# (n+1)sec Cryptographic and Protocol Review

## Equalitie

July 14, 2017 – Version 1.0

Prepared for
Ruud Koolen
Dmitri Vitaliev

Prepared by
Alex Balducci
Jake Meredith
Andy Lee

©2017 – NCC Group

Prepared by NCC Group Security Services, Inc. for Equalitie. Portions of this document and the templates used in its production are the property of NCC Group and cannot be copied (in full or in part) without NCC Group's permission.

While precautions have been taken in the preparation of this document, NCC Group the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein. Use of NCC Group's services does not guarantee the security of a system, or that computer intrusions will not occur.

# Executive Summary

## Synopsis

In Spring 2017, the Open Technology Fund[1] engaged NCC Group's Cryptography Services Practice to perform a targeted review of the Equalitie cryptographic library, (n+1)sec.[2] The (n+1)sec library aims to provide a secure group chat mechanism that provides secrecy and consistency of messages. The library is general purpose and can be used in a variety of chat clients much like Off-the-Record[3] (OTR). The engagement included security review of the cryptographic design and the end-to-end group chat protocol.

The review covered the most recent public version of the (n+1)sec library(44d7b28) and specification(5dd2055). Two consultants and one shadowing consultant performed the engagement over a span of two weeks, from April 24 to May 5, 2017 and it consisted of 20 person-days of effort.

## Scope

NCC Group's evaluation focused on issues specific to group key exchanges, ratchets used in secure messaging applications, and general cryptographic concerns. Specific targets included:

- Group Key Exchange To exchange a symmetric key among a group of participants, the (n+1)sec protocol makes use of a group key exchange protocol. This group key exchange protocol expands the scope of Diffie-Hellman[4] to collections of more than two users. The (n+1)sec protocol invokes an instance of a group key exchange protocol each time a conversation requires a new shared symmetric key.
- Denial of Service Protections The (n+1)sec protocol attempts to remove users who have maliciously disrupted conversations between honest participants.
- Protocol The (n+1)sec system is a protocol through which users of text chat systems, such as IRC or Jabber multi-user-chat, can hold cryptographically secured multi-party text chat sessions. Using an approach similar to OTR, (n+1)sec clients exchange encoded text messages and thereby construct end-to-end encrypted chat sessions that use the general-purpose chat room as a carrier.

## Key Findings

During the two weeks of assessment, NCC Group did not find any issues that would seriously undermine the security of the (n+1)sec protocol. There were issues identified that slightly reduced the security of the protocol in certain cases and are summarized below:

- Deniability There are a couple of scenarios in which deniability could be lost to an active attacker. When a new member joins a conversation, previous messages sent within the group chat could become undeniable to the joining member. This attack is not trivial and requires additional compromises or contrived scenarios.
- Forward Secrecy The shared group chat secret key refreshes (or ratchets) when the participants in a chat room change or it is initiated by a member of a chat room. There are no current recommendations for a regular ratcheting interval in the protocol and it is left up to the client. Also, in many modern secure chat protocols there is an additional symmetric key ratchet system; the (n+1)sec protocol does not have a corollary for this. In the event of key compromise (long-term keys, conversation keys, session keys), an increased number of messages may be decrypted if re-keys are not performed regularly or if a symmetric ratchet is not utilized.

## Caveats

NCC Group reviewed the full specification, however, only the portions of the library that lined up with the protocol were reviewed. NCC Group did not perform a line-by-line review of the entire library. Additionally, the review focused on the library and not a chat client implementing the library.

## Strategic Recommendations

During the course of the engagement, NCC Group observed several measures that may enhance the (n+1)sec library's long-term security posture. See Strategic Recommendations on page 9 for details.

[1] https://www.opentech.fund/
[2] https://learn.equalit.ie/wiki/Np1sec
[3] https://otr.cypherpunks.ca/
[4] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

# Dashboard

## Target Metadata

| | |
|---|---|
| **Name** | (n+1)sec |
| **Type** | Cryptographic Protocol and Library |
| **Platforms** | C++ |

## Engagement Data

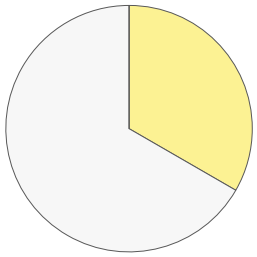
| | |
|---|---|
| **Method** | Code-assisted Cryptographic Review |
| **Dates** | 2017-04-24 to 2017-05-05 |
| **Consultants** | 2 |
| **Level of effort** | 20 person-days |

## Targets

| | |
|---|---|
| **(n+1)sec repository** | https://github.com/equalitie/np1sec/commit/44d7b2887a536d01f138b6ad43945a4b4a4b0c63 |

## Finding Breakdown

| | |
|---|---|
| Critical Risk issues | 0 |
| High Risk issues | 0 |
| Medium Risk issues | 0 |
| Low Risk issues | 1 |
| Informational issues | 2 |
| Total issues | 3 |



## Category Breakdown

| | |
|---|---|
| Cryptography | 3 |

## Key

Critical High Medium Low Informational

This section discusses the various cryptographic promises of the (n+1)sec protocol. These are described on page 6 of the protocol description.[5] Each subsection below includes the description of a cryptographic property, taken from the protocol specification, followed by NCC Group's analysis.

## Confidentiality

### Description

> The content of chat messages exchanged through a conversation is confidential. It is accessible only to (i) participants of the conversation, (ii) joining participants of the conversation for which the joining operation is in progress, and (iii) former participants of the conversation for which the leaving operation is in progress.

### Evaluation

Confidentiality is provided by AES in GCM mode.[6] All participants use the same key to send (and receive) messages. Nonces are randomly generated per message. The shared secret is generated after all participants complete the process described in Group Key Exchange and Key Authentication on page 8. The three accessibility situations described are not fully realized as described below; however, confidentiality is **not** lost.

(i) Participants in a conversation have computed a shared secret via the group key exchange. This exchange is detailed in Group Key Exchange and Key Authentication and naturally ensures that non-participating members do not learn the shared key.

(ii) Joining participants do **not** have the capability of reading the contents of chat messages. While they have obtained a copy of the state machine and are updating their copy with messages sent to the group chat, they have not performed the group key exchange, therefore, they do not have the secret key used to decrypt messages.

(iii) When a participant leaves the chat a new group key exchange is initiated by the remaining members. Any messages that are sent before this process finishes are encrypted with a key known by the leaving member. This means a participant who leaves a chat can continue reading messages until the group key exchange finishes.

## Participant Authenticity

### Description

> All participants of a conversation can verify the cryptographic identity of all participants in the conversation via their respective public keys.

### Evaluation

Authenticity stems from the long-term public keys that each user possesses. As this is simply a protocol (and a library), no method for authenticating these public keys is performed. *It is therefore required that chat clients using the library provide a method in which these long-term public keys can be validated.* Establishing trust elsewhere *relies* on these long-term public keys.

Several ephemeral keys are used when participating in carrier chats and specific chat rooms:

1. Room public keys are used to authenticate a user within a carrier chat room.
2. Conversation public keys are used to authenticate a user within a specific chat room.
3. Session public keys are used during key ratcheting steps.

#### Room Public Keys

Room public keys are authenticated to a participant with the triple Diffie-Hellman construction described in Group Key Exchange and Key Authentication. They are an artifact of previous iterations and not used elsewhere in the protocol. When a member joins a carrier chat room (the broader structure in which individual chat rooms are created within), they

[5] https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/doc/protocol.pdf
[6] http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

perform an authentication step with each member. This begins with the ROOM_AUTHENTICATION_REQUEST message, indicating that the sending user wishes to authenticate the recipient user (with their alleged long-term public and room public keys) using an authentication challenge. The authenticating user must respond with a ROOM_AUTHENTICATION request containing the proper authentication confirmation value (see Group Key Exchange and Key Authentication).

### Conversation Public Keys

Conservation public keys are unique to a particular group chat instance. These are authenticated in the same manner as the room public keys except with chat specific, not carrier room, messages. Here the authentication request is given by the CONVERSATION_AUTHENTICATION_REQUEST message and responses are given by the CONVERSATION_AUTHENTICATION message. Then the message public key is substituted with the conversation public key.

### Session Public Keys

The session public keys, used as contributions to the group key exchange, are signed using the conversation private key. Because the conversation public key is deemed trustworthy (outlined in the procedure above), the authenticity of these keys has been provided.

## Message Origin Authenticity

### Description

All participants of a conversation can verify the public key cryptographic identity of the author of each message sent to the conversation.

### Evaluation

Each message sent by an individual is signed by their conversation private key. Because trust has been established from this key to a user's long-term public key, the origin of each message can be ascertained.

## Deniability

### Description

Participation in a conversation does not leave any cryptographic proof that a conversation's participants can use to prove a participant's participation in that conversation to a third party. Any participant in a conversation, or any nonparticipant of a conversation, can forge a transcript of a conversation involving arbitrary participants and arbitrary chat contents that is cryptographically indistinguishable from the conversation that actually took place.

### Evaluation

The deniability claims can be broken into two parts:

1. The ability to deny participation in a conversation.
2. The ability to deny the contents of a participation.

As outlined in finding NCC-np1sec-2017-001 on page 11, previous conversation within a group chat is not deniable to new members. In this deniability break, a malicious user who can decrypt and record a chat message can convince a newly invited member that the chat message was sent by whoever actually sent it, even if the newly invited member was not part of the conversation when the original chat message was sent.

While participation within a particular conversation is deniable, participation in a carrier chat is not deniable as outlined in finding NCC-np1sec-2017-003 on page 13. In this case, a user (whose long-term private key and room private key are compromised) who joins a carrier chat room can prove to a third party that members of said group chat participated.

## Transcript Consistency

### Description

Participants of a conversation can verify that they have all witnessed the same sequence of chat events in a conversation. In particular, they can verify that all participants witnessed the same sequence of chat messages, and the same set of participants making up the conversation at each point in time.

### Evaluation

The protocol makes use of a conversation state machine to track the various messages and events that occur in a chat room. The overall structure is described in section 5.3 of the protocol specification. This state machine holds the list of members, the current key exchange iteration, and a few other relevant fields. When a message is received, a running hash is updated, which is calculated as follows:

```
status-checksum = Hash(state-machine || message sender || opcode || message body)
```

Here the `state-machine` represents an encoding of the previous state machine.

With this in hand, a participant can send a `CONSISTENCY_STATUS` request indicating that they will be publishing their view of the conversation state (the status-checksum) in a subsequent message. The other members in the chat room will record this event, record their current checksum, and await for the status-checksum. The original participant can now send a `CONSISTENCY_CHECK` message containing the status-checksum. Other participants can now verify that they share the same state as the participant sending the consistency check.

## Forward Secrecy

### Description

The compromise of any long term private keys that define the cryptographic identity of a participant does not compromise the above security properties for any historic chats, even for an attacker with access to a full carrier chat room transcript. In addition, any short term keys used in implementing the abstraction of a conversation are only used for a short amount of time before they are replaced; this ensures that the compromise of a short-term key only compromises a small fragment of long-running conversations.

### Evaluation

Each conversation is encrypted with keys derived from the long-term public key and ephemeral session keys as detailed in Group Key Exchange and Key Authentication, naturally providing forward secrecy. In order to provide stronger forward secrecy, the protocol allows for a "refresh" of the conversation key by having everyone in the chat room re-compute a group key exchange. The length of time between performing a new group key exchange (called a ratchet) is not currently specified by the protocol, allowing the client to decide for themself, or, in some cases, the users of the chat room to specify themselves. The protocol does not implement a symmetric ratchet per message. This means anyone with access to the group session key and access to the logs or messages, will be able to decrypt any message encrypted under that key until a group key exchange is performed again.

## Resilience to Denial of Service

### Description

As long as the carrier chat room infrastructure is not malicious, the (n+1)sec protocol can ensure the following properties:

- Any collection of honest participants can create and participate in a conversation between themselves without disruption, no matter what messages are sent by nonparticipants to the carrier chat room.
- A malicious participant of a conversation, or a group of colluding malicious participants of a conversation, cannot block a newly invited participant from joining the conversation after being invited by an honest participant. In general, a group of malicious participants cannot cause conversation processes to which an honest participant is party to be blocked or delayed indefinitely.

- A malicious participant of a conversation, or a group of colluding malicious participants of a conversation, cannot cause a pair of two honest participants of the same conversation to remove each other from the conversation. Any messages sent by the malicious participants cannot cause the honest participants to interpret each other's actions as malicious.

### Evaluation

The (n+1)sec protocol attempts to detect certain problems among group key exchanges, namely when a malicious/non-acting user is among the group. Due to the nature of the (n+1)sec group key exchange it is possible for a single malicious actor (attempting to provide invalid/null data as keys) to be detected by the group. This is because, by design, everyone shares the result of a triple Diffie-Hellman key exchange with each of their neighbors, and everyone can see everyone else's public keys. The group would then be responsible for kicking this user out of the group in order to properly secure the chat room. This property can be extended to multiple colluding users against two honest users.

## Random Number Generation

### Description

The quality of random numbers that are generated by the application directly correlates to the entropy level of encryption keys and other entropy driven values.

### Evaluation

The (n+1)sec library does not, itself, generate any random numbers. All generation of values that require high entropy come from the libgcrypt library.[7] The most current version of libgcrypt as of May 5th, 2017[8] does not have any known vulnerabilities in its random number generator, however there was a flaw[9] discovered in 2016 that could affect generated keys. In the README for the (n+1)sec codebase, it is specified for users to download the `libgcrypt-devel` library for use while building the project. This version is not susceptible to the above vulnerability.

[7] https://www.gnupg.org/documentation/manuals/gcrypt/Retrieving-random-numbers.html
[8] https://git.gnupg.org/cgi-bin/gitweb.cgi?p=libgcrypt.git;a=tree;h=9b651fb632f3697e70685c9ee340ab0cb2274bdf;hb=9b651fb632f3697e70685c9ee340ab0cb2274bdf
[9] https://formal.iti.kit.edu/klebanov/pubs/libgcrypt-cve-2016-6313.pdf

This section discusses two cryptographic procedures critical to the security of the (n+1)sec protocol. These are introduced in section 4.1 and section 4.2 of the protocol description.[10]

## Group Key Exchange

The (n+1)sec protocol performs a group key exchange to ensure that all members of the chat room share the same symmetric key. The design is based off of the GKE+P protocols as described in "Flexible Group Key Exchange with On-Demand Computation of Subgroup Keys".[11] The original GKE+P protocol does not concern itself with deniability claims: signatures with a user's long-term private key are used to assert ownership of the ephemeral keys used to create the shared secret. Instead, the (n+1)sec protocol uses the triple Diffie-Hellman construct[12] to achieve the same goal while ensuring that the protocol is deniable. Like in the GKE+P paper, participants of the group chat create ephemeral session keys in Round 1 and distribute them amongst their peers. The protocol proceeds as in the paper, substituting the Diffie-Hellman operation with the triple Diffie-Hellman operation (this step also provides authenticity between peers). As noted in "Security Weakness of Flexible Group Key Exchange with On-Demand Computation of Subgroup Keys",[13] a critical key confirmation step is missing from the GKE+P protocol. (n+1)sec includes this necessary check, ensuring that all members of the chat have derived the same key.

## Key Authentication Routine

Alongside group key exchange, members create conversation ephemeral keys. These keys will be used to specify the sender of each message by means of a cryptographic signature. Since these are ephemeral keys and not tied to a user's long-term public / private key, deniability to the outside world is maintained. As ephemeral keys, a necessary step must be inserted into the key exchange to ensure their authenticity (i.e. that the purported owner is actually the owner). To do so, each member will send a challenge nonce to each other member. To prove ownership of the conversation ephemeral key, a user will perform the following operation and send the following result to the requesting peer: $H(username + nonce + TDH)$

Where `H` is a cryptographic hash, `username` is the responding user's username, `nonce` is the challenge nonce, and `TDH` represents the triple Diffie-Hellman operation performed with the following keys:

1. Alice's long-term public key
2. Alice's ephemeral conversation key
3. Bob's long-term public key
4. Bob's ephemeral conversation key

Upon receipt of this, the requesting user has authenticated the sender, provided the returned result is correct. In the example above, either Alice or Bob can act as the requesting user (with the other acting as the responding user).

[10] https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/doc/protocol.pdf
[11] http://www.di.ens.fr/users/pointche/Documents/Papers/2010_africacryptB.pdf
[12] https://whispersystems.org/docs/specifications/x3dh/
[13] https://arxiv.org/abs/1008.1221

## Strengthen Deniability

Refresh conversation keys when new members join to ensure that previous conversations within the chat are deniable. See finding NCC-np1sec-2017-001 on page 11 for more details.

## Further Protecting Against Denial of Service

The most impact a denial of service attack could cause is the blocking of a key exchange. Due to the complexity of the timeout system, it would be beneficial to include an extra layer of defense against this possibility. To protect against an attack taking advantage of the timeout-matrix causing the keys to not be changed for a long time, each honest client could set a TTL (time-to-live) on keys. If a new key exchange has not occurred by the TTL limit, the client disconnects from the conversation. This decreases the maximum impact of the denial of service attack to destroying the conversation, rather than allowing the conversation to continue in a compromised state.

## Implement a Symmetric Key Ratchet

Currently the protocol provides forward security from the get-go (conversation start) and periodically when members choose to perform a ratcheting step. NCC Group and the library recommend that this be performed every 60 seconds. If the ratcheting step is not performed (because the particular chat client does not enforce it or the ratchet request is somehow blocked), forward secrecy within the conversation wanes. If the conversation were to continue without ratcheting and a user's keys were disclosed, all contents of the conversation since the previous ratcheting step would be revealed. By introducing a symmetric ratcheting step, this can be circumvented: each message sent will be encrypted and the symmetric key used to encrypt said message will be rotated. Typically this rotation will be a HMAC-based Key Derivation Function (HKDF) operation: a one-way transformation. This way, when a user's secret keys are revealed, previous messages are not revealed (of course subsequent messages will be), adding to the chat's forward secrecy.

This of course is complicated in the group chat setting; here every member shares the same key. If two users encrypt and send a message at the same time (after which they "forget" the encryption key), then those two users will not be able to decrypt said message. By deriving individual keys for each party, the possibility of dropping messages as described is impossible.

Consistency can be maintained because:

- When sending a message, each member encrypts said message with their own derived key.
- Each other user will receive the exact same ciphertext.
- Each other user will have knowledge of the sending user's encryption key.

# Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 14.

| Title | ID | Risk |
|---|---|---|
| Conversation Key Reuse Defeats Deniability | 001 | Low |
| Potential for Weak Key Generation | 002 | Informational |
| Carrier Room Deniability Loss | 003 | Informational |

| | |
|---|---|
| **Finding** | **Conversation Key Reuse Defeats Deniability** |
| **Risk** | **Low** Impact: Medium, Exploitability: Low |
| **Identifier** | NCC-np1sec-2017-001 |
| **Category** | Cryptography |
| **Location** | • New member joining a conversation – Section 5.3.1, page 23 of protocol.pdf<br>• Member leaving a conversation – Section 5.4.3, page 40 of protocol.pdf |
| **Impact** | When a new member joins a conversation, previous messages sent within the group chat could become undeniable to the joining member. |
| **Description** | The (n+1)sec protocol is susceptible to an attack, described below, that removes deniability of previous messages when a new member joins.<br><br>Suppose Alice and Bob are in a conversation. Alice makes some statements. Then Bob invites Charlie. Alice wants to deny to Charlie the statements she has made so far. The specification states that when the invitation process kicks off, each member, including Alice, will automatically issue an authentication-confirmation message to Charlie using their conversation key pair (i.e. $K_A$ for Alice), asserting they know the private key. Alice's conversation key is the same conversation key that was used to sign previous statements between Alice and Bob before Charlie joined. Although Charlie still needs the old session key to decrypt the previous messages, since the session key changes when he joins, Charlie can obtain the old session key from Bob. Now when Charlie looks at the decrypted plaintext, he can verify that they are signed with $K_A$. See Appendix B on page 16 for a diagram of this attack.<br><br>This issue also applies to the situation where Alice, Bob, and Charlie are in a conversation, then Charlie leaves. Now any statements Alice makes with the conversation key that she used in an authentication-confirmation with Charlie will be undeniable to Charlie. Although Charlie no longer has the shared secret, Bob can decrypt the conversation for Charlie.<br><br>The (n+1)sec specification document does not explicitly state that the conversation keys are reused when members are added or removed from the conversation. However, when asked for clarification, the (n+1)sec developer confirmed that the conversation keys are reused when members enter or leave the conversation.<br><br>The issue is worsened by the fact that Alice cannot prevent Bob from inviting Charlie. No member of a conversation can prevent another member from inviting whoever they want. |
| **Recommendation** | Refresh conversation keys when new members join. Alternatively, implement tighter controls on invitations. For example, a member might be granted veto power over invitations. A hierarchy of control could be constructed from the tree of invitee/inviter relationships.<br><br>For the sake of performance when a new member joins, members could sign with both their old and new conversation keys. They can exchange authorization confirmation messages with the invitee first, since that is the only user who cannot validate the old conversation keys – in fact, they are specifically prevented from doing so. This allows the conversation to continue to function while conversation keys are being refreshed, all while maintaining the deniability properties of statements that users would expect. When signing with two conversation keys, make sure not to sign the other signature, and to only sign the message contents, so that the association of old and new conversation keys is deniable to users that have not received authentication confirmation messages for both keys. |

**Finding** **Potential for Weak Key Generation**

**Risk** **Informational** Impact: High, Exploitability: None

**Identifier** NCC-np1sec-2017-002

**Category** Cryptography

**Location**

- https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/src/conversation.cc#L36
- https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/src/conversation.cc#L58
- https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/src/encryptedchat.cc#L98
- https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/src/keyexchange.cc#L29
- https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/src/room.h#L60

**Impact** Weak keys could be generated, potentially allowing key recovery, providing an attacker the ability to modify messages or decrypt the contents of chats.

**Description** The (n+1)sec implementation uses the libgcrypt[14] library for all cryptographic operations, including the generation of secret keys. The following method[15] is used by the (n+1)sec to generate room, conversation, and session keys:

```
141 PrivateKey PrivateKey::generate(bool transient)
142 {
143   const char* parameter_string;
144   if (transient) {
145     parameter_string = "(genkey (ecc (curve Ed25519) (flags eddsa
          transient-key)))";
146   } else {
147     parameter_string = "(genkey (ecc (curve Ed25519) (flags eddsa)))";
148   }
```

According to the libgcrypt documentation,[16] the `transient-key` option is described as:

> This flag is only meaningful for RSA, DSA, and ECC key generation. If given the key is created using a faster and a somewhat less secure random number generator. This flag may be used for keys which are only used for a short time or per-message and do not require full cryptographic strength.

Note that every example located in the Location field above calls the function with the value `True`, using the `transient-key` for all key generation. NCC Group did not investigate the quality of the libgcrypt random number generator and the precise effects of the `transient-key` option.

**Recommendation** Do not use the `transient-key` option.

[14] https://gnupg.org/software/libgcrypt/index.html
[15] https://github.com/equalitie/np1sec/blob/44d7b2887a536d01f138b6ad43945a4b4a4b0c63/src/crypto.cc#L141
[16] https://www.gnupg.org/documentation/manuals/gcrypt/Cryptographic-Functions.html

| | |
|---|---|
| **Finding** | **Carrier Room Deniability Loss** |
| **Risk** | **Informational** Impact: Low, Exploitability: Low |
| **Identifier** | NCC-np1sec-2017-003 |
| **Category** | Cryptography |
| **Location** | Room Authentication Request – Section 6.3.3, page 43 of protocol.pdf |
| **Impact** | A malicious user can prove membership within a carrier chat room to a third party, breaking minor deniability claims. Individual chat rooms within a carrier chat do maintain their membership deniability. |
| **Description** | When joining a carrier chat room, the overarching structure in which secure chats are created, a room public key is used to authenticate members. This room public key is an artifact of a previous protocol design and is not used elsewhere besides this initial authentication. This public key is authenticated during the carrier room joining process, using the process outlined in Group Key Exchange and Key Authentication on page 8, with the `ROOM_AUTHENTICATION_REQUEST` and `ROOM_AUTHENTICATION` messages. This process provides an avenue for a malicious user to defeat carrier chat room participation deniability claims. The following attack details this:<br>1. Mallory, a malicious user who wishes to prove that Alice and Bob are member of a carrier chat room, joins the room.<br>2. The invitation process described in section 6.3.3 and 6.3.4 of the protocol specification is kicked off.<br>3. At step 4 of the invitation process, the invitee will send a `ROOM_AUTHENTICATION_REQUEST` message to each participant (and likewise each participant sends one to the invitee). This request includes the room public key of the invitee and the public keys of the targets (Alice in one message and Bob in another). Mallory will choose a room public key in which she obviously does not know the corresponding private key (e.g. a key with hex encoding 'deadbeefdeadbeef…').<br>4. Alice and Bob will automatically respond with `ROOM_AUTHENTICATION` messages, performing the operation outlined in Group Key Exchange and Key Authentication with Mallory's long-term public key and room public key.<br>5. If Alice (or Bob's) long-term private key and room private key are compromised, the compromiser can now ascertain that Alice (or Bob) have been present within the carrier chat room as they are the only party who could create a valid `ROOM_AUTHENTICATION` response. |
| **Recommendation** | Because carrier chat room messages do not use the room public key, simply removing them will fix this issue. |

# Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

## Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

### Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

| | |
|---|---|
| **Critical** | Implies an immediate, easily accessible threat of total compromise. |
| **High** | Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach. |
| **Medium** | A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application. |
| **Low** | Implies a relatively minor threat to the application. |
| **Informational** | No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding. |

### Impact

Impact reflects the effects that successful exploitation upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

| | |
|---|---|
| **High** | Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level. |
| **Medium** | Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information. |
| **Low** | Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security. |

### Exploitability

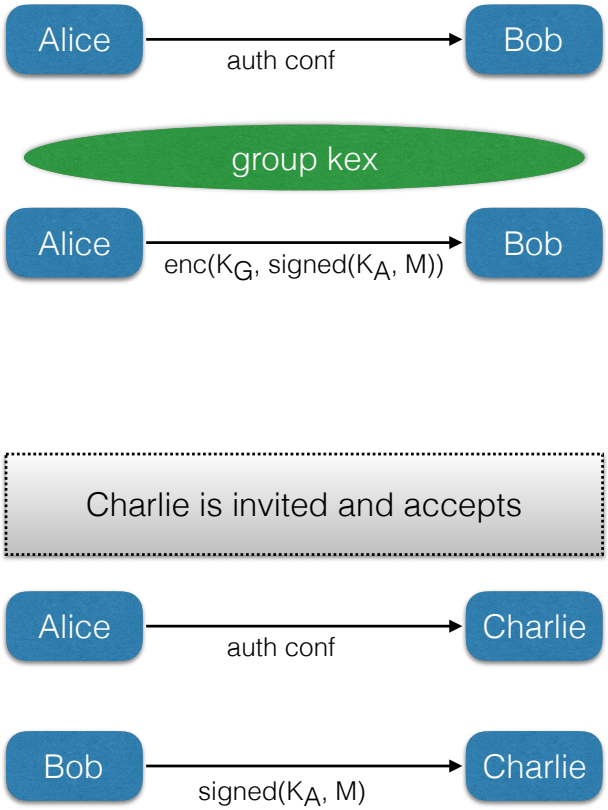
Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

| | |
|---|---|
| **High** | Attackers can unilaterally exploit the finding without special permissions or significant roadblocks. |
| **Medium** | Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding. |
| **Low** | Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely. |

## Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

| | |
|---|---|
| **Access Controls** | Related to authorization of users, and assessment of rights. |
| **Auditing and Logging** | Related to auditing of actions, or logging of problems. |
| **Authentication** | Related to the identification of users. |
| **Configuration** | Related to security configurations of servers, devices, or software. |
| **Cryptography** | Related to mathematical protections for data. |
| **Data Exposure** | Related to unintended exposure of sensitive information. |
| **Data Validation** | Related to improper reliance on the structure or values of data. |
| **Denial of Service** | Related to causing system failure. |
| **Error Reporting** | Related to the reporting of error conditions in a secure fashion. |
| **Patching** | Related to keeping software up to date. |
| **Session Management** | Related to the identification of authenticated users. |
| **Timing** | Related to race conditions, locking, or order of operations. |




1. Alice confirms her conversation-public-key to Bob.
2. A symmetric encryption key, $K_G$, is generated and shared amongst participants.
3. Alice sends a sensitive message signed with her conversation-private-key $K_A$ and encrypted with the symmetric encryption key using AES-GCM. Bob can decrypt the message since he participated in the group key exchange.
4. Charlie is invited and accepts.
5. Alice confirms her conversation-public-key to Charlie.
6. Bob sends the decrypted, signed message from Alice to Charlie. Since Alice has confirmed her conversation-public-key to Charlie, Charlie can verify that she authored the message $M$. Had Alice generated a new conversation-public-key when Charlie joined, Charlie would not be able to verify that the key used to sign message $M$ belongs to Alice.