption in order to extract the DES key, the throughput of the MIX-network was only 1 packet per second. However, since the MIX-network is a chain and each MIX operates independently, the throughput was not affected by the number of MIXes used. Adding more MIXes to the MIX-network only increased the latency between when a packet was sent and when it arrived at the message servers.

The impact of the MIX-network on the overall throughput of the system depended on whether packets were being read using the blinded read operation or not. Since, using the blinded read operation, packets could only be read from the message servers at a rate of 1 packet per second and the MIX-network was able to provide messages to the message servers at the same rate, the use of the MIX-network did not impact the throughput of the system. However, when the blinded read operation was not used, the message service was able to handle 29 packets per second. Since, as a result of the MIX-network, the message service only received packets at a rate of 1 packet per second, the total throughput of the system was limited to this rate as well. Therefore, the MIX-network was the limiting factor in the system.

6.8.2 Expected Performance

In addition to the tests above, we also ran a series of tests with the cryptographic operations removed. By comparing the results of these tests with the results of the previous tests and making use of information about the costs of cryptographic

operations using hardware, we were able to estimate the throughput of a system in which the servers use cryptographic hardware.

In the first test above, where the packets were sent directly to the message servers and were retrieved without using the blinded read operations, the servers did not have to perform any encryption. However, since the user was performing end-to-end encryption/decryption, the overall throughput of the system was affected by the encryption operations. Since, in a real system with multiple users, the computation effort of the users will not effect the overall throughput, the throughput of the system without end-to-end encryption should be more accurate. In this test, the throughput was 38 packets per second.

When using the blinded read operation, the throughput of the messages servers is reduced since the responses to requests must be computed. The throughput in this test was 27 packets per second. In order to get a most accurate estimate, however, we must factor in the costs of the cryptographic operations. First, an RSA decryption is performed to extract the DES key. While there are several hardware implementations of RSA [Bri90], the fastest one currently available can perform 512-bit RSA decryption in just under 1 ms [SV93]. Once the DES key has been extracted, the bit-vector must be decrypted. Using a DES chip, both encryption and decryption are very fast [DK90]. The speed is somewhat reduced, however, for small blocks. Since the bit-vector is only 128 bits, the decryption rate will be about 1.25 Mbits/s. At this rate, however, the decryption will take only 0.1 ms. The final step, encrypting the response, involves encrypting a 1 KByte message. For large blocks, the encryption process works at 3.6 Mbits/s. So, the

time to encrypt is 2.3 ms. Combining these estimates with the actual execution time gives an estimated throughput of just under 25 packets per second.

With the decryption operations removed from the MIX-network, the MIXes were no longer slowed down by the cost of performing RSA decryption. As a result, sending messages through the MIX-network did not impact the overall throughput of the system even when the blinded read operation was not in use. Based on these measurements, it is our conjecture that using the MIX-network, with cryptographic hardware, will not impact the throughput of the system.

Chapter 7

Conclusions

In this thesis, we have considered the privacy problems that arise as a result of the use of mobile computers for communication. The mobility of the users introduces a new type of information, location, which needs to be kept from attackers in order to maintain privacy. As was shown in this thesis, it is possible to prevent an attacker from acquiring location information even if the attacker is able to corrupt some of the servers which are responsible for hiding this information.

The primary contribution of this thesis was the design of the message service which was presented in chapter 3. The message service makes use of a blinded read operation which allows users to read messages from the service without revealing which messages they are reading. The primary advantage of our protocol for blinded reads over protocols which solve similar problems is that it makes efficient use of both computation and bandwidth. In addition, most of the computation is performed by the servers which generally have more computational power than the mobile computers. As was shown in section 6.8, the throughput of the message service using the blinded read operation is reasonable.

One of the side effects of allowing users to read messages without revealing which messages they are reading is that it is impossible for the message service to determine when the intended recipient of a message has read the message. Since the amount of memory available to the system is limited, there needs to be some way for the system to be able to delete messages while minimizing the risk that the intended recipient of a message will be unable to retrieve that message as a result of being disconnected from the network for a period of time. In order to solve this, we developed the vacation service which was described in chapter 5. The vacation service, by limiting the amount of space that each user has available to store messages in the system, gives users an incentive to inform the vacation service when they have received messages. At the same time, the protocols for interacting with the vacation service were designed in such a way that users do not need to reveal any private information to the vacation service.

7.1 Future Work

Just as the thesis presents solutions to important privacy problems, it brings to light several new problems which suggest areas of future research. First, the current design of the message service limits the potential growth of the system. Since the service relies on a single set of servers which are responsible for handling every message in the system, the potential throughput of the entire system is limited by the potential throughput of this set of servers. In order to handle systems with greater throughput requirements, without loss of privacy, it will be necessary to design a more decentralized system.

As was mentioned earlier in the thesis, the current system can not handle

real-time message traffic such as telephone conversations. Telephone conversations complicate privacy preservation in two ways. First, by observing network traffic, an attacker may be able to determine when conversations begin and end. By matching this information with other known information, the attacker may be able to locate some users. A second problem lies in the MIXes and the message service. Privacy in these systems relies on the ability to batch messages. Since messages which represent voice data in a telephone conversation have maximum latency requirements, it may not always be possible to hold messages until enough other messages have arrived to fill a reasonably sized batch. The result would either be using small batches or violating latency requirements.

One aspect of security that was not considered in the thesis was the access control problem. In most cases, the owners of a network will wish to have some control over who is able to send and receive messages over the network. This access may be limited to a specific group of people (i.e. those who work for a particular company) or may be based on payment for use of the network. In either case, there will need to be some way for the network to be able to determine whether or not a message that it receives came from an authorized user without requiring that user to identify itself.

The blinded read operation of the memory service also opens the possibility of of solving some new privacy problems. One example is reading netnews. In many cases, someone reading netnews may not want others to know which newsgroups he or she is reading. Just as anonymous remailers are used to allow users to publish articles without revealing their identities, there should be a way for users to be able to read messages in privacy. While reading every message from every

newsgroup would certainly solve the problem, it is very inefficient. Just as the memory service was modified to handle message traffic, it should be possible to design a modified version of the memory service which can handle netnews postings and read requests.

Bibliography

[AB93] Arup Acharya and B.R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

[AD94] Ashar Aziz and Whitfield Diffie. Privacy and authentication for wireless local area networks. *IEEE Personal Communications*, 1(1):25–31, First Quarter 1994.

[BAI93] B.R. Badrinath, Arup Acharya, and Tomasz Imielinski. Impact of mobility on distributed computations. *Operating Systems Review*, 27(2):15–20, April 1993.

[BBS82] Lenore Blum, Manuel Blum, and Michael Shub. Comparison of two pseudo-random number generators. In *Advances in Cryptology: Crypto 82*, pages 61–78, August 1982.

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredectable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[BCR87] Gilles Brassard, Claude Crepeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology – CRYPTO '86*, pages 234–238, August 1987.

[BCY93] Michael J. Beller, Li-Fung Chang, and Yacov Yacobi. Privacy and authentication on a portable communications system. *IEEE Journal on Selected Areas in Communications*, 11(6):821–829, August 1993.

[Bir93] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53,103, December 1993.

[BIV92] B.R. Badrinath, Tomasz Imielinski, and Aashu Virmani. Locating stategies for personal communication networks. In *IEEE Globecom*

92 Workshop on networking of personal communications applications, 1992.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 112–117, November 1982.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[BP] Pravin Bhagwat and Charles E. Perkins. A mobile networking system based on Internet Protocol(IP). Technical report, Computer Science Department, University of Maryland and IBM T.J. Watson Research Center.

[Bra88] Gilles Brassard. Modern cryptology: A tutorial. In *Lecture Notes in Computer Science*, volume 325. Springer-Verlag, 1988.

[Bri90] Ernest F. Brickell. A survey of hardware implementations of RSA. In *Advances in Cryptology - CRYPTO '89*, pages 368–370, 1990.

[BSS91] Kenneth Birman, André Schiper, and Pat Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, August 1991.

[Bur83] David Burnham. *The Rise of the Computer State*. Random House, New York, 1983.

[Car94] Ulf Carlsen. Optimal privacy and authentication on a portable communications system. *Operating Systems Review*, 28(3):16–23, July 1994.

[CASD85] Flaviu Cristian, Houtan Aghili, Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. In *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, pages 200–206, June 1985.

[CB95a] David A. Cooper and Kenneth P. Birman. The design and implementation of a private message service for mobile computers. *Wireless Networks*, August 1995.

[CB95b] David A. Cooper and Kenneth P. Birman. Preserving privacy in a network of mobile computers. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 26–38, May 1995.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[Cha92] David Chaum. Achieving electronic privacy. *Scientific American*, 267(2):96–101, August 1992.

[CM84] Jo-Mei Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[DES77] National Bureau of Standards. *Data Encryption Standard, FIPS-PUB-46*, 1977.

[DF90] Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology - CRYPTO '89*, pages 307–315, 1990.

[DFM91] Daniel Duchamp, Steven K. Feiner, and Gerald Q. Maguire, Jr. Software technology for wireless mobile computing. *IEEE Network Magazine*, 5(6):12–18, November 1991.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DK90] Stephen R. Dussé and Burton S. Kaliski Jr. A cryptographic library for the Motorola DSP56000. In *Advances in Cryptology – EUROCRYPT '90*, pages 230–244, May 1990.

[FD92] Yair Frankel and Yvo G. Desmedt. Parallel reliable threshold multisignature. Technical Report TR-92-04-02, Department of E.E. and C.S., University of Wisconsin-Milwaukee, April 1992.

[FZ94] George H. Forman and John Zahorjan. The challenges of mobile computing. Technical Report 93-11-03, Computer Science & Engineering, University of Washington, March 1994.

[GII76] Interception of nonverbal communications by federal intelligence agencies. U.S. Government Printing Office, Washington, 1976.

[Gon92] Li Gong. A security risk of depending on synchronized clocks. *Operating Systems Review*, 26(1):49–53, January 1992.

[Hen92] Nat Hentoff. *Free Speech for Me – But Not for Thee: How the American Left and Right Relentlessly Censor Each Other*. HarperCollins, New York, 1992.

[IB93a] Tomasz Imielinski and B.R. Badrinath. Data management for mobile computing. *SIGMOD Record*, 22(1):34–39, March 1993.

[IB93b] Tomasz Imielinski and B.R. Badrinath. Mobile wireless computing: Solutions and challenges in data management. Technical report, WINLAB/Rutgers Tech. Rept., February 1993.

[IB93c] Tomasz Imielinski and B.R. Badrinath. Querying locations in wireless environments. *Wireless Communications: Future Directions*, pages 85–108, 1993.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstract). In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 12–24, May 1989.

[Joh93] David B. Johnson. Mobile host internetworking using IP loose source routing. Technical Report CMU-CS-93-128, School of Computer Science, Carnegie Mellon University, February 1993.

[Kat94] Randy H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, First Quarter 1994.

[Per] Charles E. Perkins. Providing continuous network access to mobile hosts using TCP/IP. Technical report, IBM T.J. Watson Research Center.

[Pfi94] Birgit Pfitzmann. Breaking an efficient anonymous channel. In *Advances in Cryptology – EUROCRYPT '94*, pages 339–348, May 1994.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology – EUROCRYPT '93*, pages 248–259, May 1993.

[PP89] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct RSA-implementation of mixes. In *Advances in Cryptology – EUROCRYPT '89*, pages 373–381, April 1989.

[PPW91] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. ISDN-MIXes: Untraceable communications with very small bandwidth overhead. In *Proceedings of the IFIP TC11 Seventh International Conference on Information Security: Creating Confidence in Information Processing, IFIP/Sec '91*, pages 245–258, May 1991.

[PSS95] Ravi Prakash, Niranjan G. Shivaratri, and Mukesh Singhal. Distributed dynamic channel allocation for mobile computing. In *Proceedings of the Fourteenth ACM Annual Symposium on Principles of Distributed Computing*, August 1995.

[PW87] Andreas Pfitzmann and Michael Waidner. Networks without user observability. In *Computers & Security 6*, pages 158–166, 1987.

[RBvR94] Michael K. Reiter, Kenneth P. Birman, and Robbert van Renesse. A security architecture for fault-tolerant systems. *ACM Transactions on Computer Systems*, 12(4):340–371, November 1994.

[Rei93] Michael K. Reiter. *A Security Architecture for Fault-Tolerant Systems.* Ph.D. dissertation, Cornell University, July 1993.

[Rei94a] Michael K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, November 1994.

[Rei94b] Michael K. Reiter. A secure group membership protocol. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 176–189, May 1994.

[Rek] Yakov Rekhter. An architecture for transport layer transparent support for mobility. Technical report, IBM T.J. Watson Research Center.

[Ric92] Aleta Marie Ricciardi. *The Group Membership Problem in Asynchronous Systems.* Ph.D. dissertation, Cornell University, November 1992.

[RS93] Charles Rackoff and Daniel R. Simon. Cryptographic defense against traffic analysis. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 672–681, May 1993.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[SD91] Bill N. Schilit and Dan Duchamp. Adaptive remote paging for mobile computers. Technical Report TR CUCS-004-91, Department of Computer Science, Columbia University, February 1991.

[Sha81] Adi Shamir. On the generation of cryptographically strong pseudo-random sequences. In *8th International Colloquium on Automata, Languages and Programming*, pages 544–550, July 1981.

[Sim92] Gustavus J. Simmons. *Contemporary Cryptology: The Science of Information Integrity.* IEEE Press, 1992.

[SV93] M. Shand and J. Vuillemin. Fast implementations of RSA cryptography. In *1993 IEEE 11th Symposium on Computer Architecture*, pages 252–259, 1993.

[SWL90] B. Simons, J. Lundelius Welch, and N. Lynch. An overview of clock synchronization. In *Fault-Tolerant Distributed Computing*, pages 84–96, 1990.

[TT93] Fumio Teraoka and Mario Tokoro. Host migration transparency in IP networks: The VIP approach. *ACM Computer Communication Review*, 23(1):45–65, January 1993.

[vRHB94] Robbert van Renesse, Takako M. Hickey, and Kenneth P. Birman. Design and performance of Horus: A lightweight group communications system. Technical Report TR 94-1442, Cornell University, August 1994.

[VV83] Umesh V. Vazirani and Vijay V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *24rd IEEE Symposium on Foundations of Computer Science*, pages 23–33, 1983.

[WHFaG92] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[Wil95] Uwe G. Wilhelm. An architecture for more privacy in mobile communication systems. Technical report, Département d'Informatique, Ecole Polytechnique Fédérale de Lausanne, 1995.

[YW93] Tak-Shing Peter Yum and Wing-Shing Wong. Hot-spot traffic relief in cellular systems. *IEEE Journal on Selected Areas in Communications*, 11(6):934–939, August 1993.

[ZY89] Ming Zhang and Tak-Shing Peter Yum. Comparisons of channel-assignment strategies in cellular mobile telephone systems. *IEEE Transactions on Vehicular Technology*, 38(4):211–215, November 1989.

[ZY91] Ming Zhang and Tak-Shing Peter Yum. The nonuniform compact pattern allocation algorithm for cellular mobile systems. *IEEE Transactions on Vehicular Technology*, 40(2):387–391, May 1991.