Preserving Privacy in a Network of Mobile Computers

TR 95-1490*

David A. Cooper   Kenneth P. Birman†

Dept. of Computer Science
Cornell University
Ithaca, NY 14853–7501

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.

We have developed a replicated memory service which allows users to read from memory without revealing which memory locations they are reading. Unlike previous protocols, our protocol is efficient in its use of computation and bandwidth. In this paper, we will 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.

1 Introduction

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). 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.

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.

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 [6, 15] which will be discussed later in this paper). 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. In section 5, we will present such a protocol.

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,

*To appear in *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, May 1995

†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.

[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.

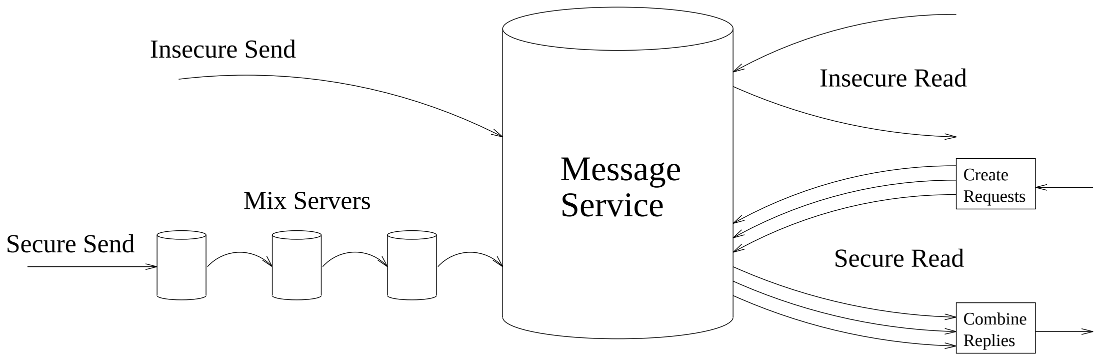

Figure 1: The Message Service

they will 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. In section 7, we will present a technique, developed by the authors, 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 and in section 8 we will show how this protocol can be used to create a message service which will allow mobile computers to read messages without revealing their location. The message service and the protocols for interacting with it are depicted in figure 1.

2 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 [10]. 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 [18]. 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. The details of message encryption and signatures are beyond the scope of this paper (see [20] for an overview of the subject).

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 [6]. 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 [11, 12, 13, 16]. The protocols in [6, 11] have security problems which were corrected in [14, 15].

In [7], David Chaum presents an information theoretically secure technique for preserving the unlinkability of the sender and 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 the 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 [4], 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 contents 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 7, we will present a protocol which satisfies our more limited requirements which has a small bandwidth overhead.

There has been some work in the area of privacy for mobile computers. In [2, 3, 5], protocols are presented which encrypt messages that are sent along wireless links thus preventing an attacker from using the contents of these messages to locate users. The main goal of these papers is to limit the computational overhead of the mobile computers. While the protocols in these papers 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. In this paper, we will present protocols which are resilient to attacks on the static network.

3 System model

3.1 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 computer's identity.

3.2 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 *area* 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 attaches 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].

3.3 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. 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. We also assume that timestamps and nonces are used as appropriate to prevent message replay attacks. 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 [17].

3.4 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*[6] to the message and sends it to the message service. Next, the intended recipient, upon discovering that the message is available, requests the message from the message service. The reason for this approach is twofold. First, it provides a means for computers to anonymously receive messages. Second,

[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 area of the base station with which it is connected.

[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 areas are neighbors of the area from which the request originated in addition to the originating base station.

[6] A label in our system is equivalent to a visible implicit address in [13].

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 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[7].

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 privacy and will be discussed in detail later.

4 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[8]. 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. $\{id_p, K_p\}_{K_a^{-1}}$[9]). The initiator, p, can then send $K_q(K_s), K_s(\{id_p, K_p\}_{K_a^{-1}}, \{id_p, id_q, m\}_{K_p^{-1}})$ to q thus allowing q to authenticate p.

5 Unlinkability of sender and recipient

Since mobile computers are anonymous by default, the relationship between senders and recipients can be hidden by carefully choosing message labels. Labels can be divided into two basic categories. Label l is *public* if it is known to every user (i.e. available in a public directory) and *private* if it is known only to the sender and the intended recipient of the message.

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 use 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.

Since the initiation message contains q's public label in plain-text, an attacker will be able to determine that an initiation message was sent to q. However, the initiation message does not contain any (unencrypted) information that would allow an attacker to determine that p sent the message. Since p's identity remains secret (to all except possibly q), an attacker will not be able to determine that p and q are communicating with each other.

In order to maintain this privacy, p's identity must remain secret for the remainder of the conversation. In order to accomplish this, p can generate a random return "address" label, r_p, and send this label to q in the initiation message. If r_p is encrypted, then only p and q will know its value thus making r_p a private label. This label can then be used by q to send a message to p without an attacker being able to determine that p is the intended recipient of the message.

In order to further reduce the amount of information that an attacker might be able to infer from messages sent between p and q (such as the number of

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

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

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

messages that q exchanged with its secret communicating party), each subsequent message (from both p and q) should contain a new randomly generated return "address" label.

6 Location privacy while sending a message

In the previous two sections, we assumed that both parties wished to maintain privacy. We made this assumption since it is impossible to prevent an attacker from obtaining the identities of the participants or the contents of the messages in a conversation if the attacker is in collusion with one of the conversation's participants (if the conversation initiator does not identify itself to the other participant, then the attacker can only acquire participant information by colluding with the initiator). On the other hand, p should be able to prevent q from determining its location even if p and q are engaged in a conversation.

If p trusts q (i.e. believes that q will not attempt to locate it or does not mind being located by q), then it is easy for p to maintain location privacy. Since, as described in the previous section, p does not include any (unencrypted) information about its identity in the messages that it sends to q, only q would be able to identify p as the message sender. Thus, an outsider will be able to determine that a message was sent from a certain location, but only q will know that p sent the message.

If p wishes to hide its location from q, then p must work to hide the location from which it sends its messages. One possible solution to this problem is to use a MIX-network [6, 15] which takes batches of messages and scrambles them so that it is impossible to determine which message came from where (see figure 2).

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[10]. 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:

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

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 $M_1, \ldots, M_{t+1}$, and S is the address of a server (the final destination of the message). (As is explained in [6, 15], the above message must be peppered with random data after each encryption step in order to prevent active attacks on the MIX-network).

[10] 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.

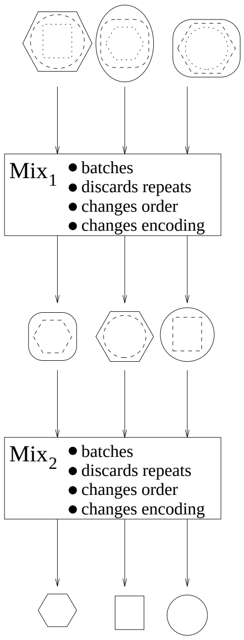

Figure 2: Mix Network (from [13])

Each MIX reads in a block of messages, decrypts them, removes the random data, reorders the messages 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.

7 A memory service with a blinded read operation

Just as the protocol of section 5 does not hide the location of the sender of a message from the message's intended recipient, the protocol also does not protect the recipient from the sender. Mobile computers must also be able to receive initiation messages which use public labels. One way to solve this problem is to use the MIX-network. A mobile computer wishing

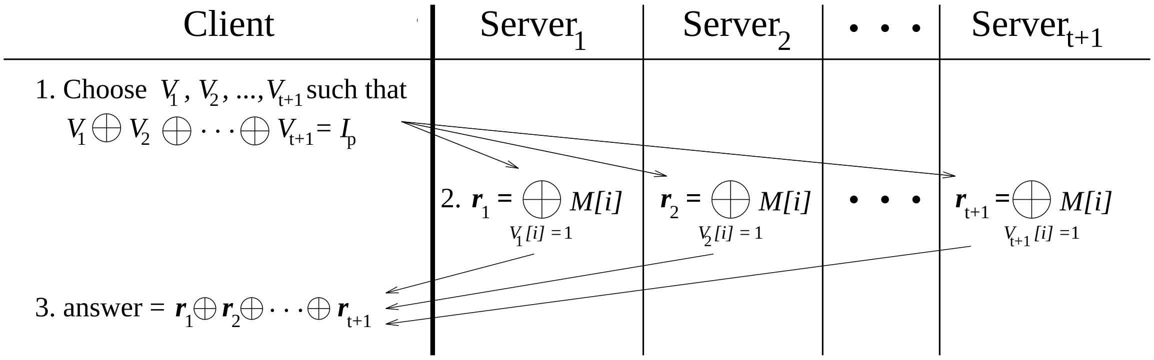

Figure 3: Bit-Vector Protocol

to receive a message would send a request message, through the MIX-network, to a message repository. The request message would include an *anonymous return address* [6, 12, 15] which could be used by the message repository to send the response (via the MIX-network).

The problem with the above technique is that the mobile computer must inform the message repository of the label in which it is interested. If the label is public or if the message repository is in collusion with the message sender (and the sender knows the identity of the intended recipient) then the message repository will know the identity of the requesting computer. Since the mobile computer may reveal its identity by sending the request message, it must hide the location from which it sends the message. While this method will guarantee location privacy for the recipient, it is very expensive.

An alternative is for the mobile computer to send request messages that do not contain any information which would allow the message repository to identify the requester (thus eliminating the need for the mobile computer to hide the location from which it sent the message). In designing a protocol to achieve this goal, our main objectives were to minimize the computational and bandwidth overhead involved (especially for the mobile computers).

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 modify this protocol for use as a message repository for mobile computers.

A memory service 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,

vector	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 4: Sample Bit-Vectors for $t = 2, p = 1$

we will assume that at most $t < n$ of the servers will be corrupted.

7.1 Reading from memory

The technique for reading from memory is similar in nature to secret sharing. The requesting computer generates a set of $t+1$ questions and sends each question to a different server. Just as in secret sharing, an attacker that is able to obtain at most t of the questions/answers will be unable to determine the secret. However, since the nature of the secret information is different in our scheme, secret sharing techniques are not appropriate for this problem.

For this section, we will assume that only read operations are performed and that the contents of the servers' memories are the same. A computer wishing to read from memory should create t random bit-vectors of length m. Next, it should create a $t+1^{st}$ bit-vector by exclusive-oring the t random bit-vectors and then flipping the p^{th} bit (in order to read cell p). This will create a set of $t+1$ bit-vectors that, when exclusive-ored together, will yield the bit-vector I_p:

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

An example of such a set is shown in figure 4. Using these bit-vectors, the contents of cell $M[p]$ can be obtained using the protocol in figure 3. While it is not shown in the figure, the bit-vectors $V_1, V_2, \ldots, V_{t+1}$ and the responses $r_1, r_2, \ldots, r_{t+1}$ must be encrypted so that only the client and server$_i$ can read the values of V_i and r_i.

7.1.1 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.*

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.

7.1.2 Sparse bit-vectors

As was shown in the previous section, 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 that sees 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 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 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 attacker's 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 $t+1^{st}$ vector. Let's call the 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 in cell p.

In order to compute the information that an attacker can infer, we will need the following values:

$$
\begin{aligned}
C &= V_1' \oplus V_2' \oplus \ldots \oplus V_t' \\
S_0 &= \{i \mid C[i] = 0\} \\
S_1 &= \{i \mid C[i] = 1\} \\
S_0' &= \{i \mid V''[i] = 0\} \\
S_1' &= \{i \mid V''[i] = 1\} \\
I_p[j] &= \begin{cases} 0 & \text{if } j \neq p \\ 1 & \text{if } j = p \end{cases}
\end{aligned}
$$

Due to the way that bit-vectors are created, we know that $V'' = C \oplus I_p$. Therefore, we can compute the a posteriori probability that $i = p$ as:

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

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

- 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, the a posteriori probability for position i is:

	S_0		S_1	
φ	P_i	$\lvert S_0\rvert$	P_i	$\lvert S_1\rvert$
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 5: Probabilities for $m = 1024$

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

Since P_i depends on $|S_0|$ and $|S_1|$, in order to be able to choose a good value for φ, we must estimate the values of $|S_0|$ and $|S_1|$. If a position, i, is not the one being read (i.e. $i \neq p$), then $C[i] = V''[i]$. Since $V''[i] = 1$ with probability φ, $C[i] = 1$ with probability φ. Since there are $m - 1$ positions, i, for which $i \neq p$, we can expect $(1 - \varphi)(m - 1)$ to be in S_0 and $\varphi(m - 1)$ to be in S_1. Since $C[p] = \neg V''[p]$, we have $p \in S_0$ with probability φ and $p \in S_1$ with probability $(1 - \varphi)$. Therefore, we can estimate $|S_0| = \varphi + (1 - \varphi)(m - 1)$ and $|S_1| = (1 - \varphi) + \varphi(m - 1)$.

In figure 5 we show the values for P_i and $|S_i|$ for different values of φ for a memory with 1024 cells (the values for P_i and $|S_i|$ were computed using the formulas above to estimate $|S_i|$). In the case of $\varphi = \frac{1}{100}$, there is a 99% chance that $p \in S_1$ and we can expect that $|S_1| \approx 11.22$. While there is a chance that p is among the approximately 1012.78 positions in S_0, it is very unlikely. So, while seeing t of the $t + 1$ vectors does not allow the attacker to rule out any of the positions entirely (for $0 < \varphi < 1$), 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.

7.2 Writing to memory

The protocol in figure 3 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 4. 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 possible ways that the above situation could occur. The first is if a read operation is performed concurrently with a write operation. The second is if a client performing a write operation sends different values to different servers or fails to inform some servers of the write operation. In order to prevent the first problem, all operations should be sent to the memory servers using a totally ordered multicast. For read operations, the $t + 1$ bit-vectors should be bundled together and sent as one message (since each bit-vector will be encrypted using a different secret key, each server will only be able to read the bit-vector intended for it even though it will receive all of the bit-vectors).

If it is necessary to guard against a malicious client (or perhaps a malicious server), then the totally ordered multicast must be tolerant to such behavior. The multicast protocol in [17] ensures that every correct server will receive the same set of messages in the same order thus preventing a malicious process from corrupting memory in this fashion[11].

8 Retrieving a message

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. In this section, we will show how to use the memory service of the previous section to implement a message service which will enable blinded read operations.

[11] Since the memory, as described, does not contain any access controls, an attacker may still cause problems by writing bad values.

Messages are sent to the servers using a totally ordered multicast (as was described in section 7.2). 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). There is a tradeoff that must be considered when choosing a table size. First, the amount of effort needed to read a cell from a table increases as m increases (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, mobile computers must wait until a table has 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 computers that read from the table.

Once a table has been filled, mobile computers may read the 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.

Since every mobile computer will need to see the digest for every table, table digests are 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 [1, 8] for more information on multicasting in mobile networks).

8.1 Reading from a table

Each mobile computer will have a list of message labels in which it is interested $(l'_0, l'_1, \ldots, l'_k)$. When it 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 mobile computer 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.

Mobile computers can read messages from the message service in one of two ways. If the label that it wishes to read is private and the computer 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 (i, j) where i is the number of the table to be read and j is the number of the cell within that table which contains the message. If the label to be read is public or if the mobile computer does not trust the message sender, then it must use the blinded read operation from section 7. 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.

8.2 Choosing a hash function

If the hash function, h, used in creating message digests is the identity function then there will never be a case where $h(l'_i) = h(l_j)$ but $l'_i \neq l_j$. This means that only mobile computers which have messages intended for them in a table will read from that table. Since an attacker can determine the location from which the mobile computers read from the table, an attacker may gain useful information about computers' locations if m is small. If m can not be made large enough to sufficiently confuse an attacker, then a hash function must be chosen which will force some mobile computers to read messages from the table that were not intended for them.

Suppose, for example, that the total number of message labels in which every mobile computer is interested is 64,000. If 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 made 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 mobile computers. Thus, an attacker seeing a mobile computer read a message from the table will know that there is only a 50% chance that the computer is the intended recipient of one of the messages in the table.

8.3 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 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 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 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 6 to hide the locations from which it sends the messages.

9 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.

One technique for solving this problem is for computers to explicitly end conversations by including an "end of conversation" marker in a message instead of a return "address" label. 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.

An alternative is to use return "address" labels with expiration times. If the recipient does not send a message using the label before the label expires, then the sender will consider the conversation to have ended. If the recipient wishes to send a message after the expiration time, it can begin a new conversation by sending an initiation message. 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 [21].

10 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 cryptographic operations. In computing our cost estimates we will primarily use the performance figures from [9]. This paper describes the implementation of RSA [18] and DES [10] using a Motorola DSP56000 along with a Western Digital WD20C03 DES chip.

10.1 Mobile computers

There are a few basic cryptographic operations that are needed by the mobile computers. The computer needs to create random numbers for use as bit-vectors, DES keys, and as padding in RSA encryption. The most secure technique for creating random data is to use a noise circuit, however, using a pseudo-random number generator can be less costly. Using a noise circuit, random data can be created at the rate of 81.9 ms/KByte. In addition, mobile computers need to encrypt and decrypt messages using RSA and DES. Estimates for the costs of these operations can be found in figure 7.

When sending a message, the message must first be encrypted using the DES session key for the conversation (2.2 ms/KByte). Then, if the message is to be sent through the MIX-network, the message must be encrypted for $t+1$ MIXes. For each MIX, 512 bits of

Operation	t		
	1	2	3
Sending (non-init)	12.2 ms + 6.6 ms/KByte (12.2 ms)	18.3 ms + 8.8 ms/KByte (18.3 ms)	24.4 ms + 11.0 ms/KByte (24.4 ms)
Sending (init, no auth)	18.3 ms + 6.6 ms/KByte (18.3 ms)	24.4 ms + 8.8 ms/KByte (24.4 ms)	30.5 ms + 11.0 ms/KByte (30.5 ms)
Sending (init, w/auth)	62.3 ms + 8.8 ms/KByte (18.3 ms)	68.4 ms + 11.0 ms/KByte (24.4 ms)	74.5 ms + 13.2 ms/KByte (30.5 ms)
Receiving (non-init)	23.1 ms + 6.6 ms/KByte (22.8 ms)	39.8 ms + 8.8 ms/KByte (39.5 ms)	56.5 ms + 11.0 ms/KByte (56.2 ms)
Receiving (init, no auth)	67.1 ms + 6.6 ms/KByte (22.8 ms)	83.8 ms + 8.8 ms/KByte (39.5 ms)	100.5 ms + 11.0 ms/KByte (56.2 ms)
Receiving (init, w/ auth)	69.1 ms + 8.8 ms/KByte (22.8 ms)	85.8 ms + 11.0 ms/KByte (39.5 ms)	102.5 ms + 13.2 ms/KByte (56.2 ms)

Figure 6: Cost Estimates for Mobile Computers

operation	estimate
creating random data	81.9 ms/KByte
RSA encryption	1 ms
RSA decryption	44 ms
DES encryption and decryption	2.2 ms/KByte

Figure 7: Cost Estimates for Basic Operations

random data must be created (a DES key along with some padding for the RSA encryption), the message must be encrypted with the DES key, and the DES key must be encrypted with the public key of the MIX. Using the estimates from figure 7, this will take 6.1 ms + 2.2 ms/KByte. If the message is an initiation message then a DES session key must be created for the conversation and encrypted with the public key of the intended recipient (along with random padding). This will cost 6.1 ms. If the initiation message includes authentication information then a signature must be created which will cost 44 ms + 2.2ms/KByte (using DES to create a message digest).

In order to read a message using the blinded read operation, t random bit-vectors must be created. Then $t+1$ bit-vectors must be encrypted and sent to message servers. The responses from the message servers must then be decrypted. For a table with 1024 entries, $1536t + 512$ bits of random data must be created ($1024t$ for the random bit-vectors and $512(t+1)$ for the DES keys and random padding). This will take $(15.4t + 5.1)$ ms. Encrypting the bit-vectors will take $0.3(t+1)$ ms and encrypting the DES keys will take 1.0(t+1) ms. Decrypting the responses from the message servers will take $2.2(t+1)$ ms/KByte. Once the responses from the message servers have been combined, the computed message must be decrypted. This will take 2.2 ms/KByte. If the received message is an initiation message then the session key must be extracted which will take 44 ms. If the initiation message also includes authentication information, then 2 ms + 2.2 ms/KByte are needed to verify the sender's public key certificate and to verify the signature.

Since many of the computations listed above do not depend on the message being sent or received, time may be saved by precomputing data. For both sending and receiving messages, the random data can be precomputed as well as the DES keys for the message servers and MIXes. In addition, since t of the $t+1$ bit-vectors used to read from the message service are random, they can be encrypted in advance. Figure 6 shows the estimated costs for a few values of t. In each box, the top line represents to total amount of computation that needs to be performed and the bottom line represents the amount that can be precomputed.

10.2 Servers

When a server (either a MIX or a message server) receives a message, it must decrypt the message. This involves using RSA decryption to extract the DES key and then using the DES key to extract the message. Since the servers will specialize in handling message requests, it is reasonable to assume that they will have special purpose hardware for RSA decryption as well as DES encryption and decryption. In [19], Shand and Vuillemin describe an RSA chip which can perform 512-bit decryption in under 1 ms. Using this chip along with the DES chip used in [9], messages can be decrypted in 1 ms + 2.2 ms/KByte. In the case of the message servers, the responses must be encrypted adding 2.2 ms/KByte.

11 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 [22] to support secure and private mobile communication. In the future, we will use this system to experiment with the techniques presented in this paper and hope to report on our experiences in future papers.

Acknowledgements

We would to thank Birgit Pfitzmann, Martín Abadi, Anindya Basu, and the anonymous referees who provided many valuable comments on drafts of this paper.

References

[1] 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.

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

[3] 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.

[4] Gilles Brassard, Claude Crepeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology – CRYPTO '86*, pages 234–238, August 1986.

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

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

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

[22] 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.