Intrusion-Tolerant Enclaves*

To be presented at the IEEE Conference on Security and Privacy, Oakland, CA, May 2002

Bruno Dutertre, Valentin Crettaz, Victoria Stavridou
System Design Laboratory, SRI International, Menlo Park, CA
{bruno,valentin,victoria}@sdl.sri.com

Abstract

Despite our best efforts, any sufficiently complex computer system has vulnerabilities. It is safe to assume that such vulnerabilities can be exploited by attackers who will be able to penetrate the system. Intrusion tolerance attempts to maintain acceptable service despite such intrusions. This paper presents an application of intrusion-tolerance concepts to Enclaves, a software infrastructure for supporting secure group applications. Intrusion tolerance is achieved via a combination of Byzantine fault-tolerant protocols and secret sharing techniques.

1 Introduction

Intrusion tolerance is the application of fault-tolerance methods to security. It assumes that system vulnerabilities cannot be totally eliminated, and that external attackers or malicious insiders will identify and exploit these vulnerabilities and gain illicit access to the system. The objective of intrusion tolerance is to maintain acceptable, though possibly degraded, service despite intrusions in parts of the system.

This paper discusses the intrusion-tolerant version of Enclaves, a lightweight platform for building secure group applications. To support such applications, Enclaves provides a secure group communication service and associated group-management and key-distribution functions. The original Enclaves system [16] and its successor [9] have a centralized architecture. An application consists of a set of group members who cooperate and communicate via a single group leader. This leader is responsible for all group-management activities, including authenticating and accepting new members, distributing cryptographic keys, and distributing group-membership information. Since the leader plays a critical role, it is an attractive target for attackers. Breaking into the leader can immediately lead to loss of confidentiality, interrupted communication, or other forms of denial of service.

The intrusion-tolerant Enclaves architecture removes this single point of failure by distributing the group-management functions among n leaders. The system uses a Byzantine fault-tolerant protocol for leader coordination and a verifiable secret sharing scheme for generating and distributing cryptographic keys to group members. This new architecture is intended to tolerate the compromise of up to f leaders, where $3f + 1 \leqslant n$. Compromised leaders are assumed to be under the full control of an attacker and to have Byzantine behavior, but it is also assumed that the attacker cannot break the cryptographic algorithms used. Under these assumptions, Enclaves ensures the confidentiality and integrity of group communication, as well as proper group-management services.

The remainder of the paper describes the intrusion-tolerant Enclaves in greater detail. Section 2 gives an overview of the architecture and goals of Enclaves. Section 3 presents the protocols and secret sharing schemes used. Section 4 describes the current implementation of Enclaves. Section 5 assesses the security of the system, and Section 6 discusses related work.

2 Overview

2.1 Enclaves Services

A group-oriented application enables users to share information and collaborate via a communication network such as the Internet. Enclaves is a lightweight software infrastructure that provides security services for such applications [16]. Enclaves provides services for creating and managing groups of users of small to medium sizes, and enables the group members to communicate securely. Access to an active group is restricted to a set of users who must be preregistered, but the group can be dynamic: authorized users can freely join, leave, and later rejoin an active application.

The communication service implements a secure multicast channel that ensures integrity and confidentiality of group communication. All messages originating from a

*This material is based upon work supported by the Space and Naval Warfare Systems Center – San Diego, under Contract No. N66001-00-C-8001.

group member are encrypted and delivered to all the other members of the group. For efficiency reasons, Enclaves provides best-effort multicast and does not guarantee that messages will be received, or received in the same order, by all members. This is consistent with the goal of supporting collaboration between human users, which does not require the same reliability guarantees as distributing data between servers or computers [16].

The group-management services perform user authentication, access control, and related functions such as key generation and distribution. All group members receive a common group key that is used for encrypting group communication. A new group key is generated and distributed every time the group composition changes, that is, whenever a user enters or leaves the group. Optionally, the group key can also be refreshed on a periodic basis. Enclaves also communicates membership information to all group members. On joining the group, a member is notified of the current group composition. Once in the group, each member is notified when a new user enters or a member leaves the group. Thus, all members know who is in possession of the current group key.

In summary, Enclaves enables users to be authenticated and to join a groupware application. Once in a group, a user A is presented with a group view, that is, the list of all the other group members. The system is intended to satisfy the following security requirements:

- *Proper authentication and access control:* Only authorized users can join the application and an authorized user cannot be prevented from joining the application.
- *Confidentiality of group communication:* Messages from a member A can be read only by the users who were in A's view of the group at the time the message was sent.
- *Integrity of group communication:* A group message received by A was sent by a member of A's current view, was not corrupted in transit, and is not a duplicate.

2.2 Centralized Architecture

The original version of Enclaves [16] and a more recent version with improved protocols [9] rely on the centralized architecture shown in Figure 1. In this architecture, a single group leader is responsible for all group-management activities. The leader is in charge of authenticating and accepting new group members, generating group keys and distributing them to members, and distributing group membership information. With such an architecture, the security requirements are satisfied if the leader and all the group members are trustworthy, but the system is not intrusion tolerant. Proper service requires that the leader be trusted and never compromised. A single intrusion on the leader can easily lead to denial of service. For example, a compromised leader can interrupt group communication or prevent authorized users from joining the group. Since the leader is always in possession of the latest group key, an attacker breaking into the leader's computer can also gain access to all group communication.

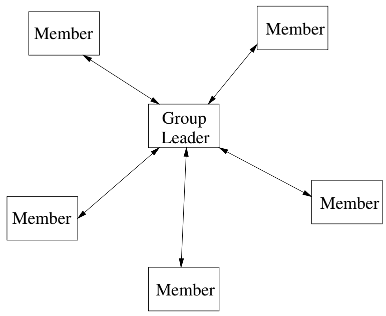

Figure 1. Centralized Enclaves Architecture

Increasing the resilience of the Enclaves infrastructure requires removing this single point of failure, to guarantee that group communication will remain secure even after some components of the system have been compromised.

2.3 Intrusion-Tolerant Architecture

The architecture of the intrusion-tolerant version of Enclaves is shown in Figure 2. The group and key management functions are distributed across n leaders. The leaders communicate with each other and with users via an asynchronous network. Messages sent on this network are assumed to be eventually received, but no assumptions are made on the transmission delays and on the order of reception of messages. The architecture is designed to tolerate up to f compromised leaders, where $3f + 1 \leqslant n$. Compromised leaders are assumed to be under the full control of an attacker; they have Byzantine behavior and can collude with each other.

The security requirements are the same as previously, and assume that a fixed list of authorized participants is specified before an application starts. The new objective is now to ensure that these requirements are satisfied even if up to f leaders are compromised.

For proper group management, any modification of the group composition requires agreement between the nonfaulty leaders. These leaders must agree before accepting a new member or determining that an existing member has left. Ideally, one would like all nonfaulty leaders to maintain agreement on the group composition. Unfortunately, this requires solving a consensus problem, in an asynchronous network, under Byzantine faults. As is well

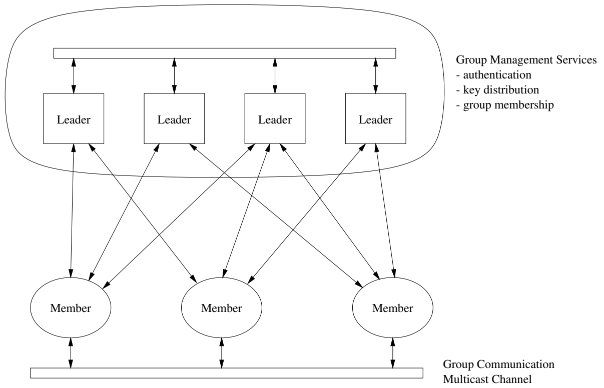

Figure 2. Intrusion-Tolerant Architecture

known, there are no deterministic algorithms for solving this problem [12]. Randomized algorithms (e.g., [3]) or algorithms relying on failure detectors (e.g., [4, 20]) could be applicable, but these algorithms tend to be complex and expensive. Instead, a weaker form of consistency property is sufficient for satisfying Enclaves's security requirements. The algorithm we use is similar to consistent broadcast protocols such as Bracha and Toueg's protocol [2, 18]. Combined with an appropriate authentication procedure, this algorithm ensures that any authorized user who requests to join the group will eventually be accepted. Unlike Byzantine agreement, this algorithm does not guarantee that users are accepted in the same order by all leaders. However, this does not lead to a violation of the confidentiality or integrity properties. If the group becomes stable, all nonfaulty leaders eventually reach a consistent view of the group.

As in previous Enclaves implementations, a common group key is shared by the group members. A new key is generated by the leaders whenever the group changes. The difficulty is to generate and distribute this key in an intrusion-tolerant fashion. All group members must obtain the same valid group key, despite the presence of faulty leaders. The attacker must not be able to obtain the group key even with the help of f faulty leaders. These two requirements are satisfied by using a secret sharing scheme proposed by Cachin et al. [3]. In the Enclaves framework, this scheme is used by leaders to independently generate and send individual shares of the group key to group members. The protocol is configured so that $f + 1$ shares are necessary for reconstructing the key. A share is accompanied with a description of the group to which it corresponds and a "proof of correctness" that is computationally hard to counterfeit. This allows group members to obtain strong evidence that a share is valid, and prevents faulty leaders from disrupting group communication by sending invalid shares.

To join an application, a user A must contact $2f+1$ leaders. Once in the group, A remains connected to these leaders and receives key and group update messages from them. A majority of consistent messages (i.e., $f + 1$) must be received before A takes any action. For example, A changes its current group key only after receiving at least $f + 1$ valid key shares from distinct leaders, and checking that these shares correspond to the same group description. This ensures A that the new group key is valid and that at least one share came from a nonfaulty leader.

Intrusion tolerance in Enclaves relies then on the combination of a cryptographic authentication protocol, a Byzantine fault-tolerant leader-coordination algorithm, and a secret sharing scheme. These protocols are presented in greater detail in the section that follows.

3 Protocols

3.1 Authentication

To join the group, a user A must first initiate an authentication protocol with $2f + 1$ distinct leaders. A is accepted as a new group member if it is correctly authenticated by at least $f + 1$ leaders. This ensures that f faulty leaders cannot prevent an honest user from joining the group, and conversely that f faulty leaders cannot allow an unauthorized user to join the group.

For authentication purposes, all users registered as authorized participants in an application share a long-term secret key with each leader. If L_i is one of the leaders, A has a long-term key $P_{a,i}$ that is known by L_i and A. In the current implementation, $P_{a,i}$ is computed from A and L_i's identities, and A's password by applying a one-way hash function. This ensures with high probability that two distinct leaders L_i and L_j do not have the same key for A. Intrusion at a leader L_i can reveal key $P_{a,i}$ to the attacker but does not reveal A's password or $P_{a,j}$. Thus, access to up to f long-term keys $P_{a,i}$ does not enable an attacker to impersonate A.

The following protocol is used by A to authenticate with L_i

1. $A \to L_i$: $\mathsf{AuthInitReq}, A, L_i, \{A, L_i, N_1\}_{P_{a,i}}$
2. $L_i \to A$: $\mathsf{AuthKeyDist}, L_i, A,$
   $\{L_i, A, N_1, N_2, K_{a,i}\}_{P_{a,i}}$
3. $A \to L_i$: $\mathsf{AuthAckKey}, A, L_i, \{N_2, N_3\}_{K_{a,i}}$.

As a result of this exchange, A is in possession of a session key $K_{a,i}$ that has been generated by L_i. All group-management messages from L_i to A are encrypted with $K_{a,i}$. Thus, a secure channel is set up between A and L_i that ensures confidentiality and integrity of all group-management messages from L_i to A. Nonces and acknowledgments protect against replay as discussed in [9]. The key $K_{a,i}$ is in use until A leaves the group. A fresh session key will be generated if A later rejoins the group.

3.2 Leader Coordination

If a nonfaulty leader L_i successfully authenticates A, L_i does not immediately add A as a new group member. Instead, the leader coordination algorithm described in Figure 3 is executed. A similar algorithm is used to coordinate leaders when a member leaves the group.

Leader L_i runs the following protocol

After successful authentication of A,
L_i sends $\langle \text{Propose}, i, A, n_i \rangle_{\sigma_i}$ to all leaders

After receiving $f + 1$ valid $\langle \text{Propose}, j, A, n_j \rangle_{\sigma_j}$
from different leaders, L_i sends $\langle \text{Propose}, i, A, n_i \rangle_{\sigma_i}$
to all leaders if it has not already done so

When L_i receives $n - f$ valid $\langle \text{Propose}, j, A, n_j \rangle_{\sigma_j}$
from $n - f$ distinct leaders, L_i accepts A as a
new member

Figure 3. Leader Coordination Protocol

The notation $\langle \ldots \rangle_{\sigma_i}$ denotes a message digitally signed by L_i. The constant n_i is used to protect against replay attacks. Each leader maintains a local integer variable n_i and its local view M_i of the current group members. M_i is updated and n_i is incremented every time L_i accepts a new member or removes an existing member. The message $\langle \text{Propose}, A, n_j \rangle_{\sigma_j}$ is considered valid by L_i if the signature checks, if $n_j \geqslant n_i$, and if A is not a member of M_i. The pair $\langle n_i, M_i \rangle$ is L_i's current view of the group. In Figure 3, L_i must include its own $\langle \text{Propose} \ldots \rangle$ message among the $n - f$ messages necessary before accepting A.

This algorithm is a variant of existing *consistent broadcast algorithms* [2, 18]. It satisfies the following properties as long as no more than f leaders are faulty.

- *Consistency:* If one nonfaulty leader accepts A then all nonfaulty leaders eventually accept A.
- *Liveness:* If $f + 1$ nonfaulty leaders announce A, then A is eventually accepted by all nonfaulty leaders.
- *Valid Authentication:* If one nonfaulty leader accepts A then A has been announced, and thus authenticated, by at least one nonfaulty leader.

The last property prevents the attacker from introducing unauthorized users into the group. Conversely, if A is an authorized user and correctly executes the authentication protocol, A will be announced by $f + 1$ nonfaulty leaders, and thus will eventually be accepted as a new member by all nonfaulty leaders.

The protocol works in an asynchronous network model where transmission delays are unbounded. It does not ensure that all nonfaulty leaders always have a consistent group view. Two leaders L_i and L_j may have different sets M_i and M_j for the same view number $n_i = n_j$. This happens if several users join or leave the group concurrently, and their requests and the associated Propose messages are received in different orders by L_i and L_j. If the group becomes stable, that is, no requests for join or leave are generated in a long interval, then all nonfaulty leaders eventually converge to a consistent view. They communicate this view and the associated group-key shares to all their clients who all also eventually have a consistent view of the group and the same group key.

Temporary disagreement on the group view may cause nonfaulty leaders to send valid but inconsistent group-key shares to some members. This does not compromise the security requirements of Enclaves but may delay the distribution of a new group key.

3.3 Group-Key Management

The group-key management protocol relies on secure secret sharing. Each of the n leaders knows only a share of the group key, and at least $f + 1$ shares are required to reconstruct the key. Any set of no more than f shares is insufficient. This ensures that compromise of at most f leaders does not reveal the group key to the attacker. In most secret sharing schemes, n shares $s_1, \ldots, s_n$ are computed from a secret s and distributed to n shareholders. The shares are computed by a trusted dealer who needs to know s. In Enclaves, a new secret s and new shares must be generated whenever the group changes. This must be done online and without a dealer, to avoid a single point of failure. A further difficulty is that some of the parties involved in the share renewal process may be compromised.

A solution to these problems was devised by Cachin et al. in [3]. In their protocol, the n shareholders can individually compute their share of a common secret s without knowing or learning s. One can compute s from any set of $f + 1$ or more such shares, but f shares or fewer are not sufficient. The shares are all computed from a common value $\tilde{g}$ that all shareholders know. In our context, the shareholders are the group leaders and $\tilde{g}$ is derived from the group view using a one-way hash function. Leader L_i computes its share s_i using a share-generation function S, the value $\tilde{g}$, and a secret x_i that only L_i knows: $s_i = S(\tilde{g}, x_i)$. Leader L_i also gives a proof that s_i is a valid share for $\tilde{g}$. This proof does not reveal information about x_i but enables group members to check that s_i is valid.

The protocol proposed by Cachin et al. [3] requires a trusted dealer to set up a number of public and private keys in an initialization phase. This can be performed offline, and the dealer is not involved in any of the subsequent computations. The share computations are performed individually by the leaders. The share validity checks and group key construction are performed individually by group members. This protocol is related to verifiable secret sharing [6, 11, 19] and, more closely, to to threshold signature schemes [7, 14, 23].

The secrecy properties of the protocol rely on the hardness of computing discrete logarithms in a group of large prime order. Such a group G can be constructed by selecting two large prime numbers p and q such that $p = 2q + 1$ and defining G as the unique subgroup of order q in $\mathbb{Z}_p^*$. The dealer chooses a generator g of G and performs the following operations:

- Select randomly $f + 1$ elements $a_0, \ldots, a_f$ of $\mathbb{Z}_q$. These coefficients define a polynomial of degree f in $\mathbb{Z}_q[X]$:

$$F = a_0 + a_1 X + \ldots + a_f X^f.$$

- Compute $x_1, \ldots, x_n$ of $\mathbb{Z}_q$ and $g_1, \ldots, g_n$ of G as follows:

$$\begin{aligned} x_i &= F(i) \\ g_i &= g^{x_i}. \end{aligned}$$

  The numbers $x_1, \ldots, x_n$ must then be distributed secretly to the n leaders $L_1, \ldots, L_n$, respectively. The generator g and the elements $g_1, \ldots, g_n$ are made public. They must be known by all users and leaders.

As in Shamir's secret sharing scheme [22], any subset of $f+1$ values among $x_1, \ldots, x_n$ allows one to reconstruct F by interpolation, and then to compute the value $a_0 = F(0)$. For example, given $x_1, \ldots, x_{f+1}$, one has

$$a_0 = \sum_{i=1}^{f+1} b_i x_i,$$

where b_i is obtained from $j = 1, \ldots, f + 1$ by

$$b_i = \frac{\prod_{j \neq i} j}{\prod_{j \neq i} (j - i)}.$$

By this interpolation method, one can compute $\tilde{g}^{a_0}$ for any $\tilde{g} \in G$ given any subset of $f+1$ values among $\tilde{g}^{x_1}, \ldots, \tilde{g}^{x_n}$. For example, from $\tilde{g}^{x_1}, \ldots, \tilde{g}^{x_{f+1}}$, one gets

$$\tilde{g}^{a_0} = \prod_{i=1}^{f+1} (\tilde{g}_i)^{b_i}. \tag{1}$$

As discussed previously, leader L_i maintains a local group view $\langle n_i, M_i \rangle$. L_i's share s_i is a function of the group view, the generator g, and L_i's secret value x_i. L_i first computes $\tilde{g} \in G$ using a one-way hash function H_1:

$$\tilde{g} = H_1(n_i, M_i).$$

The share s_i is then defined as

$$s_i = \tilde{g}^{x_i}.$$

The group key for the view $\langle n_i, M_i \rangle$ is defined as

$$K = H_2(\tilde{g}^{a_0}),$$

where H_2 is another hash function from G to $\{0, 1\}^k$ (k is the key length). Using equation (1), a group member can compute $\tilde{g}^{a_0}$ given any subset of $f + 1$ or more shares for the same group view. The security of this approach is proved in [3]. Under a standard intractability assumption, it is computationally infeasible to compute K knowing fewer than $f + 1$ shares. It is also infeasible for an adversary to predict the values of future group keys K even if the adversary corrupts f leaders and has access to f secret values among $x_1, \ldots, x_n$.

Equation (1) allows a group member to compute $\tilde{g}^{a_0}$ and K from $f + 1$ valid shares of the form $s_i = \tilde{g}^{x_i}$. However, a compromised leader L_i could make the computation fail by sending an invalid share $s_i \neq \tilde{g}^{x_i}$. L_i could also cause different members to compute different K's by sending different shares to each. To protect against such attacks, the share s_i is accompanied with a proof of validity. This extra information enables a member to check that s_i is equal to $\tilde{g}^{x_i}$ with very high probability. The verification uses the public value g_i that is known to be equal to g^{x_i} (since the dealer is trusted). To prove validity without revealing x_i, leader L_i generates evidence that

$$\log_{\tilde{g}} s_i = \log_g g_i.$$

This uses a technique proposed by Chaum and Pedersen [5]. To generate the evidence, L_i randomly chooses a number y in $\mathbb{Z}_q$ and computes

$$\begin{aligned} u &= g^y \\ v &= \tilde{g}^y. \end{aligned}$$

Then L_i uses a third hash function H_3 from G^6 to $\mathbb{Z}_q$ to compute

$$\begin{aligned} c &= H_3(g, g_i, u, \tilde{g}, s_i, v) \\ z &= y + x_i c. \end{aligned}$$

The proof that s_i is a valid share for $\tilde{g}$ is the pair (c, z). The information sent by L_i to a group member A is then the tuple

$$\langle n_i, M_i, s_i, c, z \rangle.$$

This message is sent via the secure channel established between A and L_i after authentication. This prevents an attacker in control of f leaders from obtaining extra shares by eavesdropping on communications between leaders and clients.

On receiving the above message, a group member A evaluates $\tilde{g} = H_1(n_i, M_i)$ and

$$\begin{aligned} u' &= g^z / g_i^c \\ v' &= \tilde{g}^z / s_i^c. \end{aligned}$$

A accepts the share as valid if the following equation holds:

$$c = H_3(g, g_i, u', \tilde{g}, s_i, v').$$

If this check fails, s_i is not a valid share and A ignores it. Once A receives $f+1$ valid shares corresponding to the same group view, A can construct the group key. Since A maintains a connection with at least $f+1$ honest leaders, A eventually receives at least $f+1$ valid shares for the same view, once the group becomes stable.

Cachin et al. [3] prove that it is computationally infeasible, in the random oracle model, for a compromised leader L_i to produce an invalid share s_i' and two values c and z that pass the share-verification check.

3.4 Cryptographic Material

The following cryptographic keys and secret material must be distributed to the leaders and registered users:

- Each leader L_i must own a private key to sign messages when executing the leader-coordination protocol. The corresponding public key must be known by all the other leaders.
- L_i must also hold the secret x_i used to generate shares of group keys. The corresponding verification key g_i must be known by all the registered users.
- For every registered user A and leader L_i, a secret longterm key $P_{a,i}$ is shared by A and L_i. This key is used for authentication.

4 Implementation

Enclaves is currently implemented in Java, using Sun Microsystems' Java 2 SDK 1.3.1 and the Cryptix 3.2 cryptographic libraries.[1] The source consists of around 9,000 lines of code in approximately 100 classes.

The software is organized in two main modules as described in Figure 4. A set of classes implements the core Enclaves functionalities, namely, the authentication, group management, and key-management functions described previously. On top of this basis, a user interface is available that can be customized to support diverse applications. The interface allows users to authenticate and log in to an Enclaves group and displays status information, including the list of members. Applications can be easily incorporated into this interface via a "plugin" mechanism.

[1] http://www.cryptix.org

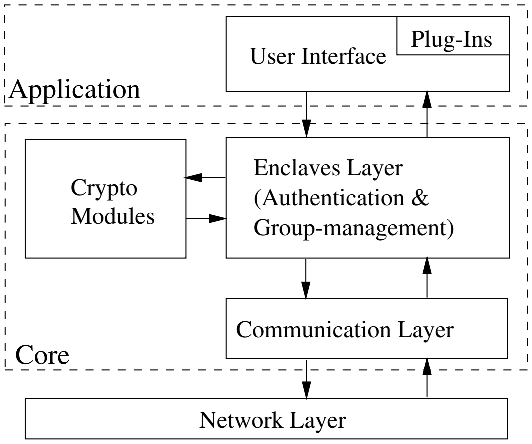

Figure 4. Main Software Modules

4.1 Core Enclaves

The core classes implement the protocols and algorithms described previously. These classes are organized in an Enclaves layer responsible for authentication and group-management services, a cryptographic module, and a communication layer that interface with Java networking functions. In the current prototype, group communication (between group members) as well as communication between leaders is implemented using IP multicast. Leader-to-client connections rely on TCP.

Enclaves uses Cryptix 3.2 as a cryptographic module, but other providers complying with the Java Security Architecture can be used. Enclaves uses a symmetric-key encryption algorithm (currently triple DES), a digital signature algorithm (DSA), and secure hashing algorithm (SHA). These can be easily replaced by other algorithms with similar functionality. The secret sharing algorithm of Section 3.3 was not available off the shelf and had to be implemented.

4.2 Plugins

Enclaves provides a simple user interface that can be customized for various applications via the use of "plugins". The plugins are loaded on startup and executed as the user requires. This architecture allows several applications to coexist and run concurrently in the same Enclaves group. The underlying support classes transparently encrypt all application messages and distribute them to all group members. Conversely, messages received from the group are decrypted and dispatched to the relevant plugin.

Communication between an application and the underlying Enclaves layer must follow the interface described in Figure 5. A plugin simply needs to implement the three methods of abstract class `PlugIn`. Method `buildGUI` is

```
public abstract class PlugIn
       extends JFrame implements ... {

  protected abstract void buildGUI();

  protected abstract void
       receiveMessage(Message m);

  protected final boolean
       sendMessage(byte[] msg);
}
```

Figure 5. Plugin Interface

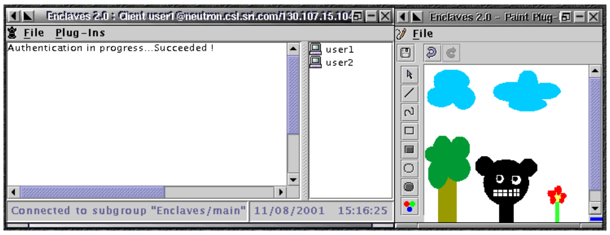

Figure 6. Whiteboard Application

invoked by the user interface for the application to initialize. Afterwards, communication between the application and the Enclaves middleware is performed via two methods for sending and receiving messages. When a plugin is ready to be deployed, the developer must package it and every resource it needs into a JAR file and put it in a specific directory. The new plugin is then loaded and available to users.

Currently, four basic plugins have been developed for Enclaves: a shared whiteboard application (*Paint*), a messaging application allowing users to send text messages (*Chat*), a file transfer application for multicasting data files (*FTP plugin*), and a *Sound plugin* for multicast of streaming audio. Figure 6 shows a screen dump of the *Paint* application.

5 Assessment

Enclaves's security requirements can be shown to theoretically hold if no more than f leaders are compromised, no group member is compromised, the attacker does not break the cryptographic algorithms, and the network assumptions are satisfied.[2] Obviously, none of these assumptions can be guaranteed to hold with certainty in the real world. What really matters is the effort it takes an attacker to compromise security by making one of these assumptions fail.

We can be reasonably confident that the cryptographic and secret sharing protocols used are hard to break. If weaknesses are discovered, the Enclaves implementation makes it easy to change cryptographic primitives.

As in any group-communication system, if an attacker can compromise a member machine and get hold of the group key, or if one member is nontrustworthy, then confidentiality is lost. Clearly, there is no absolute defense against this vulnerability as it is the function of the system to distribute data to all group members. Mitigating measures could be implemented, such as requiring members to periodically reauthenticate before sending them a new key, or relying on intrusion detection and expel members suspected of being compromised.

Enclaves can also be vulnerable to network-based denial-of-service attacks based on flooding. This is not specific to Enclaves, as current TCP/IP protocols make it difficult to defend against such attacks in any system. However, the distributed architecture of Enclaves increases the resilience of the system to such attacks. A useful property is also that group communication can continue even after a successful denial-of-service attack on the leaders. Such an attack prevents new users from joining an application and the group key from being refreshed but does not immediately affect the users already in the group.

Clearly, the intrusion-tolerant architecture of Enclaves improves security only if it is substantially harder for an attacker to penetrate several leaders than a single one. Every attempt should then be made to prevent common vulnerabilities, so that the same attack does not succeed on all leaders. This requires diversity. Leaders should use different hardware and operating systems, and, as a minimum, different implementations of the Java Virtual Machine. It is also desirable to put the different leaders under the responsibility of different administrators, as a protection against the insider threat.

Even with such measures, resilience is not absolute. With enough time and resources, a determined attacker can compromise a sufficient number of leaders to break the system. Intrusion tolerance should be seen as the ultimate defense, intended to give the defender extra time to detect and react to attacks before too much damage is done. Intrusion tolerance can be effective only if combined with traditional measures to avoid and remove vulnerabilities in system components, and with intrusion detection mechanisms to help respond to attacks before it is too late.

6 Related Work

Early work on intrusion tolerance [8,10,13] applied fault tolerance and dependability concepts to common services of distributed systems, such as file storage. These approaches already used secret sharing, encryption, and redundancy to maintain security and availability in the presence of compromised components. The same techniques

[2] And, of course, the Java code correctly implements the protocols.

are being applied, for example, for building survivable information storage [27].

Wu et al. [26] present examples of intrusion-tolerant applications that rely on threshold cryptography. A number of servers, each holding a share of an RSA private key can be contacted by a client for operations such as signing or decrypting messages. Each server performs a local computation using its key share, and the individual results from each server are combined by the client. This is similar to the group key protocol of Cachin et al. [3] that is used in Enclaves.

Enclaves relies on the same techniques as these other systems, namely, secret sharing techniques and fault-tolerant distributed algorithms. The main novelty is the type of application supported by the intrusion-tolerant Enclaves, that is, secure group communication.

Gong [15] describes a fault-tolerant protocol for authentication and key distribution that relies on several authentication servers, some of which may be compromised. The system provides an authentication service to establish secure point-to-point connections between two users A and B, and distribute a shared symmetric key to both. Proper service is guaranteed as long as the majority of the servers are not compromised. The n leaders of Enclaves provide a similar authentication service, but for a group rather than a pair of users. They are also supporting other group-management activities including renewing the group key and maintaining the current set of group members, which requires close coordination of the leaders.

Other systems such as Secure Spread [1], Ensemble [21], or Horus [25], provide secure group communication between distributed computers. These infrastructures provide group-membership and multicast services that support a variety of network-level faults, including network partitioning. Key-agreement or key-distribution protocols (e.g., [24]) are implemented on top of these services, and cryptography ensures confidentiality and integrity of group communication. These systems typically provide security against attacks on network traffic but are not designed for intrusion tolerance.

Enclaves is also related to systems such as Rampart [20] or the SecureRing [17], which use fault-tolerant protocols for consistent operation of a set of servers. These systems implement consensus in asynchronous networks under Byzantine fault models, but rely on some form of failure detection. Enclaves chose to relax the strict consensus requirement to achieve a weaker, but sufficient in practice, form of agreement between leaders. This results in a simpler and lighter protocol that does not rely on failure detection.

7 Conclusion

A combination of fairly standard fault-tolerant algorithms, secret sharing schemes, and traditional authentication protocols was used to develop an intrusion-tolerant version of the Enclaves group communication framework. The architecture provides secure services even if a number of its main components are compromised. The resulting Enclaves system remains easily deployable and lightweight, while being considerably more resilient to attacks than its predecessors.

Acknowledgements

The design and implementation of Enclaves benefited from the help of Mohamed Layouni and his many useful comments and suggestions.

References

[1] Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton, and G. Tsudik. Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, Taipei, Apr. 2000.

[2] G. Bracha and S. Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, 32(4):824–840, Oct. 1985.

[3] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. In *Proceedings of the 19th Annual Symposium on Principles of Distribtued Computing*, Portland, OR, July 2000.

[4] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, New Orleans, LA, Feb. 1999.

[5] D. Chaum and T. Pedersen. Wallet Databases with Observers. In *Crypto'92*, number 740 in Lecture Notes in Computer Science, pages 89–105, Santa Barbara, CA, Aug. 1992. Springer-Verlag.

[6] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *Proc. 26th IEEE Symp. Found. of Comp. Sci.*, pages 383–395. IEEE Press, 1985.

[7] Y. Desmedt and Y. Frankel. Shared Generation of Authenticators and Signatures. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 457–469. Springer-Verlag, 1991.

[8] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion Tolerance in Distributed Computing Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 110–121, Oakland, CA, May 1991.

[9] B. Dutertre, H. Saïdi, and V. Stavridou. Intrusion-tolerant Group Management in Enclaves. In *International Conference on Dependable Systems and Networks (DSN'01)*, pages 203–212, Göteborg, Sweden, July 2001.

[10] J.-C. Fabre, Y. Deswarte, and B. Randell. Designing Secure and Reliable Applications Using Fragmentation-Redundancy-Scattering: An Object-oriented Approach. In *First European Dependable Computing Conference (EDCC-1)*, volume 852 of *Lecture Notes in Computer Science*, pages 21–38. Springer-Verlag, Oct. 1994.

[11] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *Proc. 28th IEEE Symp. Found. of Comp. Sci.*, pages 427–437, 1987.

[12] M. Fischer, N. Lynch, and S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, Apr. 1985.

[13] J. Fray, Y. Deswarte, and D. Powell. Intrusion Tolerance Using Fine-Grain Fragmentation Scattering. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 194–201, Oakland, CA, Apr. 1986.

[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. In *Avances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer-Verlag, 1996.

[15] L. Gong. Increasing Availability and Security of an Authentication Service. *IEEE Jounral on Selected Areas in Communications*, 11(5):657–662, June 1993.

[16] L. Gong. Enclaves: Enabling Secure Collaboration over the Internet. *IEEE Journal of Selected Areas in Communications*, 15(3):567–575, Apr. 1997.

[17] K. Kihlstrom, L. Moser, and P. Melliar-Smith. The SecureRing Protocols for Securing Group Communication. In *Proceedings of the 31st IEEE Hawaii International Conference on System Sciences*, volume 3, pages 317–326, Kona, HI, Jan. 1998.

[18] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[19] T. Pedersen. Non-interactive and Information-theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1992.

[20] M. Reiter. The Rampart Toolkit for Building High-Integrity Services. In *Theory and Practice in Distributed Systems*, pages 99–110. Springer Verlag, LNCS 938, 1995.

[21] O. Rodeh, K. Birman, M. Hayden, Z. Xiao, and D. Dolev. Ensemble Security. Technical Report TR98-1703, Department of Computer Science, Cornell University, Sept. 1998.

[22] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, 1979.

[23] V. Shoup. Practical Threshold Signatures. In *Advances in Cryptology – EUROCRYPT'2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer-Verlag, 2000.

[24] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: a New Approach to Group Key Agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 380–387, Amsterdam, The Netherlands, May 1998.

[25] R. van Renesse, K. Birman, and S. Maffeis. Horus: A Flexible Group Communications System. *Communications of the ACM*, 39(4):76–83, Apr. 1996.

[26] T. Wu, M. Malkin, and D. Boneh. Building Intrusion-tolerant Applications. In *Eigth USENIX Security Symposium*, pages 79–91, Aug. 1999.

[27] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kılıççöte, and P. K. Khosla. Survivable Information Storage Systems. *Computer*, pages 61–68, Aug. 2000.