(n+1)sec: A Purely End-to-End Secure Multiparty Messaging Protocol

Vmon (Pseudonym)
eQualit.ie vmon@equalit.ie

ABSTRACT

While there have been several two-party protocols to provide an end-to-end solution to secure messaging communication, there has been no widely adopted end-to-end solution for multiparty messaging. In this paper we present the (n+1)sec protocol, which guarantees integrity, forward secrecy, consistency and deniability in a multiparty conversation which treats the server as an adversary and relies only on end-to-end security. As such, we present an implementation that can be used by users of any XMPP service.

1. INTRODUCTION

Two-party Off-the-Record messaging (OTR) was introduced in [2] as an alternative to PGP for secure casual Internet chat by additionally providing forward secrecy and deniable transcript features. Since publication in 2004, OTR has defined the standard for secure Internet chat, attracting a lot of academic attention and security analysis. Research on OTR both concentrates on analysing OTR properties, as well as extending those to a multiparty use case.

Various attempts have been made to construct an efficient authenticated multiparty key exchange protocol (also known as authenticated group key exchange protocol). OTR authors proposed a generalisation of two-party OTR to a multiparty use case in [4]. Besides not specifying the cryptographic primitives and formal definition of the adversaries, some primary performance analysis of the implementation of the key agreement protocol has been shown to be impractically slow, especially on mobile devices [5][3].

[6] proposes GOTR as an alternative to [4], aiming at improving some of its security properties. Despite the use of p2p private channels for transcript consistency between users, it disregards room consistency. GOTR introduces a notion of repudiability based on colluding 'honest' parties.

Using the Axolotl Protocol [7] for two-party messaging in a multicast scheme as in Signal, some of OTR's desired properties can be extended to multiparty messaging [8]. However, this stops short of providing participant and transcript consistency. Moreover, a multicast scheme without group key agreement provides poor data transmission efficiency - a matter of concern for applications where bandwidth consumption is critical. Additionally, multicast schemes are not contributory and are prone to key control attack.

As iterated in [8], a complete end-to-end (E2E) secure practical multiparty messaging scheme which promises participant (and transcript) consistency has not been put forward yet. This proposal introduces the $(n+1)$sec protocol and proves its strength against such expected adversaries.

Developed for the XMPP platform, $(n+1)$sec assumes its use cases. Most importantly, it allows for secure multiparty key exchange and E2E encryption without extensive computational requirements from the client.

The remainder of this proposal is organized as follows. In Section 2 we summarise the rationale of the choice of security features, the target adversaries and the protocol design. Section 3 is an overview of the current protocol implementation and its various benchmarks. In Section 4 we discuss the challenges the $(n+1)$sec protocol needs to further address.

2. RATIONALES AND DESIGN

Our approach for $(n+1)$sec design was based on the following requirements, in order of importance:

1. A protocol that is provably E2E secure in a strong adversarial model that addresses confidentiality, authenticity, forward secrecy and consistency.
2. Applicable to the XMPP use case.
3. Providing some degree of deniability when it does not negatively impact usability or our security goals.
4. Addressing security flaws in [2] and [4].
5. Addressing the inevitable conflict that arises in a group E2E protocol.

We designate the protocol suggested in [4] as our starting point and apply various modifications to reach a desirable protocol satisfying the above-stated goals.

To satisfy the above properties, $(n+1)$sec is designed in two phases.

Deniable authenticated signature key and session key exchange, where participants deniably authenticate each other and agree on session key(s), while also exchanging ephemeral signing keys and agreeing on a session key.

Communication and Transcript consistency verification, where parties send authenticated confidential messages and verify that all have received and seen an identical transcript in the same order.

2.1 Secure deniable key exchange algorithm

$(n+1)$sec's deniable authentication and key agreement protocol follows the group key agreement of [1] and Axolotl's two-round variation of Triple Diffie-Hellman (TDH) deniable authentication in parallel. Participants perform an extra phase of confirmation to assure room consistency. The signatures are signed by ephemeral keys in contrast to [1].

This will ensure:

- **Participants' deniable authenticity** based on their long-term persistent identity: although a participant in a chat can be sure of other participants' authenticity, they cannot prove this confidence to anyone who has not participated in the session or interacted with the authenticator prior to the session.
- **Confidentiality** of the encryption key which assures the confidentiality of the group conversation.
- **Forward secrecy** of the conversation, so its content remains inaccessible if a participant's long-term private key (which represents their long-term identity) is compromised after session key establishment. The two-phase TDH provides a stronger notion of forward secrecy compared to the one-phase TDH in Axolotl. The $(n+1)$sec sessions are ephemeral to ensure forward secrecy in long-lasting $(n+1)$sec sessions.
- **Room consistency**, where all participants are confident that they have participated in the same room, and that everybody in the room believes everybody else sees the same participant list as they do.

2.2 Secure Consistence Communication

(n+1)sec.
in-session message consists of session ID, confidential payload encrypted by AES-256 in CGM and the signature using ED25519 scheme. The confidential payload consists of sender ID, message ID(s), hash of the transcript (seen by the sender prior to sending the message), nonce and optional user message. Participants acknowledge the consistency of the transcript by sending empty messages.

This ensures the following properties:

- **Message origin authenticity** against both outsider intrusion and the impersonation of existing participants by other malicious participants in the session. This means that the user can be assured of the authenticity of the sender of each original message even if other participants in the room try to impersonate the sender and send messages on their behalf.
- **Confidentiality** of the conversation, so its content is not accessible or readable by an outsider.
- **Transcript consistency**, where all participants are confident they have participated in the same conversation, and, as the conversation continues, that they have been seeing the same sequence of messages.

Because $(n+1)$sec as an end-to-end does not rely on the server to enforce security properties, inevitably we are facing situations where a participant has the power to deny service to other participants. Participants can legitimately question the authenticity of other participants and refuse to participate in a common session with them. However, they should be transparent in that action. In contrast, they can covertly block others' participation or session formation by:

1. Unresponsiveness: To mitigate unresponsiveness, $(n+1)$sec participants run response timers. If a user does not reply in the grace period, they request to start a session without that user.
2. Generating wrong key share, i.e. confirming wrong session: When users get an inconsistent key or session (3 or 4) they re-run the key agreement protocol (to refresh the ephemeral keys). If an inconsistency occurs again they reveal the ephemeral keys, detect the cheater(s) and start a session without them.
3. Accusing other participants of the above: if a member of the group does not believe the reason to kick out another user, they start a session without the kicker.

Disagreement in DoS mitigation responses results in many parallel sessions in the same room. This is inevitable, as the security model of $(n+1)$sec requires the agreement of all members of the session for the session to be established.

2.3 Reliability and message order

$(n+1)$sec's primary use case is XMPP MUC protocol where the unique global order is imposed to the conversation by the server. But $(n+1)$sec can be modified to recover from an unreliable transport. For the key agreement phase:

- If only some of the participants do not receive a session establishment session, they will complain asking that the sender be kicked out of the conversation. In response, those who received the corresponding message will re-transmit the missed message.
- Message order is only important during the joining process, when the joiner sends their share. If two current participants have different views about which joiner has sent their key first, then they will go with a session with smaller hash value.

During the session:

- If a participant does not acknowledge a message, the sender will resend it.
- When transport is unreliable the transport protocol builds a partial order and computes the global order based on collapsing the partial order using the time of arrival.
- If the global order differs among participants, we can reshuffle messages based on their hash to reach consistent order between participants.

3. IMPLEMENTATION

You may go to np1sec's Github page [11] to follow and contribute to its implementation.

You may follow the README to make "Jabberite", a simple jabber client to test $(n+1)$sec using any XMPP server.

4. DISCUSSION

To our knowledge, $(n+1)$sec is the first protocol designed to answer all discussed properties and challenges in an end-to-end secure environment and is thus an evolving project. We would like to seek feedback on the various aspects of the protocol, specifically the way the conflict resolution happens in the denial of service and unreliable transport cases.

More details on the cryptographic protocol can be found at [12]. The details of DoS mitigation are accessible at [9]. For questions and comments, you can comment at [10].

5. REFERENCES

[1] M. Abdalla, C. Chevalier, M. Manulis, and D. Pointcheval. Flexible Group Key Exchange with On-Demand Computation of Subgroup Keys. volume 6055 of *LNCS*, pages 351–368. Springer.

[2] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record Communication, or, Why Not to Use PGP. WPES '04, pages 77–84. ACM.

[3] A. Filasto et al. mpOTR implementation for Cryptocat. `https://github.com/hellais/cryptocat/tree/mpotr`.

[4] I. Goldberg, B. Ustaouglu, M. D. Van Gundy, and H. Chen. Multi-party Off-the-record Messaging. CCS '09, pages 358–368. ACM.

[5] M. V. Gundy. Improved Deniable Signature Key Exchange for mpOTR.

[6] H. Liu, E. Y. Vasserman, and N. Hopper. Improved Group Off-the-record Messaging. WPES '13, pages 249–254. ACM.

[7] Perrin, Trevor. Axolotl Ratchet. `https://github.com/trevp/double_ratchet/wiki`.

[8] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure Messaging. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 232–249. IEEE.

[9] vmon. Denial of service mitigation in (n+1)sec. `https://github.com/equalitie/np1sec/blob/master/doc/dos-mitigation.pdf`.

[10] vmon et al. (n+1)sec discussion page. `https://learn.equalit.ie/wiki/Talk:Np1sec`.

[11] vmon et al. (n+1)sec library for multiparty secure messaging. `https://github.com/equalitie/np1sec`.

[12] vmon et al. (n+1)sec specification. `https://learn.equalit.ie/wiki/Np1sec`.