Attacks Against [matrix]

by Rebecca Guglin & El Jacobi

Introduction to Matrix

- **Matrix** is a communication protocol and standard that aims to provide infrastructure for decentralized, federated, and secure instant messaging. Its most popular implementer is the open-source instant messaging client Element.
- Matrix's development is overseen by the nonprofit Matrix.org Foundation, including a board of experts who ensure that changes to the protocol are in line with their values.
- The protocol has gained popularity as an open standard on which many different messaging systems are based, from Element and other chat apps, to TI-Messenger: the instant messaging standard of Germany's national healthcare system.
- **Threat Model:** For Matrix, the servers are assumed to be the adversary, as end-to-end encryption is built in and enabled by default.

Structure

- User Alice can have *i* devices $D_{A,0} \dots D_{A,i}$ and an account, which are connected to a particular **homeserver**.
- The homeserver allocates the user and device identifiers for Alice, maintains the user-device(s) link(s), and relays messages between users.
- A homeserver is shared by other users and their respective device(s) who want to have a conversation, and the group of users/devices is called a **room**. The homeserver is the relay point for messages from one user to the others.

Core Procedures

1. **Device Authentication**: Users and devices generate public keys, which are authenticated with a cross-signing framework.
2. **Session establishment:** After the keys are generated and authenticated, Olm channels are established between each pair of devices in a room using individual pairs of keys for each pair of users.
3. **Session communication**: Group Megolm channels are used to send a single message to all users from one individual, and all users decrypt the message using that individual’s public key.

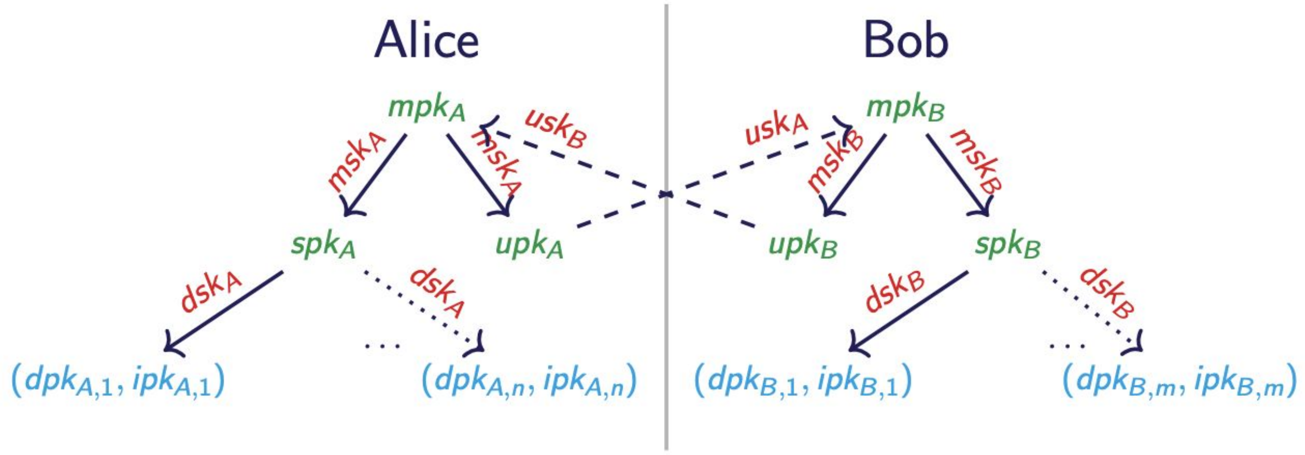
Cross-Signing Framework

A cross-signing framework is used to authenticate users.

- Alice sets up an account with a specific homeserver, and receives a user identifier *A*.
- Alice then generates 3 user secrets, which are used to establish trust:
    - Her public master key $\mathbf{mpk}_A$ serves as Alice's long-term identifier.
    - Her user-signing key $\mathbf{upk}_A$ signs other users' master keys, indicating that Alice trusts them.
    - Her self-signing key $\mathbf{spk}_A$ signs her own device key(s).

Cross-Signing Framework

Now Alice wants to talk to Bob, who has his own set of user secrets.



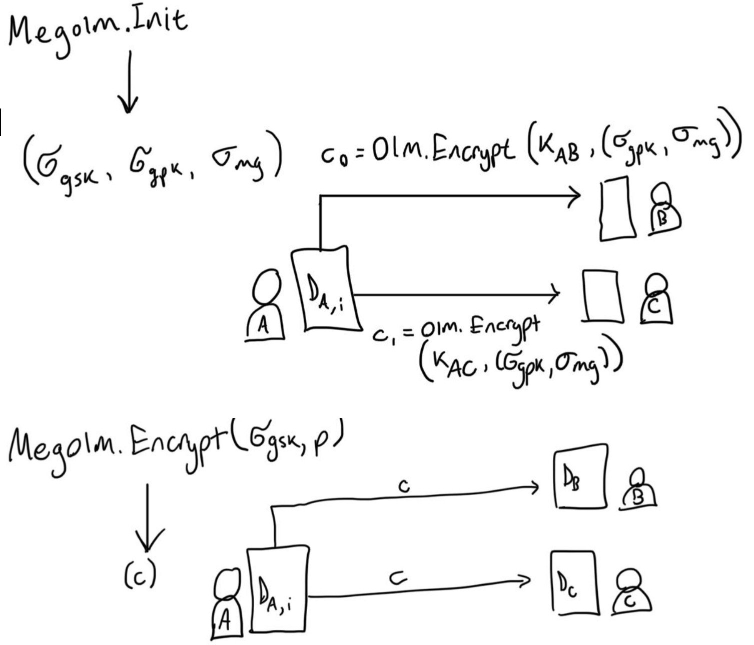
(source)

Definition: Ratchet algorithms

- **Olm channel**: An implementation of the Triple Diffie-Hellman key exchange protocol and the Signal Double Ratchet algorithm, which provides E2EE for instant messaging. Olm is peer-to-peer, so it cannot be used in applications where there will be many users receiving the same message.
  - Ratchet = an algorithm whose state can only move forward, so you can calculate future key values but have no information about past ones. The ratchet uses a Key Derivation Function (KDF) to generate a distinct key for each message.
- **Megolm**: AES-based cryptographic ratchet designed for communication between multiple users. A client receives a session key pair (public and private), then uses the value of the ratchet and the public key to encrypt a message before sending it via Olm channels to others in the conversation.
  - Megolm allows recipients to decrypt messages multiple times, so that clients only need to store session keys and the ciphertext can be stored anywhere, even an untrusted server.
- The goal of these algorithms is to provide **forward secrecy**: compromising a session key will not expose past communications.

Implementation of Olm and Megolm in Matrix

In step 1, Alice establishes a one-way Megolm channel made up of two Olm channels.


Megolm.Init
$(\sigma_{gsk}, \sigma_{gpk}, \sigma_{mg})$
$c_0 = \text{Olm.Encrypt}(K_{AB}, (\sigma_{gpk}, \sigma_{mg}))$
$D_{A,i}$
A
B
C
$c_1 = \text{Olm.Encrypt}(K_{AC}, (\sigma_{gpk}, \sigma_{mg}))$
$\text{Megolm.Encrypt}(\sigma_{gsk}, p)$
(c)
c
c
D_B
D_C

In step 2, she sends ciphertext *c* to multiple other users at the same time.

Attack 1: Trivial Confidentiality Breaks in Megolm

Main Idea: Too much trust in the homeserver allows a malicious homeserver to add its own devices

- Group membership is controlled by unauthenticated messages sent in groups
- Additionally, the list of users & devices is managed by a homeserver
- The homeserver can trivially add anyone it likes to a group by faking the add message or the device list, allowing them to eavesdrop
- Does not *technically* break confidentiality of underlying Olm sessions or cryptographic guarantees of individual Megolm sessions
- Users will be able to see the added malicious devices and must catch these attacks themselves

Attack 2: Breaking Out-of-Band Verification

Main Idea: Take advantage of user and device IDs being signed similarly to verify a malicious homeserver owner's device

- During the step where users sign each other's keys
- Short Authentication String protocol:
  - Prevents the homeserver from introducing its own devices as proxies for users
  - Users use Short Authentication String protocol, where users generate a secret, ensure their secrets match outside of Matrix, and create a secure channel using the secret to send their identities
- Tricks users into sending something a malicious homeserver controls instead of the SAS secret

Attack 2: Breaking Out-of-Band Verification

Main Idea: Take advantage of user and device IDs being signed similarly to verify a malicious homeserver owner's device

- Verification occurs between different users and between different users' devices
- In the final step of SAS, the messages are sent with either:
    - Fingerprint of master cross-signing key (mpk), if between users
    - Device ID, if between devices
- Device IDs are controlled by the homeserver
- A malicious homeserver can give a device an ID that's also a fingerprint of an mpk it controls
- When an user gets sent it, they interpret it as being sent a mpk to trust
- Gives full ability to eavesdrop

Attack 3.1: Semi-Trusted Impersonation

Main Idea: Lack of verification on accepted key shares allows attackers to impersonate devices

- New devices need keys to decrypt old messages sent to an user
- Keys are sent with KeyRequest protocol
- This authenticates devices before keys are sent to them
- Receiving devices *do not verify who is sending them keys*

Attack 3.2: Trusted Impersonation

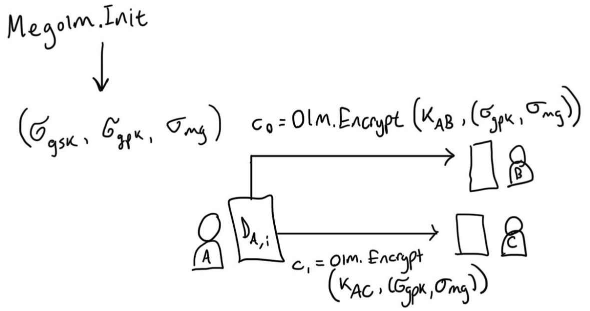
Main Idea: Use semi-trusted impersonation to achieve full impersonation

- Sending messages from devices added with KeyRequest are visually marked in some implementations as untrustworthy

Attack 3.2: Trusted Impersonation

Main Idea: Use semi-trusted impersonation to achieve full impersonation

- Recall how we initialize a connection using Olm channels


Megolm.Init
$(\sigma_{gsk}, \sigma_{gpk}, \sigma_{mg})$
$c_0 = \text{Olm.Encrypt}(K_{AB}, (\sigma_{gpk}, \sigma_{mg}))$
$D_{A,i}$
A
B
C
$c_1 = \text{Olm.Encrypt}(K_{AC}, (\sigma_{gpk}, \sigma_{mg}))$

- Nothing in the implementation stops us from starting a new connection, initialized with a Megolm channel, including the one we compromised
- Creates fully trusted impersonation in the new Megolm channel with the adversary's devices

Attack 3.3: Breaking Confidentiality

Main Idea: Use fully trusted impersonating devices to access backups

- Attacker sends a new backup key for the Secure Secret Storage and Sharing protocol
- The other users backs up all messages encrypted using this key on the homeserver, which the attacker can then decrypt

IND-CCA

- AES-CTR is used to encrypt backups and symmetric Megolm Key Backups
- The IV used is not included in the MAC
- An adversary can request the decryption of the XOR of a ciphertext of a message and the encryption of a known ciphertext
- Due to AES using XORs, they can XOR their known ciphertext with the result to get the uknown message
- Not practical to use because of the ratchet algorithms and JSON formats of ciphertexts

Sources

1. https://gitlab.matrix.org/
2. Albrecht, Martin R., Sofía Celi, Benjamin Dowling, and Daniel Jones. "Practically-exploitable Cryptographic Vulnerabilities in Matrix." Cryptology EPrint Archive, (2023). Accessed April 10, 2023.
3. https://matrix.org/blog/2021/07/21/germanys-national-healthcare-system-adopts-matrix
4. https://matrix.org/foundation/
5. https://www.gematik.de/anwendungen/ti-messenger
6. https://research.nccgroup.com/2016/11/01/public-report-matrix-olm-cryptographic-review/
7. https://leastauthority.com/static/publications/LeastAuthority-Matrix_vodozemac_Final_Audit_Report.pdf
8. https://gitlab.matrix.org/matrix-org/olm/-/raw/master/docs/olm.md
9. https://gitlab.matrix.org/matrix-org/olm/-/raw/master/docs/megolm.md
10. https://i.blackhat.com/EU-22/Wednesday-Briefings/EU-22-Jones-Practically-exploitable-Cryptographic-Vulnerabilities-in-Matrix.pdf