EECS4312 Secure Messenger Project

Anton Sitkovets (cse31027@cse.yorku.ca)
Mina Zaki (zakim@cse.yorku.ca)

November 29, 2016

You may work on your own or in a team of no more than two students. **Submit only one document under one Prism account.**

Prism account used for submission: cse31027

Revisions

Date	Revision	Description
October 30, 2016	1.0	Initial requirements document
November 29, 2016	2.0	Final requirements document

Requirements Document: Secure Messenger System

Contents

List of Figures

List of Tables

1 System Overview

The System Under Development(SUD) is a secure messenger system. The idea is that there are users (e.f. Doctors, nurses, administrators) and the messenger system must support the ability of users to send text messages. There are also groups, e.g. cardiology, nephrology, endocrinology. Users are members of groups and when they send a message it is always to a group, with the privacy condition that only members of that group may read the message. Users may become members of any number of groups. Users may send messages only to the members of a group they are registered in. Users may only access/read a message from a group they are registered in. Once a message is read by the recipient, its status changes from new to old. Users may delete their old messages.

2 Use Case Diagram

In this use case we have administrators and users as the actors. Administrators have the unique rights to add users, add groups, register users, list all the users and list all the groups. Users on the other hand can only send, read, delete and view messages. Since an administrator can also be a user, we also say that administrators can perform all the tasks that a user can. The user has a goal to be able to read, send, delete and view messages. Whereas the administrator has the goal of setting up the messenger system by adding the users, groups and registrations to the system, and simultaneously having the same goals as a user. From the diagram we can see the connections between the administrator and the use cases they trigger and the use cases the user triggers. We can see the system boundary separates the functionality of the messenger system from the actors, as the actors have no impact on how the secure messenger executes the use cases. There are relationships between the use cases, as can be seen, the List All Users case includes the Add User case because you cannot display the users if there arent any, same goes for List All Groups. We have that the case Register Users extends both cases of adding users and groups as registering users extends the operation of these two cases. On the users use cases it can be seen that the other commands all include the Send Message case because these cases never stand alone and only occur when a message has been sent already.

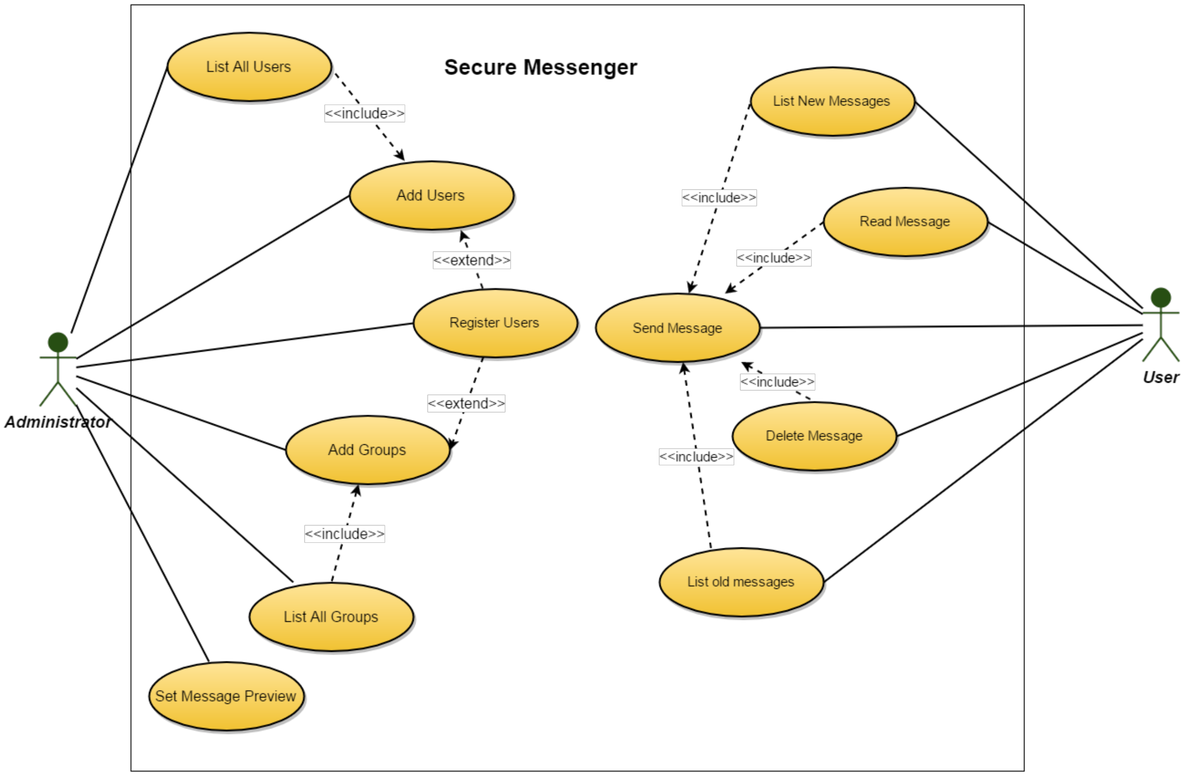


Figure 1: Use Case Diagram

3 Goals

The high-level goals (G) of the system are:

- G1—Users may be added to groups.
- G2—Users should be able to send a message to a group if they belong to that group.
- G3—Users should only be able to read a message if they belong to that group.
- G4—Users not registered to a group should not be able to read that groups messages.

4 Monitored Events

Name	Inputs	Description
add_user	id,name	Add a user with id and name to the list of users.
add_group	id,name	Add a group with id and name to the list of groups.
register_user	user_id, group_id	Add user to the group
send_message	user_id, group_id, text	Create a new message sent by the user to the group with content text
read_message	user_id, message_id	Set the status of this message for this user as read
delete_message	user_id, message_id	Delete this message for this user
set_message_preview	n	Set the length of the preview of messages
list_new_messages	user_id	Display the list of new messages for this user
list_old_messages	user_id	Display the list of old messages for this user
list_users		Display the list of users in the system
list_groups		Display the list of groups in the system

Table 1: Monitored Events

5 Abstract State

STATE	TYPE	Description
users	set[UID]	Set of the user ids in the system
names	(users) $\mapsto USER$	Function that maps a user id to its user name.
groups	set[GID]	Set of the group ids in the system
gname	(groups) $\mapsto GROUP$	Function that maps a group id to its group name
msgs	set[MID]	Set of the message ids in the system
membership	set[(users),(groups)]	Set that lists the groups that each user belongs to
info	(msgs) $\mapsto MSG\_INFO[(users),(groups)]$	Function that maps a message id to its information which is a record that stores a the sender id, recipient group id and text
ms	[(users),(msgs)] $\mapsto MSG\_STATE$	Function which maps a user, message pair to a message state (which can be read, unread or unavailable). Each user has a message state for each message.

Table 2: Abstract State

6 E/R-descriptions

6.1 R-Descriptions

6.1.1 ID Requirements

REQ1	Each user ID is unique. There will not be two users with the same ID.	Confusion is created when two users have the same ID.

REQ2	Each group ID is unique. There will not be two groups with the same ID.	Confusion is created when two groups have the same ID.

REQ3	Each message ID is unique. There will not be two messages with the same ID.	Confusion is created when two messages have the same ID.

6.1.2 Send Message Requirements

REQ4	A user may only send a message to a group if they belong to that group.	This ensures that only users with permission to send a message are permitted to send a message

6.1.3 User/Group Requirements

REQ5	A user may belong to more than one group.	Users typically communicate with more than one social circle (ie friends, family, coworkers)

REQ6	A group may not contain the same user twice.	Users should not receive duplicate copies of messages

6.1.4 Read Message Requirements

REQ7	Once a message is read by the recipient, it is no longer new for that user and is therefore moved to the old messages.	When a message is read, it should no longer be labelled new.

REQ8	A user can only read a message once	Once a user has read the message, there is no need to read it again

REQ9	A user may only read a message sent to a group if they belong to that group.	This ensures privacy, since only those given access to the group should be allowed to read messages from that group

REQ10	A user may not read a message that they sent	A user does not need to read their own message, they wrote it.

6.1.5 Delete Message Requirements

REQ11	To delete a message, the user must have read that message	If users delete a message before reading it, they might miss out on important information

REQ12	A user may not delete a message that they sent to a group	Since the user does not have access to read the message, they also are unable to delete their own message

6.1.6 Message Preview Requirements

REQ13	The message preview may be any number of characters greater than or equal to 1.	There must be a message preview and one character is the shortest possible message preview

REQ14	The default number of characters in the message preview is 15.	Fifteen is a reasonable length for a message preview.

6.1.7 Error Message Requirements

REQ15	The system should not display two error messages at the same time.	This creates confusion for the user. Present only one error message at a time to avoid bombarding the user with messages

REQ16	Prioritize error messages to display in order of importance, as specified in Section 9	The most important error messages must be dealt with first

6.1.8 Requirements on Output

REQ17	The system should be able to handle many users, messages and groups with no limit.	There can be many users on a system and there should be no limit to the number of users, messages or groups

REQ18	The system should display a unique command number which signifies the count of commands that were executed	To keep track of history.

REQ19	The system should display the status of the system	Allow the user to be notified if there is an issue with the last command.

REQ20	The system should display the list of users organized according to user ID	Allow the user to see the current users in the system.

REQ21	The system should display the list of groups organized according to group ID	:Allow the user to see the current groups in the system.

REQ22	The system should display the list of registrations of users in groups organized according to user ID	Allow the user to see the current registrations in the system.

REQ23	The system should display all the messages in the system organized according to message ID	Allow the user to see the current messages in the system.

REQ24	The system should display all the new messages in the system organized according to message ID	Allow the user to see the current new messages in the system.

REQ25	The system should display all the old/read messages in the system that have not been deleted organized according to message ID	Allow the user to see the current read messages in the system.

REQ26	If one of the list commands are executed, then the system should not display the regular output, and should instead display the command number, system status and the output list	List commands have a different output structure than the rest of the commands

REQ27	If the list command is list_users or list_groups, then the output is organized in alphabetical order	Customer preferred this organization

REQ28	If the list command is list_old_messages(id) or list_new_messages(id), then the output is organized by message id	Customer preferred this organization

REQ29	If the last command created an error, the system will not display the regular output, and will instead display the command number, system status and a meaningful error message	If an error has been created, ensure that the user is notified immediately

REQ30	When an operation can be performed, the system responds with OK	This communicates to the user that their command has been successfully executed.

6.2 Environmental Assumptions

6.2.1 Assumptions on Input

ENV31	The user id and group id received as input are integers.	Users,groups and messages require a simple unique form of identification.

ENV32	Message ids are positive integers.	This is the most natural numbering for messages

ENV33	The user name, group name and message text received as input is a string.	Users and groups require a simple unique form of identification.

ENV34	The system should be able to respond to the monitored events described in Table 1	These are the commands that the system must support

7 Mathematical Function Table

Function tables outlining the mathematical model of the system for each monitored event.

STATE	id $\notin users_{-1}$	id $\in users_{-1}$
users	users$_{-1} \cup \{id\}$	NC
names	name-1 $\upharpoonright id \mapsto name$	
groups	NC	
gname		
msgs		
membership	membership$_{-1} \upharpoonright (id, g) \mapsto FALSE$	
info	NC	
ms	ms$_{-1} \upharpoonright idxm \mapsto unavailable$	
output	OK	Error

Table 3: Mathematical Function Table for add_user

STATE	id $\notin groups_{-1}$	id $\in groups_{-1}$
users	NC	NC
names		
groups	groups$_{-1} \cup \{id\}$	
gname	gname-1 $\restriction id \mapsto name$	
msgs	NC	
membership	membership$_{-1} \restriction (u, id) \mapsto FALSE$	
info	NC	
ms		
output	OK	Error

Table 4: Mathematical Function Table for add_group

STATE	uid $\in users_{-1} \wedge$ gid $\in groups_{-1} \wedge$ uid,gid $\notin membership_{-1}$	$\neg uid \in users_{-1} \vee \neg$ gid $\in groups_{-1} \vee \neg$ uid,gid $\notin membership_{-1}$
users	NC	NC
names		
groups		
gname		
msgs		
membership	membership$_{-1} \restriction (uid, gid)$	
info	NC	
ms	ms$_{-1} \restriction idxm \mapsto unavailable$	
output	OK	Error

Table 5: Mathematical Function Table for register_user

STATE	uid $\in users_{-1}\wedge$ mid $\in msgs_{-1}\wedge$ uid $\in info(mid)(group)\wedge$ (u,m) $=$ unread $\in ms_{-1}$	$\neg uid \in users_{-1} \vee \neg$ mid $\in msgs_{-1} \vee \neg$ uid $\in info(mid)(group) \vee \neg$ (u,m) $=$ unread $\in ms_{-1}$
users	NC	NC
names		
groups		
gname		
msgs		
membership		
info		
ms	ms$_{-1} \upharpoonright uidxmid \mapsto read$	
output	OK	Error

Table 6: Mathematical Function Table for read_message

STATE	uid $\in users_{-1}\wedge$ mid $\in msgs_{-1}\wedge$ uid $\in info(mid)(group)\wedge$ (u,m) $=$ read $\in ms_{-1}$	id $\in users_{-1}$
users	NC	NC
names		
groups		
gname		
msgs		
membership		
info		
ms	ms$_{-1} \upharpoonright id \times m \mapsto unavailable$	
output	OK	Error

Table 7: Mathematical Function Table for delete_message

STATE	uid $\in users_{-1} \wedge$ gid $\in \wedge$ (uid,gid) $\in membership_{-1}$	id $\in groups_{-1}$
users	NC	NC
names		
groups		
gname		
msgs	msgs$_{-1} \cup \{m\}$	
membership	NC	
info	info$_{-1} \restriction m \mapsto (u, g, txt)$	
ms	ms$_{-1} \restriction FORALLq : user - 1$ q = u : (q ,m) $\mapsto read$ q $\neq u \wedge q \in g : (q, m) \mapsto unread$ q $\notin g \mapsto unavailable$	
output	OK	Error

Table 8: Mathematical Function Table for send_message

8 PVS Function Table

Function tables outlining the PVS implementation for each monitored event.

STATE	$\neg users_{-1}(id)$	users$_{-1}(id)$
users	users$_{-1} \cup \{id\}$	NC
names	name$_{-1}WITH[id \mapsto name]$	
groups	NC	
gname		
msgs		
membership	membership$_{-1}WITH(\forall(g : groups_{-1}) : [(id, g) := FALSE]$	
info	NC	
ms	ms$_{-1}WITH(\forall(m : msgs_{-1}) : [(id, m) \mapsto unavailable])$	
output	OK	Error

Table 9: Function table for add_user

STATE	$\neg groups_{-1}(id)$	$\text{groups}_{-1}(id)$
users	NC	NC
names		
groups	$\text{groups}_{-1} \cup \{id\}$	
gname	$\text{gname}_{-1} WITH[id := name]$	
msgs	NC	
membership	$\text{membership}_{-1} WITH(\forall(u : users_{-1}) : [(u, id) := FALSE]$	
info	NC	
ms		
output	OK	Error

Table 10: Function Table for add_group

STATE	$\text{users}_{-1}(uid) \wedge groups_{-1}(gid)$ $\wedge \neg membership_{-1}(u, g)$	$\neg users_{-1}(uid) \vee \neg groups_{-1}(gid)$ $\vee \neg membership_{-1}(u, g)$
users	NC	NC
names		
groups		
gname		
msgs		
membership	$\text{membership}_{-1} WITH[(uid, gid)]$	
info	NC	
ms	$\text{ms}_{-1} WITH(\forall(m : msgs_{-1}) :$ $[\text{ id } \times m \mapsto unavailable])$	
output	OK	Error

Table 11: Function Table for register_user

STATE	users$_{-1}(uid) \land msgs_{-1}(mid)$ $\land$ membership$_{-1}(u, info\_(mid)‘recip)$ $\land ms_{-1}(u, m) = unread$	$\neg users_{-1}(uid) or$ $\neg msgs_{-1}(mid) \lor \neg$ membership$_{-1}(u, info\_(mid)‘recip)$ $\lor \neg ms_{-1}(u, m) = unread$
users	NC	NC
names		
groups		
gname		
msgs		
membership		
info		
ms	ms$_{-1} WITH[(u, m) := read]$	
output	OK	Error

Table 12: Function Table for read_message

STATE	users$_{-1}(u) \land$ msgOf(st(i-1),u)(m)	$\neg users_{-1}(u)$ $\lor \neg msgOf(st(i-1), u)(m)$
users	NC	NC
names		
groups		
gname		
msgs		
membership		
info		
ms	ms$_{-1} WITH[(u, m) := unavailable]$	
output	OK	Error

Table 13: Function Table for delete_message

STATE	users$_{-1}(uid) \wedge groups_{-1}(gid)$ $\wedge membership_{-1}(uid, gid)$	$\neg users_{-1}(uid) \vee \neg groups_{-1}(gid)$ $\vee \neg membership_{-1}(uid, gid)$
users	NC	NC
names		
groups		
gname		
msgs	msgs$_{-1} \cup \{m\}$	
membership	NC	

info	info$_{-1} WITH$
	[m := (# sender := u,
	recip := g,
	content := txt #)]	

ms	ms$_{-1} WITH (\forall (q : user_{-1}) :$ membership$_{-1}(q, g) \mapsto$ u = q $\mapsto (q, m) := read$ u $\neq q \mapsto (q, m) := unread$ $\neg membership_{-1}(q, g) \mapsto$ (q,m) := unavailable]	
output	OK	Error

Table 14: Function Table for send_message

9 Error Message Tables

Function tables outlining the error message output for each monitored event.

		Output
uid >0 $\wedge user\_name \neq$ ”” $\wedge user\_name[0].isalpha()$ $\wedge uid \notin all\_users$		OK
$\neg$(uid >0 $\wedge user\_name \neq$ ”” $\wedge user\_name[0].isalpha()$ $\wedge uid \notin all\_users$	uid <= 0	ID must be a positive integer.
	uid $\in all\_users$	ID already in use.
	user_name = ”” $\vee$ $\neg$ user_name[0].isalpha()	User name must start with a letter.

Table 15: Error Message Table for add_user(uid, user_name)

		Output
gid $>0 \land gname \neq$ ”” $\land gname[0].isalpha()$ $\land gid \notin all\_groups$		OK
$\neg($gid $>0 \land gname \neq$ ”” $\land gname[0].isalpha()$ $\land gid \notin all\_groups)$	gid $<= 0$	ID must be a positive integer.
	gid $\in all\_groups$	ID already in use
	gname = ”” $\lor$ $\neg$ gname[0].isalpha()	Group name must start with a letter.

Table 16: Error Message Table for add_group(gid, gname)

		Output
uid $>0 \land gid>0$ $\land uid \in all\_users$ $\land gid \in all\_groups$ $\land uid \notin all\_groups[gid].users$		OK
$\neg($uid $>0 \land gid>0$ $\land uid \in all\_users$ $\land gid \in all\_groups$ $\land uid \notin all\_groups[gid].users)$	uid $<= 0 \lor gid< = 0$	ID must be a positive integer.
	uid $\notin all\_users$	User with this ID does not exist.
	gid $\notin all\_groups$	Group with this ID does not exist.
	uid $\in all\_groups[gid].users$	This registration already exists.

Table 17: Error Message Table for register_user(uid, gid)

		Output
from_id >0 $\wedge to\_id>0$ $\wedge text \neq$ ”” $\wedge from\_id \in all\_users$ $\wedge to\_id \in all\_groups$ $\wedge from\_id \in all\_groups[to\_id].users$		OK
$\neg$(from_id >0 $\wedge to\_id>0$ $\wedge text \neq$ ”” $\wedge from\_id \in all\_users$ $\wedge to\_id \in all\_groups$ $\wedge from\_id \in all\_groups[to\_id].users)$	from_id <= 0 $\vee to\_id < = 0$	ID must be a positive integer.
	from_id $\notin all\_users$	User with this ID does not exist.
	to_id $\notin all\_groups$	Group with this ID does not exist.
	test = ””	A message may not be an empty string.
	uid $\notin$ all_groups[all_messages[mid] .to_id].users	User not authorized to send messages to the specified group.

Table 18: Error Message Table for send_message(uid, gid, text)

		Output
uid $>0 \wedge mid>0\wedge$ mid $\in all\_messages \wedge uid \in all\_users$ $\wedge uid \in all\_messages[mid].unread\_ids$ $\wedge uid \notin all\_messages[mid].read\_ids$ $\wedge uid \in all\_groups[all\_messages[mid].to\_id].users$		OK
$\neg$(uid $>0 \wedge mid>0\wedge$ mid $\in all\_messages$ $\wedge uid \in all\_users$ $\wedge uid \in all\_messages[mid].unread\_ids$ $\wedge uid \notin all\_messages[mid].read\_ids$ $\wedge uid \in all\_groups[all\_messages$ [mid].to_id].users)	uid $<= 0 \vee mid< = 0$	ID must be a positive integer.
	uid $\notin all\_users$	User with this ID does not exist.
	mid $\notin all\_messages$	Message with this ID does not exist.
	uid $\notin$ all_groups[all_messages[mid] .to_id].users	User not authorized to access this message.
	uid $\in all\_messages[mid].read\_ids$ $\wedge uid \notin all\_messages$ [mid].unread_ids	Message has already been read. See ‘list_old_messages’.

Table 19: Error Message Table for read_message(uid, mid)

		Output
$\text{uid} > 0 \wedge mid > 0 \wedge$ $\text{mid} \in all\_messages \wedge$ $\text{uid} \in all\_users \wedge$ $\text{uid} \in all\_messages[mid].read\_ids$ $\wedge\, uid \neq all\_messages[mid].from\_id$		OK
$\neg(\text{uid} > 0 \wedge mid > 0 \wedge$ $\text{mid} \in all\_messages \wedge$ $\text{uid} \in all\_users \wedge$ $\text{uid} \in all\_messages[mid].read\_ids$ $\wedge uid \neq all\_messages[mid].from\_id)$	$\text{uid} <= 0 \vee mid < = 0$	ID must be a positive integer.
	$\text{uid} \notin all\_users$	User with this ID does not exist.
	$\text{mid} \notin all\_messages$	Message with this ID does not exist.
	$\text{uid = all\_messages[mid].from\_id}$ $\vee$ $\text{uid} \notin all\_messages[mid]$ .read_ids	Message with this ID not found in old/read messages.

Table 20: Error Message Table for delete_message(uid, mid)

		Output
$\text{uid} > 0$ $\wedge uid \in all\_users.keys()$ $\wedge len(all\_users[uid].unread\_messages) > 0$		OK
$\neg(\text{uid} > 0 \wedge uid \in all\_users.keys() \wedge$ $\text{len(all\_users[uid].unread\_messages)} > 0)$	$\text{uid} <= 0$	ID must be a positive integer.
	$\text{uid} \notin all\_users$	User with this ID does not exist.
	$\text{len(all\_users[uid].}$ $\text{unread\_messages)} = 0$	OK There are no new messages for this user.

Table 21: Error Message Table for list_new_messages(uid)

		Output
uid >0 $\wedge uid \in all\_users.keys()$ $\wedge len(all\_users[uid].read\_messages)$>0		OK
¬(uid >0 $\wedge uid \in all\_users.keys() \wedge$ len(all_users[uid].read_messages) >0)	uid <= 0	ID must be a positive integer.
	uid $\notin$ all_users	User with this ID does not exist.
	len(all_users[uid]. read_messages) = 0	OK. There are no old messages for this user.

Table 22: Error Message Table for list_old_messages(uid)

	Output
n >0	OK
n <= 0	Message length must be greater than zero.

Table 23: Error Message Table for set_message_preview(n)

	Output
len(all_groups) >0	OK
len(all_groups) = 0	There are no groups registered in the system yet.

Table 24: Error Message Table for list_groups

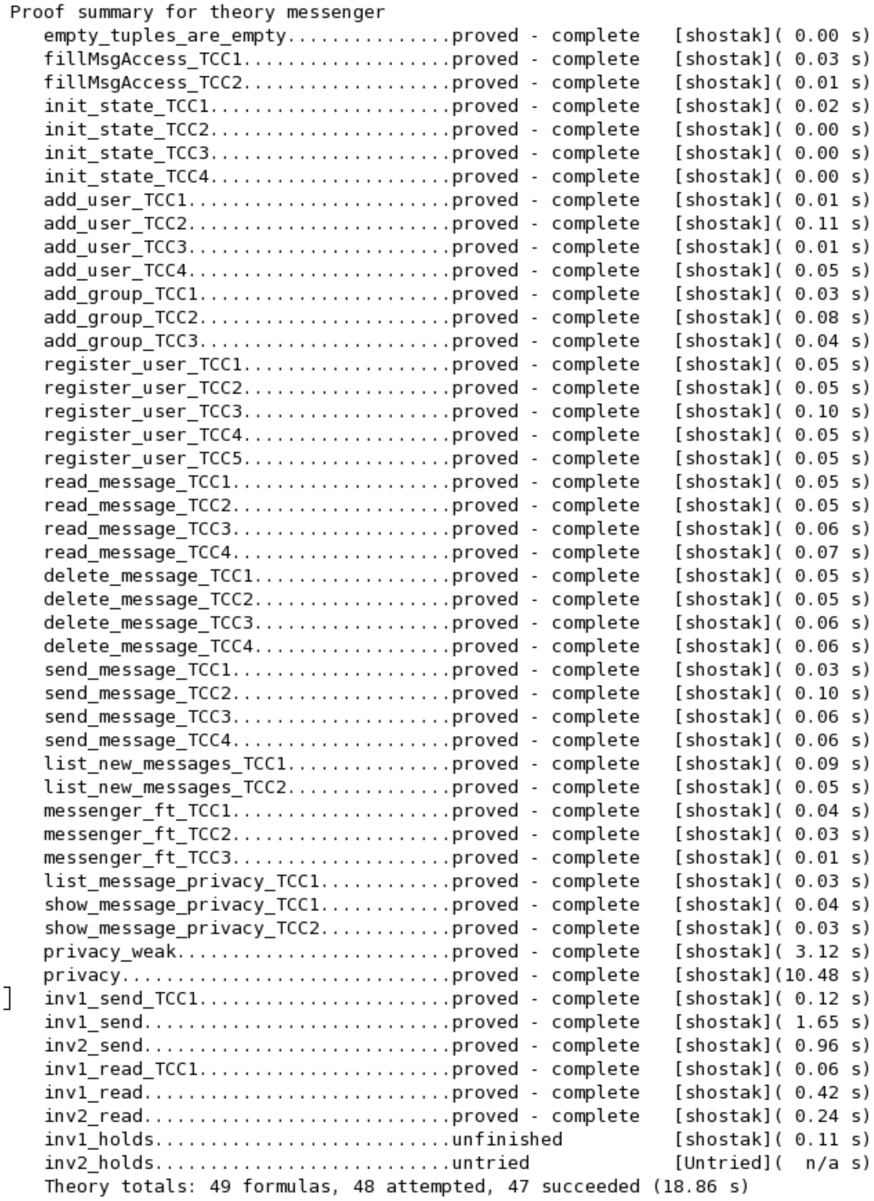
	Output
len(all_users) >0	OK
len(all_users) = 0	There are no groups registered in the system yet.

Table 25: Error Message Table for list_users

10 Validation

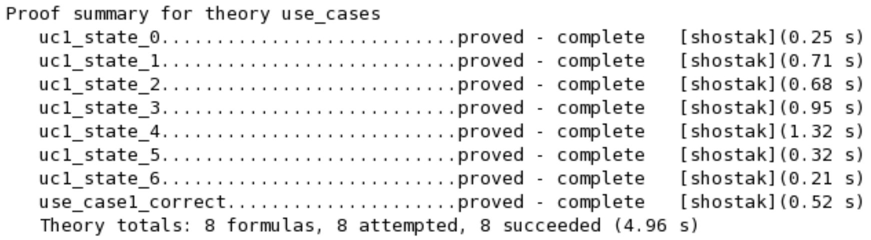
Proof of completeness, disjointness, validation, use cases and some invariants of the requirements using PVS.

PVS sources included in the appendix to this document but summarize the proofs here.



```
Proof summary for theory messenger
   empty_tuples_are_empty..............proved - complete   [shostak]( 0.00 s)
   fillMsgAccess_TCC1..................proved - complete   [shostak]( 0.03 s)
   fillMsgAccess_TCC2..................proved - complete   [shostak]( 0.01 s)
   init_state_TCC1.....................proved - complete   [shostak]( 0.02 s)
   init_state_TCC2.....................proved - complete   [shostak]( 0.00 s)
   init_state_TCC3.....................proved - complete   [shostak]( 0.00 s)
   init_state_TCC4.....................proved - complete   [shostak]( 0.00 s)
   add_user_TCC1.......................proved - complete   [shostak]( 0.01 s)
   add_user_TCC2.......................proved - complete   [shostak]( 0.11 s)
   add_user_TCC3.......................proved - complete   [shostak]( 0.01 s)
   add_user_TCC4.......................proved - complete   [shostak]( 0.05 s)
   add_group_TCC1......................proved - complete   [shostak]( 0.03 s)
   add_group_TCC2......................proved - complete   [shostak]( 0.08 s)
   add_group_TCC3......................proved - complete   [shostak]( 0.04 s)
   register_user_TCC1..................proved - complete   [shostak]( 0.05 s)
   register_user_TCC2..................proved - complete   [shostak]( 0.05 s)
   register_user_TCC3..................proved - complete   [shostak]( 0.10 s)
   register_user_TCC4..................proved - complete   [shostak]( 0.05 s)
   register_user_TCC5..................proved - complete   [shostak]( 0.05 s)
   read_message_TCC1...................proved - complete   [shostak]( 0.05 s)
   read_message_TCC2...................proved - complete   [shostak]( 0.05 s)
   read_message_TCC3...................proved - complete   [shostak]( 0.06 s)
   read_message_TCC4...................proved - complete   [shostak]( 0.07 s)
   delete_message_TCC1.................proved - complete   [shostak]( 0.05 s)
   delete_message_TCC2.................proved - complete   [shostak]( 0.05 s)
   delete_message_TCC3.................proved - complete   [shostak]( 0.06 s)
   delete_message_TCC4.................proved - complete   [shostak]( 0.06 s)
   send_message_TCC1...................proved - complete   [shostak]( 0.03 s)
   send_message_TCC2...................proved - complete   [shostak]( 0.10 s)
   send_message_TCC3...................proved - complete   [shostak]( 0.06 s)
   send_message_TCC4...................proved - complete   [shostak]( 0.06 s)
   list_new_messages_TCC1..............proved - complete   [shostak]( 0.09 s)
   list_new_messages_TCC2..............proved - complete   [shostak]( 0.05 s)
   messenger_ft_TCC1...................proved - complete   [shostak]( 0.04 s)
   messenger_ft_TCC2...................proved - complete   [shostak]( 0.03 s)
   messenger_ft_TCC3...................proved - complete   [shostak]( 0.01 s)
   list_message_privacy_TCC1...........proved - complete   [shostak]( 0.03 s)
   show_message_privacy_TCC1...........proved - complete   [shostak]( 0.04 s)
   show_message_privacy_TCC2...........proved - complete   [shostak]( 0.03 s)
   privacy_weak........................proved - complete   [shostak]( 3.12 s)
   privacy.............................proved - complete   [shostak](10.48 s)
]  inv1_send_TCC1......................proved - complete   [shostak]( 0.12 s)
   inv1_send...........................proved - complete   [shostak]( 1.65 s)
   inv2_send...........................proved - complete   [shostak]( 0.96 s)
   inv1_read_TCC1......................proved - complete   [shostak]( 0.06 s)
   inv1_read...........................proved - complete   [shostak]( 0.42 s)
   inv2_read...........................proved - complete   [shostak]( 0.24 s)
   inv1_holds..........................unfinished          [shostak]( 0.11 s)
   inv2_holds..........................untried             [Untried](  n/a s)
   Theory totals: 49 formulas, 48 attempted, 47 succeeded (18.86 s)
```

Figure 2: Proveit Summary for Messenger Theory



```
Proof summary for theory use_cases
   uc1_state_0..........................proved - complete   [shostak](0.25 s)
   uc1_state_1..........................proved - complete   [shostak](0.71 s)
   uc1_state_2..........................proved - complete   [shostak](0.68 s)
   uc1_state_3..........................proved - complete   [shostak](0.95 s)
   uc1_state_4..........................proved - complete   [shostak](1.32 s)
   uc1_state_5..........................proved - complete   [shostak](0.32 s)
   uc1_state_6..........................proved - complete   [shostak](0.21 s)
   use_case1_correct....................proved - complete   [shostak](0.52 s)
   Theory totals: 8 formulas, 8 attempted, 8 succeeded (4.96 s)
```

Figure 3: Proveit Summary for Use Cases Theory

11 Use Cases

A typical use case of how the system will be used over time by some users. This use case describes the system behaviour when the system adds a user, adds a group, registers the user to that group. Then the user sends a message to the group. Next a new user is added to the system and tries to send a message to a group they don't belong to.

```
use_case1_correct : THEOREM
          (FORALL i: messenger_ft(cmd,st,output)(i))
     AND cmd(1) = e_add_user(u1,n1)
     AND cmd(2) = e_add_group(g1,gn1)
     AND cmd(3) = e_register(u1,g1)
     AND cmd(4) = e_send(u1,g1,t1)
     AND cmd(5) = e_add_user(u3,n2)
     AND cmd(6) = e_send(u3,g1,t2)

     IMPLIES

          output(4) = OK
     AND st(4)`users(u1)
     AND st(4)`groups(g1)
     AND st(4)`membership(u1,g1)
     AND NOT empty?(msgOf(st(4),u1))
     AND output(6) = error
```

Figure 4: PVS Use Case

12 Acceptance Tests

In this section, the use cases have to be converted into precise acceptance tests (using the function table to describe pre/post conditions) to be run when the design and implementation are complete. The acceptance test cases can be found in the project directory under the acceptance folder. The following table describes where the use cases are being tested for in the acceptance tests. Each acceptance test describes different set of tests. For even deeper explanation on the reason for each acceptance test see the file acceptance_tests.txt under the docs folder.

Use Case	Acceptance Tests
Add a user to the system	at 1 - 9
Add a group to the system	at 1-2, 4-9
Register a user to a group	at 1, 5- 8
User sends a message to a group	at 1, 6-8
User reads a message in their new messages folder	at 1, 7-8
User deletes a message from their read messages folder	at 1, 7-8
User lists all their new messages	at 1, 7-8
User lists all their old messages	at 1, 7-8
User lists all the users in the system	at 1-9
User lists all the groups in the system	at 1-9
User changes message preview length	at 1

Table 26: Use Case Vs. Acceptance Test Table

13 Traceability

Matrix to show which acceptance tests passed, and which R-descriptions they checked.

R description	At1	At2	At3	At4	At5	At6	At7	At8	At9
REQ1	✓		✓						
REQ2	✓			✓					
REQ3	✓	✓				✓	✓	✓	
REQ4	✓					✓	✓	✓	
REQ5	✓				✓	✓	✓	✓	
REQ6	✓				✓	✓	✓	✓	
REQ7	✓						✓	✓	
REQ8	✓						✓	✓	
REQ9	✓						✓	✓	
REQ10							✓	✓	
REQ11	✓						✓	✓	
REQ12							✓	✓	
REQ13	✓								
REQ14	✓					✓	✓	✓	
REQ15	✓								
REQ16	✓								
REQ17	✓								
REQ18	✓								
REQ19	✓								
REQ20	✓								
REQ21	✓								
REQ22	✓								
REQ23	✓					✓	✓	✓	
REQ24	✓					✓	✓	✓	
REQ25	✓					✓	✓	✓	
REQ26	✓					✓	✓	✓	✓
REQ27	✓					✓	✓	✓	✓
REQ28	✓						✓	✓	
REQ29	✓								
REQ30	✓								

Table 27: Traceability of Requirements by Acceptance Test

14 Appendix

14.1 PVS Source Code

```
messenger_prelude : THEORY
BEGIN

precond : TYPE = {precond}

PRE (p : bool) : TYPE = { x : precond | p }

UID, GID, MID, USER, GROUP, TEXT: TYPE+

MSG_STATE : TYPE = {read,unread,unavailable}

  % Definition of an empty function
  emptyfun [T, U : TYPE] (x : {x : T | FALSE}) : RECURSIVE U =
    emptyfun(x)
    MEASURE 0


END messenger_prelude

insert [A,B,Z : TYPE,A2 : TYPE FROM A] : THEORY
BEGIN

   insertLeft (a: A, z:Z, f : [A2, B -> Z])
   (a0:(add(a,(A2_pred))),b:B) : Z =
   IF a = a0 THEN z
            ELSE f(a0,b)
   ENDIF
   insertRight (b: A, z:Z, f : [B, A2 -> Z])
   (a:B,b0:(add(b,(A2_pred)))) : Z =
   IF b = b0 THEN z
            ELSE f(a,b0)
   ENDIF
  insertRightWith (a : A, z : [ B -> Z ], f : [ B, A2 -> Z ])
  (b : B, a2 : (add(a,A2_pred))) : Z =
    IF a = a2 THEN z(b)
                    ELSE f(b, a2)
    ENDIF
END insert

message [U,G : TYPE] : THEORY
BEGIN
IMPORTING messenger_prelude

  MSG_INFO: TYPE =
    [#
       sender : U
     , recip: G
     , content: TEXT
          % sender is a member of the group the message
  % is sent to
    #]
END message

message_reader [U,G,M : TYPE] : THEORY
BEGIN
IMPORTING messenger_prelude
IMPORTING message

   readership
   ( mem  : set[[U,G]] , info : [ M -> MSG_INFO[U,G] ] )
           ( u : U, m : M ) : bool = mem(u,info(m)`recip)
  % a set of pairs (user, message) such that user is allowed
  % to read the message

END message_reader

messenger : THEORY

BEGIN
delta: posreal % sampling time
IMPORTING Time[delta]
IMPORTING message_reader
IMPORTING insert
```

```
i: VAR DTIME

STATE: TYPE =

  [#
    users: set[UID]
   ,names: [ (users) -> USER ]
   ,groups: set[GID]
   ,gname: [ (groups) -> GROUP ]
   ,msgs: set[MID]
   ,membership: set[[(users),(groups)]]
   ,info : [(msgs) -> MSG_INFO[(users),(groups)]]
     % specification of individual messages
   ,ms : [ [(users),(msgs)] -> MSG_STATE ]
          % message state: for every message and every user
          that can read it
 % a message can either be unread, read or deleted
  #]

OUTPUT : DATATYPE
  BEGIN
    OK: OK?
    error: error?
    list_msg (id: UID,ms: set[MID]): list_msg?
    show_msg (id: UID,msg:MID,txt: TEXT): show_msg?
  END OUTPUT

empty_tuples_are_empty : THEOREM
      FORALL (x2: [(emptyset[UID]), (emptyset[MID])]): FALSE

%|- empty_tuples_are_empty : PROOF
%|- (then (skeep) (typepred "x2`1") (expand "emptyset") (propax))
%|- QED

msgOf (st : STATE, u : (st`users))(m : MID): bool =
      st`msgs(m) AND st`membership(u,st`info(m)`recip)
```

```
fillMsgAccess(st: STATE, u: (st`users), g:(st`groups))
(q:(st`users)): MSG_STATE =
       COND
st`membership(q,g) -> COND
u = q -> read,
NOT u = q -> unread
        ENDCOND,
NOT st`membership(q,g) -> unavailable
ENDCOND


init_state : STATE =
  (# users := emptyset
   , names := emptyfun
   , groups := emptyset
   , gname := emptyfun
   , msgs := emptyset
   , membership := emptyset
   , info := emptyfun[(emptyset[MID]),
        MSG_INFO[(emptyset[UID]),(emptyset[GID])]]
   , ms := emptyfun[[(emptyset[UID]),(emptyset[MID])],MSG_STATE]

   #)
st : VAR [DTIME -> STATE]
output: VAR [DTIME -> OUTPUT]
u,u2  : VAR UID
un : VAR USER
gn : VAR GROUP
txt : VAR TEXT
g,g2  : VAR GID
m,m1,m2 : VAR MID

add_user (id: UID, name: USER)(st,output)(i : POS_DTIME) : bool =
  COND users_(id) -> output(i) = error AND st(i) = st(i-1)
     , NOT users_(id) ->
             st(i) = st(i-1) WITH
            [ users := add(id, users_)
   , names := names_ WITH [id := name]
   , membership := insertLeft(id,FALSE,mem_)
   , ms    :=  insertLeft(id,unavailable,ms_)
   ]
AND output(i) = OK
  ENDCOND
   WHERE
       users_   = st(i-1)`users
     , newUsers = add(id,users_)
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs

     , groups_ = st(i-1)`groups

add_group (id: GID, name: GROUP)(st,output)(i : POS_DTIME) : bool =
  COND groups_(id) -> output(i) = error AND st(i) = st(i-1)
     , NOT groups_(id) ->
             st(i) = st(i-1) WITH
            [ groups := add(id, groups_)
   , gname := gname_ WITH [id := name]
   , membership := insertRight(id,FALSE,mem_)
   ]
AND output(i) = OK
  ENDCOND
   WHERE
       users_   = st(i-1)`users
     , gname_   = st(i-1)`gname
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs
     , groups_ = st(i-1)`groups



register_user (u: UID, g: GID)(st,output)(i : POS_DTIME) : bool =
     COND users_(u) AND groups_(g) AND NOT mem_(u,g) ->
             st(i) = st(i-1) WITH
```

```
         [ membership := add((u,g),mem_)
     , ms := insertLeft(u,unavailable,ms_)
     ]
AND output(i) = OK
,    NOT users_(u)
  OR NOT groups_(g)
  OR mem_(u,g)
-> st(i) = st(i-1) AND output(i) = error
     ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs
     , groups_ = st(i-1)`groups

     % check whether we are allowed to read the message
     % change the state of the message to `read`
     % output the message content
     % to read a message, it must be in the state unread
     for that user in the prestate
read_message (u, m)(st,output)(i : POS_DTIME) : bool =
     COND users_(u) AND msgs_(m) AND mem_(u,info_(m)`recip)
     AND ms_(u,m) = unread    ->
             st(i) = st(i-1) WITH
       [ ms := ms_ WITH [ (u,m) := read ] ]
           AND output(i) = show_msg(u,m,info_(m)`content)
       ,  NOT users_(u) OR NOT msgs_(m) OR
       NOT mem_(u,info_(m)`recip)  OR NOT ms_(u,m) = unread
       ->     st(i) = st(i-1)
          AND output(i) = error
     ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs
     , groups_ = st(i-1)`groups

delete_message (u, m)(st,output)(i : POS_DTIME) : bool =
     COND users_(u) AND msgOf(st(i-1),u)(m) ->
             st(i) = st(i-1) WITH
       [ ms := ms_ WITH [ (u,m) := unavailable ] ]
        AND output(i) = OK
       ,     NOT users_(u) OR NOT msgOf(st(i-1),u)(m)
       ->     st(i) = st(i-1)
             AND output(i) = error
     ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs
     , groups_ = st(i-1)`groups

   % Allocate a fresh message id and give access to the authorized users.
   % This message's state is 'unread' for all user of the group except
   % for the sender. The state must be set to 'read' for the sender.


send_message (u:UID,g:GID,txt: TEXT)(st,output)(i : POS_DTIME) : bool =

      EXISTS (m:MID) : NOT member(m,st(i-1)`msgs) AND
      COND users_(u) AND groups_(g) AND mem_(u,g)  ->
             st(i) = st(i-1) WITH
             [ msgs := add(m, msgs_)
           , info := info_ WITH  [m :=  (# sender := u
                               ,recip := g
       ,content := txt #)  ]
    ,ms :=   insertRightWith(m,
    fillMsgAccess(st(i-1),u,g),ms_)  ]
    AND output(i) = OK
```

```
     , NOT users_(u) OR NOT groups_(g) OR NOT mem_(u,g) ->
     st(i) = st(i-1)  AND output(i) = error
      ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs
     , groups_ = st(i-1)`groups


list_new_messages (u: UID)(st,output)(i:POS_DTIME): bool =
    st(i) = st(i-1) AND
    COND users_(u) -> output(i) = list_msg(u, {m :
    (msgOf(st(i-1),u)) | ms_(u,m) = unread})
       , NOT users_(u) -> output(i) = error
    ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs
     , groups_ = st(i-1)`groups

list_old_messages (u: UID)(st,output)(i:POS_DTIME): bool =
    st(i) = st(i-1) AND
    COND users_(u) -> output(i) = list_msg(u, {m :
    (msgOf(st(i-1),u)) | ms_(u,m) = read})
       , NOT users_(u) -> output(i) = error
    ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs
     , groups_ = st(i-1)`groups

command : DATATYPE
  BEGIN
    nothing: nothing?
    e_add_user(u:UID,un:USER): add_user?
    e_add_group(g:GID,gn:GROUP): add_group?
    e_register(u:UID,g:GID): register_user?
    e_send(u:UID,g:GID,txt:TEXT): send_message?
    e_read(u:UID,m:MID): read_message?
    e_delete(u:UID,m:MID): delete_message?
    e_list_new(u:UID): list_new_message?
    e_list_old(u:UID): list_old_message?
  END command

cmd: VAR [POS_DTIME -> command]

messenger_ft (cmd,st,output)(i: DTIME): bool =
   COND i = 0 -> st(i) = init_state AND output(i) = OK
      , i > 0 -> CASES cmd(i) OF
                 nothing:       st(i) = st(i-1)
                 AND output(i) = output(i-1)
                , e_add_user(u,un):  add_user(u,un)(st,output)(i)
 , e_add_group(g,gn): add_group(g,gn)       (st,output)(i)
 , e_register(u,g):   register_user(u,g)    (st,output)(i)
 , e_send(u,g,txt):   send_message(u,g,txt) (st,output)(i)
 , e_read(u,m):       read_message(u,m)     (st,output)(i)
 , e_delete(u,m):     delete_message(u,m)   (st,output)(i)
 , e_list_new(u):     list_new_messages(u)  (st,output)(i)
 , e_list_old(u):     list_old_messages(u)  (st,output)(i)
 ENDCASES
   ENDCOND

     % When listing messages
     %(either the new messages or the new messages),
     % this property asserts that
     %   0. it is requested by a valid user
     %   1. the user has access to all the returned messages
list_message_privacy(st: STATE,output: OUTPUT): bool =
     list_msg?(output) IMPLIES st`users(id(output))
     AND subset?(ms(output), msgOf(st,id(output)))

     % When showing a message, this property asserts that
     %   0. it is requested by a valid user
     %   1. the user has access to the requested message
     %   2. the displayed text is actually the body of the message
show_message_privacy(st: STATE,output: OUTPUT): bool =
     show_msg?(output) IMPLIES
              st`users(id(output))
  AND msgOf(st,id(output))(msg(output))
  AND st`info(msg(output))`content = txt(output)

    % If we ignore the case where the user can do "nothing", we can
    % show (without induction) that list_message and show_message
    % do not leak private information.
privacy_weak : THEOREM
   FORALL (i: DTIME): messenger_ft(cmd,st,output)(i)
   AND (i = 0 OR cmd(i) /= nothing)
         IMPLIES
            list_message_privacy(st(i),output(i))
 AND show_message_privacy(st(i),output(i))

    % Like privacy_weak except that we account for doing "nothing"
    % This requires induction.
privacy : THEOREM
        (FORALL (i: DTIME): messenger_ft(cmd,st,output)(i))
   IMPLIES FORALL (i: DTIME):
            list_message_privacy(st(i),output(i))
  AND show_message_privacy(st(i),output(i))

%|- privacy : PROOF
%|- (then (skeep)
%|-  (spread (induct i)
%|-   ((then (inst -1 0) (grind)) (then (inst -1 0) (grind))
%|-    (then (skeep) (inst -3 "j+1") (grind)))))
%|- QED
```

```
inv1 (s : STATE) : bool = FORALL (u:(s`users),m:(s`msgs)):
              s`ms(u,m) /= unavailable
              IMPLIES  readership((s`membership),(s`info))(u,m)
inv2 (s : STATE) : bool = FORALL (m:(s`msgs)):
    s`membership( s`info(m)`sender,s`info(m)`recip )

inv1_send : THEOREM
  i > 0
      AND send_message (u,g,txt)(st,output)(i) AND inv1(st(i-1))
   IMPLIES inv1(st(i))

inv2_send : THEOREM
  i > 0
      AND send_message (u,g,txt)(st,output)(i) AND inv2(st(i-1))
   IMPLIES inv2(st(i))

inv1_read : THEOREM
  i > 0
     AND  read_message (u,m)(st,output)(i) AND inv1(st(i-1))
   IMPLIES inv1(st(i))

inv2_read : THEOREM
  i > 0
     AND  read_message (u,m)(st,output)(i) AND inv2(st(i-1))
   IMPLIES inv2(st(i))

inv1_holds : THEOREM
    (FORALL (i: DTIME): messenger_ft(cmd,st,output)(i))
    IMPLIES (FORALL (i: DTIME): inv1(st(i)))

inv2_holds : THEOREM
    (FORALL (i: DTIME): messenger_ft(cmd,st,output)(i))
    IMPLIES (FORALL (i: DTIME): inv2(st(i)))

END messenger

use_cases : THEORY
BEGIN
```

```
IMPORTING messenger

n1,n2 : USER
% Existence TCC generated (at line 352, column 0) for  n1: USER
  % unfinished
%n1_TCC1: OBLIGATION EXISTS (x: USER): TRUE;
gn1,gn2 : GROUP
g1 : GID
u1,u3 : UID
t1,t2: TEXT
cmd : [POS_DTIME -> command]
st : [DTIME -> STATE]
output : [DTIME -> OUTPUT]
i : VAR DTIME

distinct_users : AXIOM
  u1 /= u3

post_st6(s : STATE) : bool =
         NOT s`users(u3)
      OR NOT s`groups(g1)
      OR NOT s`membership(u3,g1)

use_case1 : bool =
        (FORALL i: messenger_ft(cmd,st,output)(i))
    AND cmd(1) = e_add_user(u1,n1)
    AND cmd(2) = e_add_group(g1,gn1)
    AND cmd(3) = e_register(u1,g1)
    AND cmd(4) = e_send(u1,g1,t1)
    AND cmd(5) = e_add_user(u3,n2)
    AND cmd(6) = e_send(u3,g1,t2)

uc1_state_0 : LEMMA
       use_case1 IMPLIES post_st6(st(0))

uc1_state_1 : LEMMA
               use_case1
       IMPLIES post_st6(st(1))
```

```
           AND st(1)`users(u1)

uc1_state_2 : LEMMA
               use_case1
       IMPLIES post_st6(st(2))
           AND st(2)`users(u1)
   AND st(2)`groups(g1)

uc1_state_3 : LEMMA
               use_case1
       IMPLIES post_st6(st(3))
           AND st(3)`users(u1)
   AND st(3)`groups(g1)
   AND st(3)`membership(u1,g1)

uc1_state_4 : LEMMA
               use_case1
       IMPLIES post_st6(st(4))
           AND st(4)`users(u1)
   AND st(4)`groups(g1)
   AND st(4)`membership(u1,g1)
   AND NOT empty?(msgOf(st(4),u1))

uc1_state_5 : LEMMA
               use_case1
       IMPLIES post_st6(st(5))

uc1_state_6 : LEMMA
               use_case1
       IMPLIES output(6) = error

% state 0: post_st6
% state 1: post_st6
%      AND users(u1)
% state 2: post_st6
%      AND users(u1) AND groups(g1)
% state 3: post_st6
%      AND membership(u1,g1)
%      AND users(u1) AND groups(g1)
% state 4:
%          post_st6
%      AND output(4) = OK
%      AND membership(u1,g1)
%      AND NOT empty?(msgOf(u1))
% state 5: post_st6
% state 6: output = error

use_case1_correct : THEOREM
        use_case1
    IMPLIES
        output(4) = OK
    AND st(4)`users(u1)
    AND st(4)`groups(g1)
    AND st(4)`membership(u1,g1)
    AND NOT empty?(msgOf(st(4),u1))
    AND output(6) = error

END use_cases
```