(n+1)sec high level API specification

eQualit.ie

September 8, 2016

1 Introduction

The (n+1)sec library defines and implements a system of secure synchronous group communication, allowing a group of people to perform text chats while enjoying communication security guarantees similar to those granted by the OTR or *off-the-record* system. Similarly to OTR, (n+1)sec ensures that chat contents remain secret to anyone not part of the conversation; provides cryptographic authentication of the identities of the conversation participants; and provides deniability of the entire conversation exchange, making it impossible for a conversation participant to show the conversation transcript to outsiders in a verifiable way. In addition, (n+1)sec can verify that different participants in a conversation are in agreement about what has been said in this conversation, as well as ensuring that only users authorized by the existing participants of a conversation can join that chat; both of these topics are security aspects that do not play a role in a two-party chat, as implemented by OTR.

The (n+1)sec library implements its secure chat systems as a cryptographic layer on top of existing text chat technologies. For example, an (n+1)sec conversation can be implemented using an XMPP Multi User Chat (*xmpp-muc*) room as the underlying chat infrastructure, in which case the (n+1)sec conversation consists of an xmpp-muc room in which the chat participants exchange encrypted messages carrying (n+1)sec content. Similar to OTR, the (n+1)sec library is designed to be usable with any underlying text chat room system (which we call the *carrier chat system*), as long as certain requirements are met.

In this document, we describe the requirements the carrier chat system needs to satisfy in order to be able to support (n+1)sec; the way the (n+1)sec library interacts with the carrier chat system; the security properties guaranteed by the (n+1)sec system; and the interaction between the (n+1)sec library and the chat client front-end. Together, this defines the high level API of the (n+1)sec

library. In Section 2, we describe the model and properties of the carrier chat system, seen from the point of view of the (n+1)sec library; Section 3 describes the properties and behavior of (n+1)sec conversations from the point of view of a client front-end. Finally, in Section 5, we describe the complications involved in the process of initially joining an (n+1)sec conversation as hosted by a given carrier chat room.

2 Carrier chat model

Conversations secured by (n+1)sec use text chat systems as a carrier group communication infrastructure. These systems consist of a *room* that users can enter or leave; while in the room, they can exchange text messages, which are sent to all users in the room. Commonly used systems of this description include XMPP Multi User Chat rooms, and IRC channels. (n+1)sec uses chat *rooms* as the basic infrastructure component; it does not concern itself with the way these rooms are part of conference servers (xmpp-muc), networks (IRC), or other forms of higher levels of aggregation.

In order to function, the (n+1)sec protocol requires that the carrier chat system satisfies certain properties. If these properties are not met, this does not compromise the security of the (n+1)sec conversation; rather, the (n+1)sec library will notice the anomaly, and will not be able to establish or continue a conversation. In particular, (n+1)sec requires the carrier chat system to satisfy the following requirements:

1. The carrier chat room contains a consistent, well-defined set of users —the *members* of the room— that are considered part of the room. All users in the room can see this member set, and all members receive all messages sent to the room. Users get notified when the set of room members changes, i.e. when new members join the room, or existing members leave the room.

2. Room members have a recognizable unique identity that stays stable for at least the duration that the member remains part of the room. Changeable unique nicknames, such as used by the IRC system, are sufficient as long as users receive a notification of the nickname changes.

3. Chat events occurring in the room —which include at least chat messages, joining members, leaving members, and nickname changes (where applicable)— have a strict and consistent order enforced by the carrier chat system. Different members of the room receive the same chat events in the same order. This includes chat events originated by the receiving member; for example, if a member sends a chat message, it should be

aware of the timing at which it is received by the room, relative to other chat events in the room.

It is a noteworthy observation that not all commonly used chat systems satisfy all these requirements. In particular, the popular IRC system does not satisfy requirement 3; users connected to different servers of the same IRC network may receive near-simultaneous chat events in different orders, and the sender of a message generally cannot tell the time it is received relative to surrounding events. This means that (n+1)sec as currently described cannot use IRC as a carrier chat system. In order to broaden the applicability of the (n+1)sec system, extensions to the (n+1)sec protocol in order to lift requirement 3 are explicitly considered as a possible future improvement step.

The (n+1)sec protocol does not rely on any of the requirements above to ensure any security guarantees; in particular, the security properties described in Section 3 still hold when the carrier chat system does not satisfy these requirements. Instead, in such a situation, the (n+1)sec library will not be able to establish secure chat sessions at all; any attempts to use the (n+1)sec system on such a carrier chat system will result in reported error situations such as network timeouts, protocol errors, et cetera. As a special case of this general point, per requirement 3, the (n+1)sec system assumes that users have a connection to the carrier chat system that is free from interference, in which attackers cannot interfere with the communication between user and carrier chat system; this can be easily accomplished by the application of proper transport security, as implented by systems such as TLS, to the connection between the user and the carrier chat system. If this assumption is violated —which can happen, for example, when a user uses a plaintext connection to connect to a carrier chat system, and an attacker modifies the message stream sent to or by the user— then the security properties described in Section 3 still hold, but the user will not be able to successfully join any (n+1)sec sessions.

To perform secure communications, (n+1)sec needs to perform actions in the role of a member of a carrier chat room. That is to say, it needs a resource consisting of a user in the carrier chat system that is a member of a chat room, and be able to perform the following operations:

- send a chat message as the (n+1)sec user to the room;
- have the (n+1)sec user leave the room.

Moreover, (n+1)sec needs to be notified of the following events happening in the chatroom:

- a new member joins the room;
- a member leaves the room;

- a member sends a message to the room. This includes messages sent by the (n+1)sec user.

Finally, (n+1)sec needs to know the following piece of information regarding the (n+1)sec user:

- the stable unique identifier of the (n+1)sec user.

A secure chat client using the (n+1)sec library needs to implement these operations, and give (n+1)sec access to the carrier chat member; when the chat events described above happen, it needs to notify (n+1)sec of this fact.

3 (n+1)sec chat model

Secure communications implemented by (n+1)sec are organized into conversations that behave similar to chat rooms; to avoid ambiguity, we shall refer to these conversations as *(n+1)sec channels*. An (n+1)sec channel is a construction similar to a chat room in most chat systems: it consists of a set of chatting users (called *participants*, again for unambiguity reasons) that can send messages to each other, and like many chat systems these chat messages are delivered only to the chat participants. But whereas most chat systems rely on server-side access control to implement security measures such as communications privacy and authentication of participants, an (n+1)sec channel guarantees these properties relying only on end-to-end cryptography.

Using this cryptography, (n+1)sec channels have the following security properties:

- The contents of messages sent to a channel are secret. The only entities able to access the message contents are the participants of the channel.
- All channel participants can verify the public-key-based cryptographic identity of all other participants.
- Verification of identities of participants is deniable: anyone can forge a transcript containing arbitrary contents and participants, which makes the contents of saved transcripts of little use as evidence of what happened during the chat.
- New participants cannot join a channel without approval of all existing participants. Participants know the exact set of participants in the channel at all times.

- Participants can verify that they are all in agreement about the events happening in a channel, a procedure we call *transcript consistency verification*. In particular, it is not possible for different participants to receive different versions of messages, or otherwise have a different view of the chat transcript, without triggering a verification alert.
- The above properties satisfy *forward secrecy*: compromise of long-term private keys defining participant identities does not compromise the security properties of historic chats, even with access to a full transcript.
- The short-term keys used to ensure forward secrecy are only used for a short amount of time, after which they are refreshed. This ensures that the compromise of a short-term key compromises only a small fragment of long-running conversations.

In order to implement these properties, the process of a participant joining or leaving an (n+1)sec channel is a relatively complex one. Whereas most chat systems model the joining and leaving of members as atomic events, both processes when applied to an (n+1)sec channel are multi-step processes. To model this, participants of (n+1)sec channels can be in one of four distinct states:

- *authenticating*: A participant is in the *authenticating* state when they have announced their intention to join the channel, but their identity has not yet been confirmed or accepted by all *active* or *joining* participants of the channel. An authenticating participant has not been established to be the person they claim they are. Authenticating participants can neither send messages to, nor decrypt messages sent to the channel. When authentication completes, the participant moves to the *joining* state.
- *joining*: A participant is in the *joining* state when they have been authenticated and approved by all *active* and *joining* members, but the key exchange process that would enable the participant to decrypt channel messages is still ongoing. Joining participants cannot reliably decrypt messages sent to the channel; however, they may be able to decrypt an unpredictable subset of messages sent to the channel. Once the key exchange process finishes, a joining participant becomes *active*.
- *active*: A participant is in the *active* state when they have completely finished joining the channel. Active members can both send messages to the channel and decrypt messages sent to the channel. When an active participant wants to leave the channel, they enter the *leaving* state.
- *leaving*: A participant is in the *leaving* state when they have announced their intention to leave the channel, but they have yet to verify the transcript consistency of recent chat. A leaving participant can decrypt an

unpredictable subset of messages sent to the chat; they cannot send messages. When the transcript consistency status of all chat before the announcement to leave the channel has become clear, the leaving procedure is completed and the leaving participant is removed from the channel.

When joining an (n+1)sec channel, participants start in the *authenticating* state, and barring complications eventually become *active* participants. Participants do not necessarily pass through the *leaving* state before leaving a channel; participants can leave without warning at any stage if they are not interested in verifying transcript consistency, and this is also what will generally happen in case of connectivity problems.

Internally, (n+1)sec channels are constructed out of a multitude of cryptographic constructions which we call (n+1)sec *sessions*. A session is an agreement between the active participants of a channel to use a particular shared key for encrypting chat messages. Unlike channels, sessions cannot be joined or left; when participants join or leave a channel, the channel spawns a new session to accomodate the changed set of participants, and the old session is eventually replaced. New sessions are also created periodically in order to refresh short-term keys, as part of the effort to limit the damage done by the compromise of a short-term private key.

As the interface to (n+1)sec channels, the (n+1)sec library provides an object representing a channel of which the user is a participant. This *channel* object contains at least the following properties:

- A set of participants, each containing a carrier-chat identifier, a public key, and a participant state;
- The participant representing the user;
- The private key of the user;
- Several settings configuring the timeouts used in several places of the (n+1)sec protocol.

The chat client can perform the following operations on the channel object:

- Send a message to the channel, assuming the user is in the *joining* or *active* state;
- Leave the channel, waiting for transcript consistency to complete;
- Leave the channel immediately.

The channel object notifies the chat client of the following events:

- A participant sends a message to the channel. This includes messages sent by the participant representing the user.
- A participant joins the channel.
- A participant leaves the channel.
- A participant changes its join state.
- An authenticating participant needs to be authenticated and authorized to join the chat. When this event happens, the chat client needs to make a decision whether or not to grant access to the authenticating participant; it needs to notify the channel object of this decision asynchronously once it has been made.
- A message sent earlier to the channel has been inspected by the transcript consistency verification system. This event either tells the chat client that the past message is properly consistent with the channel view of the rest of the participants in the channel; or it tells the chat client that the status of this message is disputed by the participants, indicating an attack.

A secure chat client can implement (n+1)sec by allocating channel objects as the user tries to join channels, and responding to the event notifications as desired. Different configurations of the timeout settings can configure the (n+1)sec protocol for different levels of network reliability; different implementations of the authentication callback can be used to define different authentication models.

4 Channel behavior

4.1 Channel consistency

The previous section introduces the concept of an (n+1)sec *channel* as the basic unit of a secure chat conversation. It also defines a number of security properties that the (n+1)sec system guarantees to be true for each channel, which each individual participant can verify by cryptographic means. If communication conditions are such that no secure chat satisfying these conditions is possible —which is the case, for example, if the carrier chat system tries to manipulate the messages sent or received by particular chat members— then the (n+1)sec system will respond by making sure that no channel can be constructed, and that affected existing channels may break down, making further chat impossible.

An (n+1)sec channel is a distributed cooperation between a set of participants, who are in agreement about such things as the list of authenticated participants, and the active cryptographic session. There is no such thing as a central authoritative version of the state of the channel; instead, the system relies entirely on

the proposition that the participants are in agreement with each other, and stay that way. If, for whatever reason, the participants of a channel find themselves in unexpected disagreement about the state of a channel —for example, if one subset of a channel is of the opinion that user Alice has been authenticated by the entire channel and is therefore promoted to a *joining* participant, whereas the remainder of the channel is of the opinion that Bob has yet to authenticate Alice— then the coherence of the channel has broken down, and the channel cannot continue in its present form. This situation can arise if the carrier chat system manipulates the contents of the carrier chat, such as by not informing part of the channel that Bob has authenticated Alice; a malicient user might also try to arrange for this situation as a denial of service attack.

When this happens, the (n+1)sec system responds by performing a procedure that ultimately leads to the disagreeing segments of the channel each continuing as a separate, independent channel. Each (n+1)sec client, when noticing that part of the channel is in disagreement with the client about the channel status, decides to unilaterally remove all participants from the channel with whom they are in disagreement. Because client agreement is an equivalence relation, the consequence of this procedure is that each "side" of the disagreement has constructed an internally consistent descendent channel; effectively, the original disagreeing channel has split itself into independent component channels that are once again in agreement. From the point of view of any particular participant of the disintegrating channel, this splitting procedure is asymmetric: all participants on the other side of the split have simply been kicked from the channel because of a protocol violation. Only from the point of view of an outsider watching the channel does the breakdown of a channel look like a symmetric split.

4.2 Denial of service

The system outline above, consisting of channels with cryptographically enforced security properties that split into consistent components whenever the security requirements can no longer be guaranteed, forms a secure chat system in the sense that all successful communications enjoy the desired security properties described in Section 3. For the resulting system to be usable in practice, however, it needs to satisfy a stronger requirement: the system must be set up in such a way that secure conditions between willing participants cannot easily be disrupted. Otherwise, it would be vulnerable to denial of service attacks, in which an attacker aims to make it impossible for honest users to participate in an (n+1)sec channel successfully.

Of course, it is not possible for the (n+1)sec system to function in *arbitrarily* adversarial conditions. An (n+1)sec channel relies on a carrier chat room to function as a message transport infrastructure; if this carrier system is malicious,

then the (n+1)sec system can never ensure that secure chat is possible. Indeed, a sufficiently malicious carrier chat system could simply refuse to accept any messages sent by a particular client, which certainly makes any attempts for that user to participate in a secure chat a hopeless endeavour. Because the (n+1)sec system must necessarily assume a sufficiently cooperative carrier chat system, it makes no attempt to achieve any successful secure conversations when using a carrier chat system that does not satisfy the requirements outlined in Section 2.

By a similar reasoning, an (n+1)sec client relies on a connection to the carrier chat system that faithfully transmits messages to and from the carrier chat system without interference or manipulation. As above, a malicious carrier chat system connection could simply stop transmitting messages at all, thus behaving similarly to a network interruption, for which no client-side recovery is possible. The common availability of countermeasures such as transport layer security provides a further rationale for the reasoning that the (n+1)sec system does not attempt to recover a secure chat conversation when faced with a malicious carrier chat connection.

As long as the above two properties hold, the (n+1)sec system is designed in such a way that honest participants can always construct and maintain secure channels, in which they can hold a secure conversation. In other words, a user of the carrier chat room cannot perform denial of service attacks by disrupting (the construction of) chat channels between honest participants; unless that user can compromise and manipulate the behavior of the carrier chat infrastructure, in which case no defense is possible. This guarantee can be formalized in greater detail as a list of requirements that the carrier chat system must satisfy, followed by a list of properties on the behavior of channels that the (n+1)sec system can guarantee are true. Specifically, the (n+1)sec system assumes the following *carrier chat system requirements*:

- The carrier chat system relays messages between members of the room, without interference, in accordance with the properties listed in Section 2.
- The carrier chat system relays messages between room members in a timely manner; that is, messages are relayed fast enough not to trigger network disconnect timeouts.

A *cooperative* member of the carrier chat room is a member whose connection to the carrier chat system relays messages without interference and in a timely manner. Based on that definition, we can formalize the following properties about the behavior of channels that the (n+1)sec system can guarantee:

- A cooperative non-participant —that is, a potential joiner of a channel— is able to learn and maintain the state of any channel in which they are interested.

- The participants of a channel always agree about the status of the channel.
- A cooperative non-participant can become an *authenticating* participant of any channel at any time.
- An *authenticating* participant eventually becomes a *joining* participant if and only if the participant is authorized by all *joining* and *active* participants in the channel.
- A cooperative *joining* participant eventually becomes an *active* participant.
- An *active* participant can decrypt all messages sent to the channel.
- A cooperative *active* participant can send messages to the channel.
- If a participant stops replying to a channel, or stops replying to particular events in a channel, eventually the participant is removed from the channel by means of a timeout.

The (n+1)sec system behaves in such a way that as long as the carrier chat system requirements hold, the above properties are guaranteed. This defines the degree to which the (n+1)sec system is resilient to denial of service attacks.

5 Joining and constructing (n+1)sec channels

The previous sections describe how (n+1)sec channels behave once one has started the procedure of joining one. What this section does not describe is how one can join a channel in the first place; or, alternatively, how to create an empty one from scratch.

Ideally, the concept of (n+1)sec channels is nearly identified with the concept of carrier chat rooms. Each carrier chat room may contain an (n+1)sec channel; if one wants to hold a secure chat inside a particular carrier chat room, one queries whether one exists; sends a join request if it does; or starts a one-person channel if it does not.

Unfortunately, this simple model isn't one that can be sustained. Several different sets of members of a carrier chat room might try to hold separate (n+1)sec channels inside the room; a split event as described in Section 4.1 can cause this to happen, and it can also be crafted deliberately as a denial of service attack. When this happens, the simple interaction model described above breaks down.

As a consequence, the procedure of joining the (n+1)sec channel inside a carrier chat room is a bit more involved. The process of joining (n+1)sec channels in a carrier chat room progresses through the following steps:

- First, the joining user broadcasts a request to the room, asking any (n+1)sec channels to identify themselves.
- The client displays a list of available (n+1)sec channels in the carrier chat room, along with the users participating in these channels, allowing the user to join one of them.
- While the user has yet to choose an (n+1)sec channel to join, the list of available channels is kept updated to reflect events happening in the available channels: new users joining, participants leaving the channel, et cetera.
- Eventually, the user either chooses to join one of the (n+1)sec channels inside the room, or creates a single-person channel.

The process of searching for (n+1)sec channels to join in a given carrier chat room consists of a model in which the (n+1)sec library maintains a list of available channels for the user to join, which is continuously updated until the user chooses a channel to join. This list is represented as an initially empty set of *channel* objects as described in Section 3, which can be displayed in a user interface. During the channel-search procedure, the following events can happen that change the concents of the available channel list:

- The search process can report the existence of a newly identified channel;
- A channel on the list can change its set of users, which happens when a user joins or leaves the channel, or a user in the channel becomes authenticated;
- A channel on the list can dissolve when its last remaining user leaves the channel;
- A new channel can be created by a user in the carrier chat room;
- A channel on the list has split into multiple channels in response to a synchronization inconsistency, as outlined in Section 4.1.

The channel-searching procedure does not have a natural end; the (n+1)sec library keeps the list of available channels up-to-date in response to room events until the chat client chooses to terminate the procedure. The chat client can perform one of the following two events to terminate a channel-searching procedure:

- It can decide to join one of the available channels on the available channel list, thus becoming an *authenticating* participant in that channel; or
- It can decide to create a new channel inside the carrier chat room, immediately becoming an *active* participant in the freshly-created channel containing only a single participant.

Either operation terminates the channel-searching procedure. Theoretically, there is no strong reason why a user could not function as a participant in multiple channels in the same carrier chat room. But for the sake of the simplicity of the user interaction model, as well as the complexity of the implementation, the (n+1)sec library does not allow a user to participate in more than one channel. Consequently, the (n+1)sec library can at any given time interact with a given carrier chat room *either* by performing a channel searching operation, *or* by participating in a single channel; if one wants to start a new channel-searching procedure in a room, any channels in that room must first be left. Of course, any third-party implementations of the (n+1)sec protocol may not hold themselves to this limitation, and chat clients must not assume that other users necessarily behave in this way. In particular, this limitation is not enforced by any cryptographic assurances, and cannot be relied upon for security purposes.