Preserving Privacy in a Network of Mobile Computers

David A. Cooper and Kenneth P. Birman*

Dept. of Computer Science
Cornell University

Abstract

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 paper we will describe a set of protocols that we have developed to allow a user with a mobile computer to communicate without compromising privacy.

1 Introduction

In some cases, the sole purpose for carrying a mobile computer is to allow others to quickly locate the carrier (e.g. the active badge system in [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. Moreover, it is common for users of communications networks to desire privacy. The focus of our work is on providing mechanisms for preserving privacy while permitting mobile communication.

Our approach centers around a *message repository* (called the message service). The message service acts like a global mailbox to which all messages intended for any mobile computer are sent. For both sending messages to, and reading messages from, the service, we will present two modes of interaction, one more secure than the other. The protocol for sending messages securely is based on a protocol by David Chaum [Cha81] and is used to hide the location from which messages are sent. Another protocol, developed by the authors, allows a computer to read a message from the message service without anyone being able to determine which message is being read. The message service and the protocols for interacting with it are depicted in figure 1.

2 Types of Privacy

In designing a system which preserves privacy for mobile users, we found it useful to partition privacy into three semi-independent components. We will refer to these components as content, participant, and location privacy.

*The work reported 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. E-mail addresses for the authors: dcooper@cs.cornell.edu and ken@cs.cornell.edu.

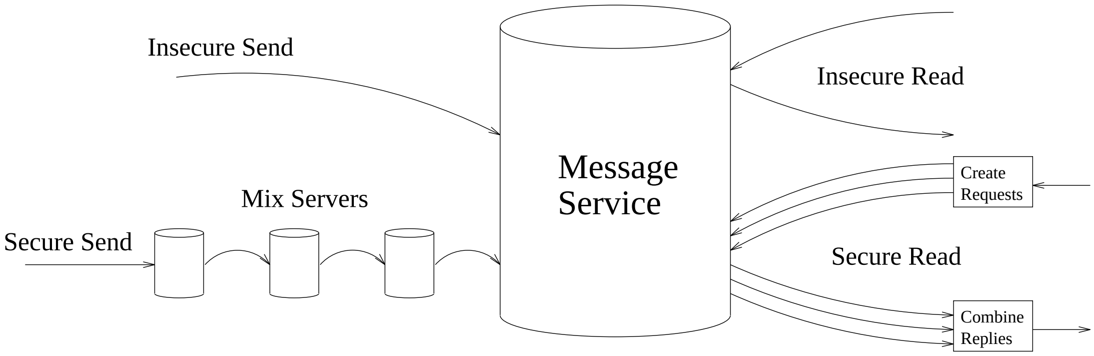

Figure 1: The Message Service

2.1 Content Privacy

A message is a piece of information that is transferred from a sender, s, to a recipient, r. Every message, m_i, that is sent in the system can be considered to have two "faces", a data "face", d_i, and a cipher "face", c_i. d_i is the information that is being transferred from s to r and c_i is the enciphered message that is actually sent over the communications medium. Message m_i, sent from s to r, has content privacy if an attacker has only a negligible chance of guessing d_i from c_i without the collusion of either s or r[1].

2.2 Participant Privacy

Even if an attacker is unable to determine the contents of a conversation between two users, it may gain useful information just by determining that two users communicated with each other. As an example, an inside trader may use knowledge of a conversation between two CEOs as an indication of an impending merger. While not nearly as useful as having the contents of a conversation, an attacker can, in general, gain a lot of useful information just by knowing who is talking to whom and when.

For any two computers p and q, we will define the *conversation* between p and q to be the set of all messages sent from p to q and from q to p. We will say that a conversation between computers p and q has participant privacy if an attacker, having access to the enciphered version of every message in the conversation, is unable to determine the identities of both participants in the conversation without the collusion of either p or q.

As will be shown later, the key to implementing participant privacy is that the identity of only one of the participants in the conversation needs to be hidden.

2.3 Location Privacy

In a mobile system, mobile computers communicate by sending messages to a nearby *base station*. At any given time, a mobile computer is connected to the base station that is closest to it. Therefore, if an attacker can determine with which base station a user's computer is communicating at any time, it can electronically "stalk" the user. In order to prevent this, the communications protocols must be designed so that an attacker can not obtain this information even if the user is engaged in

[1]Throughout this paper, 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.

a conversation with the attacker. A set of protocols that achieves this is said to provide location privacy.

3 Related Work

Of the three types of privacy described above, content privacy has received the most attention [Bra88, DES77, DH76, DH79, RSA78, Sim92]. There are two basic types of encryption schemes, secret key and public key. In secret key encryption schemes, the encryption and decryption keys are the same. In public key encryption schemes, the encryption and decryption keys differ. Usually, the decryption key is held by a single owner while the encryption key is made widely available.

In [Cha81, Cha88], David Chaum presents techniques for implementing participant privacy in a network of static computers. In [Cha81], he uses a set of computers, called *mixes*, which are responsible for sending a message to its destination while hiding its source. As will be shown later, the techniques that he presented can also be used as a building block for implementing location privacy.

Recently, there has been some work on providing privacy in a mobile network [BCY93, AD94, Car94]. In these papers, it is assumed that the static network is secure. The goal of these papers is to create secure links between mobile computers and base stations while requiring as little work as possible for the mobile computers. Since the static network is considered to be secure, these papers also deal with the problem of authenticating mobile computers that attempt to use the network.

4 System Model

4.1 Mobile Computers

The system consists of a set of mobile computers which are assumed to be *anonymous by default.* By this we mean that if every mobile computer were to send and receive the same sequence of signals, then no attacker could determine the identity of any computer. In other words, the only information an attacker can use to determine the identity of a mobile computer is any information that it can infer from the sequence of messages that the mobile computer sends and receives. Using this assumption, we will be able to demonstrate that our system preserves privacy by showing that a mobile computer does not send or receive any messages that might provide an attacker with information about the computers identity.

In the beginning, mobile computers do not share any secret information. Any information that one computer has about another is publically known (such as information stored in a directory service). Computers may, however, use public key encryption techniques to share secret information once an initial contact has been made.

4.2 Base Stations

Mobile computers communicate by sending messages to, and receiving messages from, base stations[2]. A mobile computer is said to be in the *cell* of a base station if it is able to communicate with that base station[3]. A mobile computer sends a message in the same way as a static computer. It at-

[2] In a wireless network, a base station is a network router that has a wireless connection.

[3] In the future, when we refer to the location of a mobile computer, we will mean the cell of the base station with which it is connected.

taches a header to the message with the destination address and then forwards the message to the nearest base station which will route the message towards its destination.

When a mobile computer wishes to send a query for which it expects a response[4], 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[5].

4.3 The Network

The network consists of a set of static computers and a set of communications links. The static computers are the base stations, the routers, and any servers that will be discussed later in the paper.

Each static computer has a publically known address which is used by the routers to direct messages intended for it. It is the responsibility of the communications links and routers to transmit messages from one static computer to another. Messages intended for a mobile computer are sent to the appropriate base station which is responsible for broadcasting messages to all of the computers within its cell.

We assume that the communications links and the routers can be read by an attacker, but the attacker is unable to modify or delete any of the messages. Servers, on the other hand, are considered to be secure unless specified as being corrupt. An attacker can read messages that go into or come out of a correct server but does not have access to the contents of the server's memory.

Throughout this paper, we assume that servers, both correct and corrupt, do not crash or behave maliciously. While techniques have been developed for implementing services which can handle crash failures or a limited number of malicious failures, fault tolerance is beyond the scope of this paper. Techniques for creating fault tolerant services can be found in [Rei94].

4.4 The Message Service

The message service is used to send messages to mobile computers. Sending a message to a mobile computer involves two steps. First, the sender attaches a *label* to the message and sends it to the message service. Next, a computer wishing to receive messages polls the message service for messages with a specific label. The reason for this approach is twofold. First, it provides a means for computers to anonymously receive messages Second, it allows one computer to send a message to another computer even if the second computer is not currently connected to the network. As will be shown later, most of the message labels will be chosen randomly. Therefore, labels should be large enough so that the probability of two messages having the same label is acceptably small[6].

All messages must be of the same length. Messages longer than the standard length must be fragmented and then re-assembled by the recipient. The specific means of choosing labels for messages as well as the protocols for sending messages to, and retrieving messages from, the message service are vital to providing location and participant privacy and will be discussed in detail later.

[4] As will be shown later, this includes retrieving messages sent to it by other computers.

[5] 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 cells are neighbors of the cell from which the request originated in addition to the originating base station.

[6] If two messages, intended for different recipients, use the same label, then both recipients will read in both messages. While this will not affect security or privacy, it will affect performance.

5 Content Privacy

A conversation between two computers, p and q, begins with one of the computers, say p, sending an initiation message to the other computer. Since secret key encryption is, in general, much more efficient than public key encryption, messages sent between p and q should be encrypted using a secret key. However, since p and q do not share any secret information before the initiation message is sent, some form of key exchange must be used to generate the secret key. A simple solution is to have the initiator, p, generate a secret key, K_s, and encrypt K_s with q's public key, K_q. The initiation message can then be $K_q(K_s), K_s(m)$ where m is any information that p wishes to send to q.

The initiator, p, can look up q's public key in a trusted authentication service[7]. As long as p trusts the authentication service, it will know that q, and only q, will be able to read the initiation message. As written above, however, the initiation message provides q with no way of verifying that the initiation message came from p. In some cases, the initiating computer's user may wish to remain anonymous and the receiving computer's user may not be interested in authenticating the initiator. For example, one may wish to call an airline to find out about available flights. In this case, the airline may not be interested in authenticating the caller unless the caller tries to book a reservation.

In many cases, someone receiving an initiation message may not want to send any messages to the initiator unless the identity of the initiator has been authenticated. To accomplish this, the initiator may receive a certificate from the authentication service. The certificate will be a signature, by the authentication service, of p's public key (i.e. $\{id_p, K_p\}_{K_a^{-1}}$[8]). The initiator, p, can then send $K_q(K_s), K_s(id_p, K_p, \{id_p, K_p\}_{K_a^{-1}}, m, \{id_p, id_q, m\}_{K_p^{-1}})$ to q thus allowing q to authenticate p.

6 Participant Privacy

As was mentioned earlier, messages are sent to mobile computers by attaching a label, l, to the message such that the intended recipient will eventually use label l in a request to the message service. Labels can be divided into two basic categories. Label l is a *public* label for computer p if every computer knows that p will make a request to the message service for messages with label l at some time in the future. Label l is a *private* label if the number of computers that know p will request messages with label l is small (typically only p and one other computer will know about l).

As in the previous section, a conversation between two mobile computers, p and q, begins when one of the computers, say p, sends an initiation message to the other computer. Since p and q do not share any secret information before p sends the initiation message, the only (feasible) way for p to send a message to q is to attach a label which is a public label for q (which can be found in an authentication service along with q's public key). So, an initiation message (either the authenticated or unauthenticated version from the previous section) should be sent to q by attaching q's public label to the message and then sending the message to the message service.

As was mentioned previously, in order to maintain privacy we must show that each message that is sent or received does not compromise the privacy of the mobile computer. If p sends the unauthenticated version of the initiation message, then the message contains no information about p's identity. So, if p was anonymous before sending the message, then p will be anonymous after

[7] The authentication service only needs to be trusted to provide correct information. It does not need to maintain any secret information.

[8] The notation $\{m\}_{K_a^{-1}}$ signifies the signature of m using a's private key which is $K_a^{-1}(h(m))$ where h is a one-way hash function.

sending the message. If p sends the authenticated message, then q will know who sent the message. However, since all of the information about p in the initiation message is encrypted under K_s, no computer other than p or q can determine the identity of the message sender without the collusion of either p or q. So, if q is trustworthy, p can send the initiation message to the message service while still remaining anonymous.

Even if q is not trustworthy, p should be able to maintain location privacy. In order to do so, p must send the initiation message in such a way that not even q can determine the location from which p sent the message. A method for accomplishing this was developed by David Chaum in [Cha81] and will be discussed later.

In order for q to receive the initiation message from p, it must send a request to the message service for messages with label l_q (q's public label). Since l_q is public and it is likely that q will be the only computer interested in messages with label l_q, q must be careful in requesting messages with that label. Later, we will describe a technique for q to request messages with a specific label without any computer being able to determine which label is being requested. Using this technique, q will be able to remain anonymous.

In each message that is sent between p and q (including the initiation message), the sender can include an encrypted, randomly generated label to be used by the other computer as a return "address" label. If p and q trust each other, then, since all of the labels are private, each computer can send and receive messages to and from the message service without sacrificing privacy. If one of the computers does not trust the other computer, then that computer may use the techniques which will be described later to hide its location while sending and receiving messages.

If a source of truly random numbers is not available, then a pseudo-random number generator may be used as long as it is cryptographically strong and the seed is chosen in an unpredictable manner. In order to allow one computer in a conversation to send more than one message at a time to the other computer, each computer may send the seed to a cryptographically strong pseudo-random number generator (whose output sequence will be used as the sequence of labels) instead of a single random label.

7 Sending a Message

As was discussed in the previous section, there are two basic types of message labels, public and private. If a computer, q, will be sending a message with label l, and q is concerned that an attacker may know this, then we will say that label l is public with respect to q. Otherwise, we will say that l is private. It is important notice that the notion of public and private labels differs from the one described above and may not match reality. A computer may consider an address to be private when in fact an attacker knows about the address, or the computer could be concerned about an attacker that does not exist. The idea behind these definitions is that a computer sending a message to a public label will perform extra work to guard against an attacker.

If l is private with respect to q, then q can send a message with label l by placing l in the message's header and sending the message, via the local base station, to the message service. An attacker will be able to follow the message from the base station to the message service, but if l is truly private then it will gain no information about the identity of the sender.

If an attacker knows that q will send a message with label l then, using the technique above, the attacker can determine q's location. Therefore, if q believes that l may be public, it must send the message in such a way that the attacker will be unable to follow the path of the message from the base station to the message service. A technique for doing this was developed by David Chaum in [Cha81]. The basic idea behind his work is as follows.

The static network contains a set of w *mixes* each having its own public key pair. We assume that at most $t < w$ of the mixes will be corrupted[9]. In order to send a message, a mobile computer chooses $t+1$ of the mixes (call them $M_1, M_2, \ldots, M_{t+1}$) and encrypts the message as follows:

$$K_{M_1}(R_1, K_{M_2}(R_2, \ldots K_{M_{t+1}}(R_{t+1}, l, m, S) \ldots, M_3), M_2)$$

In the above equation, l is the message label, m is the encrypted message, $K_{M_1}, \ldots, K_{M_{t+1}}$ are public keys of $M_1, \ldots, M_{t+1}$, $R_1, \ldots, R_{t+1}$ are random values, and S is the address of a server.

Each mix reads in a block of messages, decrypts them, reorders them in some random fashion, and then sends each message to the next mix in the chain or to a server. It is assumed that an attacker, due to the decryption and reordering steps, will be unable to match the incoming messages of a mix with the outgoing messages unless it has corrupted the mix. Since an attacker is unable to follow the path of a message that goes through an uncorrupted mix and at least one of the $t+1$ mixes is uncorrupted, the attacker will be unable to follow the path of the message from the base station to the server.

8 The Message Service

The message service acts as intermediate storage for messages intended for mobile computers (as shown in figure 1). Messages are sent to the service either directly or through a series of mix servers and are eventually retrieved by the intended recipient using one of two protocols. In designing the message service, the main consideration was the need to enable anonymous queries[10]. For this purpose, we developed a protocol which is similar in nature to secret sharing. In this section, we will describe the design of the message service and in section 9 we will describe the anonymous querying protocol from the mobile computer's point of view.

The message service is implemented using a set of n message servers. We will assume that at most $t < n$ of the servers will be corrupted. To perform an anonymous query, a mobile computer creates a set of $t+1$ requests, sends each request to a different server, and combines the replies (the secure read protocol in figure 1). The requests are designed so that privacy is preserved if at least one of the $t+1$ servers is correct. As in a secret sharing scheme, an attacker knowing about t or fewer of the requests/responses will gain little or no information about the label being queried.

Informally, each of the requests is a formula in which the variables represent message receptacles. A mobile computer interested in label l chooses a set of formulas $f_1, f_2, \ldots, f_{t+1}$ such that $f_1 \oplus f_2 \oplus \ldots \oplus f_{t+1} = r_l$ where r_l is a receptacle which might hold a message with label l. Since the number of receptacles must be manageable, there can not be one receptacle for each label. In our system, we store messages in a hash table and treat each position in the table as a receptacle.

8.1 The Hash Table

The hash table is divided into two distinct sections, the base section and the overflow section. The position of a message in the hash table is determined by computing a hash value using some hash function h on the label l[11]. If position $h(l)$ in the base section of the table is empty, then the message m is stored in that position (along with some header information). If position $h(l)$ is

[9] A server is corrupted if an attacker is able to read the contents of the server's memory or knows the server's private key. As was mentioned earlier, we are assuming in this paper that servers do not fail. However, all of the services described in this paper could be designed to handle a limited number of crash or malicious failures.

[10] A query is anonymous if an attacker is unable to determine which label is being queried.

[11] The hash function is only used to distribute messages within the table and does not need to be a one-way function.

already filled, then m is placed in the overflow section and a pointer to m's position is placed in the header of the message stored at position $h(l)$. If more than two messages have labels which hash to the same value, then a singly-linked list is formed.

The formulas described above treat each position in the hash table as a variable and the contents of each position as the value of that variable. Since each request involves $t+1$ formulas evaluated by $t+1$ different servers and each of the formulas must be evaluated using the same mapping of variables to values, every server must store the same messages in the same positions. Since the positions of messages in the hash table are partially determined by the order in which they are inserted into the table, requests to insert messages must be delivered to all n message servers in the same order.

8.1.1 A List of Hash Tables

In our approach, we limit each hash table to have a fixed size, hence only a limited number of messages can be stored in a table. Since, in an infinite run of the system, an infinite number of messages will arrive, it is not possible to store all of the messages in the same table. Instead, messages will be stored in a sequence of tables (later we will discuss deleting old tables). Each table will store at most MH messages. Once the current hash table has MH messages stored in it or the overflow section of the table runs out of space, a new, empty hash table will be formed and future messages will be stored in the new table.

In order to insure that all $t+1$ formulas of an anonymous query are evaluated consistently, our protocol does not allow access to a table until it has been "closed". In order to bound the period between the arrival of a message at the message service and the time that the message may be retrieved, a hash table may be "closed" and a new one started after a specified period of time even if the table is not yet filled. When a server believes that enough time has passed to "close" a table, it sends a "close table" message to the other servers. Upon receipt, each server creates a new table and begins to place new messages in that table. In order to maintain consistency, "close table" messages must be delivered by all servers in the same order relative to users' messages.

In a mail application, in which users poll the message service periodically, the latency added by waiting for a table to "close" should be acceptable. Some applications, such as a telephone conversation, may find the added latency unacceptable. In this case, an "open" hash table may be queried by bundling all $t+1$ formulas and sending the bundle to the servers using a totally ordered multicast. This will reduce latency but will be less efficient.

8.1.2 The Size of the Hash Table

There are several competing factors which must be considered when determining the size of the hash tables. As the tables get larger, mobile computers will need to do more work to anonymously query a position in a table. However, if the hash tables are too small there may be a lot of messages which hash to the same location and the mobile computer may have to query several positions in order to find the correct message (or to determine that the correct message is not in the table at all). The following formula gives the probability, for a table with m positions in the base section of the table, that exactly c elements will hash to a specific location:

$$\binom{MH}{c}\left(\frac{1}{m}\right)^{c}\left(\frac{m-1}{m}\right)^{MH-c}$$

Using this formula one can determine the expected number of positions that a computer will have to query when searching for a message with a specific label. While, in theory, all MH messages

m	c			
	0	1	2	3
1024	36.8%	36.8%	18.4%	6.1%
2048	60.6%	30.3%	7.6%	1.3%
4096	77.9%	19.5%	2.4%	0.2%
8192	88.2%	11.0%	0.6%	0.03%

Figure 2: Chain Length Probabilities (for $MH = 1024$)

could hash to the same location, it is extremely unlikely (for a reasonably sized hash table). Figure 2 shows the probabilities for a table with $MH = 1024$ for various base section sizes.

Once the size of the base section has been determined, the size of the overflow section must be set. The overflow section should be large enough so that it will usually not run out of space before MH messages are in the table. However, it should not be so large that querying the message service becomes unnecessarily expensive. The probability that at least c messages will need to be placed in the overflow section is

$$\frac{\sum_{i=c}^{MH} \binom{MH-1}{i} \binom{m}{MH-i}}{\binom{m+MH-1}{MH}}$$

Using a base section of 4096 with $MH = 1024$, the probability that an overflow section of 255 positions will become filled is 9.6×10^{-6} and the probability of a message not fitting into the table is 6.6×10^{-6}. For the given values of m and MH, this seems to be the ideal table size. The chances that a new hash table will have to be started before MH messages are placed in the current table is very small, the size of the overflow section is relatively small compared to the size of the base section, and a pointer to a position in the overflow section can be stored in one byte.

8.2 Garbage Collection

As was mentioned above, 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 computers may retrieve messages anonymously, the message service may not know when this has happened.

An approximate solution is to delete messages after some period of time Δ has passed. In our system, a hash 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 computer will be disconnected from the network for a long period of time. This can happen if the computer moves outside of the range of all of the base stations or if the computer is turned off to conserve battery power. If the computer is disconnected for too long then it may miss some of the messages that were sent to it. In order to avoid this, we have developed a vacation service.

A computer which is concerned about losing messages while it is disconnected from the network registers with the vacation service by sending it a list of message labels in which it is interested. If the vacation service does not receive a message from a registered computer within some specified period of time $\gamma < \Delta$, then the vacation service will begin to periodically check the message service for messages with any of the specified labels and will store a local copy of those messages. When the mobile computer next contacts the vacation service, the vacation service will send any messages that it stored for the mobile computer to the message service and will then stop looking for any new messages for that computer. The mobile computer can then read the messages from the message service as usual.

In order to prevent the vacation service from using the registration messages to locate the user, the mobile computer should use the techniques which were described in section 7 to hide the locations from which it sends the messages.

8.3 Message Service Operations

The message service can process three different types of requests: storing a message, retrieving a message with a particular label, and a bit-vector message retrieval request.

Since sender privacy is preserved using the mix servers, messages are added to the message service in the same way for both public and private labels. Given that messages must be delivered to all the message servers in the same order, an attacker may attempt to disrupt the system by causing the servers to deliver messages in different orders. In order to prevent this, the multicast protocol used to deliver messages to the servers must be chosen carefully.

When a mobile computer wishes to retrieve a message with a particular label, l, it can send that label to one of the message servers along with the number of the hash table to be searched. The server receiving the request will look in the appropriate table at the linked list beginning at position $h(l)$ for any messages with label l and return all such messages along with the numbers of the of the oldest and newest available hash tables[12]. If the computer requests a label from a table that has already been deleted or is not yet ready, the message server will just return the numbers of the oldest and newest available hash tables. As a general rule, requests and responses to requests of this type are sent unencrypted. Since an attacker may be able to read the contents of requests and responses associated with up to t of the message servers, it must be assumed than an attacker will be able to read the contents of a request and corresponding response sent to any one message server even if the messages are encrypted.

To retrieve a message anonymously a computer must send $t+1$ distinct, encrypted messages to $t+1$ different servers such that an attacker would need to read all $t+1$ requests and/or responses in order to determine which position is being requested. In this section we will describe the type of request a server will accept and the corresponding response that it will give in order to facilitate this type of interaction. In section 9, we will describe how a mobile computer can construct the necessary requests and combine the responses.

Let's say that each hash table has M positions in it (including the overflow section) and the message (including the header) in the i^{th} position of the j^{th} table, $0 \leq i < M$, is $HT_j[i]$. A message server will accept a request containing a hash table number, j, a bit-vector of length M, B, and a secret key, K_s. The secret key should be encrypted with the public key of the message server and the table number and bit-vector should be encrypted with the secret key. The message server will compute the response using the algorithm in figure 3. The computed response is returned along with the numbers of the oldest and newest available hash tables. Again, if the hash table requested has already been deleted or is not yet available, then only the numbers of the oldest and newest tables are returned.

[12] A hash table is available from the time that the last message is added to it until the time that it is deleted.

1. $S = \{HT_j[i] \mid (B[i] = 1) \wedge (HT_j[i] \text{ is not empty})\}$

2. $m = \bigoplus_{m' \in S} m'$, where $\bigoplus$ is the bitwise exclusive-or operator.

3. response $= K_s(m)$

Figure 3: Algorithm for Responding to a Bit-Vector

vector	bit position								
	0	1	2	3	4	5		$M-2$	$M-1$
v_1	0	1	1	0	1	0	$\cdots$	0	1
v_2	1	1	0	1	1	0	$\cdots$	1	1
v_3	1	1	1	1	0	0	$\cdots$	1	0
$v_1 \oplus v_2 \oplus v_3$	0	1	0	0	0	0	$\cdots$	0	0

Figure 4: Sample Bit-Vectors for $t = 2$

9 Receiving a Message

As in the case of sending a message, a computer, q, attempting to receive a message with label l may perceive l as being either public or private. If l is private then q can send a request to a single message server which will respond with the correct answer. Since the only pieces of information that will appear in plain-text are the label and the hash table number, neither the request nor the response needs to be encrypted[13].

If l is public, then q must perform more work. Specifically, q will need to create $t+1$ requests of the bit-vector type, send each request, encrypted, to a different message server, decrypt and then combine the responses. Say that q wishes to read the message at the i^{th} position of the j^{th} table. q should create t random bit-vectors of length M (the size of the hash table). Next, it should create the $t+1^{st}$ bit-vector by exclusive-oring the t random bit-vectors together and then flipping the i^{th} bit. This will create a set of $t+1$ bit-vectors that, when exclusive-ored together, will yield a bit-vector in which the i^{th} bit is a 1 and all the other bits are 0. An example of such a set is shown in figure 4. To get the message at the i^{th} position of the j^{th} table, q sends each bit-vector to a different server and exclusive-ors the responses together.

It should be noted that only one of the vectors actually depends on the position being queried. Therefore, q can query a new position, k, by taking the old $t+1^{st}$ vector, flipping back bit i and then flipping bit k. It can then send this vector to one of the servers (any server that did not receive one of the t random vectors) and use the response along with the responses to the t random vectors to get the message in position k. With this technique, q can read d positions in a table using only $t+d$ requests.

[13] Since the message server being queried may be corrupted, there may in fact be no advantage to encrypting the request or response.

$$P_i = \begin{cases} \frac{\varphi}{\varphi+(M-1)(1-\varphi)} & \text{if } B_1[i] \oplus B_2[i] \oplus \cdots \oplus B_t[i] = 0 \\ \\ \frac{1-\varphi}{(1-\varphi)+\varphi(M-1)} & \text{if } B_1[i] \oplus B_2[i] \oplus \cdots \oplus B_t[i] = 1 \end{cases}$$

$$\text{Expected number of positions with } B_1[i] \oplus B_2[i] \oplus \cdots \oplus B_t[i] = \begin{cases} 0 & \rightarrow & (1-\varphi)(M-1)+\varphi \\ 1 & \rightarrow & \varphi(M-1)+(1-\varphi) \end{cases}$$

Figure 5: Probability for Position i

φ	$B_1[i] \oplus B_2[i] \oplus \cdots \oplus B_t[i]$			
	0		1	
	P_i	N	P_i	N
0	0	4350	1	1
$\frac{1}{100}$	$\frac{0.0101}{4351}$	4306.51	$\frac{96.82}{4351}$	44.49
$\frac{1}{10}$	$\frac{0.1111}{4351}$	3915.1	$\frac{8.983}{4351}$	435.9
$\frac{1}{5}$	$\frac{0.2500}{4351}$	3480.2	$\frac{3.997}{4351}$	870.8
$\frac{3}{10}$	$\frac{0.4286}{4351}$	3045.3	$\frac{2.333}{4351}$	1305.7
$\frac{2}{5}$	$\frac{0.6667}{4351}$	2610.4	$\frac{1.500}{4351}$	1740.6
$\frac{1}{2}$	$\frac{1}{4351}$	2175.5	$\frac{1}{4351}$	2175.5

Figure 6: Probabilities for $M = 4351(4096 + 255)$

9.1 Creating Random Bit-Vectors

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 seeing at most t of the vectors will gain no information about the position being queried. However, using such a bit-vector can be computationally expensive for the message servers. If each hash table has MH messages in it, then, for each request, a message server will have to exclusive-or together, on average, $\frac{MH}{2}$ messages. If $MH = 1024$ and messages are 1 KByte each, then computing a response will require processing an average of 0.5 MBytes of data.

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 a 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.

In order to determine a good value for φ, we should look at the set of bit-vectors from an attackers point of view to determine how much information the attacker can infer. We will assume the worst case scenario in which the attacker has acquired t of the $t+1$ bit-vectors, one of which is the final vector[14]. Let's call the vectors that the attacker knows $B_1, B_2, \ldots, B_t$. The attacker can, for each position i, use the values in the known bit-vectors and the value of φ to determine the probability that position i is the one being queried. The formulas in figure 5 show the expected number of positions for which an even or odd number of 1 bits will be visible to the attacker and the probabilities that the attacker can infer for each position based on the parity of that position for the known vectors. Figure 6 shows the values for $M = 4351$ for several values of φ.

[14] An attacker will gain no information unless it has the final vector.

Multiplying P_i by N for the case of $\varphi = \frac{1}{100}$ shows that there is a 99% chance that one of the (about 44.49) positions with odd parity visible is the position being queried and only of 1% chance that it is one of the (about 4306.51) positions with even parity. So, while seeing t of the $t+1$ vectors does not allow the attacker to rule out any of the positions entirely (for $0 < \varphi < 1$), the attacker will be able to extract a relatively small group of positions such that the position being queried is highly likely to be in that group. However, if the number of conversations in the system is larger than the number of positions in the hash table, an attacker will have less certainty about the label being queried than the position being queried and so $\varphi = \frac{1}{100}$ may be an acceptable value.

In the formulas in figure 5 we are assuming that a computer is equally likely query any position in the table. In reality, a computer is more likely to query a position in the base section of the table than in the overflow section. In addition, a computer would never query an empty position in the overflow section of the table. Finally, if a computer usually has to wait a long time for a message to arrive, it will tend to have to query the same position in several hash tables. As a result, a computer may be more likely to query an empty position than a non-empty position. By taking advantage of these extra considerations, an attacker may, in reality, be able to compute probability distributions which provide somewhat more information than the formulas in figure 5 indicate. On the other hand, since many labels will hash to the same value, knowledge of the position being queried is not the same as knowledge of the label in which the computer is interested.

9.2 Looking at More than One Position at a Time

Usually a mobile computer will be engaged in several conversations at the same time[15]. For each conversation, the mobile computer must check every hash table for an incoming message. Most of the time, it will not find a message. Under these circumstances, it is possible to reduce the number of requests that need to be sent to the message servers by checking more than one position at a time. In order to make this possible, we must first add some information to the message headers that are added by the message service.

First, the positions in the hash table are divided into a number of segments. Each segment is allocated its own space in the message header. A message stored at a particular position must place all of its header information in the space allocated for its segment and place zeros in all other parts of the header. The header for each message should contain a pointer into the overflow section (see section 8.1) and a hashed value of the message's label. The hash function, h_2, used for the header should be independent of h (the hash function used to place messages in the table) and should always yield a non-zero value. Figure 7 shows the structure of the message headers for a hash table with a base section of 4096 positions and an overflow section of 255 positions in which the table is divided into 17 segments and h_2 maps labels to integers between 1 and 255.

Suppose that a computer, q, is interested in two different positions, i and j. As before, q can create t random vectors and a final vector. However, instead of flipping one bit, say q flips bits i and j. The result computed from combining the responses to the $t+1$ requests will be the same as exclusive-oring together the messages (and headers) in the two positions. If i and j are in different segments, then the header information for both messages will be readable. Similarly, if the hash table is divided into n segments, then up to n positions can be queried at a time[16].

[15] Remember that a mobile computer is in a conversation with another mobile computer if it expects to receive a message from that computer at some point in the future. In addition, the mobile computer is always looking for initiation messages, which can be treated as another conversation.

[16] Instead of choosing each return "address" label completely at random, q may pick sets of labels whose hash values will be evenly divided among the segments in order to minimize the number of queries needed to check every conversation. While this will constrain the value of each label, we believe that security will not be impacted.

position	byte number									
	0	1	2	3		30	31	32	33	34 - ?
0 - 255	pointer	$h_2(l)$	0	0	$\cdots$	0	0	0	0	l
256 - 511	0	0	pointer	$h_2(l)$	$\cdots$	0	0	0	0	l
$\vdots$										
3840-4095	0	0	0	0	$\cdots$	pointer	$h_2(l)$	0	0	l
o1 - o255	0	0	0	0	$\cdots$	0	0	pointer	$h_2(l)$	l

Figure 7: Message Headers

Once the response has been computed, the header information for each of the positions can be used to quickly locate messages. Say that q is looking for a message with label l at position i and that the message at position i has label l' ($l' = 0$ if the position is empty). q can get $h_2(l')$ from the header and compare it to $h_2(l)$. If $h_2(l') = 0$ then there is no message at position i and so there is no message with label l in the table. If $h_2(l) = h_2(l')$ then q should read the message at this position[17]. If $h_2(l') \neq h_2(l)$ then q can ignore the message in this position. If the linked list pointer is non-zero, then there are other messages that hash to $h(l)$ and q should check the position indicated by the pointer.

If q requests several positions at the same time and only one of the positions is non-empty, then the message in that position is available to read immediately. If $d > 1$ positions are non-empty and q is interested in the messages in all of the positions then q only needs to read in $d - 1$ of the positions one at a time. It can then use those $d - 1$ messages along with the result of the first query to extract the d^{th} message.

10 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 check every hash table for a message, it is very inefficient to have a large number of conversations when most of them are inactive.

There are basically two ways in which this can be handled. If a computer sending a message wishes for this to be the last message in the conversation, then instead of including a return "address" label in the message, the message can include a special "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 addition to saving memory, each computer will have one less conversation to check for in each hash table.

In many cases, however, when a computer sends a message it will not know whether or not that message will be the last in the conversation. In addition, in many conversations, even though the conversation may not have ended, it may be a very long time between the arrival of a message at a computer and the time that the response is sent. Since the process of maintaining a conversation

[17] It is possible, although unlikely, that $l \neq l'$ and so the mobile computer should check the full message label after reading in the message to verify that the label is correct.

(even one in which no messages are being sent) consumes resources, it is a good idea to stop such conversations. If one computer wishes to send a message to another computer after a conversation between those two computers has ended, it can start a new conversation by sending an initiation message.

To solve the problem of sporadic traffic, a computer can, when sending a return "address" label, include an expiration time for that label[18]. In order to have labels with expiration times, it is necessary to have synchronized clocks. This can be accomplished through the use of a time service. The details of clock synchronization and its use are beyond the scope of this paper. An overview of the subject can be found in [SWL90].

11 Performance

In this section, we will discuss the overhead that is associated with the operations of sending and retrieving messages in a secure manner. The primary costs associated with this privacy scheme are the public key operations. Using the RSA scheme [RSA78], the costs can be estimated by the number of modular multiplications needed to perform an operation. The amount of time necessary to perform a modular multiplication depends to a large degree on the type of hardware that is available. It should be particularly noted that the mobile computers will tend to be significantly slower than the servers.

11.1 Mobile Computers

In order to send a message using a public label, the message must be encrypted with the public keys of $t+1$ mixes. If each mix uses a public exponent of 3, then a total of $2t+2$ modular multiplications are required. If the message is an initiation message, then the message must include a secret key and that key must be encrypted with the public key of the intended recipient. This will require another 2 modular multiplications. If the initiator also includes a signature for authentication purposes, then a modular exponentiation must be performed using the initiator's private exponent. For a 512-bit modulus, this will take an average of 768 modular multiplications. However, since the initiator knows the factorization of the modulus, the Chinese Remainder Theorem can be used to significantly reduce the cost of performing the exponentiation.

The situation for receiving a message is similar. In order to look at d positions in a hash table, $t+d$ queries must be made to the message servers. Each query involves encrypting a bit-vector with a secret key and encrypting the secret key with the public key of a message server. So, using a public exponent of 3, this step involves $2(t+d)$ modular multiplications. Since the t random bit-vectors are independent of the positions being queried, these bit-vectors can be created and encrypted in advance using any spare cycles that are available. If an initiation message is received, then the message will include an encrypted secret key which must be decrypted. The cost of decrypting a message is similar to the cost of signing a message. In addition, if the initiation message includes a signature, then 2 modular multiplications will be needed for verification.

According to [BCY93], a mobile computer with a DSP chip can perform a 512-bit modular multiplication in 145 μs. Using this value, we can estimate the amount of time required for the different operations that a mobile computer will need to perform. Figure 8 shows the estimated times for a few values of t. The times listed for creating an initiation message with a signature for authentication assume that computing the signature will take 768 512-bit modular multiplications.

[18] Depending on the type and frequency of the traffic in the system, the "lifetime" of this label could be anywhere from a few minutes to several days.

Operation	t			
	1	2	3	4
Sending (non-init)	0.6 ms	0.9 ms	1.2 ms	1.5 ms
Sending (init, no auth)	0.9 ms	1.2 ms	1.5 ms	1.7 ms
Sending (init, w/ auth)	112.2 ms	112.5 ms	112.8 ms	113.1 ms
Receiving (non-init)	0.6 ms	0.9 ms	1.2 ms	1.5 ms
Receiving (init, no auth)	111.9 ms	112.2 ms	112.5 ms	112.8 ms
Receiving (init, w/ auth)	112.2 ms	112.5 ms	112.8 ms	113.1 ms

Figure 8: Time Estimates for Mobile Computers

However, if the algorithm for computing the signature takes advantage of the Chinese Remainder Theorem, the actual times to create initiation messages should be significantly less than the times shown in the table. The times for receiving a message are all computed for $d = 1$. The estimates shown for receiving an initiation message assume that the Chinese Remainder Theorem is not being used for decryption. As in the case of creating a signature, the time required will be significantly less if it is used.

11.2 Servers

When a server (either a mix or a message server) receives a message, it must decrypt the message. Just as for the mobile computer, decrypting a message involves using a private key to extract a secret key and then using the secret key to decrypt the message. The cost of handling a single message is dominated by the cost of the modular exponentiation needed to extract the secret key. Since these servers will specialize in handling message requests, it is reasonable to assume that they will have special purpose hardware for performing modular exponentiation. With such hardware, the secret key can be extracted in under 20 ms [Bri89].

12 Conclusions

We have presented a set of protocols which work together to preserve privacy for users of mobile computers. A major concern in designing such protocols is the limited computing power of the mobile computers. In our approach, the computing costs of each of the protocols can be dynamically tuned by each of the mobile computers based on the amount of computing power available and the degree of any perceived threat to privacy. In addition, the protocols were designed to place most of the computational burden on the servers which can use specialized hardware in order to perform the necessary operations quickly.

We are in the process of extending the Horus architecture [vRHB94] to support secure and private mobile communication. In the future, we will use this architecture to experiment with the techniques presented in this paper and hope to report on our experiences in future papers.

References

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

[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.

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

[Bri89] Ernest F. Brickell. A survey of hardware implementations of RSA. In *Advances in Cryptology - CRYPTO '89*, pages 368–370, 1989.

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

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

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

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

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

[DH79] W. Diffie and M.E. Hellman. Privacy and authentication. *Proceedings of the IEEE*, 67(3):397–427, March 1979.

[Rei94] 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.

[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.

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

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

[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.

[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.