Intrusion-Tolerant Group Management in Enclaves

DSN'01, Göteborg, Sweden, July 2001

Bruno Dutertre, Hassen Saïdi, and Victoria Stavridou
System Design Laboratory, SRI International
333 Ravenswood Avenue, Menlo Park, CA 94025, USA
{*bruno,saidi,victoria*}*@sdl.sri.com*

Abstract

Groupware applications require secure communication and group-management services. Participants in such applications may have divergent interests and may not fully trust each other. The services provided must then be designed to tolerate possibly misbehaving participants. Enclaves is a software framework for building such group applications. We discuss how the protocols used by Enclaves can be modified to guarantee proper service in the presence of nontrustworthy group members. We show how the improved protocol was formally specified and proven correct.

1. Introduction

Group-oriented applications deployed over insecure networks such as the Internet can involve a set of participants who collaborate on common tasks but may not fully trust each other. Even if all group members trust each other, external intruders could attempt to disrupt the application by compromising a member's host machine. To support groupware applications in such environments, it is necessary to provide robust authentication, communication, and group-management services that can tolerate misbehaving members and members whose machine has been compromised.

Enclaves^{TM} is a platform supporting such secure group-oriented applications. It is designed to be lightweight and portable, and relies on software-implemented cryptography. An application consists of a set of members who coordinate and cooperate via a group leader. The leader is responsible for all group-management activities, including authenticating and accepting new members, distributing cryptographic keys, and distributing group-membership information. In the current implementation, all group members are assumed trustworthy and there is little protection against misbehaving members. As a consequence, compromise of a single computer hosting one of the members can lead to the failure of group-management services.

This paper presents an improved protocol for authentication and group management in Enclaves, and shows how this protocol was formally verified. The protocol ensures that a noncompromised user is guaranteed correct group-management services as long as the leader is not compromised. The modeling and verification approach is a new combination of techniques for proving secrecy properties of cryptographic protocols, and of state-machine abstraction based on verification diagrams. Section 2 gives an overview of the Enclaves architecture and protocols, and discusses weaknesses of these protocols in the presence of nontrustworthy group members. Section 3 presents the improved protocol for user authentication and for group-management activities. Sections 4 and 5 describe the formalization and verification of this protocol, respectively. Section 6 discusses related work.

2. An Overview of Enclaves

2.1. Architecture

A group-oriented application enables users to share information and collaborate via a network such as the Internet. The group involved is usually dynamic. An application is started when a user initiates a session, and new users are allowed to join and later leave the session. Multicast is the main mode of communication: Messages originating from one group member are received by all the users in the current session. In many cases, groupware applications have restricted access policies and other security requirements. Access to an active session is limited to a predefined set of users or to users having appropriate credentials (e.g., they have paid for the service provided). In such contexts, user authentication, key distribution, data confidentiality, and data integrity are essential.

Enclaves [5] is a lightweight software framework that provides the infrastructure to support such secure group applications. The overall architecture of an Enclaves application is shown in Figure 1. The group is organized around a group leader who starts and ends the application, and is responsible for all group-management activities. Regular

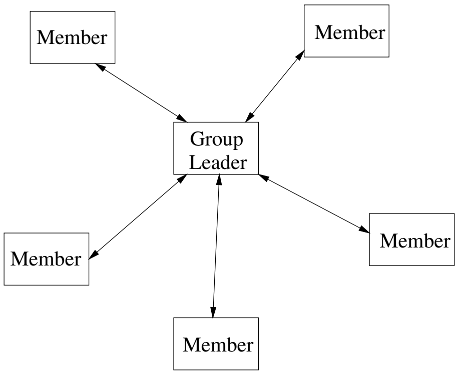

Figure 1. Enclaves Architecture

group members establish a bidirectional point-to-point link with the leader when they join the application.

Enclaves is designed to be easily portable. It does not use IP multicast but relies on common point-to-point protocols. As a consequence, all group communications are mediated by the group leader. Messages from a member to the group are sent to the leader for relay to the other members. To ensure confidentiality and integrity of communication, Enclaves relies on standard cryptographic techniques based on symmetric-key encryption and message-authentication codes.

Messages are encrypted using a group key K_g that is distributed to the members when they join the group. The key can be changed by the leader according to an application-dependent policy. In addition to the group key, which is common to all the members, each member A has a separate session key K_a that is used for group-management activities and is shared between A and the leader. This key is generated when A joins the application and remains in use until A leaves.

Enclaves provides various group-management protocols for supporting applications. Such protocols perform operations such as user authentication or key distribution. Examples are described in the next section. These protocols have been extracted from the Java$^{\mathrm{TM}}$ implementation of Enclaves described in [6] that differs significantly from a previous implementation [5].

2.2. Example Protocols

For user authentication, Enclaves assumes that each potential group member has a long-term password that must be known in advance to the group leader.[1] To join an application, a user A first signals to the leader L his intention to join the group. L can either accept or deny access to A depending on the application security policy:

$$\begin{array}{lll} 1. & A \rightarrow L: & A, req\_open \\ 2. & L \rightarrow A: & L, ack\_open \text{ (or } connection\_denied). \end{array}$$

If the connection is accepted, A initiates the following authentication protocol:

$$\begin{array}{lll} 1. & A \rightarrow L: & A, \{A, L, N_1\}_{P_a} \\ 2. & L \rightarrow A: & L, \{L, A, N_1, N_2, K_a, I.V., K_g\}_{P_a} \\ 3. & A \rightarrow L: & A, \{N_2\}_{K_a}. \end{array}$$

In message 1, A encrypts a triple composed of A's identity, the leader's identity and a nonce N_1 and sends the result to the leader. This encryption uses a key P_a derived from A's password, so P_a is known by both A and L. On reception of this message, L checks that the two encrypted identities are correct and extracts N_1. L then generates a new nonce N_2, a new shared key K_a, and an initialization vector $I.V.$. L sends all these components together with N_1 and the current group key K_g, all encrypted with P_a. In message 3, A acknowledges receipt of message 2.

At the end of this protocol, A is ready to participate in group activities. The leader informs all the group that a new member A has joined and sends to A the identity of all the other group members. These messages are encrypted using the group key.

The group leader generates a first group key K_g when the first member is accepted. A new group key can be generated and distributed at any time by the leader, depending on the application security policy. Typically, new keys can be generated when new members join, when members leave, or on a periodic basis. For distributing a new group key K'_g, L sends an individual message to all the members and waits for an acknowledgment:

$$\begin{array}{lll} 1. & L \rightarrow A: & L, new\_key, \{K'_g, I.V.\}_{K_a} \\ 2. & A \rightarrow L: & A, new\_key\_ack, \{K'_g\}_{K'_g}. \end{array}$$

A user A willing to leave the session simply sends the following message:

$$1. \quad A \rightarrow L: \quad A, req\_close$$

The leader then sends an acknowledgment to A and informs the rest of the group that A has left:

$$\begin{array}{lll} 2. & L \rightarrow A: & L, close\_connection \\ 3. & L \rightarrow B_i: & L, mem\_removed, \{A\}_{K_g}. \end{array}$$

A variation of this protocol can be used to expel some members of the group.

2.3. Protocol Weaknesses

The previous protocols are vulnerable to various attacks – for example, based on message replay. Furthermore, the

[1] Authentication using public-key cryptography is also possible, but is not currently implemented.

protocols were designed under the assumption that group members (past and present) are trustworthy. In case this assumption is not valid, a misbehaving member can easily disrupt the application and cause various security failures.

For example, the pre-authentication exchange can lead to a simple denial-of-service attack. It may seem economical to check whether A is allowed to join the group before performing the authentication protocol, but A has no guarantees that the reply ack_open or $connection\_denied$ actually came from the group leader. To prevent a legitimate user A from joining the group, an attacker can forge a $connection\_denied$ reply and send it to A.

The transmission of group-membership information is also weak. The message $L, mem\_removed, \{A\}_{K_g}$ is intended to signal to all members that A has left the group, but there is little evidence that this message is fresh, and there is no evidence that it was sent by the leader. Such a message can be easily forged by any group member since it is encrypted with the common group key. A malevolent A can then convince a member B that A has left the group. As a result, B has an inaccurate view of who is part of the group, and this may cause B to send information to the group that he did not intend A to receive.

There are similar problems with the distribution of new group keys. The message $L, new\_key, \{K'_g, I.V.\}_{K_a}$ informs a group member A that K'_g is the new group key. Unfortunately, nothing guarantees to A that this message is fresh. An attacker can then force A to reuse an old group key K'_g by replaying an old key-distribution message. Such an attack can cause loss of confidentiality if the attacker is in possession of K'_g. Obtaining an old group key such as K'_g may be difficult for an outsider but is trivial for any group member. The attack can then be performed by a past member of the group who has left the application but has kept the old key K'_g. The rekeying procedure is then insecure unless all present and past participants in the current application are trustworthy.

3. Intrusion Tolerance in Enclaves

3.1. Objectives

Our objective is to develop a more robust version of the Enclaves protocols, that fixes the various flaws of the current implementation and tolerates nontrustworthy or compromised members. The main objective of Enclaves is to ensure user authentication, confidentiality and integrity of group communication, and to provide accurate group membership information to the group members.

Confidentiality requires that group communication must be accessible only to current group members. Clearly, this cannot be guaranteed in the presence of nontrustworthy members as any such member can leak the information outside the group. Instead, we focus on making sure that compromised group members cannot interfere with the user authentication process or with the group management services. As shown by the example attacks discussed previously, it is important for each user to have an accurate view of who is in the group, and to have evidence that key distribution messages are timely and originate from the leader. To ensure these properties, we define an improved protocol for performing user authentication and distributing arbitrary group-management messages to users.

The new protocol relies on the same architecture and means of authentication as previously. We assume a set of agents connected via an insecure asynchronous network. The agents include users who can participate in the group application and other agents (the outsiders). The participants consist of a central group leader L and a set of prospective group members. Each prospective member A has a long-term key P_a initially known only by A and L. We say that a participant is not compromised, trustworthy, or nonfaulty if it behaves as specified by the protocol. Outsiders and compromised participants behave arbitrarily but we assume that they cannot break the encryption primitives used. A compromised participant may be one who intentionally misbehaves or an honest user whose host machine or operating system has been corrupted. Since the network is insecure, compromised participants and outsiders can read all the messages exchanged, replay old messages, and send arbitrary messages they can construct (as described formally in Section 4.2). In particular, compromised participants can leak secrets, including their long-term key, to outsiders or other compromised participants. Thus collusions between attackers are possible.

The protocol specifies how a participant A joins and later leaves the group and how group-management messages are sent by L to group members. The requirements are as follows:

- *Proper User Authentication.* If a user is accepted as group member A by the leader then this user is actually A.
- *Proper Distribution of Group-Management Messages.* All the group-management messages accepted by a group member A have been sent by the group leader; they are accepted by A in the same order as they were sent by L; no group-management message accepted by A is a duplicate.

We want these requirements to be satisfied provided both A and L are not compromised, even in the presence of an arbitrary number of nontrustworthy agents. Each time A enters the group, L generates a new session key for A, and the requirements must be satisfied even if old session keys are compromised and known to nontrustworthy agents.

3.2. Improved Protocol

As previously, each user A has a secret long-term key P_a that is initially known by A and by L. To join the application, A initiates the following protocol:

1. $A \rightarrow L:\ \mathsf{AuthInitReq}, A, L, \{A, L, N_1\}_{P_a}$
2. $L \rightarrow A:\ \mathsf{AuthKeyDist}, L, A, \{L, A, N_1, N_2, K_a\}_{P_a}$
3. $A \rightarrow L:\ \mathsf{AuthAckKey}, A, L, \{N_2, N_3\}_{K_a}.$

With small variations, this is the same authentication protocol as previously. The main differences are the removal of the pre-authentication exchange, the presence of a fresh nonce N_3 in message 3, and the absence of the group key K_g. K_g must be distributed to A in subsequent group-management messages and nonce N_3 is used in the distribution of such messages.

If authentication succeeds, A becomes a member of the group and is in possession of the session key K_a. As long as A is in session, L can send group-management messages to A and A must acknowledge each such message. The messages and acknowledgments are encrypted with K_a and nonces are used to protect against replay. Both L and A memorize a nonce N_{2i+1} that was generated by A. This nonce is either the N_3 communicated to L at the end of the authentication protocol, or a nonce that L received from A in the most recent acknowledgment message. The group-management exchange is as follows:

1. $L \rightarrow A:\ \mathsf{AdminMsg}, L, A, \{L, A, N_{2i+1}, N_{2i+2}, X\}_{K_a}$
2. $A \rightarrow L:\ \mathsf{Ack}, A, L, \{A, L, N_{2i+2}, N_{2i+3}\}_{K_a}.$

Message 1 contains N_{2i+1} to prove to A that the message is not a replay, and communicates to A the nonce N_{2i+2} that L generates. The field X is the actual group-management message. For example, X may specify a new group key and initialization vector, or indicate that a member has joined or left the session. Message 2 contains N_{2i+2} to prove to L that the acknowledgment is not a replay, and communicates to L a fresh nonce N_{2i+3} to be used in the next exchange.

A can leave the session at any time by sending the following message to L:

1. $A \rightarrow L:\ \mathsf{ReqClose}, A, L, \{A, L\}_{K_a}.$

In this message, K_a is used to guarantee that the message originated from A and to prove freshness. The message cannot be a replay since there can be at most one closing message per session and hence per session key. On reception, L simply closes the session with A: K_a is discarded and no further group-management message is sent to A.

4. Formal Protocol Model

To analyze the protocol, we first build a system model based on state-transition systems. This model uses notations and concepts presented by Millen and Rueß [10], and closely follows Paulson's inductive approach to modeling cryptographic protocols [11].

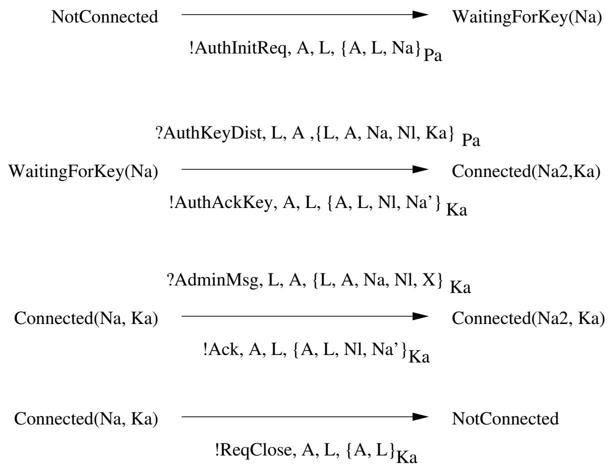

Figure 2. State-transition Model of User A

We represent the message space in a standard way, as described in [10] or [11]. Each message consists of a label, an apparent sender, an intended recipient, and a content. Labels represent the type of each message – for example, $\mathsf{AuthInitReq}$ or $\mathsf{AuthKeyDist}$. Message contents are elements of the set of fields $\mathcal{F}$ defined as follows:

- Agent identities, keys, and nonces are primitive fields.
- Given two fields X and Y, their concatenation, denoted by $[X, Y]$, is a field.
- Given a field X and a key K, the encryption of X with K, denoted by $\{X\}_K$, is a field.

The set of agent identities is denoted by $\mathcal{A}$, the set of nonces by $\mathcal{N}$, and the set of keys by $\mathcal{K}$. These three sets are mutually disjoint. Keys are either long-term keys of the form P_a or session keys such as K_a. All the keys are symmetric.

We also use "oops" events to model the compromise of session keys (cf. [11]). An oops event is written $\mathsf{Oops}(X)$ and means that field X (typically a session key) is communicated to all agents. $\mathsf{Oops(X)}$ is treated like an ordinary message whose content is the field X. The set of messages and oops events is denoted by $\mathcal{M}$.

4.1. Users and Leader

A nonfaulty user A is modeled by the state-transition system of Figure 2. The states of this systems are of three forms:

- $\mathsf{NotConnected}$: A is out of the group and has not started the authentication process.

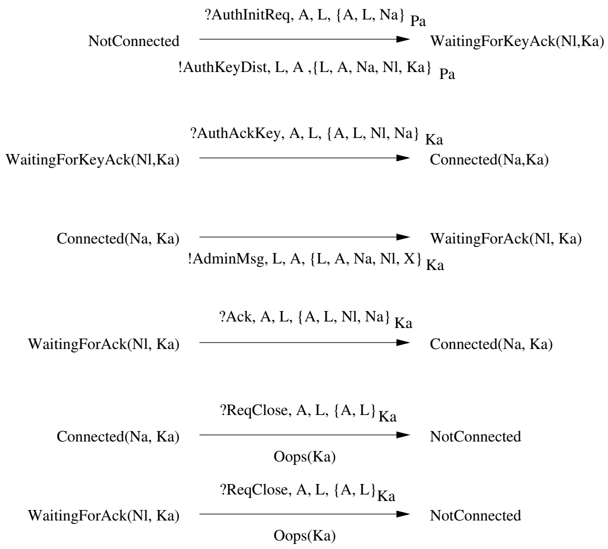

Figure 3. Leader Communication with A

- $\mathsf{WaitingForKey}(N_a)$: A has sent an $\mathsf{AuthInitReq}$ message containing N_a as a fresh nonce, and A is waiting for a reply from the leader
- $\mathsf{Connected}(N_a, K_a)$: A has joined the group, and has received K_a as a session key from the leader. N_a is the last nonce A has generated and sent to L. N_a is then the nonce A expects in the next group-management message from L.

The leader is modeled as the composition of separate transition systems, one for each user. Each of these systems defines the responses of the leader to requests from a user A, and the transmission of group-management messages to this user. The system is shown in Figure 3 and its states are of the following forms:

- $\mathsf{NotConnected}$: A is not connected.
- $\mathsf{WaitingForKeyAck}(N_l, K_a)$: L has generated a fresh session key K_a for A and is waiting for a key acknowledgment containing nonce N_l.
- $\mathsf{Connected}(N_a, K_a)$: this is the normal state when A is a member of the group. K_a is the session key that L uses to communicate with A. N_a is the most recent nonce that L received from A and will be included in the next group-management message.
- $\mathsf{WaitingForAck}(N_l, K_a)$: L has just sent a group-management message to A and is waiting for an acknowledgment containing nonce N_l.

On reception of a $\mathsf{ReqClose}$ message, L closes the session and K_a is discarded. The $\mathsf{Oops}(K_a)$ event attached to the corresponding transitions models the compromise of old session keys: K_a is released and becomes public as soon as the session is closed.

4.2. Global Model

We use Paulson's approach [11, 10] to model the behavior of nontrusted agents. Given a set of fields S, the following sets are used: $\mathsf{Parts}(S)$, $\mathsf{Analz}(S)$, and $\mathsf{Synth}(S)$. $\mathsf{Parts}(S)$ is the set of fields and subfields that occur in S. $\mathsf{Analz}(S)$ is the set of fields that can be extracted from elements of S without breaking the cryptosystem. $\mathsf{Synth}(S)$ is the set of fields that can be constructed from elements of S by concatenation and encryption. Formal definitions can be found in [11] or [10].

Overall, our model is the asynchronous composition of an honest user A, an honest leader L, and other nontrusted agents. The behavior of a nontrusted agent B is determined by the keys and other fields that B knows initially, and by the messages that B has observed so far. Given an agent $G \in \mathcal{A}$, we denote by $I(G)$ the set of fields that G knows initially. We assume that $\mathsf{Parts}(I(G))$ does not contain any nonce or session key. Furthermore, if $G \neq A$ and $G \neq L$, we assume $P_a \notin \mathsf{Parts}(I(G))$. This means that initially only A and L know A's long-term key P_a.

Let Q denote the global state space of the system. In a system state $q \in Q$, we denote by $trace(q)$ the set of messages and oops events that have occurred so far and by $\underline{trace}(q)$ the message contents that occur in $trace(q)$. We assume that all agents are able to observe all the events that have occurred so far. In a state q, the set of fields that G can access is then

$$\mathsf{Know}(G, q) \quad = \quad \mathsf{Analz}(I(G) \cup \underline{trace}(q)).$$

This is the set of fields that G can obtain from its initial knowledge $I(G)$ and the messages seen so far. We also define the set of nonces and sessions keys that are not used in state q as follows:

$$\begin{aligned}
\mathsf{FreshNonces}(q) &= \mathcal{N} - \mathsf{Parts}(\underline{trace}(q)) \\
\mathsf{UsedKeys}(q) &= \{K \mid \exists X : \{X\}_K \in \mathsf{Parts}(\underline{trace}(q))\} \\
\mathsf{FreshKeys}(q) &= \mathcal{K}_S - \\
&\quad (\mathsf{Parts}(\underline{trace}(q)) \cup \mathsf{UsedKeys}(q)),
\end{aligned}$$

where $\mathcal{K}_S$ denotes the set of session keys. The set of fields that G can generate in a state q is

$$\mathsf{Gen}(G, q) \quad = \quad \mathsf{Synth}(\mathsf{Know}(G, q) \cup \mathsf{FreshFields}(q)),$$

where $\mathsf{FreshFields}(q)$ is the union of $\mathsf{FreshNonces}(q)$ and $\mathsf{FreshKeys}(q)$. G can then synthesize new messages from fields it knows and from fresh keys or nonces it generates.

The overall model is now given by a collection of transition relations $T_G \subseteq Q \times \mathcal{M} \times Q$, one for each agent $G \in \mathcal{A}$. Each triple (q, M, q') of T_G represents a global transition corresponding to agent G sending message M in state q. Such a transition is denoted by $q \xrightarrow{M} q'$ by G, and its effect on $trace$ is given by

$$trace(q') = \{M\} \cup trace(q). \tag{1}$$

An important constraint is that an agent G sends only messages it can generate:

$$q \xrightarrow{M} q' \text{ by } G \quad \Rightarrow \quad \underline{M} \in \mathsf{Gen}(G, q), \tag{2}$$

where $\underline{M}$ is the content of message M. The label of M, the apparent sender, and the intended recipient can be arbitrary.

The transition relations T_A and T_L corresponding to A and L are extracted from Figures 2 and 3 in a straightforward way. For all other agents B, T_B is an arbitrary relation that is assumed to satisfy constraints (1) and (2). The global system behavior is characterized by the following rule:

$$q \longrightarrow q' \quad = \quad \exists G \in \mathcal{A}, M \in \mathcal{M} : \; q \xrightarrow{M} q' \text{ by } G.$$

The system evolves by selecting an agent G and executing a corresponding transition. This corresponds to the asynchronous composition of the systems T_G. Since we look only at safety properties, there are no fairness assumptions.

5. Verification

Let A be an arbitrary non-faulty user. The proof that the protocol satisfies the requirements of Section 3.1 is decomposed in four steps. We first show two secrecy properties:

- A's long-term key P_a is kept secret. Nobody other than A and L can ever access key P_a.
- As long as a session key K_a is in use, it is secret: only A and L can use K_a.

These properties are state invariant: they are satisfied in all the reachable states of the system. The proofs use the notions of protocol regularity, and of ideals and coideals presented in [10].

Using these two invariant properties, we apply a verification method, based on verification diagrams, that was proposed by Rushby [15]. The idea is to construct an abstraction of the transition system from which the requirements can be easily established. We construct a verification diagram that corresponds to our informal understanding of the protocol and we show that this diagram is actually a valid abstraction of the system. The final step is to show that proper authentication and proper distribution of group-management messages are implied by the diagram.

The following sections describe the main stages of the proof. A more detailed presentation is given in [4]. The whole formalization and verification have been performed using the PVS theorem prover. The PVS formalization required two person weeks of effort. No error in the protocols was found, but the use of PVS was essential to fix flaws in our hand proofs, including errors and omissions in several candidate verification diagrams we constructed. The resulting PVS developments are available at `http://www.csl.sri.com/~bruno/pvs/enclaves.txt`.

5.1. Secrecy of A's Long-Term Key

We have to prove that the following property is satisfied by any reachable state q:

$$\forall G \in \mathcal{A} : P_a \in \mathsf{Know}(G, q) \quad \Rightarrow \quad G = A \vee G = L.$$

We prove this by using the *Regularity Lemma* of Millen and Rueß [10]. The idea is to show that, in any reachable state q, P_a does not occur in $trace(q)$, that is, $P_a \notin \mathsf{Parts}(\underline{trace}(q))$. This property is satisfied because the protocol is *regular*: Neither A nor L ever send P_a in a message. Formally, we have to show

$$q \xrightarrow{M} q' \text{ by } A \quad \Rightarrow \quad P_a \notin \mathsf{Parts}(M)$$

$$q \xrightarrow{M} q' \text{ by } L \quad \Rightarrow \quad P_a \notin \mathsf{Parts}(M).$$

The proof is an easy case analysis. Using the regularity lemma of [10], this implies that, in any reachable state q, we have $P_a \notin \mathsf{Parts}(\underline{trace}(q))$. Now, by elementary properties of Analz and Parts (cf. [10]), we have

$$\begin{aligned} \mathsf{Know}(G, q) &= \mathsf{Analz}(I(G) \cup \underline{trace}(q)) \\ &\subseteq \mathsf{Parts}(I(G) \cup \underline{trace}(q)) \\ &= \mathsf{Parts}(I(G)) \cup \mathsf{Parts}(\underline{trace}(q)). \end{aligned}$$

If q is a reachable state and P_a belongs to $\mathsf{Know}(G, q)$, we must then have $P_a \in \mathsf{Parts}(I(G))$. By the assumptions on the initial knowledge of G, this implies $G = A$ or $G = L$.

5.2. Secrecy of Session Keys

We say that a session key K_a is in use in a state q if L is in a local state that contains K_a as a component:

$$\begin{aligned} &\mathsf{InUse}(K_a, q) \;\; = \\ &\quad (\exists N : lead_A(q) = \mathsf{WaitingForKeyAck}(N, K_a)) \\ &\quad \vee (\exists N : lead_A(q) = \mathsf{Connected}(N, K_a)) \\ &\quad \vee (\exists N : lead_A(q) = \mathsf{WaitingForAck}(N, K_a)). \end{aligned}$$

We want to show that the following property is satisfied in any reachable state q for all K_a and G:

$$\mathsf{InUse}(K_a, q) \; \wedge \; K_a \in \mathsf{Know}(G, q) \; \Rightarrow G = A \; \vee \; G = L.$$

As long as K_a is in use, no agent other than A and L has access to K_a.

As discussed in [10], to show that a key K_a is known only by A and L we have to consider the *ideal* generated by the set $S = \{K_a, P_a\}$. This ideal is denoted by $\mathcal{I}(S)$ and is the smallest set of fields such that

- $S \subseteq \mathcal{I}(S)$
- if $X \in \mathcal{I}(S)$ or $Y \in \mathcal{I}(S)$ then $[X, Y] \in \mathcal{I}(S)$
- if $X \in \mathcal{I}(S)$ and $K \notin S$ then $\{X\}_K \in \mathcal{I}(S)$.

$\mathcal{I}(S)$ contains all the fields from which K_a or P_a can be extracted. For example, $\{X, Y, K_a\}_{P_b}$ belongs to $\mathcal{I}(S)$ as any agent in possession of P_b can obtain K_a from this field. For K_a to remain secret, we must make sure that fields from $\mathcal{I}(S)$ never occur in the trace as long as K_a is in use.

The complement of $\mathcal{I}(S)$ is called a *coideal* and is denoted by $\mathcal{C}(S)$. Proving that K_a remains secret amounts then to showing that $\underline{trace}(q)$ is included in $\mathcal{C}(S)$. Coideals satisfy two important properties [10]:

$$\mathsf{Analz}(\mathcal{C}(S)) = \mathcal{C}(S). \quad (3)$$
$$\mathsf{Synth}(\mathcal{C}(S)) = \mathcal{C}(S). \quad (4)$$

Another useful result is that, for any set of fields E,

$$\mathsf{Parts}(E) \cap S = \emptyset \quad \Rightarrow \quad E \subseteq \mathcal{C}(S).$$

This is the Ideal-Parts Lemma of [10]. To apply these results to Enclaves, we start with the following lemma whose proof is an easy induction.

Lemma 1 *For any reachable state q and any session key K_a, we have*

$$\mathsf{InUse}(K_a, q) \Rightarrow K_a \in \mathsf{Parts}(\underline{trace}(q)).$$

This simply means that once K_a is in use, it is no longer fresh and thus any key that nontrusted agents might generate in the future will be distinct from K_a.

Lemma 2 *If G is an agent other than A and L, and q is a reachable state such that $\mathsf{InUse}(K_a, q)$ and $\underline{trace}(q) \subseteq \mathcal{C}(S)$, then the two following properties are satisfied:*

$$\mathsf{Know}(G, q) \subseteq \mathcal{C}(S)$$
$$\mathsf{Gen}(G, q) \subseteq \mathcal{C}(S).$$

Proof: By assumption, we have $\mathsf{Parts}(I(G)) \cap S = \emptyset$ and then $I(G) \subseteq \mathcal{C}(S)$. Since $\underline{trace}(q) \subseteq \mathcal{C}(S)$, we obtain $I(G) \cup \underline{trace}(q) \subseteq \mathcal{C}(S)$, and then

$$\begin{aligned} \mathsf{Know}(G, q) &= \mathsf{Analz}(I(G) \cup \underline{trace}(q)) \\ &\subseteq \mathsf{Analz}(\mathcal{C}(S)) = \mathcal{C}(S). \end{aligned}$$

The proof of the second inclusion is very similar and relies on the preceding lemma. □

The remainder of the proof of secrecy consists of showing that the property

$$\mathsf{InUse}(K_a, q) \quad \Rightarrow \quad \underline{trace}(q) \subseteq \mathcal{C}(S) \quad (5)$$

is invariant. It is trivially true in the initial state q_0. Now, let q be a state that satisfies (5) and let q' be a successor of q. We have

$$q \xrightarrow{M} q' \text{ by } G$$

for some message M and agent G. Assuming K_a is in use in q', we have to show $\underline{trace}(q') \subseteq \mathcal{C}(S)$, that is, $\underline{trace}(q) \cup \{\underline{M}\} \subseteq \mathcal{C}(S)$. Three cases must be considered:

- K_a is not in use in q. K_a has then been freshly generated by L in response to an $\mathsf{AuthInitReq}$ message: $G = L$ and $K_a \in \mathsf{FreshKeys}(q)$. This means that $K_a \notin \mathsf{Parts}(\underline{trace}(q))$. As we have shown in the previous section, $P_a \notin \mathsf{Parts}(\underline{trace}(q))$. Since $S = \{K_a, P_a\}$, the properties of coideals give $\underline{trace}(q) \subseteq \mathcal{C}(S)$. The message M is of the form

  $$\mathsf{AuthKeyDist}, L, A, \{A, L, N_a, N_l, K_a\}_{P_a},$$

  and it is easy to see that the content $\underline{M}$ of this message – namely, the field $\{A, L, N_a, N_l, K_a\}_{P_a}$ – does not belong to $\mathcal{I}(S)$. So we have $\underline{trace}(q') \subseteq \mathcal{C}(S)$.

- K_a is in use in q and G is an agent other than A and L. We know that $\underline{M} \in \mathsf{Gen}(G, q)$. Since K_a is in use in q, we have $\underline{trace}(q) \subseteq \mathcal{C}(S)$. By Lemma 2, this implies $\underline{M} \in \mathcal{C}(S)$ and then $\underline{trace}(q') \subseteq \mathcal{C}(S)$.

- K_a is in use in q and G is either A or L. By inspection, we can see that the content of message M is of the form $\{\dots\}_{K_a}$. Such a field does not belong to $\mathcal{I}(S)$ so we have $\underline{M} \in \mathcal{C}(S)$. We can then conclude that $\underline{trace}(q') \subseteq \mathcal{C}(S)$ as in the previous case.

By induction, we have then shown that property (5) is true in all reachable states. We conclude by the following proposition, which is an easy consequence of the previous result, of Lemma 2, and of the fact that $K_a \notin \mathcal{C}(S)$.

Proposition 3 *Given a session key K_a, an agent G and a reachable state q, we have*

$$\mathsf{InUse}(K_a, q) \wedge K_a \in \mathsf{Know}(G, q) \quad \Rightarrow \quad G = A \vee G = L.$$

5.3. System Abstraction

Figure 4 shows a verification diagram (cf. [15, 8]) that represents an abstraction of the overall transition system. Each box in the diagram is labeled with a predicate $Q_i(q)$ that relates $usr_A(q)$, $lead_A(q)$, and $trace(q)$. (Only the

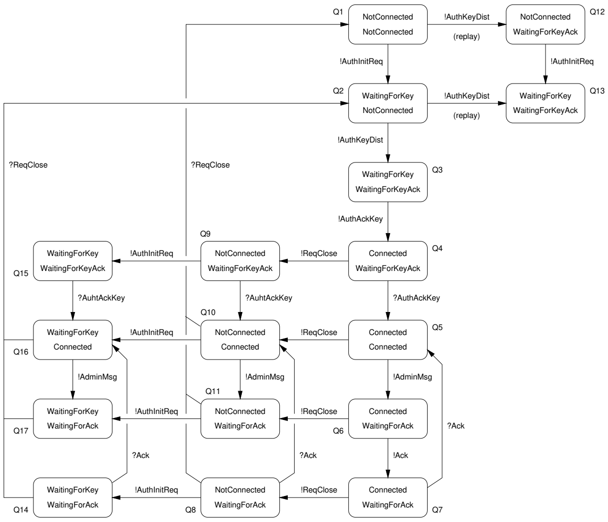

Figure 4. Protocol Abstraction

local user and leader states are shown in the figure.) For example, the predicates Q_1, Q_2, and Q_{12} are as follows:

$$\begin{aligned}
&Q_1(q) = \\
&\quad usr_A(q) = \mathsf{NotConnected}\ \wedge\ lead_A(q) = \mathsf{NotConnected}
\end{aligned}$$

$$\begin{aligned}
&Q_{12}(q) = \exists N_l, K_a : \\
&\quad usr_A(q) = \mathsf{NotConnected} \wedge \\
&\quad lead_A(q) = \mathsf{WaitingForKeyAck}(N_l, K_a) \wedge \\
&\quad \forall N :\ \{A, L, N_l, N\}_{K_a}\ \notin\ \mathsf{Parts}(\underline{trace}(q))
\end{aligned}$$

$$\begin{aligned}
&Q_2(q) = \exists N_a : \\
&\quad usr_A(q) = \mathsf{WaitingForKey}(N_a) \wedge \\
&\quad lead_A(q) = \mathsf{NotConnected} \wedge \\
&\quad \forall N, K :\ \{L, A, N_a, N, K\}_{P_a}\ \notin\ \mathsf{Parts}(\underline{trace}(q)).
\end{aligned}$$

The whole diagram can be constructed in a systematic way as explained in [15]. In our case, the construction is based on examining the successive transitions A or L can execute, starting from a state that satisfies Q_1. A complete list of the abstraction predicates is given in [4].

To prove that the diagram is a correct abstraction of the protocol model, we must show for each box i labeled by predicate Q_i that the diagram captures all the possible transitions out of a state where Q_i holds. If the successor boxes of i are numbered $i_1, \ldots, i_k$, the proof obligation is

$$\begin{aligned}
\forall q, q', M, G :\ Q_i(q)\ \wedge\ q \xrightarrow{M} q' \text{ by } G\ \Rightarrow& \\
Q_{i_1}(q') \vee \ldots \vee Q_{i_k}(q').&
\end{aligned}$$

Although this is not shown explicitly in Figure 4, i is always a successor of itself. In addition, we must also show that q_0, the initial state, satisfies predicate Q_1.

For $i = 1$, we must then show that for states q and q', all agents G and all messages M, we have

$$Q_1(q)\ \wedge\ q \xrightarrow{M} q' \text{ by } G\ \ \Rightarrow\ \ Q_1(q') \vee Q_2(q') \vee Q_{12}(q').$$

The proof is a simple case analysis. If $G = A$, q' satisfies Q_2; if $G = L$, q' satisfies Q_{12}; otherwise, q' satisfies Q_1. The proof that $Q_2(q')$ is satisfied when $G = A$ or that $Q_{12}(q')$ is satisfied when $G = L$ uses the fact that the nonces created by the transition are fresh.

In most other cases, we use the secrecy theorems established previously to discharge the proof obligations. For example, the proof obligation for Q_3 is

$$Q_3(q)\ \wedge\ q \xrightarrow{M} q' \text{ by } G\ \ \Rightarrow\ \ Q_3(q') \vee Q_4(q'), \quad (6)$$

where $Q_3(q)$ and $Q_4(q)$ are as follows:

$$\begin{aligned}
&Q_3(q)\ =\ \exists N_a, N_l, K_a : \\
&\quad usr_A(q) = \mathsf{WaitingForKey}(N_a) \wedge \\
&\quad lead_A(q) = \mathsf{WaitingForKeyAck}(N_l, K_a) \wedge \\
&\quad (\forall N, K : \{L, A, N_a, N, K\}_{P_a}\ \in \mathsf{Parts}(\underline{trace}(q))\ \Rightarrow \\
&\qquad\qquad\qquad N = N_l \wedge K = K_a) \wedge \\
&\quad (\forall N :\ \{A, L, N_l, N\}_{K_a}\ \notin \mathsf{Parts}(\underline{trace}(q)))\ \wedge \\
&\quad \{A, L\}_{K_a}\ \notin \mathsf{Parts}(\underline{trace}(q))
\end{aligned}$$

$$\begin{aligned}
&Q_4(q)\ =\ \exists N_a, N_l, K_a : \\
&\quad usr_A(q) = \mathsf{Connected}(N_a, K_a) \wedge \\
&\quad lead_A(q) = \mathsf{WaitingForKeyAck}(N_l, K_a) \wedge \\
&\quad (\forall N : \{A, L, N_l, N\}_{K_a} \in \mathsf{Parts}(\underline{trace}(q))\ \Rightarrow N = N_a) \wedge \\
&\quad (\forall N :\ \{L, A, N_a, N\}_{K_a}\ \notin \mathsf{Parts}(\underline{trace}(q)))\ \wedge \\
&\quad \{A, L\}_{K_a}\ \notin \mathsf{Parts}(\underline{trace}(q)).
\end{aligned}$$

Assume that q satisfies Q_3 and that $q \xrightarrow{M} q'$ by G. It is easy to see that no transition of L is enabled in q so we must have either $G = A$ or G is an agent other than A and L.

- If $G = A$ then A has just received a key-distribution message of the form

  $$\mathsf{AuthKeyDist}, L, A, \{L, A, N_a, N, K\}_{P_a}$$

  and has answered with the key acknowledgment

  $$\mathsf{AuthAckKey}, A, L, \{A, L, N, N'_a\}_K,$$

  where N'_a is a freshly generated nonce. A moves to state $\mathsf{Connected}(N'_a, K)$.

  Since A has received it, the field $\{L, A, N_a, N, K\}_{P_a}$ belongs to $\mathsf{Parts}(\underline{trace}(q))$. By definition of Q_3, we must have $N = N_l$ and $K = K_a$. The only field of the form $\{A, L, N_l, N\}_{K_a}$ in $\underline{trace}(q')$ is the content of the message that A has just sent and then $N = N'_a$. Since nonce N'_a is fresh, there cannot be a field of the form $\{L, A, N'_a, N'\}_{K_a}$ in $\underline{trace}(q')$. It is also clear that $\{A, L\}_{K_a}$ does not belong to $\underline{trace}(q')$. This shows that q' satisfies predicate Q_4.

- If $G \neq A$ and $G \neq L$, we know $\underline{M} \in \mathsf{Gen}(G, q)$ and we must show that $Q_3(q')$ is satisfied. This amounts to showing that $\underline{M}$ cannot contain a part of the form $\{L, A, N_a, N, K\}_{P_a}$, unless $N = N_l$ and $K = K_a$, and that $\underline{M}$ cannot contain a subfield $\{L, A, N_l, N\}_{K_a}$ or $\{A, L\}_{K_a}$. Now, $Q_3(q)$ implies $\mathsf{InUse}(K_a, q)$ so we can use the preceding results – namely, $P_a \notin \mathsf{Know}(G, q)$ and $K_a \notin \mathsf{Know}(G, q)$. Then, by definition of Synth (cf. [10]), G cannot synthesize fields of the form $\{\ldots\}_{P_a}$ or $\{\ldots\}_{K_a}$, but can only replay them. For $\{L, A, N_a, N, K\}_{P_a}$ to be in $\mathsf{Gen}(G, q)$, we must then have

$$\{L, A, N_a, N, K\}_{P_a} \quad \in \quad \mathsf{Know}(G, q).$$

Since $\mathsf{Know}(G, q) \subseteq \mathsf{Parts}(I(G)) \cup \mathsf{Parts}(\underline{trace}(q))$, this implies that

$$\{L, A, N_a, N, K\}_{P_a} \quad \in \quad \mathsf{Parts}(\underline{trace}(q)).$$

By the third clause of Q_3, $N = N_l$ and $K = K_a$.

By a similar reasoning, we obtain that no part of $\underline{M}$ can be of the form $\{L, A, N_l, N\}_{K_a}$ or $\{A, L\}_{K_a}$ since no such field belongs to $\mathsf{Parts}(\underline{trace}(q))$. We can then conclude that q' satisfies Q_3.

The remainder of the verification relies on proof obligations that are very similar to property (6). The reasoning is always the same: transitions by A or L from a state that satisfies Q_i lead to a state that satisfies a new predicate Q_j. Transitions performed by other agents leave Q_i invariant, essentially because agents other than A and L do not have access to P_a and K_a.

5.4. Diagram Analysis

The proof diagram of Figure 4 is the support for further analysis of the Enclaves protocol. In particular, the fact that group-management messages are received in the order they were sent without duplication can be almost read off the diagram. In more detail, we first extend the transition system by recording the list of group-management messages sent by L and received by A. In a state q, these two lists are kept in state-variables $snd_A(q)$ and $rcv_A(q)$, respectively. The lists are initially empty. A new element is appended to $snd_A(q)$ when L sends an AdminMsg M, and a new element is appended to $rcv_A(q)$ when A executes a transition triggered by the reception of an AdminMsg M. $rcv_A(q)$ is emptied when A leaves a session and $snd_A(q)$ is emptied when L receives ReqClose from A. It is then easy to prove from the verification diagram that the list $rcv_A(q)$ is a prefix of $snd_A(q)$ in all reachable states.

Proper user authentication can be derived from the diagram by a similar method. L accepts A as a member when the system enters a state where $lead_A(q) = \mathsf{Connected}(\ldots)$. This happens whenever L accepts a message of type AuthAckKey. Similarly, A requests to enter by sending a message of type AuthInitReq. Proper user authentication requires that the nth AuthAckKey accepted by L was preceded by the nth AuthInitReq from A. This can be proved by showing that the list of acceptance events from L is a prefix of the list of requests to join from A.

Another important property follows from the verification diagram: Whenever A and L are both in a Connected state, they agree on the session key and on the most recent nonce produced by A:

$$\begin{array}{l} usr_A(q) = \mathsf{Connected}(N, K) \ \wedge \\ \quad lead_A(q) = \mathsf{Connected}(N', K') \\ \qquad \Rightarrow \ N = N' \ \wedge \ K = K'. \end{array}$$

The diagram also shows that whenever A is in possession of a session key K_a, then L is also in possession of K_a, that is, $\mathsf{InUse}(K_a, q)$ is satisfied.

6. Related Work

Existing approaches for achieving intrusion tolerance in distributed systems rely on redundancy and secret sharing (e.g., [3]). In the Enclaves context, our objective was more limited and our approach was to design a robust cryptographic protocol. Such an approach can apply to other systems such as Antigone [9] that are based on a centralized group leader. The advantage of such an architecture is simplicity. User authentication is centralized and membership information is easily collected and distributed. On the other hand, such an architecture is not scalable and the leader is clearly a single point of failure. The protocol we propose tolerates compromised users but still requires the leader to be trusted and trustworthy. For better resilience to intrusions, more complex systems such as Rampart [13] or the SecureRing [7] use fault-tolerant protocols to distribute all decisions about group management. Centerpieces of such systems are intrusion-tolerant distributed agreement protocols that tolerate some forms of Byzantine failures by relying on failure detectors, or other forms of Byzantine agreement protocols (e.g., [2]). These distributed architectures can provide stronger intrusion-tolerance guarantees, but the agreement protocols are complex and can cause important performance degradation. These infrastructures are reserved for critical applications with high-integrity requirements. In other systems such as Secure Spread [1], Ensemble [14], or Horus [17], security services are built on top of an existing group communication infrastructure. The infrastructure provides group-membership and multicast services that support a variety of network-level faults, including network partitioning. Key-agreement or key-distribution protocols (e.g., [16]) are implemented on top of these services, and cryptography ensures confidentiality and integrity of group communication. These systems typically provide security against outsider attacks but do not consider misbehaving group members.

The formalization and verification methods we use are based on Paulson's inductive approach to verifying cryptographic protocols [11], with extensions proposed by Millen and Rueß [10]. Proof diagrams were proposed by Manna and Pnueli [8]. The type of diagrams we use was discussed by Rushby [15] and has been applied to the verification of fault-tolerant protocols [12].

7. Conclusion

We define a new Enclaves protocol for performing user authentication and for distributing group-management messages. This protocol improves over the existing Enclaves protocols by providing correct services even in the presence of faulty insiders (past or present group member). We give a proof of correctness of the protocol that combines modeling and verification techniques developed for cryptographic protocols, and system abstraction techniques.

The main limit of the current Enclaves architecture is its reliance on a central group leader. In future work, we intend to develop a more robust and scalable version of the system where the single leader is replaced by a distributed set of group managers.

Acknowledgments

The work presented in this paper was partially funded by the Defence Evaluation and Research Agency under contract CU009-0000002017, and by the Space and Naval Warfare Systems Center under contract N66001-00-C-8001.

Our thanks go to Sathish Gopalakrishnan who implemented the new Enclaves protocol. Our PVS formalization and verification relied on libraries developed by Harald Ruess and Jon Millen.

References

[1] Y. Amir et al. Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments. In *20th IEEE International Conference on Distributed Computing Systems*, Taipei, April 2000.

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

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

[4] B. Dutertre and H. Saïdi. Verification of Enclaves Group-Management Services. Technical report, System Design Laboratory, SRI International, July 2000.

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

[6] S. Keung and L. Gong. Enclaves in Java: APIs and Implementations. Technical Report SRI-CSL-96-07, SRI International, 1996.

[7] K. Kihlstrom, L. Moser, and P. Melliar-Smith. The SecureRing Protocols for Securing Group Communication. In *IEEE 31st Hawaii International Conference on System Sciences*, pages 317–326, January 1998.

[8] Z. Manna and A. Pnueli. Temporal Verification Diagrams. In *International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, pages 726–765, Sendai, Japan, April 1994. Springer-Verlag, LNCS 769.

[9] P. McDaniel, A. Prakash, and P. Honeyman. Antigone: A Flexible Framework for Secure Group Communication. In *8th USENIX Security Symposium*, pages 99–114, Washington, DC, August 1999.

[10] J. Millen and H. Rueß. Protocol-Independent Secrecy. In *IEEE Symposium on Research in Security and Privacy*, pages 110–119, Oakland, CA, May 2000.

[11] L. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[12] H. Pfeifer. Formal Verification of the TTP Group Membership Algorithm. In *FORTE/PSTV 2000*, Pisa, Italy, October 2000.

[13] 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.

[14] O. Rodeh et al. Ensemble Security. Technical Report TR98-1703, Department of Computer Science, Cornell University, September 1998.

[15] J. Rushby. Verification Diagrams Revisited: Disjunctive Invariants for Easy Verification. In *Computer Aided Verification (CAV 2000)*, pages 508–520, Chicago, IL, July 2000. Springer-Verlag, LNCS 1855.

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

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