Department of Computer Science

Rochester Institute of Technology

MS Project Proposal

Security in an Ad Hoc Network using Many-to-Many Invocation

Revision:	1.2
Date:	February 11, 2003
Author:	Jefferson S. Tuttle (**jst1734@cs.rit.edu**)

Approvals:

____

Chair	Alan R. Kaminsky	Date

____

Reader	Hans-Peter Bischof	Date

____

Graduate Coordinator	Hans-Peter Bischof	Date

1 Summary

This project involves providing a mechanism for applications running Many-to-Many Invocation (M2MI) to establish a secure communication channel. The first part involves establishing and maintaining group keys. These group keys will be used for authentication. The next step is to use these group keys to establish session keys that will be used to encrypt M2MI method invocations, thus implementing the secure communication channel. The final step involves integrating these session keys into M2MI.

2 Overview

As with any communication in a public arena, there exists a desire to send sensitive data through that communication channel, but to limit whom can see the data being communicated. There are two parts to this requirement. The first is to authenticate whom you are talking to. The second is to hide the data that is being communicated. These ideas are nothing new. Authentication and encryption have been around for a while, and there are some strong algorithms in place for performing these two steps. A relatively new area of security is secure group communication. Within this field, secure group communication in an ad hoc network is an area of current research. Ad hoc networks bring two challenges to secure group communication. The first is the lack of a central server. Central servers are typically used to provide the authentication step in establishing a secure communication channel. Maintaining membership lists and storing public keys of group members on the devices that want to communicate can overcome this problem. This has drawbacks, especially in mobile ad hoc networks. In mobile ad hoc networks, the memory sizes of communicating devices are typically quite limited, and are not capable of storing large membership lists or a large number of keys.

This project overcomes these challenges by splitting the authentication step into two phases. The first phase involves establishing and maintaining group authentication keys. Each device that will communicate in a mobile ad hoc network will store an authentication key (or key pair) on the device for each group that it belongs to. This authentication key will be established and maintained while the device can communicate with a central server (either directly or via a host computer, then downloaded from the host computer using a secure communication channel, e.g. PDA cradle, IrDA, etc.). As members of each group are added or removed, the device will get updates to the authentication key when it connects to the central server (or host computer). These authentication keys must be kept securely on the device to prevent compromise. A password-protected file may be used for this. This issue will be researched during the project.

The second phase involves using these group authentication keys to establish a session key that will be used during a single group communication session. All members of a group that have received the group authentication key will be able to participate. Each member will use the group authentication key to perform an encrypted key exchange in establishing a session key. Only members who have the authentication key will be able to comprehend the session key. The protocol for performing this encrypted key exchange will be refined during the project. An initial version of this protocol is provided in Section 3 Functional Specification.

The session key that is established with the encrypted key exchange will then be used to secure all communication among the group. For this particular project, the data that is encrypted with the session key will be the Many-to-Many invocations of a test client.

3 Functional Specification

This project will enhance the current M2MI class library to be able to perform M2MI over a secure channel. The steps that need to be performed in order to establish a secure channel are listed below. Some of these steps will be described in detail in this proposal. Others will be listed as issues that need to be addressed, either during this project, or outside this project. If it will not be addressed by this project, this will be explicitly indicated.

The following steps needed to establish a secured connection:

1. Establish a group authentication key. (Group Key Establishment)
2. Maintain the group authentication key. (Group Key Management)
3. Establish a communication session key. (Session Key Establishment)
4. Encrypt/decrypt M2MI method invocations. (M2MI Integration)

These basic steps are described in more detail in sub-sections below.

In addition to these basic steps, there are several issues that need to be addressed by the project. These are:

1. How to handle multiple devices starting a communication session at the same time. (Multiple Startup)
2. How to merge communication sessions that use the same group authentication key, but were formed separately from each other. (Group Merge)
3. Rolling over communication session keys so that the same session key is not used for long periods of time. (Key Rollover)
4. Defining/choosing an encryption/decryption algorithm that can be implemented on small devices. (Encryption Algorithm)
5. Differentiating between M2MI messages encrypted with the session key and random messages sent by an imposter. (Imposter Detection)
6. Measuring memory and CPU usage to determine if the algorithms developed are appropriate for small devices. (Measurements)

Finally, in order to show that this project works, a test client needs to be generated, or an existing M2MI test client needs to be modified, in order to show the full integration of the secure channel. (Test Client)

Section 3.1 describes the steps for establishing the secure connection. Section 3.2 describes in more detail the issues that need to be addressed in this project. Section 3.3 describes the test client that will be implemented by this project.

3.1 Establishing the Secure Connection

3.1.1 Group Key Establishment

The group authentication key can be established in a number of ways. It is assumed that a central server will maintain the group authentication key, which will be downloaded to devices that want to perform secure M2MI. The method of downloaded the key may vary between devices, as described in this section.

On some devices, a secure connection to a central server may be possible directly from the device (e.g. Public Key Infrastructure). In this case, the device can establish the secure connection and download the group authentication key directly from the central server.

On some devices, a secure connection to a central server may be impossible to establish directly from the device. However, if the device has a direct connection to a PC host computer, e.g. cradle, IrDA, etc., the host computer can download the group authentication key from the central server, then download the group authentication key to the device.

Because the types of devices that will be using M2MI will vary greatly, this project will not implement this step. This project will assume that a group authentication key has been downloaded to the device using some secure communication channel (either to the central server or to a host computer), and is stored in a file, or other non-volatile memory on the device.

3.1.2 Group Key Management

The group authentication key(s) for a device will be maintained in non-volatile memory on the device. For purposes of this project, it will be assumed that these keys will be stored in a disk file. However, the interface for retrieving the group authentication key from non-volatile memory will not assume that it is a disk file.

If members are added to or removed from the group, the group authentication key should be updated on the central server. This allows for forward and backward secrecy.

Any changes to a group authentication key at the central server will be propagated to the device using the same methods described in Section 3.1.1 Group Key Establishment. In other words, the device will need to download the new key, either directly from the central server, or through its host PC.

3.1.3 Session Key Establishment

The following interface will be used to establishing a session key.

```
interface SessionKeyEstablisher {
      void requestSessionKey ( String groupName );
      void reportSessionKey (      String groupName, byte[] encryptedKey,
                        long keyID);
}
```

Devices that want to perform secure M2MI will use interface `SessionKeyEstablisher` to obtain a session key. At startup, they will invoke method requestSessionKey to try to obtain an existing session key. A timer will then be set to limit the amount of time the device will wait to obtain a session key. Each device will also export an object that implements interface SessionKeyReceiver. If method reportSessionKey gets invoked on the exported object before the timer goes off, the timer will be canceled, and the session key received in reportSessionKey will be used as the session key. If the timer goes off, the device will generate its own session key, and invoke reportSessionKey to tell anybody else who might be out there that they have established a new key.

Each device will invoke reportSessionKey under the following conditions:

- Upon reception of a requestSessionKey invocation, to let other devices know what the current session key is. However, to avoid a broadcast storm (see [6]), each device will create a short, random backoff timer. When that timer expires, it will respond with a reportSessionKey invocation. If the device receives a reportSessionKey invocation before the backoff timer expires, the timer will be canceled, and it will not invoke reportSessionKey.

- Periodically, to help with merging of communication groups and to let other devices know that the key is still active.

- After a given expiration period, at which point in time the device will generate a new session key and broadcast the new key. Whenever a device receives a reportSessionKey with a higher key id, it will use the new session key and key id received, and reset its expiration timer. This expiration timer first set on the first reportSessionKey the device receives, or when the requestSessionKey timer expires.

More details on these and other conditions are provided in Section 3.2 Protocol Issues.

3.1.4 M2MI Integration

The goal is to leave the M2MP layer alone, and to only modify the M2MI layer. A new message format will be defined for secure M2MI. The new message format will contain a different message prefix, which will be used to distinguish between secure and unsecured M2MI messages. Messages with the old message prefix will be processed as they were before the project. The new M2MI security module will process messages with the new prefix.

The new message format will be defined as part of the project.

3.2 Protocol Issues

This section describes issues that need to be resolved during the project.

3.2.1 Multiple Startup

If many clients startup at about the same time, how will they arrive at a single sessionKey?

As described in Section 3.1.3 Session Key Establishment, each device will invoke requestSessionKey, and then listen for a reportSessionKey invocation. If a reportSessionKey invocation occurs within a given timeout period, the key received will be used as the current session key. If no key is received, the device will create its own key with a unique key id, and broadcast these using requestSessionKey. If another device starts up about the same time, there are three possible outcomes:

1. Second device receives reportSessionKey of first device within its timeout period. In this case, both devices will use the session key and key id sent by the first device.

2. Second device starts up before the first device, and declares its session key and key id before the first device times out waiting for reportSessionKey. This is the same as case 1, but looking at it from the other device’s perspective.

3. Both devices timeout before receiving any reportSessionKey invocations. In this case, both devices will invoke reportSessionKey. In this case, the reportSessionKey invocation containing the largest key id wins.

This scenario was given with two devices. This could easily be extended to N devices by looking at any two devices in the set of N, and deciding the outcome of those two devices.

3.2.2 Group Merge

If a session key has been established, and a new session key is being reported by reportSessionKey, how do these different "groups" get merged?

Nothing special needs to be done in this case. If a device receives a reportSessionKey invocation with a larger key id than is currently associated with the current session key, the new session key becomes the current session key, and new key id is saved.

3.2.3 Key Rollover

In cryptography, the longer a key is used, the more likely it is that the key may be cracked. The security package developed in this project must take care to limit the amount of time that a session key is used. If a session key has been used for more than a TBD amount of time (which may be configurable), then it should invalidate its key and create a new session key.

This is easily accomplished with this protocol. When a device determines that its session key has been used for longer than its [configurable] amount of time (i.e. its expiration timer expires), it will generate a new session key and a new, higher, key id. The new key and key id are then broadcast using reportSessionKey. All other devices in the group will receive the new key id, see that it is larger, and switch over to using the new session key and key id.

3.2.4 Encryption Algorithm

Need to define or choose an encryption/decryption algorithm that can be implemented on small devices. This algorithm should use fairly large keys for the group authentication key, and relatively small (64-256 bits) for the session key. Perhaps the key sizes can be configurable, within these defined limits. The algorithm should be able to run on small devices, which means that it should be able to run in a reasonable amount of time on a slow CPU, and should not require a lot of memory to process. The processing speed and memory requirement limits need to be defined.

3.2.5 Impostor Detection

The security package should be able to differentiate between M2MI messages encrypted with the session key and random messages sent by an impostor.

To overcome this issue, each message that needs to be encrypted will be sent through a one-way hash to create a fixed length message digest. This message digest will be concatenated to the original message. The combined message and message digest will then be encrypted with the appropriate key (session key or group authentication key, depending on which type of message was being encrypted). On the receiving end, the message will be decrypted with the appropriate key. The message will be separated into the original message and the fixed length message digest. The original message will be sent through the one-way hash function, and compared with the message digest. If they match, the message is authenticated, and can be used. If they don't match, the message will be discarded, and perhaps a counter will be incremented to show how many times a bad message was received.

3.2.6 Measurements

In order to ensure that this package will work on a small device, the following measurements will be taken and evaluated.

Measurement	Description/Purpose
Volatile Memory Usage	This measurement will be used to determine the amount of RAM used by this package.
Non-Volatile Memory Usage	This measurement will determine the amount of NVRAM needed by this package. This will include the code size plus the amount of memory needed to store all keys and IDs.
Session Key Generation Time	Amount of time required to exchange messages in order to establish a session key. This time will be measured with various numbers of clients wishing to join a communication group at the same time.
Message Encryption Time	Amount of time required to encrypt a message. This time will be measured with small and large messages.
Message Decryption Time	Amount of time required to decrypt a message. This time will be measured with small and large messages.

3.3 Test Client

An existing M2MI test client will be modified to use this security package in two ways. One modification will send secure M2MI method invocations. The other will act as an impostor and inject bogus messages into the system. The goal is to make sure that an impostor cannot interfere with secure M2MI communication. The impostor will attempt to send bogus session keys as well as bogus method invocations. The goal is to make sure that the secure test client does not use these bogus session keys, and that the secure test client does not invoke the bogus method invocations.

The test client that will most likely be modified is the chat program, thus creating a secure chat program.

4 Deliverables

- Modifications to the Java Archive implementing the edu.rit.m2mi package. This archive will also include the test client and classes needed to implement the group key management. This will also include an “impostor” version of the test client, which will show the robustness of the software. All code will be contained in a new edu.rit.m2mi.security package. The test clients will be included in a separate package under edu.rit.m2mi.test.
- Final API describing the interfaces used, and an overview of how to use the new security portion of

the M2MI package, including how to feed the group name and session keys to the security package and how to tell M2MI to encrypt messages or not. (10-20 pages)

- Technical report of 20-30 pages containing the following:
- Security overview describing:
- Alternative security methods considered
- Reason for choosing this method
- Detailed description of security protocol, including the format of encrypted M2MI messages, the algorithm(s) used to encrypt/decrypt data, and what security issues are addressed by the protocol.
- Performance measurements and discussion.
- Issues that were addressed in the project.
- Future work.
- References

5 References

[1] Benaloh, Josh and de Mare, Michael. One Way Accumulators: A Decentralized Alternative to Digital Signatures. In *Advances in Cryptology (EUROCRYPT '93), Proceedings of the Workshop on he Theory and Application of Cryptoghapic Techniques*, pages 274-285, May 1993.

[2] Branchaud, Marc. *A Survey of Public Key Infrastructure*. March 1997.

[3] Jablon, David P. Strong Password-Only Authenticated Key Exchange. *ACM SIGCOMM Computer Communications Review*, October 1996.

[4] Kaminski, Alan and Bischof, Hans-Peter. Many-to-Many Invocation: A New Object Oriented Paradigm for Ad Hoc Collaborative Systems. In *Proceedings of the 17th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 2002)*, November 2002.

[5] Menezes, A., van Oorschot, P., Vanstone, S. *Handbook of Applied Cryptography*. CRC Press, 1996.

[6] Ni, S.-Y., Tseng, Y.-C., Chen, Y.-S., and Sheu, J.-P. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proceeding of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 151-162, August 1999.

[7] Steiner, Michael, Tsudik, Gene, Waidner, Michael. Diffie-Hellman Key Distribution Extended to Group Communication. In *ACM Symposium on Computer and Communication Security*. March 1996.

[8] Stinson, Douglas R. *Cryptography: Theory and Practice*. CRC Press, 1995.

6 Schedule (Milestones)

- Complete proposal and get approved. (Mid January 2003)

- Code/unit test session key establishment mechanism. (Mid February 2003)
- Integrate into M2MI package. (Early March 2003)
- Code test clients/final testing. (Late March 2003)
- Complete technical report, and prepare presentation. (Mid April 2003)
- Master’s defense. (Early May 2003)