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: February 22, 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 the conversation, as long as at least two participants are honest. The protocol uses the elliptic-curve Diffie-Hellman key agreement protocol to generate a set of shared secrets between a group of participants. The group is initially formed by an Organizer. The Organizer role is abandoned when key agreement is accomplished, leaving the original organizer as a one of N peers in the messaging network. All setup operations prior to key agreement can 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 '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 key-agreement strategy. Work on the problem continues to date, most notably with[3], however the key-agreement problem remains unsolved. Liu *et.al.*[4] recently improved the GOTR/mpOTR key agreement mechanism.

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. The group is initially constituted by an Organizer. Following the formation of the group, the Organizer role is abandoned and the initial Organizer assumes a peer role, the same as all other Participants.

II. KEY POINTS

- Each group key agreement is facilitated by an Organizer, who is also Participant P_0. Following the key agreement stage, the Organizer is no longer required. The Organizer data is securely deleted and the Participant P_0 role is that of a peer like any other participant.
- Elliptic curve Diffie-Hellman key operations are performed on curve25519 with 32 byte public and private keys.
- 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.
- Putative participants authenticate to the Organizer, prior to activation of the group protocol.

- Participants may not be added to the group. This would require a reinitialization of the group secrets. A Participant may leave the group.

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

III. NOTATION

The following notation is used:

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

- $\oplus$ is the XOR operator

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

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

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

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

- $||$ is the concatenation operator

- 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 curve25519 and ECDH() is the Diffie-Hellman operator on curve25519, *i.e.* for private key u, the corresponding public key is $U = \text{ECDH}(u, X)$. The shared secret between y and z is then $\text{ECDH}(y, Z) = \text{ECDH}(z, Y) = \text{ECDH}(y, \text{ECDH}(z, X))$.

IV. PROTOCOL STATE

Each Participant P_i will maintain the following variables in persistent storage:

- RK - the root key
- HK - the header key
- NHK - the next header key
- mk - the message key
- v - the group private ratchet key
- $\{R_i\}$ - the set of public ratchet keys from each participant P_i
- i - Participant P_i'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

V. KEY AGREEMENT

The Organizer and Participants complete the following steps:

1. Participant P_0 establishes himself as the group Organizer O and solicits keys from other potential participants.

2. Organizer O determines the total number of participants N.

3. Organizer O assigns a participant index number i to each participant.

4. Organizer O generates elliptic curve Diffie-Hellman (ECDH) persistent group identity, and ephemeral group handshake keys (u, U) and (w, W).

5. Organizer O forwards U, W, N, and i to each participant P_i.

6. Each participant P_i generates ECDH persistent identity, ephemeral handshake and ephemeral ratchet keys (b_i, B_i), (k_i, K_i) and (r_i, R_i).

7. Each participant P_i forwards B_i and K_i to Organizer O.

8. Each participant P_i forwards R_i to all other participants in the group.

9. Organizer O authenticates the identity of each participant P_i with identity key B_i.

10. Each participant P_i authenticates the group identity U with Organizer O.

11. Organizer O computes:
    $L_i = \text{hash}(\text{ECDH}(u, K_i) \,||\, \text{ECDH}(w, B_i) \,||\, \text{ECDH}(w, K_i))$,
    $G_i = \text{ECDH}(w, R_i) \oplus \oplus_{j \neq i} L_j$.

12. Organizer O forwards G_i to participant P_i for all participants.

13. Participant P_i computes:
    $\text{MK} = \text{hash}(\text{ECDH}(k_i, U) \,||\, \text{ECDH}(b_i, W) \,||\, \text{ECDH}(k_i, W) \oplus G_i \oplus \text{ECDH}(r_i, W)$

14. Participant P_i computes his/her initial ratchet state $\hat{\mathcal{A}}_i$:
    $\text{RK} = \text{KDF}(\text{MK}, 0\text{x}00)$,
    $\text{HK} = \text{KDF}(\text{MK}, 0\text{x}01)$,
    $\text{NHK} = \text{KDF}(\text{MK}, 0\text{x}02)$,
    $\text{mk} = \text{KDF}(\text{MK}, 0\text{x}03)$,
    $v = \text{KDF}(\text{MK}, 0\text{x}04)$,
    $\{R_i\}$ from participants,
    N from Organizer O,
    i from Organizer O,
    resync_required = *False*.

15. The Organizer role is complete and all Organizer data is securely erased by participant P_0.

VI. 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} and sets $R_j = R_j^{new}$.
3. Participant P_j computes preliminary ciphertexts $c_h = e(\mathrm{HK}, j \,||\, R_j^{new})$ and $c_m = e(\mathrm{mk}, m)$. He the forms $c' = c_h \,||\, c_m$.
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 updates his/her ratchet state $\hat{\mathcal{A}}_j$ as follows:
   $\mathrm{RK} = \mathrm{hash}(\mathrm{RK} \,||\, \mathrm{ECDH}(v, \oplus_i R_i))$,
   $\mathrm{HK} = \mathrm{NHK}$,
   $\mathrm{NHK} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}02)$,
   $\mathrm{mk} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}03)$.
7. Participant P_j broadcasts c to all participants P_i.

VII. 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 $\mathrm{hash}(v, c') = c_{hmac}$. If they do not match, he/she raises a Bad_HMAC exception and goes no further.
3. Participant P_j splits c' into header c_h and message c_m components and obtains $q \,||\, R_q^{new} = d(\mathrm{HK}, c_h)$ and $m = d(\mathrm{mk}, c_m)$. If either of these operations fail, 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 IX A.

4. Participant P_j sets $R_q = R_q^{new}$.

5. Participant P_j updates his/her ratchet state $\hat{\mathcal{A}}_i$ as follows:
   $\mathrm{RK} = \mathrm{hash}(\mathrm{RK} \,||\, \mathrm{ECDH}(v, \oplus_i R_i))$,
   $\mathrm{HK} = \mathrm{NHK}$,
   $\mathrm{NHK} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}02)$,
   $\mathrm{mk} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}03)$.

VIII. 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.

7. Decryption will need to be added to the housekeeping section.

IX. 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 IX A 1
0x01	Instant Messaging Within SMMP	Section IX A 2

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:* $b = 0\text{x}00$

It is possible, during communication within the group, that a participant's ratchet state may become unsynchronized. Transport layer failures due to lost packets or packet 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 sets $v = m$.

2. Participant P_j updates his/her ratchet state $\hat{\mathcal{A}}_j$ as follows:
   $\{R_i\} = \{\text{hash}(v \,||\, i)\}$,
   $\text{RK} = \text{hash}(\oplus_i R_i \,||\, \text{ECDH}(v, \oplus_i R_i))$,
   $\text{HK} = \text{KDF}(\text{RK}, 0\text{x}01)$,
   $\text{NHK} = \text{KDF}(\text{RK}, 0\text{x}02)$,
   $\text{mk} = \text{KDF}(\text{RK}, 0\text{x}03)$.

3. Participant P_j sets his resync_required flag to *False*.

2. Decrypting Private Messages Within SMMP: $b = 0\text{x}01$

tbd...

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 ratchet key v^{new}.
2. participant P_j computes preliminary ciphertext $c' = e(v, 0\text{x}00 \,||\, v^{new})$.
3. Participant P_j computes hmac $c_{hmac} = \text{hmac}(v, c')$.
4. Participant P_j forms $c = c' \,||\, c_{hmac}$.
5. Participant P_j delays according to a backoff algorithm that should be transport-specific and is unspecified here.
6. 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.
7. Participant P_j broadcasts c to all participants P_i.
8. Participant P_j updates his/her ratchet state $\hat{\mathcal{A}}_j$ as follows:
   $v = v^{new}$,
   $\{R_i\} = \{\text{hash}(v \,||\, i)\}$,

$\mathrm{RK} = \mathrm{hash}(\oplus_i R_i \,||\, \mathrm{ECDH}(v, \oplus_i R_i))$,

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

$\mathrm{NHK} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}02)$,

$\mathrm{mk} = \mathrm{KDF}(\mathrm{RK}, 0\mathrm{x}03)$.

9. Participant P_j sets his resync_required flag to *False*.

X. DISCUSSION

Various protocols exist for facilitating secure one-on-one messaging, including Off the Record Messaging (OTR)[5–8], Silent Circle Instant Messaging Protocol (SCIMP)[9], and Axolotl[10]. 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 user 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.

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. Thus far, to our knowledge, the key-agreement problem has not been solved. As a result, this protocol has never been successfully implemented.

An improvement of the GOTR/mpOTR protocol was made by Liu *et.al*[4].

In this paper, we have presented a secure, peer-to-peer multi-party messaging protocol (SMMP) that provides PFS, PFuS, and PD, and has a simple key agreement algorithm. 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.

---

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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