Enclaves: Enabling Secure Collaboration over the Internet

Li Gong
SRI International

Enclaves: Enabling Secure Collaboration over the Internet

Li Gong

SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, California 94025, U.S.A.
(gong@csl.sri.com)

Abstract

The rapid expansion of the Internet means that users increasingly want to interact with each other. Due to the openness and unsecure nature of the net, users often have to rely on firewalls to protect their connections. Firewalls, however, make real-time interaction and collaboration more difficult. Firewalls are also complicated to configure and expensive to install and maintain, and are inaccessible to small home offices and mobile users.

The Enclaves approach is to transform user machines into "enclaves," which are protected from outside interference and attacks. Using Enclaves, a group of collaborators can dynamically form a secure virtual subnet within which to conduct their joint business.

This paper describes the design and implementation of the Enclaves toolkit, and some applications we have built using the toolkit.

1 Motivation

Most user interaction and collaboration over the Internet have been primarily via electronic mail. More recently, groupware applications including teleconferencing have become more widely deployed, especially within large organizations. However, due to the general lack of network security support provided by these systems, groupware has been used mostly within a local-area network environment or over private networks consisting of leased lines.

On the other hand, the openness and lack of security provisions in the Internet have made firewalls a popular instrument for companies and institutions to protect their Internet connections. Such an approach poses two major problems. First, against the widely perceived potentials of the Internet, firewalls make (real-time) user interaction and collaboration much more difficult, if not impossible, especially across organizational boundaries [4]. Second, firewalls cannot be universally applied. For example, small companies or individuals users may not be able to afford the installation and maintenance of firewalls. Moreover, in the case of home users and mobile users, providing an extra, physically separate firewall machine is not practical.

We advocate a method of providing the necessary security while still taking advantage of this broad Internet connectivity through the use of secure "enclaves," where users are protected from outside interference and resistant to security attacks. (Many users "own" their desk-top or lap-top machines, so in theory they can turn off unwanted network server daemons on these enclaves.) For example, it would be very desirable for a group of colleagues, who are in different physical locations and are connected through the Internet, to be able to form a secure virtual subnet within which to conduct their joint business. It is also more convenient and economical if the secure formation of these virtual subnets does not depend on firewall machines or extra hardware other than the user machines.

In a fictitious application scenario shown in Figure 1, a group of colleagues are connected via the Internet, where each member is protected by the local enclave (denoted by the "circle" around each member). Those in possession of the correct authentication token (the "rose") can securely join the group, which is organized by the group leader. One member is using a mobile computer, and another member is a notary service whose job is to take down notes of transactions occured within this session.

In this paper, we describe the design and implementation of a toolkit, called "Enclaves", which is a con-

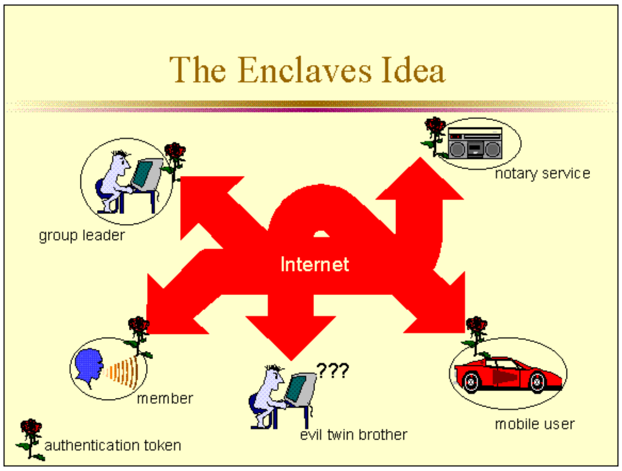

Figure 1: Using Enclaves

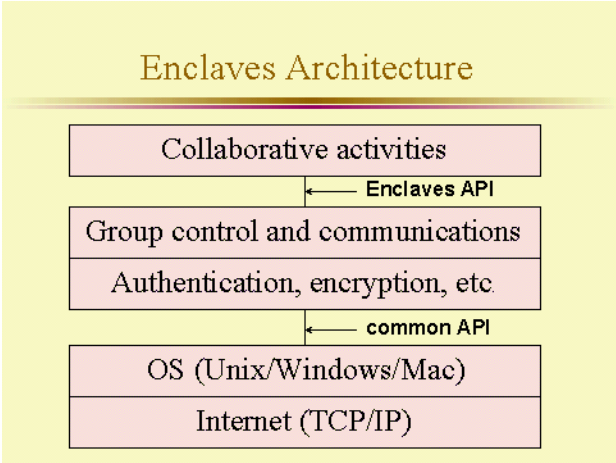

Figure 2: Layered architecture

crete demonstration of how to integrate security into user-level group-oriented applications. It shows that it is feasible for multiple users to collaborate closely, efficiently, and securely over the otherwise unsecure Internet. Our design practices should be useful to systems security architects and our toolkit should facilitate future constructions of secure, network-based collaborative applications.

2 Enclaves Design

In this section, we first describe the abstract system architecture of Enclaves. Then we explain the ideas behind this design and also compare it with prior related work.

2.1 System Architecture

Enclaves consists of subsystems layered on top of each other. Directly over the Internet and the operating system is a mechanism for authentication, which involves encryption primitives. Using authentication is the group management layer that handles group initiation, membership changes, and group dispersal. Built on top of this is a layer of abstraction for secure point-to-point communication and secure multicast over the Internet. Finally, on top of these are secure user-level group applications, an example of which is a facility for secure file sharing among group members.

As shown in Figure 2, the Enclaves toolkit depends on commonly available APIs, such as TCP/IP, and provides application builders with a layer of Enclaves API. This new API automatically provides mechanisms for user authentication, key distribution, secure group management, and secure multicast among group members.

The logical connections between a group of enclaves is as follows. The group leader occupies a special and important role in that all control flow (such as authentication and admission control) is mediated by the group leader. Any data flow that affects the entire group, such as modification to a shared file, is also mediated by the leader. Other non-crucial data can flow directly between group members to improve efficiency, after such channels are securely set up (again mediated by the group leader).

2.2 Design Rationale

Enclaves follows a design paradigm based on the following observations. First, user-level groups have different characteristics from process-level groups (such as in Isis [3] and Rampart [14]) and require different system-support mechanisms. In particular, group members have common interests but do not normally engage in identical activities.

Second, the object of sharing among group members should be the target of an application rather than the method or the medium used for the application. This is in contrast with many existing groupware systems [5], where sharing typically occurs too far away from the applications. For example, through simple replication, users may share a common text editor or the same X-window display. Sharing at such low levels has a number of problems. A major problem is security. To share a reasonably powerful editor (such as Emacs) or an X window where one can start a Unix shell, all users become one in terms of access privileges and must be totally and mutually trustworthy. This is clearly not an secure or desirable way of sharing. Another problem is undesirable interference. Suppose an Emacs editor is shared among three users, and one user starts a search for a word. Because the editor is shared, the windows used by the other two users will be affected even though they have no interest in looking for this word (e.g., the other users could not type from their keyboards without having interference from the search command that is in effect).

Moreover, user groups need to be flexible in their security controls and must be able to be dynamically formed and dispersed. In today's commonly available technology, to let a user (who might work for a competing company) be part of a group requires setting up a user account and granting the user more access than he is entitled to, sometimes with serious security implications. In Enclaves, users can be admitted to or barred from a group by a simple control mechanism, and group members share exactly those resources that have been explicitly introduced into the group for sharing, not more and not less.

Enclaves runs completely in user space without needing root privilege, and does not need much infrastructural support (such as a multicast kernel).

2.3 Technical Approaches

Enclaves assumes that initial key assignment to group members is dealt elsewhere – certification authorities can be very useful for assigning keys. More specifically, each member is assumed to share a (possibly long term) symmetric key with the group leader. Authentication (for joining in a group session) uses DES-style shared-key cryptosystems. If we can assume that members have public-key capabilities, then Enclaves can use alternative protocols that provide very strong password protection [8].

Enclaves provides an abstract layer of secure multicast. For each group session, different keys are distributed to members for encrypting multicast data. The implementation of this layer is expected to change in future versions to better handle issues such as failure recovery and efficiency [9].

The Enclaves toolkit does not yet directly support remote objects. To facilitate a simple kind of secure remote object invocation, our approach is to define a platform-independent object manipulation language for each type of shared remote objects. Each local operation, if deemed to have global effect on a remote object, is then translated into this manipulation language before being multicast to the group. There are emerging softwares with built-in remote object capabilities and we will take advantage of them as they become widely available (except that we may have to retrofit these software with security features).

Finally, our design enforces the security of groups in Enclaves through the following mechanisms:

1. The group leader and members mutually authenticate each other via authentication tokens.

2. The session key is securely distributed to new members, and is securely revoked and updated whenever a member leaves the session.

3. Group communication is secured by encrypting network traffic.

4. Group members can invoke only those predefined procedures (or operations) on remote machines (or objects) and can do so only through the secure protocol interfaces.

2.4 Comparison with Related Work

There are related works on group security in the context of distributed computing systems. The Rampart system [14] functions on the fundamental concept of secure process groups that are also virtually synchronized. A major difference between Rampart and Enclaves is that the target application domains differ significantly. Rampart aims to be a toolkit for developing highly secure and fault-tolerant systems. As a result, its protocols are relatively complicated and expensive to run. Enclaves, on the other hand, deals with user-level groups that operate in a different fashion. For example, since users do not necessarily engage in identical activities, maintaining a virtual synchrony between group members is not meaningful and can only waste system resources and slow down the applications. Nevertheless, certain types of group activities in Enclaves may benefit from the stronger semantics guaranteed by the reliable and atomic multicast protocols.

Another related work is IP-multicast [6] and its security considerations [2]. Superficially, IP-multicast and Enclaves have a lot in common: they both form user groups over the Internet and facilitate group-oriented activities. However, the two have very distinctive characteristics in reality. The typical group formed using IP-multicast technology over the M-bone [7] is large in scale, loose in user interaction, and difficult to control in terms of security. In contrast, Enclaves seeks to support close user interaction and collaboration, good real-time response, and high quality of security control. Moreover, Enclaves is extremely lightweight and runs directly over TCP/IP protocols. In comparison, the deployment of IP-multicast requires special routers or tunnels and sometimes the recompilation of operating system kernels to incorporate IP-multicast software.

The topic area popularly known as *groupware* or *computer-supported cooperative work* (CSCW), which includes collaborative systems for workflow management, shared whiteboard, teleconferencing, and shared editing (e.g., [11, 15]), is obviously related. A major difference is that these systems have not yet paid serious attention to security issues. A recent survey of the field [5] shows that only a tiny number of vendors discuss security at all or show concerns over privacy issues resulting from collaboration. The discussion often stops at how to provide password controlled access and encrypted electronic mail. Enclaves, on the other hand, is built on the basis of a secure group management layer and a secure multicast abstraction, and thus can provide very strong security guarantees.

A few systems that provide secure file sharing (such as [16]) have drawbacks, compared with Enclaves, in that they do not necessarily support real-time interaction; they also depend on good user behavior to maintain consistency of shared files. Moreover, they may have to share a real (rather than a virtual) file system.

Recently, there have been proposals to use firewalls (such as SunScreen from Sun Microsystems) to structure secure virtual networks over the open Internet. The idea is to use a pair of firewalls on each link between different parts of an organization. With automatic encryption and other support by the firewalls, machines behind these firewalls can function as if they are connected via a private network.

The secure subnet concept in the Enclaves system extends to individual users and machines, and thus can provide security functionality in the absence of firewalls, such as in the case of small companies, where maintaining firewalls is too expensive, or in the case of mobile users (e.g., using laptop computers) where a separate firewall is often unavailable. In this respect, Enclaves is closer to the concept of "joint-ventures", where partners with limited mutual trust must perform restricted collaborative activities [4]. The Enclaves system can also supplement firewall protection in that it is more user-friendly to configure corporate firewalls to support an acceptable level of protection policy while leaving certain fine-grained controls to individual users. In fact, by running Enclaves as firewalls and proxies, secure subnets can be formed in a similar fashion to using pairs of firewalls.

3 Toolkit Implementation

Enclaves 1.0 is developed on a Sun SPARC-IPX running SunOS 4.1.3, and has been tested on a SPARC-IPC and a SPARC-20 running Solaris 2.5, and on a 100-MHz PC 486 running the Slackware 2.3 distribution of Linux kernel 1.2.8. All the machines are connected to the Internet via Ethernets.

Enclaves 1.0 software is programmed using mostly the scripting language Tcl 7.3, its graphical companion Tk 3.6, and a TCP/IP extension of Tcl called Tcl-DP 3.2 [13]. In addition, there is a small amount of C and Unix Shell programming. Encryption is done with the DES in Cipher Block Chaining (CBC) mode.

3.1 Secure Group Management

The typical user-level group considered in Enclaves is a distributed group, with members located in different places. The group membership changes dynamically, and the group typically exists for a relatively short time. Because of the group's exclusive membership and the openness of the Internet, the first line of defense in establishing a group and maintaining this exclusiveness is to properly authenticate the group leader and members.

Users wishing to participate in shared activities are initially equipped with seed secrets (such as passwords or public-key certificates) which are registered with the group leader. The process of obtaining these can be facilitated via another layer of indirection of authentication. In fact, to make the toolkit easy to use for a wide range of users, we try hard to minimize the infrastructure needed to run Enclaves. Thus, in a very common mode of group operation, it is often sufficient that users can be authenticated to the group leader via (long-term or one-time) passwords that are prearranged by out-of-band methods.

The design for the protocols to authenticate group members to the group leader, and among themselves, follows established practice, so we do not dwell on these protocols except by giving one example, as shown in Table 1. The notation $\{x\}_K$ denotes x encrypted with K, and "," denotes concatenation. We use M to denote a group member and L to denote the group leader.

In message 1, the member requests joining the group. The leader checks the validity of the request message, and if satisfied, then checks the secure group policy to see if such a user can be admitted. If yes,

msg.1. M → L:	group_join, M_ID, $\{$M_ID, L_ID, leader_host_name, port_number, group_join, Time$\}_K$
msg.2. L → M:	group_admit, $\{$L_ID, M_ID, leader_host_name, port_number, group_admit, group_key, Time$\}_K$

Table 1: A simple authentication protocol

it returns message 2, which includes the group key for communication among group members. If no, the leader returns a message "authentication failed." Note that the checking of the group policy is not represented in the simple protocol description of Table 1, nor are the failure semantics. This protocol is implemented with a single RPC.

Synchronized clocks are not required to use Enclaves. If clocks may not be accurate, then random numbers (known as *nonces*) are used to prove freshness in a different, four-message authentication protocol shown in Table 2. Enclaves provides a random number generator, based on well-known algorithms [12]. In this situation, although three messages is the theoretical minimum for mutual authentication, the four-message protocol is simple to implement using two nested RPCs.

msg.1. M → L:	group_join, nonce_1
msg.2. L → M:	nonce_2
msg.3. M → L:	group_join, M_ID, $\{$M_ID, L_ID, leader_host_name, port_number, group_join, nonce_2$\}_K$
msg.4. L → M:	group_admit, $\{$L_ID, M_ID, leader_host_name, port_number, group_admit, group_key, nonce_1$\}_K$

Table 2: A nonce-based authentication protocol

In designing the protocols, we followed the robustness principle [1] and the specific format of fail-stop protocols [10]. Therefore, the protocols are somewhat less efficient than they can be; optimizations are discussed later in this paper. We do not discuss the security of these protocols except by pointing out that, in Enclaves, a user name is an arbitrary string with no spaces, and is usually bound to a secret key (e.g., a password). Once a user initiates or joins a group, his or her name is also bound to a fully qualified host name (such as *anchor.enclaves.com*) and a particular port number. This implies that a user cannot have more than one instance that is active within the same group.

When a user is admitted to the group, the leader arranges a state transfer so that the new member catches up with existing members. Currently, the group state transferred includes the group membership list and references of all shared resources such as text files.

There can be numerous types of groups, which range from totally open ones, to moderately controlled ones, and to ultra-secret ones [9]. These groups have vastly different admission policies. When a leader initiates a group, it has the option to decide upon the particular admission policy for that group. The most commonly used policy in Enclaves is what we call *leader controlled*. In such a group, the leader has the authority to invite users to join the group, decides the access-control policy, and is in charge of all group-wide activities. For example, it is the group leader who announces membership changes. To represent the policy, we use a simple access control list containing pairs of user names and their keys. The group leader can make policy changes by modifying this list "on-the-fly".

During the course of our development, we found it difficult to present the various group policies with a suitable user interface such that a user can easily visualize the implications of the policy and can conveniently change it while still maintain some coherent access control semantics. A leader-controlled group policy, the default policy, is one of the easiest to implement.

The procedure of admitting a new member is depicted in Figure 3. Note that if nonces are used, then an extra, nested RPC should be added. It would be reasonable to delay the state-transfer process till a later time, e.g., when the user explicitly requests for some state information.

When a member leaves the group, the leader initiates a group-key change protocol so that the departing user can no longer take part in group activities uninvited (using the residue group key). A key change may also be prompted when the group leader suspects a compromise somewhere. Such key updates are totally transparent to the users and in fact should be done periodically.

The group leader can invoke a program (or a protocol) to be run by a group member, and vice versa. Currently, no directly executable code is shipped across the network connecting the group members. Moreover, all such remotely executable programs are predefined, and security checking ensures that only these

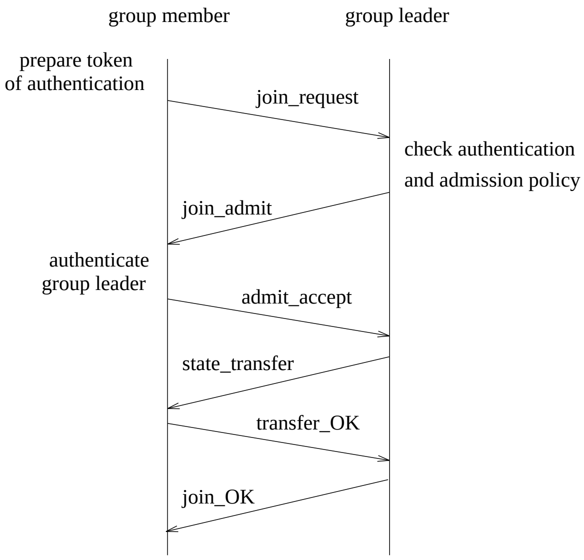

Figure 3: Admitting a new group member

programs can be invoked. Security checks are also performed on parameters when these programs are invoked.

A directory service, serving at a well-known port on a well-known machine, provides a listing of currently active groups. This service may also restrict information dissemination to certain Internet sites and to users with certain privileges. Such a service is straightforward to implement and we do not discuss further details. Of course, a group site may want to run unadvertised. In our on-going work, this service is integrated into the world-wide-web browser technology, so that the service is located at a well-known URL.

3.2 Secure Group Communication

The secure group communication abstraction provides both point-to-point communication and secure multipoint communication. Point-to-point messages are encrypted with a key shared by the two ends. Multicast messages are encrypted with the group key. For data flows, we use DES encryption in CBC mode, which provides data secrecy and integrity. Except for the first group-join request message, which has a segment of clear text noting the name of the requester, all messages are fully encrypted. (Anonymity can be achieved through other means that are not discussed here.) Following the fail-stop and the explicitness principles [1, 10], we make every message unique by including in the message the identities of the sender and of the recipient, a timestamp, and a sequence number, as follows: $\{$`Sender_ID, Recipient_ID, Sequence_Number, Timestamp, Message_Body`$\}_K$. Such a format ensures that a message cannot be reused successfully in a different place, time, or context. This rather conservative approach results in quite long messages even when the real message body is short. This can be optimized when the encryption key can identify the sender or the context cannot be misunderstood.

Currently, the secure multicast mechanism performs best-effort communication and does not provide additional semantics such as group-wide causality or atomicity [3]. For example, messages within the group are not necessarily causally or totally ordered.

There are a few reasons for this design choice. First, the weaker the multicast semantics, the faster the mechanism can be. Second, in a leader-controlled group, all crucial traffic is serialized at the leader site; therefore, no synchronization is necessary at lower levels. Moreover, users tend to tolerate failures more intelligently than processes. For example, suppose there is a temporary network link failure so that updates to a file cannot be communicated between a leader and a group member. A member can queue his changes locally and then "flush" them when the network becomes functional again. He can also push a button to refresh the file manually to pull in all the updates from the group leader. In other words, a weak low-level secure multicast semantics ensures efficiency, while upper-level mechanisms handle failures and exceptions. In the extreme case where a member has been cut off because of network partition, he can join the group again and instantly obtain the fresh group state. We believe that such an approach provides the best "economy" for general users.

When a leader becomes inactive (e.g., because of system crash or other failures), there is currently no mechanism to automatically reestablish the group with a new leader. It is not clear that such a mechanism is required or even desirable in user-level groups. For example, certain group tasks depend on a specific person as the group leader (e.g., a lead drafter of a contract), and it does not make sense for the group to continue in his absence. Certain types of user groups may benefit from automatic leader election, and this is an area for further study.

3.3 Performance Measurement

To see if our design and implementation are practical, we took some measurements on the time to complete crucial group-oriented tasks. We measured response time, that is, the time lapse between when a task is invoked and when it is completed.

The first task is initiating a group or session. This includes initializing group control data structures, reading in from disk an access control file, starting a server listening on a socket, and displaying a control panel.

The second task we measured is joining an existing group. This includes the member requesting joining by sending an encrypted message, and the group leader checking authentication and group admission policy and responding with an encrypted message to distribute a group key. These messages are followed by a state transfer phase, again via an encrypted channel. Note that this measurement obviously depends on the size of the current state, the complexity in checking the group admission policy, network latency, and speed of the crypto software.

Our measurement is done within a lightly loaded local-area network, a speeded-up software implementation of DES, a small ACL file (a dozen or so entries) and a minimal group state. Also note that the time it takes to notify all group members of this new member is not measured, because notification should be done asynchronously and some delay in this step is not crucial in a leader-controlled group.

The third task is leaving a group. In this signing-off process, the group member sends a request message, which is securely encrypted, and the leader verifies authentication and sends an acknowledgment. Note that to completely remove the member from the group, the leader must also clear all state information that depends on the member (such as locks on files) and execute the group-key change protocol. These various activities, however, are not measured since they should be done after the member is acknowledged. Their costs can vary a lot, depending on the size of the group and of its state information.

The measurements are represented in Table 3. It is not surprising that joining a group takes the longest time because it involves more interactions between the group member and the group leader.

	time in seconds
starting group	2.201
joining group	3.634
leaving group	0.569

Table 3: Measured response time of operations in a leader-controlled group (October 1995)

We must emphasize that all execution except encryption is done in a *scripting* and *interpreted* language (and in user space). Compiled code should have about 10 times speedup. Also, the prototype paid no special attention to code optimization. Given that these tasks are initiated by human users, who choose menu items, type in data (such as password), and click on buttons, a few seconds latency in each task is insignificant and tolerable. Other tasks, such as introducing a new shared file, updating a file, or posting a group message, take much less time.

We expect that these numbers will be significantly improved in our re-implementation of the Enclaves system, which has already begun. Apart from the obvious code optimization, there are a number of design choices to improve efficiency. One is to precompute a stream cipher using the group key and DES. This would save the time to initialize DES (e.g., the time to load in the key and arrange the key table), which is a very expensive part of encryption. A second method is to arrange session keys (in addition to the group key) between each member and the group leader, and then use these temporary keys in point-to-point encryption. The benefit is that such keys are fresh and can identify the sender. Thus, the message length can be reduced; for example, timestamps and sender-recipient identifications are no longer necessary.

4 Demonstration Applications

To demonstrate that Enclaves can facilitate the development of secure group applications, we now describe three simple applications we built using Enclaves 1.0. These are whiteboard, secure file sharing, and secure co-editing of text files.

4.1 Secure Whiteboard

A whiteboard for posting group messages is implemented on top of secure multicast. This board is a shared object in the sense that every message sent to the board appears on all whiteboards maintained by the group members, and messages sent by the same member do not arrive out of order. No other constraint is placed on the board. For example, a member can clear his board when it becomes full, and this clear operation does not affect other whiteboards.

In the example shown in Figure 4, a user is composing a draft response to a comment posted to the group whiteboard by a fellow group member.

The whiteboard serves as the communication medium in the absence of (or in addition to) voice and video connections between group members. The

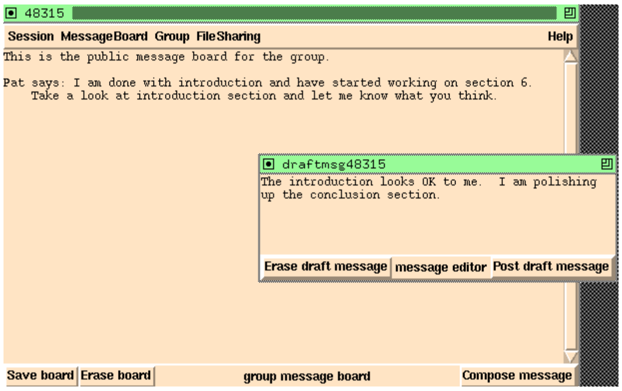

Figure 4: Composing a message for the whiteboard

content of the board can also be saved to a system log for auditing purposes.

4.2 Secure File Sharing

A rule followed in the Enclaves design is that, apart from group state information such as group key or group membership, each enclave is protected from other enclaves and the outside world in that nothing is shared until a group member explicitly introduces it into the group.

Consequently, when an Enclaves group is formed, the group shares a virtual file system space that is in most cases initially empty. This file space can be thought as something like `$GROUP_PATH/enclaves/` (where `GROUP_PATH` is a user-defined Unix shell variable) and may map to completely different positions within the file systems used by individual group members. In fact, two members could locally use entirely different types of file systems. When a member introduces a directory from his local file system into the group, the subtree becomes available to all group members. The availability of individual files depends on the protection bits on each file.

When a file is explicitly introduced into the group, a reference is entered into a shared-file list (i.e., the virtual file space in Enclaves) at every member's site. The file now is available for other group members (including the leader) to access. A member can click on the entry to access the file (e.g., view a gif file or edit a text file). Because the group leader coordinates all changes to shared files, the leader has the most up-to-date version. Thus, the open request is serviced by the leader, not by the member who originally introduced the file. This design choice can be changed, depending on the actual group structure.

4.3 Secure Co-Editing

For consistency reasons, even after opening a text file inside an editor, the member cannot edit the file (i.e., key strokes would have no effect) until he (or she) locks a region.

A region in this context is a continuous section of a file that contains zero or more characters. Locking is done by using the mouse to highlight a region of the file and then clicking the `lock region` button (or choosing from the menu bar). The lock request is sent to the server, which determines if it conflicts with existing locks, because two locked regions must not overlap. If there is no conflict, the server registers the lock and acknowledges it to the requesting member.

At this point, the locked region changes in color (say to blue) to visually signify that the lock has been granted. (We are considering non-color-based indicators for color-blind users.) Now the lock holder can edit within this region. Another member who also has the same file opened will see that this new region turns into red, signaling that it is locked by another group member, and will also see changes made by the other member reflected immediately in the local editor. When a member opens a file, all existing locks are reflected by color coding.

If a lock request is denied, the requestor is informed as to whose current locked region overlaps with the requested region. It is possible to assign priorities to users such that one with a higher priority can "grab" a locked region from one whose priority is lower. (Whether this is a desired group behavior is a separate question.) A lock can be explicitly released by clicking on the `unlock region` button. When a member exits from the editor, or leaves the group, his locks are automatically released.

To facilitate system-independent file editing, we have devised a simple language for file manipulation. We will not give all the details here, except for an example. In this language, inserting a character A is represented by a tuple of the form (`file_name, insert, A, cursor_index, region_name`). Here, the index is the relative distance from the start of the locked region.

This relative indexing has a few significant benefits. First, because the member has exclusively locked the region for writing, to maintain consistency, it is sufficient if the communication regarding this file between this member and the group member responsible for coordinating the file (in our case, the group leader) maintains a FIFO order. In other words, serialization is done via the group leader, so the regions can "float" as a result of simultaneous editing, and yet there is still no need for a global causal or total ordering among file operations. Another advantage is that local batching becomes possible. For example, if a member is typing rapidly, the changes can be saved and then sent to other members in one batch. Of course batching can be done at lower levels, such as at the multicast level, although low-level batching is best directed by higher-level applications to obtain the best desirable effect.

One way to further improve efficiency is to design the object manipulation language to be as compact as possible. For example, a series of insertions can be replaced by a single insert command. Also, there is a lot of redundancy in commands such as the following, where the operation to delete a character from a shared file translates into a long command of the form (`file_name, delete, start_index, end_index, region_name`). If the redundancy in these commands is squeezed out, the amount of encryption is also reduced.

Based on this system-independent language, different group members can in theory use their own favorite editors (which of course must have been enhanced with this language capability).

Note that because a user has ultimate control over his own enclave, he can use methods outside of Enclaves to violate a lock and to update a shared file. However, any effect of such editing is strictly local because the group leader would not accept or propagate such an editing command.

So far we have focused on how a region within a single file can be locked and shared among group members. It is not difficult to generalize such a basic facility to sharing or locking entire files or even parts of the directory structure.

Finally, files can be saved to local disks and can also be signed with digital signatures. In fact, it is easy to implement "notary" services. A traditional object notary, as in a paper-based environment, receives submissions in the form of objects or files and returns (and maintain a record of) signed certificates. A session notary, on the other hand, can be implemented as a group member who takes notes of every change made to the shared files and commits these transaction records to a non-volatile storage with the proper digital signatures.

4.4 A Quick Review Tour

To review the overall workings of the demonstration applications built on Enclaves, let us suppose that Peter, Pat, and John are to add the final touches to a joint research paper, but John is traveling and Pat is working from home. Using today's typical technology, they would probably communicate via phone and electronic mail, and one of them would coordinate changes to the manuscript. This is inconvenient.

With Enclaves, things are quite different. Peter starts the control panel by typing `enclaves` at the shell prompt. At this point, Peter can either initiate a new session or join an existing session. Peter chooses to initiate a new session for finalizing the paper. By choosing `Start new session` from the `Session` menu, Peter is prompted for a few details, including the name of an access control file (ACL). In this scenario, the ACL file (called .access-control) contains two entries for John and Pat. This process is illustrated in Figure 5.

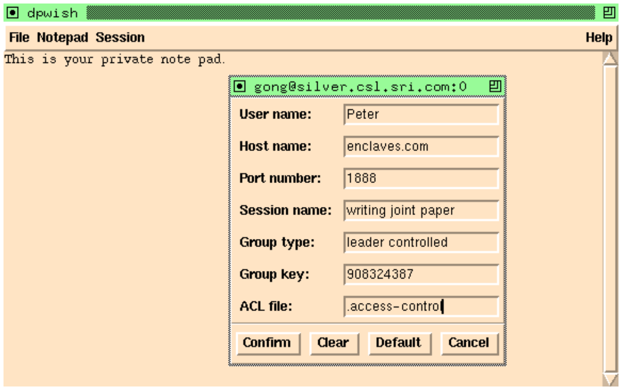

Figure 5: Starting up

By clicking the `Confirm` button, Peter starts a new session and gets a leader window for this session. Choosing the `FileSharing` menu (see Figure 6), he introduces into the group the file containing the joint paper to be shared among group members. Peter then locks the first few lines of the paper and starts refining the sentences. The locked region is automatically highlighted by a blue foreground and a white background.

A few moments later, Pat chooses to join the existing session. After giving the correct name and password, Pat is admitted to the group and gets a member window, which is similar to that of a group leader, except that it does not have leader functions such as access control. As Pat joins, a message is automatically posted on the group message board notifying this membership change. Figure 6 shows Peter's display at this point.

Pat can see from the shared-file list (under the `FileSharing` menu) that the joint paper is already inside the group and thus opens it. Immediately, he sees

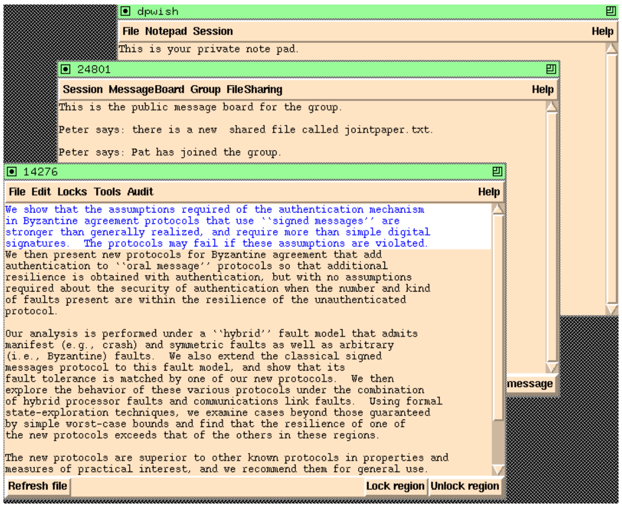

Figure 6: A group leader

that Peter has locked the first few lines and is working on them. He can instantly observe the changes as Peter makes them. Pat then locks the next paragraph and starts polishing.

Soon afterward, John also joins from the other end of the country via a local modem dial-up and then an Internet connection. The three of them communicate via the message board, working on the sections they choose. Figure 7 is a snapshot of Pat's editor window.

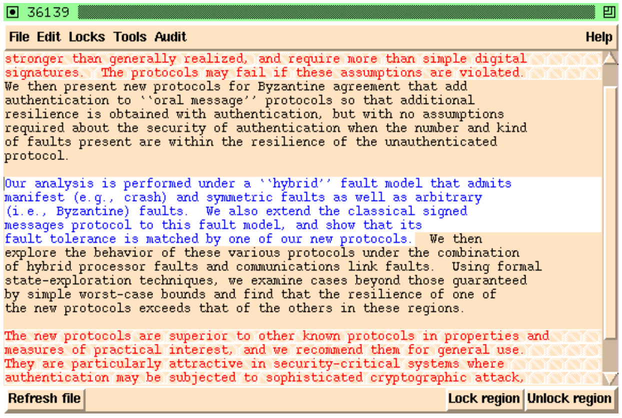

Figure 7: Co-editing

The top region is heavily shaded to indicate that it is currently locked by someone else (Peter, in this case). Similarly, the bottom region is locked by John. The region locked by Pat is the middle section with a solid white background. The rest of the paper – the gray regions – is currently unclaimed. (The locked and unlocked regions are color coded and will be more pleasant to see on a color display.) Each member can modify only a region locked by himself or herself and the group leader mediates the lock-granting process so that no two regions can overlap. These controls ensure that the replicated file copies maintain consistency.

When the completed paper is saved on disk and sent to printers, the group is simply dispersed. Throughout this session, all crucial communications over the Internet are encrypted. These communications include the authentication tokens and messages (passwords are never sent in the clear), messages posted on the group board, the text file, and any changes to it. An outsider cannot join the group for lack of a valid password, and cannot eavesdrop on useful information because of encryption. (We do not consider covert channels or traffic analysis.)

5 Summary and Ongoing Work

We have presented the design and implementation of Enclaves, which is an Internet-environment toolkit that makes it easier to build secure group-oriented applications. The toolkit contains mechanisms for dynamic group formation, secure group management, and secure group communication. Our prototype and preliminary performance measurements indicate that organizing secure, closely coupled user groups over the Internet is practical with currently available technology. Enclaves does not require privilege to install or run, and does not require a complicated access control mechanism.

To demonstrate the usability of the toolkit, we have built a secure file-sharing application that allows members of the same group to introduce files for sharing and co-editing within the group. The application allows concurrent editing via the concept of a locked region, which can be as small as containing zero or a single character, and a system-independent text file manipulation language.

We are currently developing version 1.1 to use Tcl7.5/Tk4.1, which are supported on Unix platforms as well as on Microsoft Windows and MacOS machines. One of the goals is to reduce platform dependent codes, which are mostly related to cryptographic operations. Parallel to this effort, we are developing Enclaves 2.0 from scratch, where integrating with web technology is a major objective. Another priority item is to incorporate secure audio and video into Enclaves, which will greatly improve coordination among group members. We are in the process of making the API and the code available.

Acknowledgements

This work is supported by SRI's internal research and development program. I am grateful for managerial support from the Engineering Research Group. Many thanks to Sam Owre for helping me out of various implementation traps, and to Don Nielson and Peter Neumann for technical and editorial comments.

References

[1] R.J. Anderson. Why Cryptosystems Fail. *Communications of the ACM*, 37(11):32–40, November 1994.

[2] A.J. Ballardie and J. Crowcroft. Multicast-Specific Security Threats and Counter-Measures. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, San Diego, California, February 1995.

[3] K.P. Birman. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM*, 36(12):37–53/103, December 1993.

[4] W.R. Cheswick and S.M. Bellovin. *Firewalls and Internet Security.* Addison-Wesley, 1994.

[5] D. Coleman and R. Khanna, editors. *Groupware Technology and Applications.* Prentice-Hall, Upper Saddle River, New Jersey, 1995.

[6] S. Deering. Host Extensions for IP Multicasting. Request for Comments 1112, Internet Network Working Group, August 1989.

[7] H. Eriksson. MBONE: The Multicast Backbone. *Communications of the ACM*, 37(8):54–60, August 1994.

[8] L. Gong. Optimal Authentication Protocols Resistant to Password Guessing Attacks. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 24–29, County Kerry, Ireland, June 1995.

[9] L. Gong and N. Shacham. Multicast Security and Its Extension to a Mobile Environment. *ACM-Baltzer Journal of Wireless Networks*, 1(3):281–295, October 1995.

[10] L. Gong and P. Syverson. Fail-Stop Protocols: An Approach to Designing Secure Protocols. In *Proceedings of the 5th IFIP Working Conference on Dependable Computing for Critical Applications*, Dependable Computing and Fault-Tolerant Systems, pages 44–55, Urbana-Champaign, Illinois, September 1995. Springer-Verlag.

[11] M. Knister and A. Prakash. Issues in the Design of a Toolkit for Supporting Multiple Group Editors. *Computing Systems*, 6(2):135–166, Spring 1993.

[12] D.E. Knuth. *The Art of Computer Programming, Vol.2: Seminumerical Algorithms.* Addison-Wesley, Reading, Massachusetts, 1969. Revised edition.

[13] J.K. Ousterhout. *Tcl and the Tk Toolkit.* Addison Wesley, Menlo Park, California, 1994.

[14] M. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, Fairfax, Virginia, November 1994.

[15] M. Roseman and S. Greenberg. Building Real Time Groupware with GroupKit, a Groupware Toolkit. *ACM Transactions on Computer Human Interaction*, 1996. To appear.

[16] T. Takahashi, A. Shimbo, and M. Murota. File-Based Network Collaboration System. In *Proceedings of the 5th USENIX UNIX Security Symposium*, pages 95–104, Salt Lake City, Utah, June 1995.