Version 0.5.0

SMMP: A Secure Multi-Party Messaging Protocol

David R. Andersen* and Mark S. Andersland

Department of Electrical and Computer Engineering,

The University of Iowa, Iowa City, IA 52242

Tycho J. Andersen

Canonical, Inc.

Madison, WI 53703

(Dated: March 13, 2014)

Abstract

We describe a new secure, multi-party, synchronous communication protocol. The protocol follows a peer-to-peer model and provides perfect forward secrecy, perfect future secrecy, and plausible deniability for participants in a multi-party conversation, as long as at least two participants are honest. The protocol uses the Burmester-Desmedt group key agreement protocol to generate a shared secret between a group N of participants. Conversation participants authenticate to each other during key agreement via a triple Diffie-Hellman exchange. Individual message keys are updated after receipt of each message by incorporating new key material distributed by the sending participant. No security requirements are imposed on the underlying transport layer, and our protocol leaks no metadata beyond that exposed by the transport layer. Conversation transcript universality is assured by a conversation digest that is updated upon receipt of each message, and all conversation messages are signed to verify proof of origin. All setup operations prior to group key agreement take place over an insecure channel.

I. INTRODUCTION

Secure multi-party messaging has been an elusive goal of security researchers for a long time. The initial group off-the-record (GOTR) 'virtual server' approach[1] was deemed unsatisfactory because it has the unfortunate drawback of not permitting all participants to confirm that they were receiving unmodified copies of all messages. A further attempt to solve this problem[2] was incomplete because of the lack of a simple, secure key-agreement strategy. Work on the problem continues to date, most notably with[3]. Liu *et.al.*[4] recently improved the security of GOTR/mpOTR using a Burmester-Desmedt group key agreement mechanism imposed on top of a network comprised of binary encrypted channels between participants.

In this paper, we describe a novel secure multi-party messaging protocol with a simple key-agreement algorithm that provides perfect forward secrecy (PFS), perfect future secrecy (PFuS), and plausible deniability (PD) for participants in a group conversation. Our protocol is a peer-to-peer protocol, whereby each participant (peer) contributes new key material to the group key set with each message sent. This key-update mechanism is called a ratchet. In contrast to the various mpOTR/GOTR proposals, no constraints on the security of the underlying transport layer are required.

II. KEY POINTS

- Burmester-Desmedt[5] group key agreement takes place using finite-field, cyclic-group Diffie-Hellman key agreement as the underlying protocol.

- During the conversation, elliptic curve Diffie-Hellman key operations are performed, minimizing the required key-exchange bandwidth.

- The protocol is synchronous. A method for resynchronizing those participants that lose synchronization due to lost packets from collisions or transport failure is included in the protocol.

- Group setup is via an insecure channel.

- No security constraints are imposed on the underlying transport layer.

- Participants may be added to the group via a reinitialization of the group secret. A participant may leave the group by simply stopping participation and not updating his ratchet state.

- A uniform message transcript is assured over all participants via a conversation digest that is updated with each message. If the digest does not match what a receiver expects, a resynchronization of the protocol is requested.

- The underlying symmetric encryption algorithm used for communication is unspecified.

III. BACKGROUND

Various protocols exist for facilitating secure one-on-one messaging, including Off the Record Messaging (OTR)[8–12], Silent Circle Instant Messaging Protocol (SCIMP)[13], and Axolotl[14]. Each of these protocols provides security through the use of ephemeral keys for symmetric encryption. The method of generating and exchanging these keys varies widely between the three protocols. All of the protocols provide forward secrecy. Some also provide plausible deniability. And one (Axolotl) also provides perfect future secrecy (compromise of the current keyset does not cause compromise of future keysets). However, each of these protocols is limited to the one-on-one messaging case.

OTR was the first such protocol. It is based on an Advertise → Acknowledge → Use method for key updating. OTR has been widely used for secure instant messaging and applications containing OTR plugins are available for XMPP, IRC, and other messaging systems.

SCIMP is a proprietary protocol developed by Silent Circle as their solution for providing secure communications as part of their product line. The key advancement algorithm for SCIMP is essentially a hash. Each symmetric encryption key is hashed to obtain the key for the next ratchet step. As a result, a key compromise at any point will permit an attacker to follow the conversation from that point on.

Axolotl is the first one-on-one messaging protocol to provide future secrecy. In this protocol, randomly-generated key data is mixed in to the ratchet state with each message send/receive operation, so that a compromised keyset permits an attacker access to only one message. Following the conversation requires a new key compromise with each message sent.

The first proposal for a secure group ($N \geq 3$) messaging protocol was that of Bian *et.al.*[1] [Group Off The Record (GOTR)]. The protocol is based on a virtual server, a participant that sets up a secure one-on-on channel with each member of the group. When members of the group who are not the virtual server wish to communicate with each other, they relay their messages through the virtual server. The problem with this approach is that there is no way for users to verify that they are getting a complete transcript of the group conversation. Trust in the virtual server is required, and a compromise of the virtual server would permit an attacker to follow the entire conversation, as well as learn the membership of the group.

A second proposal for a secure group messaging protocol was made by Goldberg *et.al.*[2] [multi-party Off The Record (mpOTR)]. While this protocol solved some of the problems associate with the Bian messaging scheme, it did not give a complete key-agreement algorithm. As a result, this protocol has never been successfully implemented.

An improvement of the security of the mpOTR/GOTR protocol was made by Liu *et.al*[4] using Burmester-Desmedt[5] for group key agreement. The improved mpOTR/GOTR is imposed on an underlying network of secure, binary communication channels that exist between participants. The improved mpOTR/GOTR protocol requires an underlying network of binary encrypted channels between adjacent users in the network, six exchanges to complete group key agreement, and does not provide either PFS or PFuS. The Liu group demonstrated their protocol with a Pidgin[6] plugin.

In this paper, we present a secure, peer-to-peer multi-party messaging protocol (SMMP) that provides PFS, PFuS, and PD, and has a simple key agreement algorithm requiring three exchanges to complete key agreement (including authentication between all peers). The protocol allows all group members to confirm that they are receiving a complete transcript of the group conversation, and provides a robust mechanism for resynchronization of the ratchet state if messages are lost because of failure of the underlying transport mechanism.

IV. NOTATION

The following notation is used:

- N is the total number of participants in the group (including the Organizer)

- $\oplus$ is the bitwise XOR operator

- $\{N_i\}$ is the set of all N_i and $\{N_i\}_m = N_m$.

- $\oplus_{j \neq i} x_j$ means bitwise XOR of all x_j except x_i.

- $\oplus_i x_i$ means bitwise XOR of all x_i.

- $\hat{\mathcal{A}}_i$ is the ratchet state for participant P_i.

- $||$ is the concatenation operator

- $||_i x_i$ means concatenation of all x_i running from the smallest index i to the largest

- Lower case keys are private ECDH keys.

- Upper case keys are public ECDH keys.

- hash() is a secure, one-way cryptographic hash function.

- hmac(k, m) is a secure hashing message authentication function that hashes a message m using key k.

- KDF() is a secure key derivation function, *e.g.* pbkdf2.

- $c = e(k, m)$ and $m = d(k, c)$ are the underlying (unspecified) symmetric encryption and decryption algorithms that use a key k to transform a plaintext m into ciphertext c and *vice versa*.

- MK is a master key from which the initial ratchet state $\hat{\mathcal{A}}_i$ is computed by each participant P_i. This parameter is securely erased upon computing the initial ratchet state.

- X is the generator for the elliptic curve and ECDH() is the Diffie-Hellman operator on the elliptic curve, *i.e.* for private key u, the corresponding public key is $U = \mathrm{ECDH}(u, X)$. The shared secret between y and z is then $\mathrm{ECDH}(y, Z) = \mathrm{ECDH}(z, Y) = \mathrm{ECDH}(y, \mathrm{ECDH}(z, X))$.

V. PROTOCOL STATE

Each Participant P_j will maintain the following state variables in persistent storage:

- RK - the root key
- HK - the header key
- mk - the message key
- v - the group private ratchet key
- conv_digest - the digest of all conversation messages so far
- r_j - the participant private ratchet key (for participant P_j, $\mathrm{ECDH}(r_j, X) = R_j$)
- r_j^{init} - the initial value of participant P_j's private ratchet key
- $\{R_i\}$ - the set of public ratchet keys from each participant P_i
- $\{R_i^{init}\}$ - the set of initial values of the public ratchet keys from each participant P_i
- j - Participant P_j's participant index number
- N - the group size
- group_name - the group name (this may be different for each participant)
- resync_required - a flag used to determine if a resynchronization of the ratchet state is required

VI. KEY AGREEMENT

When a group of participants desires to establish a secure group conversation channel, the following preliminary steps are completed (each participant will be known by a long-term private identity key b_i and the corresponding public identity key B_i - proof of knowledge of the private key constitutes proof of identity):

1. The total number of participants in the group N is determined and this number is distributed to all participants.

2. A unique participant index i is assigned to each participant. The method of assigning this index is application-dependent, and may *e.g.* be assigned based on the ordering of the participant public identity keys.

Each each group participant P_j then completes the following steps:

1. Participant P_j generates an ephemeral ECDH ratchet key pair (r_j, R_j), and an ephemeral cyclic-group DH handshake key pair (h_j, H_j).

2. Participant P_j broadcasts B_j and R_j.

3. Participant P_j computes the set of keys:
   $k_{TDHi} = \mathrm{hash}(\mathrm{ECDH}(b_j, R_i) \,||\, \mathrm{ECDH}(r_j, B_i) \,||\, \mathrm{ECDH}(r_j, R_i))$
   and hmacs:
   $\mathrm{mac}_i = \mathrm{hmac}(k_{TDHi}, H_j)$.

4. Participant P_j sends $H_j \,||\, \mathrm{mac}_i$ to participant P_i for all participants.

5. Participant P_j receives $H_i \,||\, \mathrm{mac}_j$ from each participant P_i.

6. For each participant $j \neq i$, participant P_j computes the set of received keys:
   $k_{r-TDHi} = \mathrm{hash}(\mathrm{ECDH}(r_j, B_i) \,||\, \mathrm{ECDH}(b_j, R_i) \,||\, \mathrm{ECDH}(r_j, R_i))$
   and hmacs:
   $\mathrm{mac}_{r-i} = \mathrm{hmac}(k_{r-TDHi}, H_i)$.

7. Participant P_j tests if $\mathrm{mac}_{r-i} = \mathrm{mac}_j$. If this test fails, the identity of participant P_i is not confirmed and P_j goes no further.

8. Participant P_j computes: $K_j = (H_{j+1}/H_{j-1})^{h_j} \mod p$ where the index j is taken in a cycle. P_j broadcasts K_j to all participants P_i.

9. Participant P_j receives K_i from each participant P_i and computes:
   $\mathrm{MK} = \mathrm{hash}(H_{j-1}^{Nh_j} \cdot K_j^{N-1} \cdot K_{j+1}^{N-2} \cdots K_{j-2} \mod p)$

10. Participant P_j computes his/her initial ratchet state $\hat{\mathcal{A}}_j$:
    $\mathrm{RK} = \mathrm{KDF}(\mathrm{MK}, 0\mathrm{x}00)$,
    $\mathrm{mk} = \mathrm{KDF}(\mathrm{MK}, 0\mathrm{x}01)$,
    $v = \mathrm{KDF}(\mathrm{MK}, 0\mathrm{x}04)$,

$\text{conv\_digest} = 0\text{x}00 * 32$,

r_j from step 1,

$r_j^{init} = r_j$,

$\{R_i\}$ from participants,

$\{R_i^{init}\} = \{R_i\}$,

$\text{resync\_required} = \mathit{False}$.

VII. SENDING MESSAGES

Participants wanting to send a message will proceed as follows:

1. When a participant P_j wishes to communicate with the other participants, he forms a message m.

2. Participant P_j generates a new ephemeral ratchet key (r_j^{new}, R_j^{new}).

3. Participant P_j computes a new conversation digest $\text{conv\_digest} = \text{hash}(m) \oplus \text{conv\_digest}$.

4. For all $i \neq j$, participant P_j computes the set of $N-1$ preliminary ciphertexts $c'_i = e(\text{mk}, j \,||\, R_j^{new} \oplus \text{hash}(\text{ECDH}(r_j, R_i)) \oplus \text{conv\_digest} \,||\, m)$.

5. Participant P_j computes the set of $N-1$ hmacs $c_{hmaci} = \text{hmac}(v, c_i)$.

6. Participant P_j forms the set of $N-1$ ciphertexts $c_i = c'_i \,||\, c_{hmaci}$.

7. Participant P_j updates his/her ratchet state $\hat{\mathcal{A}}_j$ as follows:
   $(r_i, R_i) = (r_i^{new}, R_i^{new})$,
   $\text{RK} = \text{hash}(\text{RK} \,||\, \text{ECDH}(v, ||_i\, R_i))$,
   $\text{mk} = \text{KDF}(\text{RK}, 0\text{x}01)$.

8. Participant P_j sends c_i to participant P_i for all participants.

VIII. RECEIVING MESSAGES

Participants receiving a message will proceed as follows:

1. Upon receipt of a ciphertext c, participants P_j separates c into c' and c_{hmac} parts.

2. Participant P_j tests if $\mathrm{hmac}(v, c') = c_{hmac}$. If they do not match, he/she raises a Bad_HMAC exception and goes no further.

3. Participant P_j obtains $q \,||\, R \,||\, m = d(\mathrm{mk}, c')$. If this operations fails, he/she raises a Message_Undecryptable exception, sets the resync_required flag *True*, and passes c' and control to the message-received housekeeping routine described in Section X A.

4. Participant P_j computes a new conversation digest $\mathrm{conv\_digest} = \mathrm{hash}(m) \oplus \mathrm{conv\_digest}$.

5. Participant P_j sets $R_q = R \oplus \mathrm{hash}(\mathrm{ECDH}(r_j, R_q)) \oplus \mathrm{conv\_digest}$.

6. Participant P_j updates his/her ratchet state $\hat{\mathcal{A}}_i$ as follows:
   $\mathrm{RK} = \mathrm{hash}(\mathrm{RK} \,||\, \mathrm{ECDH}(v, ||_i \, R_i))$,
   $\mathrm{mk} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}01)$.

IX. ONE-ON-ONE MESSAGING WITHIN SMMP

Participants in the group may exchange one-on-one messages with other group participants using the group infrastructure. If participant P_i wishes to message participant P_j, he/she proceeds as follows:

1. Participant P_i forms a message m.

2. If this is the first private message with P_j, participant P_i makes a copy of his current private ratchet key r_i (s_i) as well as participant P_j's current public ratchet key R_j (S_j). If this is not the first private message P_i has exchanged with P_j, s_i and S_j already exist.

3. Participant P_i generates a new one-on-one ratchet key (t_i, T_i).

4. Participant P_i forms preliminary ciphertexts $c_h = e(v, S_j), 0\mathrm{x}01 \,||\, T_i)$ and $c_m = e(\mathrm{ECDH}(s_i, S_j), m)$.

5. Participant P_i forms $c' = c_h \,||\, c_m$ and computes $c_{hmac} = \mathrm{hmac}(v, c')$.

6. Participant P_i sends $c = c' \,||\, c_{hmac}$ to P_j.

Decryption is a straightforward reversal of this process.

X. PROTOCOL HOUSEKEEPING

Here we list several protocol housekeeping operations that may be necessary. The list may be extended if other useful operations are identified.

A. Message Received Housekeeping

Housekeeping messages will be routed based on a one-byte header prepended to the payload of the housekeeping message. The byte values and their corresponding housekeeping tasks are (currently there are 2):

Byte Value	Housekeeping Task	Location
0x00	Resynchronizing the Ratchet State	Section X A 1
0x01	Instant Messaging Within SMMP	Section ??

To determine the proper routing, the participant proceeds as follows:

1. Participant P_i computes $b \,||\, m = d(v, c')$. He/she then finds the value corresponding to byte b in the table above, and sends message m and control to that section.

1. *Resynchronizing the Ratchet State:* $b = $ 0x00

It is possible, during communication within the group, that a participant's ratchet state may become unsynchronized. Transport layer failures due to lost messages or message collisions can cause a participant to be unsynchronized. If this is the case, a Message_Undecryptable exception will be raised and control of the decryption will be passed here. The receiving participant will then proceed as follows:

1. Participant P_j decomposes $m = q \,||\, v^{new} \,||\, R_q^{new}$.

2. Participant P_j sets $R_q^{init} = R_q^{new}$ and $\{R_i\} = \{R_i^{init}\}$.

3. Participant P_j sets $v = \text{hash}(v \,||\, v^{new})$.

4. Participant P_j updates his/her ratchet state $\hat{\mathcal{A}}_j$ as follows:
   $\text{RK} = \text{hash}(v \,||\, \text{ECDH}(v, ||_i\, R_i))$,

$\mathrm{mk} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}01),$

$\mathrm{conv\_digest} = 0\mathrm{x}00 * 32,$

$\mathrm{resync\_required} = False.$

2. *Adding Members To the Group*

Members may be added to the group by running the key agreement part of the protocol again, and including the new participant.

3. *Removing Members From the Group*

Members may drop from the group simply by not updating their ratchet state. Further, the group may remove a participant by running the key agreement part of the protocol again without the dropped participant.

B. Message Send Housekeeping

It may be necessary at some point to send housekeeping messages. This section describes procedures to be followed in this case.

1. *Resynchronizing the Ratchet State: resync_required = True*

When the resync_required flag is set to *True*, the ratchet state is unsynchronized, and decrypting further conversation messages is impossible. A participant wishing to correct this situation should follow the following procedures (this may be automated):

1. Participant P_j generates a new group private key v^{new}.

2. Participant P_j generates a new ratchet key pair (r^{new}, R^{new}).

3. participant P_j computes preliminary ciphertext $c' = e(v, 0\mathrm{x}00 \,||\, j \,||\, v^{new} \,||\, R^{new})$.

4. Participant P_j computes hmac $c_{hmac} = \mathrm{hmac}(v, c')$.

5. Participant P_j forms $c = c' \,||\, c_{hmac}$.

6. Participant P_j delays according to a backoff algorithm that should be transport-specific and is unspecified here.

7. Participant P_j tests resync_required to see if it is still *True*. If not, a resynchronization message was received and no further action is necessary. If *True*, participant P_j continues with the next step.

8. Participant P_j broadcasts c to all participants P_i.

9. Participant P_j updates his/her ratchet state $\hat{\mathcal{A}}_j$ as follows:
   $v = \text{hash}(v \,||\, v^{new})$,
   $r_j = r^{new}$,
   $r_j^{init} = r^{new}$,
   $R_j^{init} = R^{new}$,
   $\{R_i\} = \{R_i^{init}\}$,
   $\text{RK} = \text{hash}(v \,||\, \text{ECDH}(v, \oplus_i R_i))$,
   $\text{mk} = \text{KDF}(\text{RK}, \text{0x01})$,
   $\text{conv\_digest} = \text{0x00} * 32$,
   $\text{resync\_required} = False$.

XI. SECURITY AND EFFICIENCY ANALYSIS

A. Security Model

Following the work of Goldberg, *et.al.*[2], we propose a security model with the following attributes:

- Confidentiality - While a participant is willing to disclose certain information to members of a group, the group communication should remain hidden to those outside the group.

- Entity authentication - Prior to joining the group, members should be authenticated so that during the group conversation, group members can be confident that messages purportedly from a particular group member were actually authored by that group member.

- Origin authentication - All messages should be authenticated as to their participant of origin.

- Forward secrecy - The protocol should provide PFS for all messages sent as part of the group communication.

- Future secrecy - The protocol should provide PFuS for all future messages to be sent as part of the group communication.

- Plausible Deniability - The protocol should provide PD such that a transcript of the group conversation cannot be used to prove the membership in the group of any participant during the conversation or after the conversation has completed.

Further, we adopt the same threat model as did Goldberg *et.al*[2]. In particular, we analyze the robustness of the protocol to three types of adversaries, a security adversary, a consensus adversary, and a privacy adversary. We further assume that at least two group participants are honest. Online judges (members of the group) can be considered a subset of the dishonest participants. Both online and offline judges will be used to determine if the goals of the protocol have been met.

B. Network Model

The network model used by SMMP is that of a fully-connected graph. Operation of the network may be simplified somewhat by using an echo server to relay messages to all participants. The requirements for this echo server are not great. It should not have decryption capability, and as a result does not need to maintain its own ratchet state. It merely needs to route messages based on routing information supplied by the sending participant. Providing such routing information will not leak additional metadata to an adversary, because the adversary could also track the message flow along the connected graph, obtaining the same information.

The connected graph network suffers from scalability issues - for large enough groups, maintaining the ratchet state becomes very resource intensive. However, we do not believe this is a significant disadvantage, because person-to-person group conversations suffer from the same scalability problem. It would be unusual to see a group conversation between a number of participants larger than *e.g.* $N = 20$ or so. Larger conversations tend to break into sub-conversations. Extending SMMP to cover multiple sub-conversations with a group will be described in a later paper.

Finally, we note that we have developed a reference implementation[15] of SMMP. The reference implementation uses an echo server to forward messages to users.

---

* k0rx@uiowa.edu; Also at Department of Physics and Astronomy, The University of Iowa.

1 J. Bian, R. Seker, and U. Topaloglu "Off-the-Record Instant Messaging for Group Conversation," *IRI '07: Proceedings of Information Reuse and Integration,* pp. 79-84, IEEE Computer Society, 2007.

2 I. Goldberg, M. D. Van Gundy, B. Ustanoglu, and H. Chen, "Multi-Party Off-the-Record Messaging," *CSS '09: Proceedings of the 16th ACM Conference on Computer and Communication Security,* pp. 358-368, ACM, 2009.

3 Cryptocat Messaging Blog `https://github.com/cryptocat/mpotr`, accessed Feb. 22, 2014.

4 H. Liu, E. Y. Vasserman, and N. Hopper, "Improved Group Off-the-Record Messaging," *WPES'13: Proceedings of the ACM Workshop on Privacy in the Electronic Society,* ACM, 2013.

5 M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," *EUROCRYPT'94: Advances in Cryptology,* volume 950 of Lecture Notes in Computer Science, 1995.

6 "Pidgin," `https://www.pidgin.im`, accessed Feb. 27, 2014.

7 M. Blum, P. Feldman, and S. Micali, "Non-Interactive Zero-Knowledge and Its Applications," *STOC 1988:Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 103112, ACM, 2008.

8 "Off the Record Messaging", `https://otr.cypherpunks.ca/`, accessed Feb. 22, 2014.

9 N. Borisov, I. Goldberg, and E. Brewer, "Off-the-Record Communication, or, Why Not To Use PGP," *WPES'04: Proceedings of the ACM Workshop on Privacy in the Electronic Society,* ACM, 2004.

10 M. D.Raimondo, R. Gennaro, and H. Krawczyk, "Secure Off-the-Record Messaging," *WPES'05 Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society,* pp. 81-89, ACM,

2005.

[11] C. Alexander and I. Goldberg, "Improved User Authentication in Off-The-Record Messaging," *WPES'07: Proceeedings of the ACM Workshop on Privacy in the Electronic Society,* ACM, 2007.

[12] R. Stedman, K. Yoshida, and I. Goldberg, "A User Study of Off-The-Record Messaging," *SOUPS 2008: Symposium on Usable Privacy and Security*, pp. 1-10, SOUPS, 2008.

[13] Vinnie Moscaritolo, Gary Belvin, and Phil Zimmermann, "Silent Circle Instant Messaging Protocol Protocol Specification", `https://silentcircle.com/static/download/SCIMP%20paper.pdf`, accessed Feb. 22, 2014.

[14] Trevor Perrin and Moxie Marlinspike, "Axolotl Ratchet", `https://github.com/trevp/axolotl/wiki/newversion`, accessed Feb. 22, 2014.

[15] D. R. Andersen, M. S. Andersland, and T. J. Andersen, "Reference implementation of SSMP: A Secure Multi-party Messaging Protocol," `https://notpublicyet`, 2014.