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.

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 periodically contributes key material to the group key set. 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 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 $U = \mathrm{ECDH}(u, X)$. The shared secret between y and z is then $\mathrm{ECDH}(y, Z) = \mathrm{ECDH}(z, Y) = \mathrm{ECDH}(y, \mathrm{ECDH}(z, X))$.

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

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

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

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' to the message-received housekeeping routine described in Section VIII 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. 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 is only 1):

Byte Value	Housekeeping Task	Location
0x00	Resynchronizing the Ratchet State	Section VIII A 1

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: $b = 0\mathrm{x}00$

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*.

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:

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 random 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:
   $v = v^{new}$,
   $\{R_i\} = \{\text{hash}(v \,||\, i)\}$,
   $\text{RK} = \text{hash}(\oplus_i R_i \,||\, \text{ECDH}(v, \oplus_i R_i))$,
   $\text{HK} = \text{KDF}(\text{RK}, 0\text{x}01)$,
   $\text{NHK} = \text{KDF}(\text{RK}, 0\text{x}02)$,
   $\text{mk} = \text{KDF}(\text{RK}, 0\text{x}03)$.

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

---

[*] 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. 14, 2014.

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