Off-the-Record Instant Messaging for Group Conversation

Jiang Bian
Department of Computer Science
University of Arkansas at Little Rock
Little Rock, Arkansas 72004
Email: jxbian@ualr.edu

Remzi Seker
Department of Computer Science
University of Arkansas at Little Rock
Little Rock, Arkansas 72004
Email: rxseker@ualr.edu

Umit Topaloglu
Department of Computer Science
University of Arkansas at Little Rock
Little Rock, Arkansas 72004
Email: umtopaloglu@gmail.com

***Abstract*—Instant Messaging (IM) is becoming an integral part of social as well as business life. The main concern with IM systems is that the information being transmitted is easily accessible. Some protection could be achieved with the use of a secure tunneling (i.e. VPN etc.). Nevertheless, even the use of VPN-like technologies does not provide end-to-end secrecy. Off-the-record (OTR) [1], designed by Borisov. N. et al., is a protocol which enable IM users to have private conversations over the open and insecure public Internet. However, the OTR protocol currently does not support multi-user chat rooms via various popular IM services. And there is a need for such a product, which provides users the opportunity to meet in a IM-based, virtual, and encrypted chat room. This project implements an extension of the two-party OTR protocol, named Group OTR–GOTR. GOTR enables users to have a free and secure multi-user communication environment with no proprietary software requirement. The case study describes a proof of concept plug-in of GOTR developed for the GAIM [2], as well as the plug-in implementation details. Such a product is believed to be beneficial to small businesses to keep their privacy and their competitiveness.**

I. Introduction

Current trend in message exchange systems around the world is Instant Messaging (IM). IM systems are becoming available even on cellular phones, pagers, and it is expected that more mobile devices will support at least one IM technology and these technologies will be employed for business solutions. Given the status quo of measures implemented for privacy maintenance or intellectual property, there is still much space for improvement.

Users seem to prefer IM systems because they are not as intrusive as phone calls, yet are more interactive than e-mails. Some of the popular IM systems include: Microsoft's MSN (or Windows) Messenger (MSN) [3], American Online Instant Messaging (AIM) [4], Google Talk [5], etc. and these systems are changing the way people communicate with friends, family and business partners. On the other hand, confidentiality has not been addressed in the IM environment. Most IM protocols were implemented on the top of the existing public Internet service, where there is no guarantee of the secrecy of the transmitted messages. A message exchanged between users sitting next to each other may still need travel a path through several routers. Furthermore, if there is no proper encryption and/or authentication in place, messages are almost open to any eavesdropping, account hijacking, man-in-the-middle, denial of service, and similar types of attacks that is potentially harmful for most of the current distributed network applications.

As IM systems become part of the social and business infrastructure, concerns related to protecting the content of messages arise. Although it can be extended to other application areas, the focus of this paper is limited to protecting the intellectual property of small businesses. The reason for focusing on small businesses is that, a vast majority of them cannot afford implementing a service which ensures secrecy of their intellectual property. Unveiling the company secrets may harm its competitive advantages and result in capital loss.

A. IM system's architecture

One could probably consider Internet Relay Chat (IRC) as the first IM system. IRC provides a real-time communication service among a group of people regardless of their physical locations. In IRC, each participant needs to connect to a centralized IRC server and join a conversation channel (or topic). There are two types of conversations in IRC: one is public in which messages could be read by everyone in the same channel and the other is private in which messages are exchanged between only two parties, who may or may not be on the same channel [6]. Later on, the new generation IM systems such as AIM, MSN, ICQ, Yahoo Messenger, Google Talk etc. appeared. These systems share an identical concept and provide similar functionalities such as real-time chat (peer-to-peer or/and chat room) , file transfer and so on.

Instant Messaging is a typical client-server distributed network application and as such can be partitioned into two fundamental communication models:

- **Client-Server-Client:** AAll messages being exchanged among participants need to go through a centralized server regardless of the messages type, either system messages (i.e. the messages that are taking place between the client and the server to exchange status information like: users buddy lists, IP addresses, client status, etc.) or the actual conversation payload.
- **Peer-to-Peer:** Only the system queries and control messages are exchanged between the clients and the server. Upon initiating a conversation, one party queries the

server to get another's IP address and uses this information to establish a communication channel. In this model, the server stores user specific information such as user profiles, clients' IP addresses, client version, etc. and provides a querying service for clients to find other party's address information.

B. Potential Vulnerabilities

The message packages in public IM systems need to traverse through the public Internet regardless of the model and structure utilized. In general, these messages are not encrypted and an eavesdropper could easily stand on one router between the IM users, sniff their communication and access the messages. Retrieving the contents of IM messages is a rather trivial task once one grabs the packets traversing in the network. Eavesdropping is considerably easier on a LAN network, since the packages are broadcast and every node could reach the distributed packet unless the LAN network is switch based. For the switch based network, an eavesdropper could still sniff the traffic from the line that connects the switch to the router. There are also other available mechanisms for sniffing traffic on a switched LAN; however, it is beyond the scope of our paper. The discussion about an eavesdropper sniffing traffic is to justify the need for protecting the content of the IM messages utilizing an encryption method. Nonetheless, the problem still remains unresolved as to how one would distribute the keys securely, as well as, how the identity of a user can be verified.

Account Hijacking is another form of attack by which one could hijack another user's IM account and impersonates that user in subsequent conversations with others. The reason this attack is so successful, is because most of the IM systems have been using vulnerable authentication mechanisms. Session identifiers, widely used by most of the IM protocols, are not difficult to forge. An attacker could produce a similar legal key in accordance with the identifier's format.

Man-in-middle attack is a type of attack that exposes improper key exchange schemes of IM systems. For instance, the Diffie-Hellman(DH) [7] key agreement protocol, which is used by lots of IM security products, is vulnerable to this attack. Suppose Alice and Bob want to communicate privately and Alice initiates the Diffie-Hellman key exchange protocol. In case of an eavesdropper, Carol, intercepts Alice's public key and sends her own public key to Bob. When Bob replies, Carol gets Bob's public key, substitutes it with hers and replies to Alice. After all, Alice and Bob will think they are sharing a private secret with each other and all their messages encrypted by this secret are safe. But in fact, they are talking through a middle-man (Carol). Carol could even go one step further by modifying the messages and then re-encrypt with the appropriate key before delivering to the other party. The lack of authentication scheme employed in Diffie-Hellman key agreement protocol allows aforementioned vulnerability. The issue can be overcome by adding fingerprints (or digital signatures) to each message.

The above mentioned security breaches in an IM environment are usually addressed by employing encryption and authentication algorithms. Nevertheless, utilizing only confidentiality and authentication schemes are not good enough to provide an off-the-record conversation environment, in which the deniability property is also satisfied. In 2004's Workshop on Privacy in the Electronic Society (WPES), Borisov, N., et. al. proposed and implemented the OTR protocol [1]. In a later version of the OTR protocol, they have fixed a security flaw pointed out by Raimondo, et al [8]. The OTR protocal has two distinguishable security properties: perfect forward secrecy and deniability. These features will be discussed in Section III.

Yet, the OTR protocol only addressed the security issue of a two-party conversation. On the other hand, chat rooms, where several people can meet virtually to talk, could be useful not only for fun, but also for business life. Lots of corporations and organizations tend to move forward from conventional teleconference to the cheaper, more efficient online meeting environments, IM systems. For those companies, governments, and institutions, the protection of communications is extremely critical. Therefore, our intention is to address this need by utilizing the original two-party OTR protocol to create a guarded, assured and riskless multi-user IM ambiance.

Because of the differential characteristics between two-part conversation and multi-user message exchange system, we are facing amount of unique pitfalls in designing a security scheme. The growing size of chatting members apparently accumulates the complexity of the key exchange stage, which would require more time and CPU power for both encryption and decryption processes. Likewise, the synchronization of the shared keys is a further hurdle. Similar to the clock problem in distributed operating system (i.e., a universal clock system is unobtainable), it is hardly possible to generate and maintain an all-inclusive key among various users. We will go deep into these problems in Section IV, when we are dealing with our application design.

This paper is composed of four sections and the remaining sections are organized as follows: Section II presents a introduction of current popular mechanisms used for secure Instant Messaging; Section III discusses the main concepts behind the OTR project and signifies the need for chat room support, a feature missing in the OTR protocol; Section IV presents our implementation of secure group conversation based on the OTR protocol. Conclusion and future work are given in Section V and VI.

II. Related Work

Security in distributed applications is supported by usually five typical security services which are defined by the International Organization for Standardization (ISO). Those are access control/authorization, identification/authentication, confidentiality, integrity, and non-repudiation [9]. However, for an IM system, some security features need to be reconsidered, particularly, non-repudiation. In order to enable IM users to talk off-the-record, deniability (repudiation) is needed instead

of non-repudiation. In other words, a user should have the ability to deny what he or she said in the past. Several open source projects and some commercial software have addressed the security weaknesses that exist in the current IM systems. Security in these applications is, in general, supported by adding encryption and authentication schemes.

A few of such secure IM clients which are in use today will be surveyed, and our focus is the ones that have encryption and/or authentication support. A complete survey of all the secure IM clients is beyond the scope of this paper.

SimpPro [10] is a commercial software developed by Secway, a European company, providing privacy protection in IM systems. It includes:

- Encryption of messages
- Key infrastructure support for user authentication
- Encrypted file transfers for MSN and ICQ/AIM
- Secure recording of conversations

Moreover, in SimPro encryption algorithm and authentication key agreement can be customized by the users and the following symmetrical algorithms : AES (128 bits), 3DES (Triple DES, 128 bits), CAST (128 bits), Twofish (128 bits) and Serpent (128 bits), are provided for encrypting messages; and the asymmetrical algorithms such as, RSA (2048 or 4096 bits), Diffie-Hellman, ElGamal/DSA, and Elliptic curves, are used for authentication and key agreement. [11]

Gaim-Encryption [12] is a security plug-in for GAIM [1] and it uses NSS (Network Security Services) to provide transparent RSA encryption. It can automatically generate the public/private key pairs upon loading the plug-in and exchange the public keys during the initiation of conversations. Although Gaim-Encryption does not provide free choice of encryption algorithms, its API can be used as a wrapper and it can be easily extended to support different encryption algorithms. Obviously, without authenticating the participants, it suffers from man-in-the-middle attacks.

Gaim-e is again an encryption plug-in for GAIM. It uses GNUPG (GPG) to securely transfer the session keys encrypted with RC5, a block cipher notable for its simplicity [13]. The Gaim-e plug-in currently works with AOL, MSN, and Yahoo IM systems (at the time of this writing, other protocols have not been tested) [14].

Trillian is another IM client software with multi-protocol support (like GAIM) [15]. It claimed to support encrypted IM communication for AIM and ICQ protocols. Like most the others, authentication and encryption are sustained. It uses the combination of DH asymmetrical key-exchange agreement with symmetrical 128 bits Blowfish cipher algorithm to secure the messages and assure the identities.

As the literature survey shows, most of the available products for IM security address only part of the security services defined by the ISO. Often, message integrity, perfect forward secrecy, and deniability are not addressed. The next protocal we will survey examines these widely omitted concerns within the IM domain.

III. Off-the-Record (OTR) Instant Messaging System

A. Basic concepts behind the OTR

The OTR protocol contains four basic cryptographic primitives:

Perfect forward secrecy: [16] Confidentiality (i.e. only the two communicating parties, Alice and Bob, should be able to read the conversation messages), is introduced by using short-lived encryption/decryption key(s). The basic idea is that the two parities, Alice and Bob, should forget used keys after they process the old messages[2]. It is computationally infeasible to generate the previously used keys from the current key and the long-term keys. The OTR mechanism guarantees that even if an eavesdropper has the current key and can compute the shared secret being used at the moment, the compromised current key does not allow decrypting and reading previous messages. OTR improves the well known Diffie-Hellman key agreement protocol [7] to provide perfect forward secrecy. Each key is used to secure one message only and a new key is generated for transmitting the next message securely.

Digital signatures and non-repudiation: Digital signatures are used to make up for the lack of authentication in the Diffie-Hellman key agreement protocol. It is a popular approach in authentication protocols to use digital signatures that act as long-lived keys. These long-lived keys are solely for authentication purposes and are not used to encrypt IM messages in the OTR protocol. On the other hand, the signature along with the message leads to another problem. It enforces the non-repudiation property that the signatures can be verified by a third party without the cooperation of the owners, and this property conflicts the deniability service required by an OTR IM session. The solution is to use a Message Authentication Code (MAC) to authenticate the messages instead of the user's digital signature. In other words, the digital signatures authenticate the keys instead of the messages and authentication of keys provides the identification service, because only the person who has the right key can read the cipher texts. In the implementation, the previous key is used to authenticate the new key at every key refreshing stage.

Message Authentication Code (MAC) and deniability: Deniability is the ability to deny the content of conversations and it is addressed in the OTR protocol by using MACs. The way to generate each MAC is to use a one-way cryptographic hash function with a secret MAC key shared by conversation members. Alice uses her copy of the MAC key to compute a MAC of her message, and sends this MAC along with her message over a secure transmission channel; Bob verifies the integrity and authenticity of the message by computing the MAC for the received message using his copy of the shared MAC key and comparing with the MAC sent by Alice [1].

[1]GAIM is an open source, multi-protocol IM client and it is available for Linux and Windows operating systems. [2]

[2]an old message is a message for which the encryption-transmission-decryption cycle is completed

Deniability is provided by using these MACs for IM: Carol, a third party, cannot prove that the message was sent by Alice, since she does not know the MAC key shared between Alice and Bob. Even Bob cannot make a proof to the public that the message is really came from Alice. Both of them know the same MAC key, and so the message could have been forged by Bob.

Malleable encryption and forgeability: Forgeability is a stronger property than repudiation provided by the OTR. Once a key expires, the associated MAC keys for message authentication are revealed. The reason for revealing old MAC keys is to allow forgeability of messages whose encryption keys have expired. This feature enhances the ability of Alice to deny that the messages were sent by her, because anyone could calculate a MAC based on a modified message (even though it's encrypted) and validate it with one of the revealed MAC keys. OTR protocol uses a malleable encryption scheme (i.e. any change made to a cipher text will cause a meaningful change in the right position of the plaintext). Revealing of MAC keys together with a malleable encryption scheme, gives a third party the ability to forge the messages sent by Alice. Hence it gives Alice the ability to deny the fact that she has sent those messages. Moreover, anyone who recovers the MAC key in the future is unable to verify the authenticity of the messages sent in the past, which means even if somebody reads the messages it is hardly possible to track the messages' origin.

B. Security Weakness in OTR

Mario Di Raimondo, et al. [8] pointed out three major security flaws or vulnerabilities after they examined the OTR protocol:

1) An authentication failure
2) A key refreshment flaw, and
3) Unreliable support of the deniability

All these three weaknesses are caused by the choice of the key agreement protocol, which are adopted from DH protocol. First, the OTR protocal inherits a possible "identity misbinding" attack originally discovered by Diffie et al. [17]. Suppose that Alice, Bob and Eve are in the same channel. The key $(g^x, Sign_{Alice}(g^x))$ sent from Alice to Bob could be passed on by Eve but this time with her own digital signature (i.e. under Eve's name). When Bob receives the signed (by Eve) key, he thinks that he is talking to Eve, and replies $(g^y, Sign_{Bob}(g^y))$. Furthermore, Eve relays Bob's response back to Alice and gets Alice's reply. After all, Alice thinks Bob is on the other side, but actually it is Eve who is talking to her; while Bob believes the key was exchanged with Eve. Moreove, all of them had computed the same shared secret. (i.e. since Eve did not changed Alice's public key (g^x) when she was relaying the message.) This is critically harmful. For a real life example, a criminal, Eve, can use this authentication failure to mislead a customer, Alice, and a bank, Bob. One simple solution is to include identity information in the digital signature, but it will surely dismiss the deniability property.

Because signed keys were improperly used, the OTR protocol suffered from a possible man-in-the-middle attack. Suppose that Alice wants to start a private conversation with Bob by sending her signed public key $(g^x, Sign_{Alice}(g^x))$. Eve, however, stands between them and sniffs their messages. Eve replaces Alice's signature with hers, and passes it to Bob. When Bob receives this message, he thinks that Eve wants to talk with him, and he replies with his signed key $(g^y, Sign_{Bob}(g^y))$ under his name. Eve relays this message to Alice with no change. After all, Alice thinks that she is talking to Bob, since she got Bob's key under his name, but actually it is Eve who is on the other side. Therefore, in this case, Eve impersonates Bob. For this attack, one simple solution is to include identity information in the digital signature (i.e. $g^x, Sign_{Alice}(g^x, Alice)$), so that no one could modify the signature of a message. However, this change will definitely void the deniability property.

Moreover, the revealing of an ephemeral private key could cause an impersonation attack. An attacker could easily relay the message, $g^x, Sign_{Alice}(g^x)$, with $g^x, Sign_{Attacker}(g^x)$, to Bob, which was received from Alice, and he/she could compute the session key upon regardless of g^y (i.e. Bob's pubic key) responded from Bob. This session key will be valid as long as the long-term private key of Alice is not revoked. As known, this flaw defeats the goal of a secure and well-designed key protocol, the only way for a devil impersonate into the conversation is the disclosure of the long-lived private key rather than a piece of information used in the session. Therefore, they suggest doing full key refreshment periodically, which ensures that the revealing of an ephemeral private key will not affect the next fully-refreshed conversation.

Furthermore, the improper mechanism of revealing MAC keys weakens the secrecy of encryption keys. Since the MAC keys are generated as a one-way hash over the encryption key, the attacker can easily use this knowledge to mount a "dictionary attack", although it is probably computationally too expensive. And, the choice of using stream cipher may also cause troubles, especially, when one is trying to manage the encryption counters to avoid re-use of counter values.

Consequently, they suggested three alternate Authentication Key Exchange (AKE) algorithms, SIGMA [18], SKEME [19] (i.e. which is an early voice of a protocol designed to provide deniability to IPsec's IKE protocol.) and HMQV [20], and discussed both the advantage and disadvantage of using these three protocols.

This discussion leads the OTR developers reconsider their design, which resulted in a second version of the OTR protocol, where:

1) They fixed the identity-binding flaw (the impersonate attack vulnerability) simply by adding an additional identification message at the beginning of the conversation session.
2) No longer revealing the users' public keys to passive eavesdroppers and this helps in privacy-preserving for the internal application's OTR messages.
3) And, additionally, made a support of fragmentation OTR

messages, since a lot of Instant Messaging protocols have limitation on each message's size.

C. Lack of Chat Room Support

Chat room systems are often being utilized by companies to increase the success of business for its efficiency. A chat room system is more suitable for online discussions than conventional mailing-list systems due to its interactive nature. Many small businesses use chat room systems (and/or IM systems) for daily business discussions, customer service, etc. Using such technologies cuts down the operation costs of a business and enables employees to multitask when necessary. Many open source projects also use chat rooms to conduct development meetings, since most project members are located in disparate locations. And normally, these projects are relied on donations, which limit their expense on phone meetings. There is, however, no privacy protection in most modern chat room systems. Security concerns associated with chat rooms restrict their use for many businesses.

Some IM systems have chat room support built into them. For example, MSN and Yahoo IM protocols support the chat room concept and once a user invites another user to an ad-hoc chat room, they can invite other parties to have a meeting in that established chat room. Considering security related issues (associated with both IM systems and chat rooms) mentioned previously, it would be beneficial to extend the OTR protocol to support a secure chat room facility.

A secure chat room that utilizes the existing IM infrastructure would bear virtually no cost on the participants. An IM-based secure chat room will also avoid the need for a VPN or dedicating a local server and the challenges that come with having such systems and their management. Hence, we extend the OTR protocol with a scheme to support multi-party conversations and implemented a GAIM plug-in. Our implementation currently supports secure chat room over the MSN IM protocol.

D. MSN protocol

Our first GOTR implementation will base on the famous MSN protocol, which has a native support of chat rooms. The MSN Messenger [3], developed by Microsoft, was released in July, 1999. It became popular with the wide use of the Windows operating system. The end users' MSN applications are called an "MSN Client"; it connects to the "MSN Server" hosted by Microsoft to acquire information about the user's personal profile, buddy list and so on. Whenever a user modifies his/her profile, the client sends this information to the server and the server notifies other users in your buddy list.

The MSN protocol has a built-in support for chat rooms. There are no major differences between a two-party conversation session (referred to as 'SwitchBoard') and a chat room session other than the number of users in the session. If a user wants to invite another to the chat room, s/he will send an invite command and the invited user will receive an RNG command containing the session id, buddy list of the chat room, etc. Notice that, all messages exchanged in the chat room session are broadcast to every user along with sender's account name, the message body and the timestamp.

IV. Methodology and Implementation

A. Initial Design

The main concept of our implementation is to create a virtual server. The term "virtual" is used in the context of acting as a server in the chat room. The server, which could be any one of the members in the same conversation session, will perform key exchanges with every other participant the same way as if s/he would for a regular peer-to-peer OTR conversation. Therefore, the virtual server is sharing a secret with other chat room members. In another words, every one other than the virtual server itself will establish a private channel, each having its own shared secret with the host. The server is responsible for relaying and routing all the IM messages. This implies that the virtual server needs to process and deliver all the messages from any one member to every one else in the same chat room session, as shown in Figure 1. The idea is simple. It is similar to the scenario that happens in the WLAN network for DHCP system. When a computer joins an anonymous wireless network, it needs to make an IP address reservation first. The client computer will broadcast its request and the server will response it with a specific empty IP address slot. And therefore, they made a connection. However, there is a major difference in our GOTR protocol. The server instead of the clients broadcasts its request (i.e. request for a key exchanging) meanwhile it informs every member that it is the dedicated virtual server.

For example, in a GOTR chat room, we have three joiners: Alice, Bob and Carol, and we suppose Alice to be the virtual server. Later on, after all the key-exchange processes conclude, we should have:

- Bob and Alice have a shared secret $SS_{Alice-Bob}$.
- Carol and Alice have a shared secret $SS_{Alice-Carol}$.

There comes to a

Problem: *How can Bob communicate with Carol?*

Bob cannot send his GOTR encrypted messages directly to Carol since the two do not have a common secret, and even if he could, Carol would not be able to decrypt his messages and read them. It is true that they could start their own OTR session and talk to one another without Alice knowing (Alice would not be able to see their messages). But either way will defeat the purpose of a chat room. (i.e. every one in the same chat room should have the same screen of conversations.) However, both Bob and Carol, each have a shared secret with the Virtual Server, Alice. Therefore Bob can send the OTR messages to Alice first and then Alice decrypts Bob's messages by using $SS_{Alice-Bob}$, re-encrypts them with $SS_{Alice-Carol}$ and gets done with sending them to Carol. Now Carol has no problem to decipher the messages. All of our GOTR implementation is based on this straightforward idea.

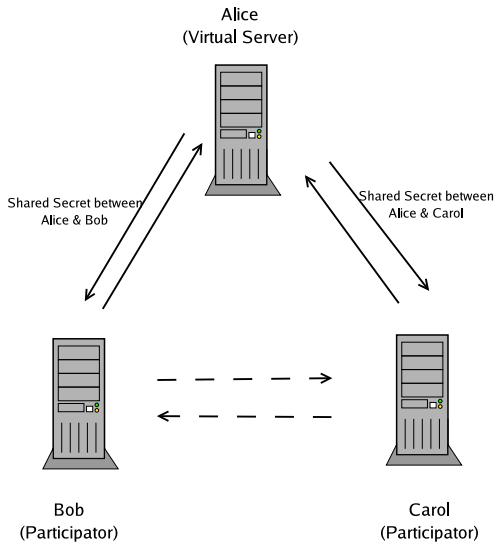

Fig. 1. OTR chat room scheme: a virtual server in the middle

B. Design and Implementation Details

Design Problem 1:

The MSN IM protocol specifies that the messages sent out in a chat room are broadcast to every user in that IM session. Therefore, in proposed design, it is hard to tell the real receiver of the message. When a message goes through and is relayed by the virtual server, the MSN protocol is not really helpful to tell the end receiver where the message indeed came from. It could be a word said by the server itself or any other user in the chat room, since in either case the message has to be passed over by the server. In our virtual server scenario, a user only has the capability to decrypt the instant messages coming from the server and all other messages are discarded. Since only the senders explicitly know the messages are encrypted with which shared key and for whom, we need an identifier at the beginning of a message to determine the receiver of the message.

Now, we would like to show some more examples of our idea. Assume we have an OTR chat room via MSN IM system with three users: Alice, Bob, and Carol. Again, suppose that Alice is the virtual server for this secure chat session. And thereupon, after the key exchange stage, as we said in the previous example:

- Bob and Alice have a shared secret $SS_{Alice-Bob}$.
- Carol and Alice have a shared secret $SS_{Alice-Carol}$.

Example 1:

Alice, the virtual server, sends a message (no matter what kind of messages, including OTR system messages, which are used for key management) to Bob,which should be encrypted with $SS_{Alice-Bob}$ and formatted into the following manner:

```
Alice->Bob:
?RECV?Bob@hotmail.com?ENDRECV?
 + <Encrypted Message>
```

When Carol receives this message, she will check the receiver tag (i.e. prefix of the encrypted messages) first and find out that the message is not hers (i.e. in terms of she can not decrypt it and read it) according to the account name in between $?RECV?$ and $?ENDRECV?$ markers, she just discards this message. When Bob receives the same message, and after he checks the tag and finds out that the message does belong to him, he will send this message to the OTR message encryption and decryption routine (using the OTR library) and use the secret shared between him and Alice ($SS_{Alice-Bob}$) successfully decode the message.

Example 2:

Bob says something in the chat room, but Carol could not read it, since the two do not have a common shared key. Hence, Bob has to send his message first to the virtual server, Alice, with the receiver tag [$Alice@hotmail.com$]. And the message will be something like this:

```
?RECV?Alice@hotmail.com?ENDRECV?
 + <Encrypted Message from Bob>
```

When Alice receives the message, she will decrypt it with the key she shared with Bob ($SS_{Alice-Bob}$), write the message to her screen, then encrypt it again with $SS_{Alice-Carol}$ and set Carol to be the receiver as:

```
?RECV?Carol@hotmail.com?ENDRECV?
 + <Encrypted Message from Alice>
```

Now, on the Carol's side, she will receive both messages encrypted with different keys, one from Bob and another one from Alice. She will simply dismiss the first, since she does not know the right key; but process the second one to the decryption routine and retrieve the decoded content of the message.

It seems to be perfect, but there remains another issue:

Design Problem 2:

Since the server is basically a router which is responsible for reformatting and transferring all the messages. It is hardly possible to know the real sender without any additional effort. If the previous example is revisited: when Carol gets the message from Alice, the virtual server, although she could decipher the message, she wouldn't know the real sender, which could be either Alice or Bob. This is because of that the messages do not contain any source information. Conceivably, Carol will assume the message was started by Alice, since it is Alice that Carol received the message from. However it was originated by Bob. There is no such support mechanism for message tracking" or "transitive authentication" in either the MSN protocol or GAIM project. The solution proposed is to add another tag after the receiver tag to indicate the real sender like what we did to classify the receiver. In accordance with the previous example IV-B, now all the messages will appear to be the following format:

Part One: Message from Bob to Alice

```
Bob->Alice:
?RECV?Alice@hotmail.com?ENDRECV?
+ ?SEND?Bob@hotmail.com?ENDSEND?
```

```
+ <Encrypted Message>
```

Part Two: Retransfered message from Alice to Carol

```
Alice->Carol:
?RECV?Carol@hotmail.com?ENDRECV?
+ ?SEND?Bob@hotmail.com?ENDSEND?
+ <Encrypted Message>
```

Till to this point, the messages include all the necessary tags to identify both the real sender and the receiver.

Design Problem 3:

In the MSN IM protocol, every one in a chat room has the same privilege, which means there is no special power for one to be the "owner" of the chat room established via the MSN IM server. Every one in the chat room can invite another buddy without restriction. Such scenario causes the GOTR protocol a serious problem. For example, assume a new user has been invited to the GOTR session, in order to keep the chat room in protection, the new user should first do a key-exchange with the virtual server and build up the private connection like every one else. But neither the MSN protocol nor the GAIM implementation supports the ability to tell the "owner" of a chat room, which would be our virtual server. Since we would like to offer participants a degree of security via OTR, there is a work around to the aforementioned problem in the following way: Each member of the chat room keeps some additional information and they are recorded in a file named $otr.chatinfo$ located in the $.gaim$ folder, which is used by GAIM to keep configuration files. This file is designed to have the following format:

```
?AC?[account name] ?CID?[chat_id]
?HOST?[host name] ?STAT?[security level]
```

- **AC**: indicate the account name of the user who owns the current conversation window.
- **CID**: $chat_id$ is used by GAIM to identify different chat rooms.
- **HOST**: the user who made to be the virtual server for the MSN chat session.
- **STAT**: indicate the security level used by OTR library; the security level can be:
  - 0 indicates no private conversation.
  - 1 indicates private session over.
  - 2 indicates private session.

Implementation Assumptions:

It is assumed that, the user who initiates the private conversation, would be the virtual server. For example, Alice starts a private conversation, and therefore, she is the virtual server. So her $otr.chatinfo$ would look like:

```
?AC?Alice@hotmail.com ?CID?1
?HOST?Alice@hotmail.com ?STAT?2
```

For all other users, when a private conversation is requested by the virtual server (e.g. Alice). The invited user will write the following information to his/her $otr.chatinfo$ file:

```
?AC?Bob@hotmail.com ?CID?2
?HOST?Alice@hotmail.com ?STAT?0
```

Notice that the conversation's security status is initiated to be 0 (not private). After they finish the first key exchange round and establish the private communication channel, the security status will be changed to 2 (private level) accordingly:

```
?AC?Bob@hotmail.com ?CID?2
?HOST?Alice@hotmail.com ?STAT?2
```

Now, we are confident to say that Alice, Bob and Carol are talking privately under our GOTR system.

V. Conclusion

There is a need for secure chat room environments via the existing IM infrastructure, and therefore an approach to extend OTR to provide secure chat room support via IM is provided. The proposed approach is useful in addressing the needs of individuals (e.g. small businesses where confidentiality of information is crucial) to have off-the-record and secure meetings with virtually has no cost. As proof of concept, a plug-in for GAIM was developed. Although the current implementation only supports chat rooms via the MSN IM network, the idea can be easily extended to other IM protocols such as Yahoo IM, AOL IM, etc.

Some additional network traffic might be an overhead introduced by the proposed approach, it is considered to be preferable rather than dealing with the complicated issue of group key-exchanging protocols particularly in the Deffie-Hellman key agreement protocol. The performance of group key management algorithm remains an issue. Although having some light and small extra packet payloads, will not hinder the performance as much as a complex group key management protocol would. Actually, since the payloads and the network bandwidth traffics are distributed to every chat member, the performance is not an issue for those participants except for the virtual server. But, the result of the initial test shows that the performance for the virtual server is still ideal.

VI. Future Work

The issue of how to deal with a new user (no matter who invited him/her, could or could not be the virtual server) remains. Ideally, the dedicated virtual server should be responsible to respond the change of the secure status due to a new joined member. In other words, when a new user, say Eve, joins the private chat room and breaks the security (i.e. since Eve does not have any shared secrets with any members in that chat room), the virtual server, Alice, should react accordingly. Alice could restart the whole key-exchange process pair-wisely, which is the same process as a fully key refreshment, but including the new member, Eve, this time. And after the reestablishment of the keys, it becomes a normal OTR chat room session. Or, as mentioned ***Design Problem 3***, the server could individually do the key exchange with the new user, which will fitly keep the privacy of the whole chat room. Now, the solution to this problem is the virtual server needs react manually. But in our following research, the next version

of GOTR chat room, this problem will be addressed in terms of reestablish the private conversation session automatically. Basically, we have two choices: one, as we said, is to do a full key-refreshment together with the new member; and another one is only do a peer-to-peer OTR key-exchange between the virtual server and the new user. Either way could solve this problem, but we prefer the first scenario, since it will help to enhance the security by the extra full key refreshment.

Our design brings up an additional security issue and it is remaining to be solved. There is always a possibility that the virtual server gets attacked. It has all the ability, not only to read but also to modify the messages, since it is responsible for delivering them. Hence, if the virtual server turns into a malicious node, it could easily read messages from a user, modify it, and then send it to the other users. We think the integrity of the messages can be assured by maintaining MD5 (any one-way hash table should be applicable) values of the original plain-text messages on each user's computer and verifying them periodically. In order to keep it simple, let us continue with the previous examples:

Carol receives a message which is under Bob's name, but how could she verify that and the message has not been changed by Alice? So when Bob says something, he could attach an additional message digest (i.e. hash value generated by MD5 algorithm), a unique identifier (i.e. timestamp) along with the original message. And as we discussed, in MSN IM protocol, messages are broadcast in a chat room session, so that Alice has no control to prevent Carol from receiving the authentic message digest (i.e. Carol would not be able to decrypt the message part, but this does not hinder her from getting the message's MD5 hash value. She could simply discard the unreadable message, but keep the hash value). Thus, Carol could use this knowledge to verify the messages' integrity, when she receives the re-encrypted copy from Alice. In detail, the only thing she needs to do is: decrypts the message from Alice, calculates the MD5 value over the plain-text message and compares this value with the one she gained from Bob. If they are same, which means, Alice has not changed the content of what Bob said; otherwise, Alice has been cheating. This approach will cause further traffic. However, an automated way of checking integrity of messages would be more beneficial than drawback.

REFERENCES

[1] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use pgp," in *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. New York, NY, USA: ACM Press, 2004, pp. 77–84.

[2] G. Project. (2006, Oct.) What is gaim? [Online]. Available: http://gaim.sourceforge.net/about.php

[3] M. Corp. (2006) Windows live messenger. [Online]. Available: http://get.live.com/messenger/features

[4] A. Online. (2006) American online, aim. [Online]. Available: http://aimexpress.aol.com/

[5] Google. (2006) Chat history saving. [Online]. Available: http://www.google.com/talk/chathistory.html

[6] D. Caraballo and J. Lo. (2006, 06) The irc prelude. [Online]. Available: http://www.irchelp.org/irchelp/new2irc.html

[7] T. I. S. N. W. Group. (1999, 06) Rfc2631:diffie-hellman key agreement method. [Online]. Available: http://www.ietf.org/rfc/rfc2631.txt

[8] M. D. Raimondo, R. Gennaro, and H. Krawczyk, "Secure off-the-record messaging," in *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. New York, NY, USA: ACM Press, 2005, pp. 81–89.

[9] *Information Processing Systems - Open Systems Interconnection Reference Model - Security Architecture*, International Organization for Standardization Std. 7498-2, 1988.

[10] Secway. (2006) Simppro: Instant messengers, instant security. [Online]. Available: http://www.secway.fr/us/products/simppro/

[11] ——. (2006) Secway products: compare table. [Online]. Available: http://www.secway.fr/us/products/compare.php

[12] G.-E. Project. Gaim-encryption. [Online]. Available: http://gaim-encryption.sourceforge.net/

[13] R. L. Rivest, "The rc5 encryption algorithm," in *In the Proceedings of the Second International Workshop on Fast Software Encryption*. FSE, 1994, pp. 86–96.

[14] G. e Project. (2002, 06) Gaim-e, encryption plug-in for gaim. [Online]. Available: http://gaim-e.sourceforge.net/

[15] C. Studios. Cerulean studios: Learn about trillian. [Online]. Available: http://www.ceruleanstudios.com/learn/

[16] D. P. Jablon, "Strong password-only authenticated key exchange," *Computer Communication Review*, vol. 26, no. 5, pp. 5–26, 1996. [Online]. Available: citeseer.ist.psu.edu/jablon96strong.html

[17] W. Diffie, P. C. V. Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Des. Codes Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[18] H. Krawczyk, "Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols." in *CRYPTO*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 400–425.

[19] ——, "Skeme: a versatile secure key exchange mechanism for internet," *sndss*, vol. 00, p. 114, 1996.

[20] ——, "Hmqv: A high-performance secure diffie-hellman protocol." in *CRYPTO*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 546–566.