Off-the-Record Instant Messaging for Group Conversation

Jiang Bian, Remzi Seker, Umit Topaloglu
Department of Computer Science, University of Arkansas at Little Rock, Arkansas, U.S.A.
{jbian, rxseker}@ualr.edu, 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. Although some protection could be achieved with the use of a secure tunneling (i.e. VPN etc.), they do not provide end-to-end secrecy. Off-the-record (OTR) is a protocol which enables 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. There is a need for such a product that provides users an opportunity to meet in an 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, 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.

1. 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 room 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) [2], American Online Instant Messaging (AIM) [10], 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 for the secrecy of 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.

The message packages in 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' contents. 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. Encryption can be utilized for protecting the content of IM messages. 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.

Nevertheless, utilizing only confidentiality and authentication are not good enough to provide an off-the-record environment which includes deniability property. In 2004's Workshop on Privacy in the Electronic Society (WPES), Borisov, N., et. al. proposed the OTR protocol [1] which aims to provide deniability. However, it had several security flaws as pointed out by Raimondo, et al [11], which have been fixed in a later version. The OTR protocol has two distinguishable security properties: perfect forward secrecy and deniability. These features will be discussed in Section 2.

The OTR protocol provides a perfect secure conversation environment for two-party. The protection of communications is critical, especially for private companies, while unveiling the company secrets may harm their competitive ad-

vantages and result in capital loss. Therefore, our intention is to address this need by utilizing the original two-party OTR protocol to create a guarded and assured multi-user IM ambiance.

Because of the differential characteristics between the two-party conversation environments and the multi-user chatting systems, we are facing a set of unique challenges in designing a security schema. The increased size of conversation members increases the complexity of key exchanging processes and requires more computational resources for both encryption and decryption routines. Likewise, the synchronization of the shared keys is another hurdle. Similar to the clock problem in distributed operating systems (i.e., a universal clock system is unobtainable), it is hardly possible to generate and maintain an all-inclusive key among multiple IM users. We will revisit these problems as well as implementation details in Section 3. Conclusion and future work are given in Section 4 and 5, respectively.

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

Security in distributed applications is usually supported by five typical security services defined by the International Organization for Standardization (ISO). Those are access control/authorization, identification/authentication, confidentiality, integrity, and non-repudiation [4]. However, for an off-the-record 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. In other words, a user should be able to deny what s/he has said in a past conversation. Most of the secure IM products address only part of the required security services. Often, message integrity, forward secrecy, and deniability are missing. However, the OTR protocol addresses these widely omitted concerns within the IM domain.

2.1. Basic concepts behind the OTR

The OTR protocol contains four basic cryptographic primitives:

Perfect forward secrecy: [6] Confidentiality is introduced by using short-lived encryption/decryption key(s). The basic idea is that both parties should forget the used keys after they process the old messages[1]. It is computationally infeasible to guess the used keys from the current key or 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 is useless to decrypt and read previous messages, since each key is used to secure exactly one message.

Digital signatures and non-repudiation: Digital signatures are used as long-term keys to address the lack of authentication mechanism in the conventional Diffie-Hellman protocol. However, directly attaching the signature to every IM message leads to another problem. It enforces the non-repudiation that a signature can be verified by a third-party without the cooperation of the owners, which conflicts with the deniability property of the OTR protocol. The solution is to use a Message Authentication Code (MAC) on each message instead of user's digital signature. Therefore, the digital signatures authenticate the keys only rather than the entire messages. The authenticated keys can still provide the identification service, because only the person who has the right key can read the cipher texts.

Message Authentication Code (MAC) and deniability: Deniability is the ability to deny the content of conversations and it is addressed in the OTR by the use of MAC codes. A MAC is basically a code generated by a one-way cryptographic hash function with a secret MAC key shared by conversation members. Alice uses the shared key compute a MAC of her message, and sends it along with her message over a secure transmission channel; Bob verifies the integrity and authenticity of the message by computing the MAC on the received message by using the same agreed MAC key and comparing the MAC he computed with the MAC sent by Alice [1]. 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 private MAC key. And, even Bob cannot make a proof to the public that the message is really from Alice. Both of them know the same MAC key, and so it could be a message forged by Bob himself.

Malleable encryption and forgeability: Forgeability is a little stronger than repudiation. Once a key expires, the associated MAC key is revealed. This mechanism enhances the ability of Alice to deny the content of the conversation, because the message could have been forged by anyone, since everyone knows the MAC key. Moreover, the 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 in the plaintext). Technically, the malleable encryption is not a "deniable encryption" algorithm, but its deniability refers to the inability of an third-party to prove the authenticity of a conversation.

2.2. Security Weakness in OTR

Mario Di Raimondo, et al. [11] pointed out three major security flaws after they examined the OTR protocol:

1. An authentication failure

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

2. A key refreshment flaw, and

3. Unreliable support of the deniability

First of all, the OTR protocol inherits a possible "identity misbinding" attack originally discovered by Diffie et al. [3]. Suppose, an attacker, Eve, stands between two communicators, Bob and Alice, and if she attacks properly, Eve could make Alice think she is talking to Bob but actually she is talking to Eve. For a real life example, Eve, can use this authentication flaw to mislead a customer, Alice, and a bank, Bob. A simple solution is to include identity information in the digital signature, which will invalidate the deniability property.

Second, the revealing of an ephemeral private key could cause an impersonation attack. A talented attacker could use this information to produce a valid session key as long as the long-lived key is not revoked. This possibility defeats the goal of a well-designed key protocol, where the only way for an attacker to impersonate into a conversation is the disclosure of the long-term private key rather than a piece of information used in a short session. Therefore, Mario Di Raimondo, et al. [11] 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 use this knowledge to mount a "dictionary attack", although such an attack is computationally too expensive. Moreover, the choice of using stream cipher may also cause troubles, especially, when one is trying to manage the encryption counters to avoid reuse of counter values.

Consequently, Mario Di Raimondo, et al. [11] suggested three alternate Authentication Key Exchange (AKE) algorithms, SIGMA [8], SKEME [7], and HMQV [9].

The critic of OTR by Mario Di Raimondo, et al. [11] resulted in the second version of the OTR protocol, where,

1. The OTR team fixed the identity-misbinding flaw by adding an additional identification message at the beginning of the conversation session.

2. No longer revealing of the users' public keys to passive eavesdroppers.

3. Additionally, support for fragmenting OTR messages was implemented, since most of public IM protocols have a limitation on message size.

2.3. Lack of Chat Room Support

Chat room systems are often utilized to increase business efficiency. Many small businesses use chat room (and/or IM) systems for business meetings, some even for customer service, etc. Utilizing such technologies cuts down the operation costs and enables employees to multi-task when necessary. Also, many Open-Source Software (OSS) project groups use IMs and chat rooms to conduct developer meetings. Most OSS developers are volunteers around the world and most of such projects rely on donations and so there is virtually no funded physical meeting opportunity. Despite all the advantages they have, there is no privacy protection built-in most contemporary chat room systems. It would be a great benefit to extend the OTR protocol to support a secure chat room facility.

A secure chat room via the existing IM infrastructures would bear virtually no cost on the participants. An IM-based secure chat room will also avoid the need for a VPN or a dedicating local server and the challenges that come with having such systems and their management. Hence, we extended the two-party OTR protocol to support multi-party conversations, named GOTR, and we implemented a GAIM plug-in for the proof of our concept. The implementation currently provides a secure chat room system over the MSN protocol.

3. Methodology and Implementation

3.1. Initial Design

The main concept of our implementation is to create a virtual server, which is a chat member literally acting as a server. The server, which could be any one of the participants, will perform key exchanges with every others the same way as if s/he would for a regular peer-to-peer OTR conversation. Therefore, the virtual server shares a secret with each member, respectively. In another words, every one other than the virtual server itself establishes 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.

For example, assume we have three members in a GOTR chat room: Alice, Bob and Carol; and Alice is the virtual server (Figure 1). After 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 $SS_{Alice-Carol}$.

Problem: *Bob and Carol have no shared secret. How can they communicate?*

Bob cannot send his GOTR encrypted messages directly to Carol, and even if he could, Carol would not be able to

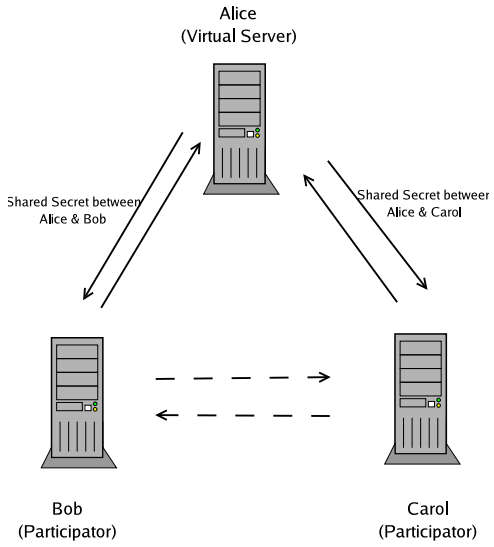

Figure 1. A GOTR chat room example where one of the users acts as the virtual server

read them since the two do not have a common secret. It is true that they could start their own OTR session and talk to one another without Alice knowing. 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 relays the messages to Carol. Now Carol has no problem to decipher Bob's messages.

3.2. Design and Implementation Details

Design Problem 1:

It is important for us to know the real receiver of a message, because there is no point to waste computing power on processing a message encrypted with an unknown key, which, when decrypted, will produce a meaningless message. However, when a message is relayed by our virtual server, the MSN protocol is not really helpful to identify the end receiver of each message. In our design, a user only has the capability to decrypt the IMs that have been encoded with the secret key s/he shares with the virtual server, which is a subset of messages arrive from the virtual server. All other messages should be discarded. Since only the senders explicitly know the messages are encrypted with which key and for whom, we need the sender to attach an identifier of the receiver at the beginning of each message.

Let us revisit the previous example where we had a GOTR chat room including three users. Again, suppose that Alice is the virtual server. After the key exchange stage concludes, Bob and Alice share a secret $SS_{Alice-Bob}$ while Carol and Alice have a shared secret, $SS_{Alice-Carol}$.

Example 1:

Alice, the virtual server, sends a message to Bob,which is encrypted with $SS_{Alice-Bob}$ and formatted in the following manner:

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

When Carol receives this message, she checks the receiver tag (i.e. prefix of the encrypted messages) first and find out that the message is not hers (i.e. she can not decrypt it and read it), she simply discards this message. Meanwhile, Bob receives an exactly same message and notices that it is a message for him, he will route this message to the OTR encryption/decryption routine (using the OTR library) and decode that encrypted message using the secret shared between him and Alice ($SS_{Alice-Bob}$).

Example 2:

Bob says something in the chat room, but Carol could not read it, since the two do not have a shared key. Hence, Bob has to send his message first to the virtual server, Alice. The message may look like:

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

When Alice receives the message, she decrypts it with the key she shared with Bob ($SS_{Alice-Bob}$), writes the message to her screen, then encrypts it again with $SS_{Alice-Carol}$ and sends to Carol as:

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

Now, on 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 one, because she does not know the right key(i.e. the first message is encrypted with a key only known between Alice and Bob); but process the second one and display the decoded message.

There remains another issue to be addressed:

Design Problem 2:

The virtual 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 would not know who said that, which could be either Alice or Bob. This is because 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 originated from Bob. There is no such support mechanism for message tracking" or "transitive authentication" in either the MSN protocol or GAIM project. The proposed solution 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, now all the messages will appear to be in 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>
```

At 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 protected, 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**: indicates the owner of the current account.
- **CID**: $chat\_id$ is used by GAIM to identify different chat rooms.
- **HOST**: the user assigned to be the virtual server.
- **STAT**: indicates the security level used by OTR library; the security level can be:
  - 0 indicates no private conversation.
  - 1 indicates the private session is over.
  - 2 indicates this session is private.

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 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 initially 0 (not private). After the users finish the first key exchange 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.

3.3. Evaluation of the GOTR protocol

There is always performance concerns when a security solution is implemented. One question would be how many users a GOTR chat room session could hold. Obviously, having more users in a chatroom means more network traffic and longer processing time which eventually will cause noticeable lag. In addition, a GOTR chatroom session uses a virtual server which routes all the messages, which increases the number of network packets transmitted during a conversation. We have not performed any sophisticated test such as comparing transit time of each message, counting how many extra packets are caused by a GOTR chatroom etc., but we performed several user experience tests. We have tested our GAIM plug-in with up to ten users in a single chat room and there was no noticeable delay. Theoretically, the limitation of the number of a chat room session, is the maximum number of users which a MSN server can handle in a chat room session. We will perform more detailed tests in our next version of GOTR.

4. Conclusion

There is a need for secure chat room environments that utilize the existing IM infrastructure. We provided an approach to extend OTR to provide secure chat room support via IM. 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 at virtually no cost. As a 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 as an overhead introduced by the proposed approach is considered to be preferable to dealing with the complicated issue of group key-exchanging protocols. Therefore, having some extra network packets (IM packets are relatively small in size) will not hinder the performance as much as a complex group key management protocol would. Actually, the performance is not an issue for the participants except for the one acting as the virtual server. But, as we mentioned, the result of the initial test shows that the virtual server can handle a room of up to ten users with no problem.

5. Future Work

There are three issues we will address in the next versions of the GOTR project. The first issue is dealing with a new user joining an already existing chatroom. The second issue is failure of the virtual server and the third issue is knowing whether the virtual server has altered contents of messages it has been forwarding.

The ideal solution for the first issue would be the dedicated server automatically responding to any changes of the status caused by a new comer. When a new user, say Eve, joins the private chat room and breaks the security (i.e. because Eve doesnot have any shared secret with any other member 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 full key refreshment, but this time including the new member, Eve. Or, as mentioned in ***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. As for the second issue, if there is a failure of the virtual server and it is unresponsive, some one else in that chat room should pick up the responsibility, restart the whole GOTR conversation by initiating a full key refreshment. The participants can be alerted to do so via utilizing a timer. The final issue to be address is to make sure the virtual server doesnot change the content of IMs it forwards. 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.

6. Acknowledgment

This work was funded in part, by grants from the National Science Foundation (CNS- 0619069) and Acxiom Corporation (# 281539).

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*, pages 77–84, New York, NY, USA, 2004. ACM Press.

[2] M. Corp. *Windows Live Messenger*. http://get.live.com/messenger/features, 2006.

[3] W. Diffie, P. C. V. Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 2(2):107–125, 1992.

[4] I. O. for Standardization. *Information Processing SystemsOpen Systems InterconnectionBasic Reference ModelPart 2: Security Architecture*. Number ISO 7498-2. 1988.

[5] Google. *A Google approach to instant communications*. http://www.google.com/talk/, 2007.

[6] D. P. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, 1996.

[7] H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. *sndss*, 00:114, 1996.

[8] H. Krawczyk. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer, 2003.

[9] H. Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005.

[10] A. Online. *American Online, AIM*. http://aimexpress.aol.com/, 2006.

[11] 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*, pages 81–89, New York, NY, USA, 2005. ACM Press.