Olm Cryptographic Review

Roomys

November 1, 2016 – Version 2.0

Prepared for
Matthew Hodgson
Richard van der Hoff

Prepared by
Alex Balducci
Jake Meredith

Document Change Log

Version	Date	Change
1.0	2016-10-04	Initial report released to Olm
1.1	2016-10-13	Updated report with additional findings
1.5	2016-10-22	Updated report with remediation details
2.0	2016-11-01	Public release

Executive Summary

Synopsis

In September 2016, Matrix, along with financial support from the Open Technology Fund,[1] engaged NCC Group's Cryptography Services Practice to perform a targeted review of their cryptographic library Olm. The review covered two major components of the Olm library: the double ratchet used for peer-to-peer communications, and Megolm, the group ratcheting mechanism. Matrix has produced several reference implementations that make use of the Olm library including the client-server SDK for JavaScript, matrix-js-sdk.[2] Matrix-js-sdk was *not* reviewed during the engagement; however, certain remediations to issues were applied to this implementation and *not* Olm.

The review covered the 1.3.0 release of the Olm library. Two consultants performed the engagement over a span of two weeks (September 19 to September 30, 2016) and consisted of 15 person-days of effort. A follow-up review of fixes was performed over the latter half of October.

Scope

NCC Group's evaluation focused on issues specific to double ratchets used in secure messaging applications, general cryptographic concerns, and potential vulnerabilities introduced by the C programming environment. Specific targets included:

- Olm Double Ratchet: the main construction providing a secure, deniable method for exchanging keys.
- Olm API: the API used by applications who wish to harness the double ratchet.
- Megolm Rachet: the main cryptographic construction used for group messaging.
- Input Entry Points: fuzzing of entry points into the library as well as pickling and unpickling functions.

Key Findings

NCC Group identified several issues that could undermine the integrity of Olm and Megolm. Noteworthy attacks include:

- Unknown Key-Share Attacks. Both Olm and Megolm suffer from variations of unknown key-share attacks. This attack can allow a user or group of users to conspire to confuse other users about who they are actually speaking with.
- Messages in Group Chats Can be Replayed. A group chat member, or an attacker with access to encrypted chat messages, can replay messages from any user in a chat. This may lead to undesired business, personal, or ethical implications for the user whose messages are being replayed.
- Future and Backward Secrecy Concerns. While group chats are effectively protected by ephemeral keys, these chats may be long lived. Compromise of these keys can lead to the decryption of past and future messages.

Caveats

The original scope of the engagement was solely based on reviewing the Olm cryptographic ratchet. After the first week of the assessment (five person-days), Matrix informed NCC Group that Megolm should take precedence over Olm as it is the primary mechanism to be defined in the Matrix specification. The remainder of the engagement was spent reviewing Megolm with some additional time devoted to Olm. While the engagement focused less on Olm than originally expected, NCC Group does not believe that Olm's coverage in the allotted time was adversely affected.

Test Details & Strategic Recommendations

NCC Group has detailed main portions of the testing plan and observations in Test Details on page 6.

During the course of the engagement, NCC Group observed that the following measures may enhance the Olm library's long-term security posture:

- Perform an Additional Review of Megolm. Several issues were found in Megolm after a short review period. These issues need to be addressed in order for group messengers utilizing Megolm to have similar guarantees as one-on-one messages with Olm. Once these changes are made to the Megolm protocol, there should be additional review to verify that the fixes have been implemented securely.
- Add Additional Identity Information into Pre-Key Messages in Olm and Megolm. Providing additional information in the initial message between users prevents unknown key-share attacks. The additional information can help users and applications know that they are communicating with the intended user. This information can be any item that uniquely identifies a user, such as user ID, phone number, or address. Privacy issues should be accounted for in this solution as to not give out too much personal information.

[1] https://www.opentech.fund/
[2] https://github.com/matrix-org/matrix-js-sdk

See Strategic Recommendations on page 9 for more details.

Remediation Review

Matrix began remediating issues after the initial report was delivered to them. Remediation results have been identified and included in each individual finding and the dashboard updated to reflect the one issue that has been fully addressed.

Matrix has addressed the majority of the findings in their higher-layer protocol matrix-js-sdk and not in Olm. For a majority of these issues addressing them within Olm would add undue complexity to the library and higher layer protections were more apt for the fixes. Because of this, Matrix has clearly documented the shortcomings in their end-to-end encryption guide[3] and the Olm specification[4] as well as stating what developers must to do protect against these various issues.

[3] https://matrix.org/docs/guides/e2e_implementation.html
[4] https://matrix.org/docs/spec/olm.html

Dashboard

Target Metadata

Name	Olm
Type	Cryptographic Library
Platforms	C/C++
Environment	Tag 1.3.0

Engagement Data

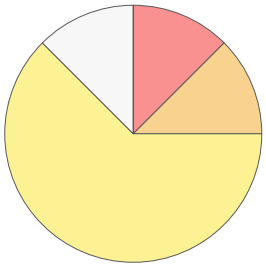
Type	Cryptographic Review
Method	Source Review
Dates	2016-09-19 to 2016-09-30
Consultants	2
Level of effort	15 person-days

Targets

Olm 1.3.0 Release	https://matrix.org/git/olm/tag/?h=1.3.0

Finding Breakdown

Original Assessment		Remaining	
Critical Risk issues	0	Critical Risk issues	0
High Risk issues	1	High Risk issues	1
Medium Risk issues	1	Medium Risk issues	1
Low Risk issues	6	Low Risk issues	5
Informational issues	1	Informational issues	1
Total issues	9	Total issues	8



Category Breakdown

Cryptography	8	

Component Breakdown

Megolm	5	
Olm	3	

Key

Critical High Medium Low Informational

In this section we discuss various cryptographic details of the Olm library and NCC Group's assessment of it. The Olm ratchet is based off of the work of Trevor Perrin and Moxie Marlinspike as detailed in https://github.com/trevp/double_ratchet/wiki. The Megolm ratchet is partially detailed in https://matrix.org/docs/spec/megolm.html.

Cryptographic Keys and Authentication

Cryptographic keys are used for two major reasons in a cryptographic chat protocol: to provide authentication of a peer and to provide confidentiality and integrity of messages. Both Olm and Megolm share the following sets of keys:

- A long-term signing and identity fingerprinting Ed25519 key.
- A long-term identity Curve25519 key.

While not in scope in this engagement, users submit their Ed25519 key and Curve25519 key, signed by their Ed25519 key, to a central server. When a user wishes to engage a peer, they first download a bundle containing the peer's Ed25519 key, Curve25519 key, and signature. Users must then out-of-band verify the Ed25519 key fingerprint of their peer, which happens in the application layer (Matrix). This out-of-band verification provides authentication of peers.

Confidentiality is provided by AES256 in CBC mode. Authenticity is provided by HMAC-SHA256.

Olm

In addition to the above, the Olm ratchet makes use of ephemeral, one-time Curve25519 keys. These keys are published on a central server and used to facilitate asynchronous messaging. For example, Alice can grab one of Bob's published keys and perform a Triple Diffie-Hellman (Triple DH) handshake, all without Bob's input. When Bob eventually logs in and receives a message from Alice, he can complete his own Triple DH and decrypt the message. Authentication is inherently built into the Triple DH operation.

NCC Group discovered the Olm protocol is vulnerable to an unknown key-share attack, which weakens the authentication between users. More information can be found in finding NCC-Olm2016-009 on page 23.

Megolm

The Megolm ratchet makes use of an ephemeral, one-time Ed25519 key. This ephemeral key is used to sign messages within a single group chat setting, allowing the peers to verify the origin of each message.

NCC Group discovered the Megolm protocol is also vulnerable to an unknown key-share attack but from the perspective of a group of colluding users in a single chat room. It weakens the authentication of users across group chat. More information can be found in finding NCC-Olm2016-010 on page 11.

Forward Secrecy

Forward secrecy is a critical aspect of secure messaging applications: if a long-term private key is revealed, previous messages sent should not be compromised. Below we discuss forward secrecy over the lifetime of all conversations as well as forward secrecy within a particular conversation.[5]

Olm

The Olm protocol makes use of a Triple DH exchange, per-conversation, to create shared secrets for peer-to-peer communication. In Triple DH, the following operations are performed:

1. Diffie-Hellman (DH) between Alice's long-term Curve25519 key and Bob's ephemeral Curve25519 key.
2. DH between Alice's ephemeral Curve25519 key and Bob's long-term Curve25519 key.
3. DH between Alice's ephemeral Curve25519 key and Bob's ephemeral Curve25519 key.

Because the resultant shared secret involves ephemeral keys, forward secrecy across conversations is conserved.

Additionally, the pseudo-random function (PRF) used during the double ratchet ensures that a compromise of a particular chain key, used to later derive symmetric encryption keys, does not reveal previous chain keys. This property

[5] For this case we focus on the compromise of per-conversation symmetric encryption keys instead of long-term keys.

ensures that forward secrecy across messages within a conversation is conserved.

Megolm

The standalone Megolm protocol session sharing message is, in essence, a transfer of a symmetric ratchet key[6] between peers where the key is unique for this particular group chat instance. The Megolm protocol relies on Olm to distribute these symmetric keys between peers. Because Olm exhibits forward secrecy in this regard, Megolm ensures that forward secrecy is kept across conversations.

Forward secrecy within a conversation is not as straight forward; Megolm makes use of a PRF to ratchet the symmetric key, thus under ordinary circumstances the exposure of said key does not compromise previous messages. However, each member of a group chat will permanently "remember" the initial symmetric key, which is never ratcheted forward.[7] If this initial symmetric key is revealed then messages previously sent may be decrypted by simply ratcheting the key forward. To relieve some of these forward secrecy concerns, higher-layer protocols should periodically refresh these symmetric keys (see Strategic Recommendations on page 9). In particular, when new users join or existing members leave, the ratcheting keys should be "re-negotiated". These higher-layer protocols also have the choice of balancing security and usability in their hands: the more they refresh the keys (adding security) the less usability is given to the user (the ability to decrypt previous messages without storing them locally). It is up to these applications to balance these items.

Backward Secrecy

Backward secrecy is a critical aspect of secure messaging applications: if a long-term private key is revealed, future messages sent should not be compromised. Below we discuss backward secrecy over the lifetime of all conversations as well was backward secrecy within a particular conversation.[8]

Olm

As discussed in the Forward Secrecy section above, Olm makes use of a Triple DH exchange, which will also provide backward secrecy across conversations.

Additionally, Olm institutes a double ratchet in order to provide the self-healing property of backward secrecy. Any encryption key obtained during an Olm session can only be used to decrypt that one message. As long as only one person is sending messages, the keys for future messages will still be available as it is an HMAC of the previous key with a static value. As soon as the second person replies, a new DH key exchange will occur, which won't allow the attacker to access the new keys and messages. This essentially "heals" the leaking of one encryption key so that it does not allow all future messages to be decrypted, providing backward secrecy within a conversation.

Megolm

As discussed in the Forward Secrecy section above, Megolm makes use of a symmetric ratchet key unique per-conversation sent via Olm. This ensures backward secrecy across conversations.

The Megolm protocol has a single session state that is exchanged for all members of the group. This is used as the encryption key and is ratcheted forward via an HMAC mechanism with a static key for every additional message. Because the Megolm protocol does not include the healing properties that Olm provides, it does not stop a user from being able to obtain all future keys given the session state; hence, it does not exhibit the backward secrecy property within a conversation.

As discussed in the Forward Secrecy section above, it is up to the higher-layer protocol to periodically refresh this session state to enable backward secrecy protections.

[6] In reality this symmetric key is more aptly named 'initial ratchet state'.
[7] This is designed to enable users to decrypt previous messages from the group chat without having to store every message locally.
[8] For this case we focus on the compromise of per-conversation symmetric encryption keys instead of long-term keys.

Deniability

A desirable aspect of secure messaging applications is deniability: a user can authenticate their peer and messages sent from them without being able to cryptographically prove, at a later date, that the content of those messages were genuinely sent by said peer. Additionally, deniability may or may not include the ability to deny that a conversation ever occurred with a specific user.

Olm

The Olm library implements Triple DH, which provides deniability because the ephemeral keys are published, allowing anyone to potentially use them as identity keys. The possession of a signed (or unsigned) ephemeral key provides no evidence of a relationship.

Megolm

The initial Megolm session sharing messages are signed with an Ed25519 key; however, this key is ephemeral for this particular group chat session. Any user can create this session sharing message for any other user. These messages should be sent via Olm to authenticate the actual user. Because of this, Megolm group chat messages are deniable.

Consistency in the Megolm protocol

As currently designed, the Megolm protocol does not exhibit consistency: any user can send differing messages to the group chat members with help from a colluding central message passing server. In general, consistency in group messages is an open problem in multi-party encrypted chat. To give an example of one solution to this problem, multi-party OTR[9] (mpOTR) uses a consistency check at the end of a group chat. To facilitate this, every message of the chat is hashed together and hashes are compared at the end of the protocol. While this will allow users the ability to discover consistency issues in a protocol, the mpOTR design has issues, including:

1. Undelivered messages will naturally cause participants to have different views of the transcript.
2. Attackers may subvert the exchange of transcript hashes, disallowing the discovery of consistency issues.
3. Revelation of transcript inconsistency at the end of a conversation does not make for a good user experience.

These trade-offs should be carefully considered, and any consistency or usability shortcomings of the final protocol should be documented clearly.

Defensive Coding and Fuzzing Efforts

While the focus of the engagement was on cryptographic issues, NCC Group also examined the codebase for native code issues. In general, NCC Group found the Olm and Megolm code to be well written and without many of the common secure coding issues. Additionally, NCC Group extended Olm's existing fuzzing suite, using the American Fuzzy Lop[10] fuzzer, to target various functions that include user input. One of these functions (`_olm_decode_group_message`) was found to have a minor defect during fuzzing as detailed in finding NCC-Olm2016-011 on page 19.

[9] https://www.cypherpunks.ca/~iang/pubs/mpotr.pdf
[10] http://lcamtuf.coredump.cx/afl/

Olm:

- Consider signing ephemeral Olm keys. The basic version of Triple Diffie-Hellman (in which ephemeral keys are merely published as is) is susceptible to weak forward secrecy in certain cases, particularly if one member of the chat exchange is offline. If Bob and Alice are exchanging messages, Bob signing his ephemeral key prevents that key from being forged. Even though these ephemeral keys are signed by long-term identity keys, deniability is kept if they are published.
- The MAC of the initial Olm message should contain additional uniquely identifiable information. Without this unique additional data (user ID, phone number, address, etc.), there is an unknown key-share attack in the Olm protocol. This can allow an attacker (Mallory) the ability to dupe Bob into believing he is speaking with Alice when he is actually speaking with Mallory.

Megolm:

- Perform additional review of Megolm. Several issues were found in Megolm after a short review period. These issues need to be addressed in order for group chat using Megolm to have similar guarantees as one-on-one messages with Olm. Once these changes are made to the Megolm protocol, there should be additional review to verify the fixes have been implemented securely.
- Ensure the Megolm library's session sharing message is performed over a secure channel. Because the Megolm session sharing message is in essence a static key exchange, the confidentiality of the message is pivotal; anyone who is in possession of this message can encrypt future group chat messages. Matrix currently shares this message via the secure channel Olm.
- Periodically refresh Megolm keys. Because Megolm group chats have weak forward and backward security, keys should be refreshed periodically. Matrix currently refreshes keys when group chat members enter and leave conversations.
- Clearly document the shortcomings and assumptions of the Megolm library functionality. Certain functionality, such as the ability to decrypt past chat messages in a group chat, is by design; however, this undermines forward secrecy. All privacy violating design choices should be clearly articulated in the specification and documentation.

General Recommendations

- Third parties should erase randomness buffers. The Olm library requires that calling users provide their own source of randomness. In some cases this randomness is used to create ephemeral secret keys. Developers should securely erase all random buffers to ensure secret key material is not leaked.
- Consider replacing existing cryptographic libraries with side-channel resistant libraries. While NCC Group did not perform a full-fledged review of the cryptographic libraries in use by the Olm library, the AES primitives used appear to not be side-channel resistant. This may lead to the recovery of secret keys used to encrypt messages.

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 26.

The astute reader will notice that findings "008" and "007" are missing. These were false positives identified during the engagement. "007" made the erroneous assumption that the Megolm session sharing message was signed by a user's long-term identity key, which it is not. "008" detailed the fact that signatures are replaceable on the Megolm session sharing message, but because an attacker would not be in the position to swap in their own ephemeral key (this message should be sent securely via Olm), the finding was omitted.

Megolm

Title	Status	ID	Risk
Unknown Key-Share Attack in Megolm	Not Fixed*	010	High
Group Chat Messages Replayable Within Groups	Not Fixed*	006	Medium
Lack of Transcript Consistency in Group Chat	Risk Accepted	003	Low
Lack of Backward Secrecy in Group Chats	Not Fixed*	004	Low
Weak Forward Secrecy in Group Chats	Not Fixed	005	Low
Improper Bounds Checking May Lead to Denial of Service	Fixed	011	Low

Olm

Title	Status	ID	Risk
Lack of Signature on Ephemeral Keys	Not Fixed*	001	Low
Unknown Key-Share Attack in Olm	Not Fixed*	009	Low
Pre-Key Message Contains Un-MACed Fields	Not Fixed	002	Informational

* While these findings are addressed in matrix-js-sdk, the Olm library does not address the issues.

Finding	**Unknown Key-Share Attack in Megolm**
Risk	**High** Impact: High, Exploitability: Medium
Identifier	NCC-Olm2016-010
Status	Not Fixed (Matrix provides application-level mitigations in matrix-js-sdk)
Category	Cryptography
Component	Megolm
Location	The Megolm Protocol found at https://matrix.org/docs/spec/megolm.html#the-megolm-ratchet-algorithm.
Impact	A pair of attackers (Mallory & Donald) may dupe a user (Bob) into believing he is receiving messages from another user (Alice) when in fact Mallory & Donald are simply forwarding selected messages sent by Alice in a different group chat. In this case, Alice is participating in a group chat with Mallory & Donald; Bob believes he is speaking in a group chat with Alice.
Description	In this variant of the unknown key-share (OKs) attack, an attacker will extend finding NCC-Olm2016-009 on page 23 to allow highly targeted, known messages to be sent to Bob. In this scenario, Bob will still believe he is talking with Alice. Here, two parties (Donald and Mallory), who may be the same person, will collude against Alice in a group chat situation (Megolm). Donald will be performing the unknown key-share attack. Mallory will be an instigator (attempting to elicit messages from Alice that will later be sent to Bob) who will be able to read the contents of the group chat. 1. Donald publishes Bob's keys (his public long-term identity key and public ephemeral pre-keys) as his own and they all (Alice, Donald, Mallory) enter into a group chat. 2. Donald cannot read any of the messages, because he lacks the (Bob's) private keys. However, Mallory can, so she can sync up the ciphertext and plaintext of each of Alice's messages. 3. Mallory and Donald now have known plaintext/ciphertext pairs (thanks to Mallory) and a Megolm session sharing message designed for Bob (thanks to Donald). 4. Donald can perform the UKS attack on Bob with the Megolm session sharing message. This occurs in the same manner as finding NCC-Olm2016-009 on page 23. 5. Bob will think this message is coming from Alice and may enter into the group conversation. 6. Donald can then send the appropriate ciphertext depending on the plaintext message he wishes to send (using the list from Mallory). 7. While Donald cannot directly decrypt responses from Bob, this is a group chat. Mallory may have been invited and can relay Bob's messages to Donald.
Recommendation	There are two ways in which this may be fixed: 1. **Uniquely** identifying information (user ID, email address, phone number, etc.) can be MACed into the initial message Alice sends. The ID info for each party would be included and verified by each peer when a message is received. This information must be provided by a higher level protocol. 2. A stronger method would be to prove possession of the private key corresponding to the

long-term identity key. One method would be to perform a challenge-response or other zero-knowledge protocol during the initial fingerprint verification.

Retest Results

Because these messages are sent via Olm, the fix to finding NCC-Olm2016-009 on page 23 addresses this issue in matrix-js-sdk as well.

While this is fixed in matrix-js-sdk, the Olm library does **not** address the issue.

Finding	**Group Chat Messages Replayable Within Groups**
Risk	**Medium** Impact: Medium, Exploitability: Medium
Identifier	NCC-Olm2016-006
Status	Not Fixed (Matrix provides application-level mitigations in matrix-js-sdk)
Category	Cryptography
Component	Megolm
Location	The Megolm protocol found at https://matrix.org/docs/spec/megolm.html#the-megolm-ratchet-algorithm.
Impact	A replayed message in a group chat could affect the integrity of a group chat session and potentially lead to undesired business, personal, or ethical implications for the user whose messages are being replayed.
Description	As detailed in finding NCC-Olm2016-005 on page 17, each participant in a group chat holds two ratchet states for each peer in a conversation: • An `initial_ratchet` state: a static state representing the first ratchet state known to a peer. • A `latest_ratchet` state: a modified ratchet representing the state of the last message the peer has received. The `initial_ratchet` is kept for design reasons: messages can be stored encrypted on a server and pulled by a peer at any time. This allows the subsequent decryption of past messages. Because of this, a man-in-the-middle attacker or nefarious peer in the group chat, can intercept a message and replay it to the members of the group. The message being sent must have an index between the first message received by a user in the group and the most recent message in the group, in order to be successfully decrypted. In the case of a man-in-the-middle attacker, the content of the message would be unknown, so this limits the ability for them to use the message maliciously. However, if the message were to contain a generic response or perhaps an actionable request, it could potentially lead to undesired behavior for other members of the group. In the more problematic case of a nefarious group member, the content of the message would be known, allowing highly targeted replays that have a higher risk of undesired implications for the victim user.
Recommendation	Several approaches may be taken, however, drawbacks exist for each. • Do not allow the decryption of messages past the current `latest_ratchet` index (i.e., remove the `initial_ratchet` structure). This would not allow the decryption of messages that are received out-of-order. • Keep a list of all ratchet indices seen. When an index seen before is received, the peer should somehow be notified that this particular message may have been replayed. This does not protect against a malicious group member who can delay the reception of messages. It should be noted that it is the intended functionality of the Megolm ratchet to allow the decryption of previous messages, limiting the applicability of the solutions listed above.

Retest Results

Matrix has opted to remember ratchet indices and indicate an error ("Duplicate message index, possible replay attack") if a duplicate message is received during a group chat session.[11] These indices are remembered until the client ever re-downloads historical messages in a group chat allowing them to decrypt previous messages (else they would receive errors as historical message ratchet indices have inherently already been recorded).

Documentation has been updated explaining the concept to developers.[12]

While this is fixed in matrix-js-sdk, the Olm library does **not** address the issue.

[11] https://github.com/matrix-org/matrix-js-sdk/pull/241
[12] https://github.com/matrix-org/matrix-doc/pull/414

Finding	**Lack of Transcript Consistency in Group Chat**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-Olm2016-003
Status	Risk Accepted
Category	Cryptography
Component	Megolm
Location	The Megolm Protocol found at https://matrix.org/docs/spec/megolm.html#the-megolm-ratchet-algorithm.
Impact	A user can send two different messages to two different users of a group. The responses to those queries could be misinterpreted by other members of the group.
Description	As detailed in finding NCC-Olm2016-005 on page 17, a ratchet state session sharing message is shared with all other members of the group chat (i.e., each peer receives the same ratchet state). Subsequent chat messages are encrypted using keys output from this shared ratchet state and are packaged into the following message: 1. Version of the protocol. 2. The current ratchet index. 3. The encrypted message. 4. A MAC covering the preceding items. 5. A signature from the current ephemeral key covering the entire message. Due to the nature of group chat in the Megolm protocol, there is currently no way for participants to know if they received the same message from a member of the group. If the keys and ratchet index provided with the message are valid then the user will decrypt the message and it will appear as though everyone in the chat received the same message. *Note* to perform this attack, the server must be working in collusion with the attacker so that different messages are actually routed to different users. An example: A group participant (Bob) initiates a conversation between three or more parties in the chat and sends a different proposal to different parties. Unaware, Bob's correspondents see the replies to his messages and obtain a false impression on their parties' intentions/responses.
Recommendation	Transcript consistency and message ordering (especially with non-fixed groups) is an open problem in multi-party encrypted chat. Ensure that this limitation and property is clearly documented.
Retest Results	As this is a hard problem to solve, Matrix has opted to make this limitation clear.[13]

[13] https://matrix.org/git/olm/commit/?id=df04cd509a13fb1a3c8b953de4f987b15fcde9b1

Finding	**Lack of Backward Secrecy in Group Chats**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-Olm2016-004
Status	Not Fixed (Matrix provides application-level mitigations in matrix-js-sdk)
Category	Cryptography
Component	Megolm
Location	The Megolm protocol found at https://matrix.org/docs/spec/megolm.html#the-megolm-ratchet-algorithm.
Impact	Any access to the group ratchet state would allow for decrypting any future group chat messages, weakening backward secrecy within a particular group chat.
Description	Once a group session is instantiated the base index for the ratchet is stored for the lifetime of the session. Due to the lack of self-healing properties, such as a double ratchet, once this ratchet state is obtained, all future ratchet states and messages are calculable. A self-healing property would, at defined points, re-establish the ratchet state to prevent this issue. The Olm protocol continuously performs DH operations for self-healing as the second ratchet in its double-ratchet algorithm.
Recommendation	Other double-ratchet protocols perform pairwise messaging between group participants for group messaging using their normal double-ratchet protocol. This would essentially mean that for any group, each pair of members of the group would communicate within their own Olm session and the application would handle the idea of a group as opposed to the protocol.
Retest Results	The Matrix matrix-js-sdk library will automatically refresh the Megolm keys after a certain amount of group chat messages or time has elapsed since the previous refresh (defaulting to 100 messages / one week).[14] In addition, the specification has been updated to call out backward secrecy concerns and how developers can alleviate some of these concerns.[15] While this is partially mitigated in matrix-js-sdk, the Olm library does **not** address the issue.

[14] https://github.com/matrix-org/matrix-js-sdk/pull/240
[15] https://matrix.org/git/olm/commit/?id=df04cd509a13fb1a3c8b953de4f987b15fcde9b1

Finding	**Weak Forward Secrecy in Group Chats**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-Olm2016-005
Status	Not Fixed
Category	Cryptography
Component	Megolm
Location	src/inbound_group_session.c:36
Impact	A compromise of a peer's `OlmInboundGroupSession` would lead to the decryption of previous messages, breaking forward secrecy.
Description	When receiving a group message from a peer for the first time, an `OlmInboundGroupSession` structure is created.[16]

```
struct OlmInboundGroupSession {
    /** our earliest known ratchet value */
    Megolm initial_ratchet;

    /** The most recent ratchet value */
    Megolm latest_ratchet;

    /** The ed25519 signing key */
    struct _olm_ed25519_public_key signing_key;

    enum OlmErrorCode last_error;
};
```

As detailed above, this structure holds the first ratchet state sent to the user in `initial_ratchet` and the current ratchet state in `latest_ratchet`. When subsequent messages are received, the `latest_ratchet` state will be ratcheted; `initial_ratchet` remains static for the entire length of the session. If messages received indicate a ratchet step in the past (i.e., before the current `latest_ratchet` state), a copy of the static `initial_ratchet` is used to decrypt the message.

While this design allows the decryption of previous messages and is intended by Matrix, it also allows an attacker the ability to decrypt an entire history of messages if they can recover the `OlmInboundGroupSession`.

Recommendation	To facilitate the decryption of missed and unordered messages, consider advancing the `initial_ratchet` up to the state of the first missing message. Continue advancing the `latest_ratchet`. After `latest_ratchet` has diverged a certain amount from `initial_ratchet`, consider any missing messages lost and advance `initial_ratchet` up to the current state. The "divergent value", the difference between the `latest_ratchet` and `initial_ratchet`, should be tuned to maintain some usability (the ability to decrypt skipped messages that were unreliably delivered) and security.
Retest Results	The specification has been updated to call out backward secrecy concerns and how developers can alleviate some of them.[17]

[16] https://matrix.org/git/olm/tree/src/inbound_group_session.c?h=1.3.0#n36
[17] https://matrix.org/git/olm/commit/?id=df04cd509a13fb1a3c8b953de4f987b15fcde9b1

Client Response Matrix may include a "seal" option that would remove all old session data (that is, the `initial_ratchet` keys for conversations). This may either be a specific action a user can take via a button press or a configurable, automated process.

Finding **Improper Bounds Checking May Lead to Denial of Service**

Risk **Low** Impact: Low, Exploitability: High

Identifier NCC-Olm2016-011

Status Fixed

Category Denial of Service

Component Megolm

Location src/message.cpp

Impact An attacker can send a malicious group chat message that may lead to a denial of service.

Description The `_olm_decode_group_message` function is used in two places when receiving a Megolm group chat message:

- Reporting back the size of the underlying plaintext message (when called by `olm_group_decrypt_max_plaintext_length`).
- Returning the plaintext (when called by `olm_group_decrypt`).

`_olm_decode_group_message` parses out the tag-length-value (TLV) structure in the received message, ignoring (for now) the MAC and signature attached to the message. To facilitate this, some setup is performed before the parsing occurs:

```
void _olm_decode_group_message(
    const uint8_t *input, size_t input_length,
    size_t mac_length, size_t signature_length,
    struct _OlmDecodeGroupMessageResults *results
) {
    std::uint8_t const * pos = input;
    std::size_t trailer_length = mac_length + signature_length;
    std::uint8_t const * end = input + input_length - trailer_length;
    std::uint8_t const * unknown = nullptr;

    bool has_message_index = false;
    results->message_index = 0;
    results->ciphertext = nullptr;
    results->ciphertext_length = 0;
    ...
```

Here `mac_length` and `signature_length` combine to a static size of 72 bytes. `end` will point to the last byte of the message minus the length of the MAC and signature (i.e., the end of the TLV structure).

Continuing where we left off:

```
    if (input_length < trailer_length) return;
    results->version = *(pos++);

    while (pos != end) {
       pos = decode(
            pos, end, GROUP_MESSAGE_INDEX_TAG,
            results->message_index, has_message_index
        );
        pos = decode(
            pos, end, GROUP_CIPHERTEXT_TAG,
```

```
            results->ciphertext, results->ciphertext_length
        );
        if (unknown == pos) {
            pos = skip_unknown(pos, end);
        }
        unknown = pos;
    }
```

After ensuring that the input length is at least the size of the MAC and signature, the version of the message is parsed from the TLV structure and the current position point pos is advanced. Finally, the loop that will iteratively parse out the appropriate items from the TLV message is entered, skipping tags it does not recognize.

If an attacker sends a message that is exactly 72 bytes (the size of the MAC and signature), right before entering the loop end will point at a location in memory 1 byte *before* pos. The decode functions will parse out the particular data for that tag and advance pos forward. If no tags are found, the skip_unknown function will be called.

```
static std::uint8_t const * skip_unknown(
    std::uint8_t const * pos, std::uint8_t const * end
) {
    if (pos != end) {
        uint8_t tag = *pos;
        if ((tag & 0x7) == 0) {
            pos = varint_skip(pos, end);
            pos = varint_skip(pos, end);
        } else if ((tag & 0x7) == 2) {
            pos = varint_skip(pos, end);
            std::uint8_t const * len_start = pos;
            pos = varint_skip(pos, end);
            std::size_t len = varint_decode<std::size_t>(len_start, pos);
            if (len > std::size_t(end - pos)) return end;
            pos += len;
        } else {
            return end;
        }
    }
    return pos;
}
```

As seen in the skip_unknown function, pos is advanced if the byte it points at bitwise AND seven is zero or two. In all other cases, the pointer end is returned. In this else case, the loop will finally be terminated.

Because of this, the loop will keep running until the byte pointed at by pos is not zero or two after the bitwise AND operation. It should be noted that the byte pointed at by pos is outside the original length of the buffer in this case.

Recommendation Before advancing the pointer pos after parsing the version, ensure that pos is not equal to end. Additionally, consider ensuring that input_length is at least the size of the MAC, signature, version, GROUP_MESSAGE_INDEX_TAG structure, and zero-length message GROUP_CIPHERTEXT_TAG structure.

Retest Results Commit 1ff64391edf9f2e3180238271858698a5a6f30c6,[18] partially shown below, fixes the issue by including the equality check as described in the Recommendation section of this

[18] https://matrix.org/git/olm/commit/?id=1ff64391edf9f2e3180238271858698a5a6f30c6

finding.

```
     if (input_length < trailer_length) return;
+
+   if (pos == end) return;
     results->version = *(pos++);

     while (pos != end) {
+        unknown = pos;
          pos = decode(
```

Finding	**Lack of Signature on Ephemeral Keys**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-Olm2016-001
Status	Not Fixed (Matrix provides application-level mitigations in matrix-js-sdk)
Category	Cryptography
Component	Olm
Location	The Olm Protocol
Impact	A man-in-the-middle attacker or dishonest server may swap out ephemeral keys for a peer. If the peer's long-term identity key is later exposed, past messages may be decrypted.
Description	When bootstrapping the ratchet, Olm performs a so-called 'Minimal Triple Diffie-Hellman' to first generate the shared secrets. In this Triple DH operation, a long-term Curve25519 static identity key and a Curve25519 ephemeral key are exchanged between each parties (in reality, these ephemeral keys are published to a central server and the initiating peer will retrieve one when starting a conversation). After this exchange, a DH operation is performed between the following keys: 1. DH Alice Identity, Bob Ephemeral 2. DH Alice Ephemeral, Bob Identity 3. DH Alice Ephemeral, Bob Ephemeral Because these ephemeral keys are not signed by a peer's long-term identity key, an attacker or dishonest server could replace them with ephemeral keys for which they know the private key. In this case, the initiator can still send messages to the recipient, although the recipient will not be able to decrypt them. At a later date, if the recipient's long-term key is compromised, the attacker can retroactively decrypt these (one-sided) conversations.
Recommendation	Users should sign their ephemeral keys before publishing them to the central server. In this case, an attacker or dishonest server will not be able to replace these keys with known versions, preventing retroactive decryption.
Retest Results	Matrix has fixed the issue in matrix-js-sdk by signing the ephemeral keys before submitting them to the server and verifying the signatures when receiving an ephemeral key.[19] Documentation has been updated detailing the necessary changes developers should make.[20] While this is fixed in matrix-js-sdk, the Olm library does **not** address the issue.

[19] https://github.com/matrix-org/matrix-js-sdk/pull/243
[20] https://github.com/matrix-org/matrix-doc/pull/416

Finding	**Unknown Key-Share Attack in Olm**
Risk	**Low** Impact: Low, Exploitability: Medium
Identifier	NCC-Olm2016-009
Status	Not Fixed (Matrix provides application-level mitigations in matrix-js-sdk)
Category	Cryptography
Component	Olm
Location	The Olm Protocol
Impact	An attacker (Mallory) may dupe a user (Bob) into believing he is speaking with another user (Alice) when in fact Mallory is simply forwarding messages (originally intended for Mallory) from Alice to Bob. In this case, Alice believes she is speaking with Mallory and Bob believes he is speaking with Alice.
Description	The Olm protocol is subject to an unknown key-share (UKS) attack as follows: Mallory wishes to send a message to Bob from Alice. In the attack, Bob will believe he is speaking with Alice. A few setup items: • Alice has long-term key A and ephemeral pre-keys A1, A2, …, An. • Bob has long-term key B and ephemeral pre-keys B1, B2, …, Bn. Mallory will take Bob's keys (his public long-term identity key and public ephemeral pre-keys) and publish them as her own by subverting the server or colluding with the server or publishing only Bob's "last resort"[21] pre-key, if it exists. She will then out-of-band authenticate "her" keys with Alice. If Alice attempts to message Mallory now: 1. Alice grabs one of Mallory's pre-keys (which is actually one of Bob's!), say B1. 2. Alice performs Triple Diffie-Hellman and sends the following (simplified) message: A || A1 || B1 || AE(key=ss, msg="Something bad") 1. Mallory will not be able to decrypt the message, but can then forward it to Bob. Bob will believe that he is speaking with Alice and will be able to decrypt and read the message. At this point, Mallory can continue sending these messages to Bob until Bob responds and a chain-key ratcheting step is performed (creating a new shared secret for subsequent messages). At this point Mallory cannot send more messages as she is not in possession of the new symmetric key or messages encrypted to this new symmetric key; Bob will not know that this is the case.
Recommendation	There are two ways in which this may be fixed: • **Uniquely** identifying information (user ID, email address, phone number, etc.) can be MACed into the initial message Alice sends. The ID info for each party would be included and verified by each peer when a message is received. This information must be provided by a higher-level protocol. Since the MAC can only be computed by honest peers with access to the shared secret (Alice and Bob, not Mallory), an attacker will not be able to modify and swap out the identifying information.

[21] Usually pre-keys are removed when a user requests them, but what happens when Bob's pre-keys are exhausted and he has not posted new ones? In these cases, the last pre-key is held indefinitely and sent to peers until Bob replenishes his stash.

- A stronger method would be to prove possession of the private key corresponding to the long-term identity key. One method of doing such would be to perform a challenge-response or other zero-knowledge protocol during the initial fingerprint verification.

Retest Results

Matrix has updated their documentation to call out unknown key-share attacks and how developers should mitigate this issue.[22]

Particular to the Matrix standard, documentation has been updated showing the mitigations they have applied to the specification.[23] Matrix has opted to include the sender and receiver identification numbers and the sender and receiver Ed25519 fingerprints into each encrypted message.[24] During decryption, these values are sanity checked and the message is then only returned to the user if all of these values are correct. This addresses the issue in matrix-js-sdk.

While this is fixed in matrix-js-sdk, the Olm library does **not** address the issue.

[22] https://matrix.org/git/olm/commit/?id=d48dc8197680dce2bb810c5714f17d1a35dcb3d0
[23] https://github.com/matrix-org/matrix-doc/pull/412/files
[24] https://github.com/matrix-org/matrix-js-sdk/pull/239/files

Finding	**Pre-Key Message Contains Un-MACed Fields**
Risk	**Informational** Impact: None, Exploitability: Low
Identifier	NCC-Olm2016-002
Status	Not Fixed
Category	Cryptography
Component	Olm
Location	src/session.cpp:317
Impact	An attacker can modify fields in the pre-key message leading to possible denial of service conditions that would disrupt communication between two peers.
Description	The initiator of a peer-to-peer conversation must send a specially crafted first message to their peer so they can bootstrap the first root key. This message, called a pre-key message, includes the following fields: 1. Version 2. One-Time Key of my peer 3. Base Key of myself 4. One-Time Key of myself 5. Encrypted, authenticated (MACed) message While there is a message authentication code present in the pre-key message (specifically HMAC-SHA256), it only provides authentication of the encrypted message. The other fields can be freely modified by an attacker. 1. Modifying the version may lead to a downgrade; however, only one version exists at this time. 2. Modifying the One-Time Key of the peer will cause a denial of service. The peer will ensure they are in possession of the private key corresponding to this One-Time Key before continuing execution of the protocol. 3. Modifying the Base Key will cause the peer to derive a shared secret different from the initiator, causing a denial of service. 4. Modifying the One-Time Key of the initiator will cause the peer to derive a shared secret different from the initiator, causing a denial of service.
Recommendation	Consider extending the MAC placed on the encrypted message (item five from the pre-key message listed above) to cover all items in the pre-key message.
Client Response	As this has no impact besides denial of service and because there is currently only one version of the protocol, this will be fixed at a later date.

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.