Secure Group Communication for First Responders

C. Edward Chow and Ganesh Godavari

Department of Computer Science
University of Colorado at Colorado springs
1420 Austin Bluffs Parkway
Colorado springs, CO 80933-7150
USA
Email: {gkgodava, chow}@cs.uccs.edu

Abstract

In this paper, we present the design and implementation of a secure groupware for first responders, called SGFR, that is capable of secure group chat, remote file download and remote display control. It integrated Jabber instant messaging system and Keystone group rekeying system. Users are authenticated through the use of digital certificates. Group key are issued when members are joined or leaves to ensure the security policy. The performance of SGFR is also presented. The system was first developed on Linux PC then ported to an IPaq PDA running Linux as a secure information delivery platform.

1. Introduction

In these days of terrorism threats, the ability of first responders like firemen and emergency technicians to be able to reach the site of an accident or attack early is essential. The ability to co-ordinate first responders who are coming to help at a site would improve response times and also efficient utilization of resources. An important consideration of such communication is security. It is possible for terrorists and malicious elements to eavesdrop on first responder communication and use it for further destruction. One possibility could be to use the movements of emergency responders to plan further attacks so that they could be targeted. Hence there is a need to use security in order to mask the communication between first responders.

The general objective of Secure Communication is an attempt to create a framework for secure Group Communication. SGFR uses Instant Messaging platform for communication between various clients. In addition to text chatting, SGFR provides file transfer and remote display.

1 This work is based on research sponsored by the Air Force Research Laboratory, under agreement number F49620-03-1-0207. The view and conclusions contained herein are those of the authors and should not be interpreted as necessarily represented the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratories or the U.S. Government. It is sponsored by a NISSC Summer 2003 grant.

2. Related work

Secure group communications has been a hot topic for research in the recent years. There has been some work done by Lawrence Berkeley research Labs (LBL) in building a reliable multicast transport protocol (RMTP) [1] similar to TCP as ip multicast is unreliable in a peer to peer model.

Specifying policy framework for secure group communication has been studied by the Antigone [2] project at University of Michigan.

3. Instant Messaging (IM)

IM uses Internet technology to allow people to send text messages that are delivered in real time. One needs to download instant messaging software and install it on his/her computer. After the software is installed and you've registered a unique name with the IM service provider, you log into a central server that indicates that you are available. The messages are sent either through the service's central server or, directly from one computer to another using peer-to-peer technology. There are a number of instant messaging services like AOL [3], Yahoo [4], and MSN [5] which are widely used by people. One of the Major problems between these various IM's is interoperability. AOL users can't talk to MSN, and MSN users can't talk to Yahoo. If you want to use IM to communicate with someone, they must have the same service.

Currently Two groups within the Internet Engineering Task Force (IETF) are actively working to make interoperability between the various instant messengers, a reality by defining a common protocol for IM: the Extensible Messaging and Presence Protocol (XMPP) Working Group and the SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) Working Group.

3.1 Jabber - Open Source IM service

Jabber is an open XML protocol for the real-time exchange of messages and presence between any two points on the Internet [6]. It's based on the XMPP protocol. The first application of Jabber technology is an asynchronous, extensible instant messaging platform,

along with an IM network that offers functionality similar to legacy IM systems such as AIM, ICQ, MSN, and Yahoo. Jabber uses client-server architecture, not a direct peer-to-peer architecture. This means that all Jabber data sent from one client to another must pass through at least one Jabber server. Jabber clients are free to negotiate direct connections, for example to transfer files, but those "out-of-band" connections are first negotiated within the context of the client-server framework. XML is an integral part of the Jabber architecture because it is of utmost importance that the architecture be fundamentally extensible and able to express almost any structured data. When a client connects to a server, it opens a one-way XML stream from the client to the server, and the server responds with a one-way XML stream from the server to the client. Thus each session involves two XML streams.

The Jabber server plays three primary roles:

- Handling client connections and communicating directly with Jabber clients.
- Communicating with other Jabber servers.
- Coordinating the various server components associated with the server.

The only things a Jabber client must do are:

- Communicate with the Jabber server over TCP sockets.
- Parse and interpret well-formed XML "chunks" over an XML stream.
- Understand the core Jabber data types (message, presence, and iq…).

There are a number of Jabber IM clients that run on various operating systems. A list of the Jabber IM client is available at http://www.jabber.org/.

JabberX [7] is a console-mode client for the Jabber instant-messaging IM platform. With JabberX, you can send and receive messages, browse and use Jabber services, participate in Jabber groupchats and search Jabber user directories.

4. Group Communication and Group Key Management

Group Communication can be explained as "Communication between two or more people, with a common goal, in which every person can participate with other members". Group communication is a critical area that currently inspires a lot of research. One of the major aspects of group communication is security. Network based applications like online stock markets and command-and-control systems use group Communications. Internet Research Task force has formed a Group Security (GSEC) [8] to identify problems related to Group communication

A group key management Server establishes and maintains group keys for groups of clients. Keystone [9] uses key graph technique to manage keys, thereby providing a scalable group key management scheme. A key graph is a directed a-cyclic graph with 2 kinds of nodes, u-nodes representing users and k-keys representing keys.

Keystone has the following components

- "keyserver0" is a key server program with embedded registrar.
- "keyserver" is a key server program without embedded registrar.
- "registrar" is a registrar program.
- "specwriter" is a specification writer program.
- "libks.a" is a library for client control functions

Figure 4.1 shows the architectural overview of Keystone. For the clients to register with the keyserver the authentication protocol used is SSL/TLS [10]. The keyserver can provide access control using certificates. Once the client is authenticated, the keyserver generates the client's individual key, which is used to protect further communications between them. As the keyserver can become a bottleneck, keystone provides one or more registrars. Different registrars may use different authentication services to authenticate different set of clients at the same time.

The control manager of a client is responsible for client control functions i.e. sending requests and processing rekey messages. Each client has a data processor, which is not a part of the keystone. The key server processes requests from client, changes keys and distributes new keys to client using the rekey messages using unicast or multicast.

5. SGFR Design and Analysis

SGFR integrates JabberX with Keystone and Jabber Server and provides secure group communication between various JabberX clients using the keystone. It provides facilities like secure group chatting, file transfer and remote display.

The Design of SGFR is taken to fit in using the Keystone architecture, yet providing an independent framework for secure Instant Messaging platform. The figure shows how the JabberX client interacts with the control manager and Jabber server for authentication the JabberX client sends the data to the conference module of the Jabber Server which broadcasts data to various JabberX clients.

Association of the JabberX client with the Keyserver with Jabber server follows the following rules:

1) User logs into the Jabber server
2) If login successful, the client registers with the Keyserver.
3) On joining/beginning of a group conference the Keyserver gives a key to the client.
4) On leaving the group the keyserver generates a key for the remainder of group that is different from the earlier one.

The messages displayed in Figure 5.1 is because we are not connected to the outside world so its not able to contact jabber server for updates. if we look into the error message it shows that its trying to contact jabber.org

Figure 5.2 shows the output produced by the keystone when two JabberX clients joined the group. The data enclosed in the red box shows the key generated when each client joined in the group.

The data sent to the group is encrypted using blowfish [11]. The message is sent out as a normal message to jabber server. Upon receiving the message the client tries to decrypt the key using the group key given by the keyserver. If decryption fails message is ignored. Figure 5.4 shows the packet sent between client and server captured using ethereal []

User "ganesh" joined an existing conference started by user "ayen". So he cannot read what has taken place in the group before he joined the conference.

5.1 File Transfer and Remote Display

One of the client can send a file to all other clients in the group. Once the File transfer is complete it will be automatically displayed on the all the groups web browser. The reason for choosing a web browser is its inherent ability to display/open various applications depending on the type of the file. As the Jabber clients are free to negotiate direct connections for transferring files within the context of the client-server framework. In a group communication if all the clients try to establish a peer-to-peer connection with the sender of the file, the sender of the file can become overloaded. Moreover first responders like fire fighters cannot be envisioned to carry heavy equipment like laptops.

We decided to send file as normal group chat message but with a message type of 'filetransfer'. The client on receiving the message interprets the message as a 'file transfer' rather than 'group chat' message. One of the draw back with this approach is conference module of the Jabber Server is not optimized to receive a lot of big chunks of messages.

We have successfully ported JabberX client onto IPAQ PDA running Linux. One of the problems faced is formatting of data on screen. JabberX uses 'iconv' which is a part of the glibc for formatting of messages into UTF-8. The older version of glibc libraries has a bug in iconv. So the iconv libraries had been updated but the problem persists. We had to display stuff in UTF-8 format instead of any other local format set using the LANG environment variable.

6. SGFR Testing Results

We have tested the time taken for client registration, group join and group leaving. Table 6.1 shows that on average the time for client registration is 0.2 sec, join a group 0.42 sec, and leave a group is 0.36 sec.

We have tested the time taken for file transfer. Table 6.2 shows that on average time taken for each kilobyte of file transfer is 5043.92 (ms). The reason for this poor performance is because conference module of the jabber server is not designed for handling large chunks of data.

7. Lessons learned

One of the critical goal of this project is to provide a framework for a secure group communication using the existing tools for instant messaging server like Jabber, key server management like Keystone and instant messaging clients like JabberX.

Getting these various tools and their dependencies to work with each other was a great learning experience. Some of the problems faced were

1) Cryptolib -1.2 [12] is a part of keystone key management system. work on Cryptolib libraries were stopped way back in 1995. Cryptolib libraries has no problem running on Solaris but it has a problem running on Linux as multiple jumps to the same location between function call is overridden on Linux.
2) Remove the conflict between the Keystone client library and P-thread library caused by redefinition of variables constants.
3) Remove the conflict between function declaration of Cryptolib and OpenSSL caused by md2, md4 and md5.
4) Porting the JabberX client onto ipaq PDA caused a lot of problems of formatting data onto screen. The problem was solved when we upgraded the iconv libraries and setting the 'LANG' environment variable to UTF-8 encoding.

8. Conclusion and Future work

The goal of SGFR was to work on creating a framework for secure group communication for Instant Messaging using group communication tools like Keystone, Jabber server and Jabber client. We need to extend the work by improving the file transfer capability using Reliable Multicast Transport Protocol.

We are also working on implementing wireless ad-hoc mode of communication between various client devices. Thereby improving the range of communication between various client devices like PDAs, palmtops etc.

We are also working on improving keystone's error handling mechanism between keyserver/registrar and client manager. We are also focusing on improving keystone client manager by moving it into socket layer and providing socket layer API between a client manager and data processor.

9. References

[1] Reliable and secure group communication http://www-itg.lbl.gov/CIF/GroupComm/

[2] Antigone, policy definition and analyzer for group communication http://antigone.eecs.umich.edu/content/antigone-2.0.12/docs/html/index.html

[3] America on Line. Instant Messenger ™ of AOL http://www.aol.com/

[4] Yahoo ™ Instant Messenger of Yahoo http://www.yahoo.com/

[5] MSN ™ Instant Messenger of Microsoft http:// msn.com/

[6] Jabber Software Foundation http://www.jabber.org/

[7] JabberX, Jabber client http://jabberx.jabberstudio.org/

[8] IRTF research group, Group security (GSEC) http://www.irtf.org/charters/gsec.html

[9] Chung Kei Wong and Simon S. Lam "Keystone: A Group Key Management Service" *Proceedings International Conference on Telecommunications*, May 2000

[10] RFC2246 - "The TLS Protocol Version 1.0" ftp://ftp.isi.edu/in-notes/rfc2246.txt

[11] blowfish encryption algorithm http://www.schneier.com/blowfish.html

[12] J. Lancy, D. Mitchell, and M Blaze. Cryptolib-1.2 http://www.homeport.org/adam/crypto/cryptolib.phtml

[13] http://www.ethereal.com/

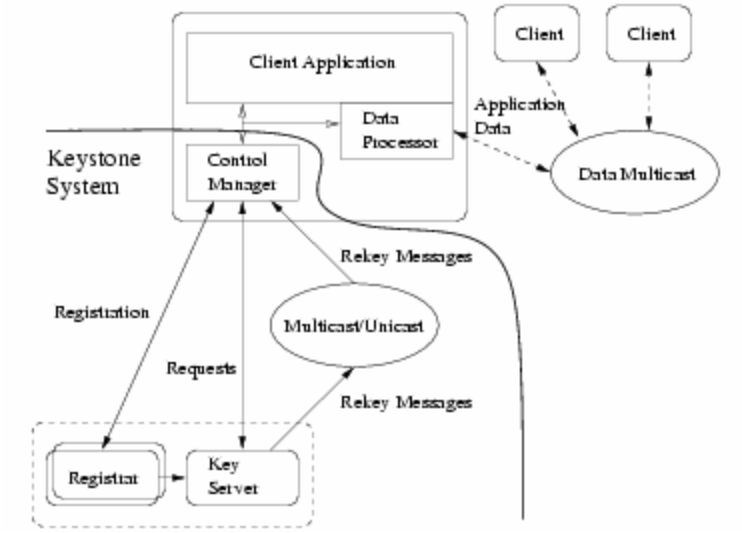

Figure 4.1 architectural overview of Keystone

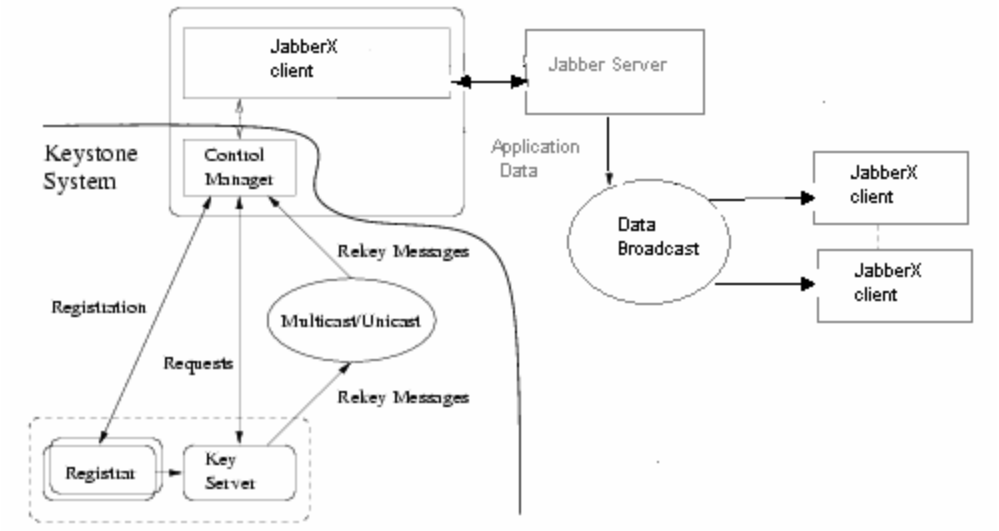

Figure 5.1 shows the interaction between various components

```
File  Edit  View  Terminal  Go  Help
[root@oblib jabber-1.4.2]# ./jabberd/jabberd -c jabber.xml
20030916T10:09:23: [notice] (-internal): initializing server
20030916T10:09:54: [notice] (update.jabber.org): bouncing a packet to jsm@update
.jabber.org/1.4.2 from localhost: Server Connect Timeout
20030916T10:09:54: [alert] (localhost): hostname maps back to ourselves!
20030916T10:09:54: [notice] (localhost): failed to establish connection
20030916T10:09:54: [warn] (localhost): dropping a packet to localhost from jsm@u
pdate.jabber.org/1.4.2: Server Connect Failed
```

Figure 5.2 shows the output of the Jabber server running on a machine

```
[root@oblib Keystone1.0]# ./keyserver0
pid 23976 in progress
pid 23976 exited
group g1 key (100000,2): 5def1274 eca51de5 5d30b65f 9cf37007 5def1274 eca51de5
req rekey: [N(100002,1)] [N(100000,2)] (108)
join rekey
0105006c 00000000 00000002 00000001
1351d29c 44625901 42e5f4b5 b9852684
d5892548 061fdf6a 1885d461 a168d3e1
c7da83ba 6eae79ec 5857d567 77906ade
f635e06c a3ba820a dbda1127 9004f194
388eb20e c6857b75 8a9fa8f8 1a168074
9240821e b3cf284b 3e1624f1
JL_JL1:
rekey msg 0 (g1):
pid 24020 in progress
pid 24020 exited
req rekey: [N(100003,1)] [N(100000,3)] (108)
join rekey
0105006c 00000000 00000003 00000001
b316f5e9 9244c27f e7bfc2d5 c40f3ccd
46ea5a55 58316b96 488ad2e3 c8d012a2
17b481c6 b2c72901 905b97ee 45986e56
0a7131ef c8dc57ac 92b575a6 94294a8f
b600cc55 5ca76321 728022af 4a07ad99
e684e16a 7e9612b6 e3643ec2
JL_JL1:
rekey msg 0 (g1): [j(100000,3)(100000,2)]
rekey msg 132
01040084 00000000 00000001 00000001
0402002c 000186a0 00000002 000186a0
00000003 1fbacec6 2146f863 6d1c2425
0569e904 755c0800 37c32ae8 07000048
00000000 d6f50b30 911f653b bdae8c07
cf337be1 5bdcd195 d9fb4e2d 678fb7f4
82631594 329be29a bbb32e24 4e73c9f6
920ead76 20024322 4ea758de f77360fb
300a7d46
group g1 key (100000,4): 4dcd385a f96e9452 ac8cb02c e705cdae 4dcd385a f96e9452
```

Figure 5.3 shows the output of the Keystone

```
0040  d1 24 3c 6d 65 73 73 61  67 65 20 74 79 70 65 3d   .$<messa ge type=
0050  27 67 72 6f 75 70 63 68  61 74 27 20 74 6f 3d 27   'groupch at' to='
0060  67 6b 67 6f 64 61 76 61  40 6f 62 6c 69 62 2e 75   gkgodava @oblib.u
0070  63 63 73 2e 65 64 75 2f  4a 61 62 62 65 72 58 27   ccs.edu/ JabberX'
0080  20 66 72 6f 6d 3d 27 67  31 40 63 6f 6e 66 65 72    from='g 1@confer
0090  65 6e 63 65 2e 6f 62 6c  69 62 2e 75 63 63 73 2e   ence.obl ib.uccs.
00a0  65 64 75 2f 67 61 6e 65  73 68 27 20 63 6e 75 3d   edu/gane sh' cnu=
00b0  27 27 3e 3c 62 6f 64 79  3e 78 47 62 61 72 37 39   ''><body >xGbar79
00c0  33 78 31 67 3d 3c 2f 62  6f 64 79 3e 3c 2f 6d 65   3x1g=</b ody></me
00d0  73 73 61 67 65 3e                                  ssage>
```

Figure 5.4 shows the encryption of the message from client to server.

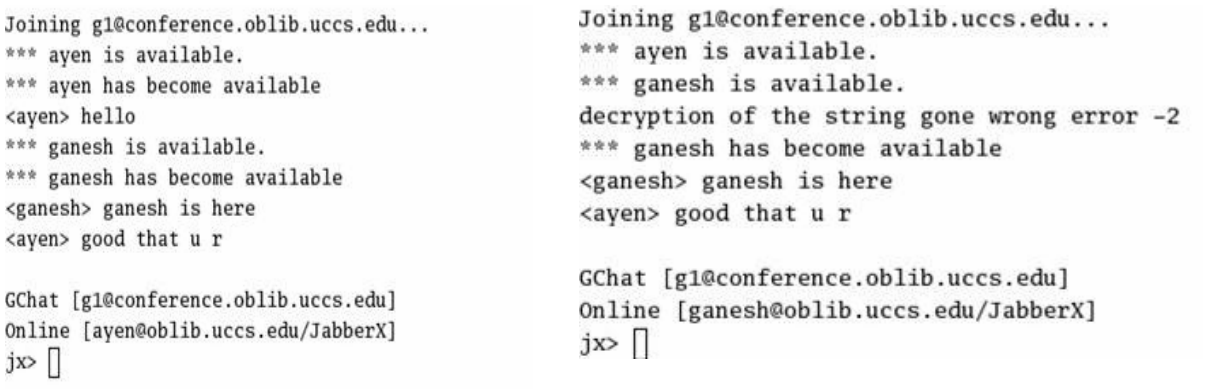

Figure 5.5 shows the chatting between 2 clients

Runs	Client Registration time (ms)	group join time (ms)	group leave time (ms)
1	279.62	233.46	135.54
2	249.28	652.74	126.78
3	253.93	706.04	769.08
4	259.46	118.15	434.12
Avg/Run	260.5725	427.5975	366.38

Table 6.1 time taken for client registration group join, group leave

File size	Time Taken (ms)
8.5K	35302.47
25K	105986.05
60K	305934.53
195K	1007949.38

Table 6.2 time taken for file transfer

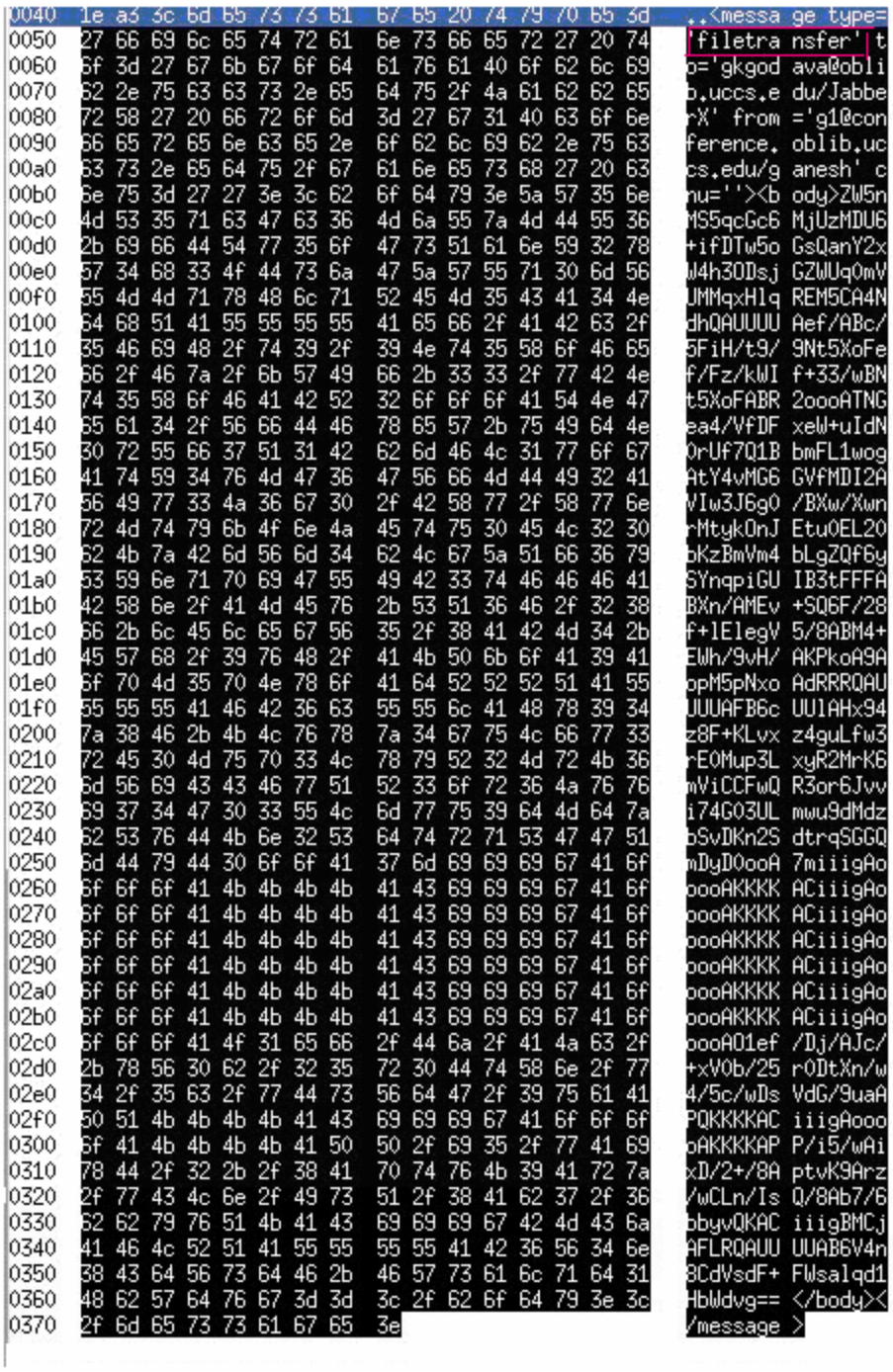

Figure 5.6 shows the message between 2 clients during file transfer