Decentralised Communication:
The challenge of balancing interoperability and privacy.

matthew@matrix.org
http://www.matrix.org
Privacy in Matrix
Two basic types of privacy:

1. Can attackers see what you're saying?

2. Can attackers see who you're talking to, and when?
Matrix can protect the contents of what you're saying using end-to-end encryption.

Neither the servers nor the network can decrypt the data; only invited clients.
End to End Crypto with Olm

https://matrix.org/git/olm
End to End Encryption

• Based on Open Whisper Systems’ “Double Ratchet” algorithm as used in Signal etc.
• Public audit by NCC Group
• Started beta roll-out in Sept 2016 on Web
• Beta launched Nov 21 2016 on iOS+Android
• Keys are per-device, not per-user (currently)
• So encryption is per-device.
• Supports flexible history privacy per-room.
Olm

• Apache License C++11 implementation of Double Ratchet, exposing a C API.
• Supports encrypted asynchronous 1:1 communication.
• “Megolm” layer adds group communication too.
• ~150KB x86-64 .so, or ~250KB of asm.js
Olm + Megolm C API

Megolm Group Ratchet

Account
  • Keys

Session
  • Initial Key Exchange

Ratchet
  • Encrypt
  • Decrypt

Crypto
  • Curve25519
  • AES
  • SHA256
Alice and Bob both generate identity (I) & ephemeral (E) elliptic curve key pairs

Initial Shared Secret (ISS) =
- \( \text{ECDH}(E_a, I_b) + \)
- \( \text{ECDH}(I_a, E_b) + \)
- \( \text{ECDH}(E_a, E_b) \)

Discard \( E_a \)
Derive chain key from ISS (HMAC)
Derive message key (\( K_0 \)) from chain key (HMAC)
Derive new chain key \( \leftarrow \text{hash ratchet} \)

\( M_0 = \) Message plaintext
\( C_0 = \) Authenticated Encryption of \( (M_0, K_0) \)
\( R_{a_0} = \) generate random ratchet key pair
\( J_{a_0} = \) incremental counter for each hash ratchet advancement

\( I_a, E_a, E_b, R_{a_0}, J_{a_0}, C_0 \)
Alice

Bob

A Double ratchet.
Kinda sorta.

Compute same Initial Shared Secret =
ECDH(Ea, Ib) +
ECDH(ia, Eb) +
ECDH(Ea, Eb)

Compute same $K_0$
$M_0 = \text{Authenticated decryption of } (C_0, K_0)$

To respond, B starts new ratchet chain:
$R_{b1} = \text{generate random ratchet key pair}$
New Initial Shared Secret =
ECDH(Ra$_0$, R$_{b1}$) $\leftarrow$ ECDH Ratchet

$C_0 = \text{Authenticated Encryption of } (M, K_0)$
$Ra_0 = \text{generate random ratchet key}$
$Ja_0 = \text{incremental counter for each hash}$
ratchet advancement

R$_{b1}$, J$_{b1}$, C$_1$
Alice

Sending | Receiving
MK  | CK  | RK  | CK  | MK
--  | --  | --  | --  | --

---

ECDH(A₀,B₀)

---

ECDH(A₁,B₀) +

---

CK—A₁—B₀

---

MK—₀ ———

MK—₁ ———

MK—₂ ———

---

ECDH(A₂,B₁) +

---

CK—A₂—B₁

---

MK—₀ ———

---

CK—A₂—B₂

---

MK—₀ ———
Group chat

- Adds a 3rd type of ratchet: “Megolm”, used to encrypt group messages.
- Simple hash ratchet, which can be fast-forwarded to ease sharing ratchet details.
- Each sender maintains its own ratchet per room.
- Establish 'normal' 1:1 ratchets between all participant devices in order to share the initial secret for a sender’s group ratchet session.
- Ratchets are replaced when users leave, on demand, or every N messages.
Flexible privacy with Megolm

- Rooms can be configured to have:
  - No ratchet (i.e. no crypto)
  - Full PFS ratchet
  - Selective ratchet

  - Deliberately share megolm "session keys" to support paginating partial eras of history.
  - Up to participants to trigger the ratchet (e.g. when a member joins or leaves the room)
Olm: What’s next?

• Debugging!
• Backing up & restoring megolm session ratchet data
• Sharing session ratchet data with new devices or new room participants
• Cross-signing device keys?
• Better device verification
• Better push notification UX for E2E rooms
• Better primitives & performance
• Turning on E2E by default for rooms with private history
• Negotiating E2E with legacy clients(?)
So, what about protecting metadata?

(i.e. hiding who's talking to who and when?)
Matrix is all about pragmatically fixing today's vendor lock-in problem.

You can't bridge existing networks without exposing who's talking to who.
Bridges expose metadata

Existing App

Unavoidable Metadata leak!
That said, Matrix also exposes metadata on Home Servers:
Home Servers expose metadata too
Can we do better?

Apps like Pond show that you can obfuscate metadata quite effectively:
Pond servers (Tor hidden services)

Pond clients, storing encrypted history

Pond preserves sender privacy through Group Signatures – only the client can decrypt who the message was from.
Matrix was designed to evolve and support future network architectures and privacy strategies.
Thought Experiment: Could Matrix adopt a Pond-like strategy?
• Move home servers onto the client.

• Use pond-style Tor hidden services for store-and-forward of encrypted messages.

• Migrate incrementally from 'classic' DAG federation.
Matrix with Pond strategy
Advantages over pure Pond

• Supports any and all Matrix clients via the existing standard client-server API

• Supports decentralised conversation history by tunnelling HS federation over Pond

• Supports bridging to other networks via existing Matrix AS API or classic Matrix Federation – at expense of privacy. Mitigated by disabling bridging/federation per-room.
Thank you!

matthew@matrix.org
http://matrix.org
@matrixdotorg