UPDATE 2019/10/14: This document is superceded by https://github.com/matrix-org/matrix-doc/pull/1228

MSC1228: Removing MXIDs from events

Background

For GDPR compliance, we may like to be able to break the association between a user's id (such as @richvdh:sw1v.org) and their activity in a room.

The stretch goal is to also remove server names, since for many users, they are the only user on a server and it is reasonable to be able to ask that sw1v.org be removed.

The general idea is to use a pseudonym in many places where we currently use user IDs. The current @user:server then becomes a user alias; the mapping between alias and the pseudonymous ID is public but can be removed in the future.

User IDs currently appear in the following places in a room:

- 'sender' of each event
- 'creator' key in m.room.create[1]
- state_key of m.room.member events
- 'users' list in m.room.power_levels events
- ‘creatorUserId’ in the content of m.widget

Server names appear in the following places:

- 'origin' of each event
- keys in the 'signatures' dict
- Room IDs and Event IDs
- Room Aliases
- state_keys in 'room_aliases' events

[1] It is suggested (below, and in https://github.com/matrix-org/matrix-doc/issues/1193), that we remove this field, since it's pointless.

Proposal

[This is v3 of this proposal, which in summary is: do proposal v1, but also introduce a requirement for a global user_key at the same time, which in future will replace mxids as the user's One True Identity. v1 and v2 are included for reference below.

- Each user (currently identified by an mxid) will also have a **user_key**. In time, this will replace the mxid as your One True Identity; however for now they will live in parallel.
    - A user_key is represented like `~1:dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh`, where `1:` is a version (to allow other systems to be used in future) and the rest is an (unpadded urlsafe-base64ed) ed25519 public key.
    - Homeservers are responsible for making up keys for their users.
    - For now, each homeserver maintains a one-to-one mapping between user_key and mxid for each of their users. In future, we will look to break this link to allow portability of accounts.
- Room IDs also become ed25519 public keys.
    - They look like: `!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI`.
    - The server which creates the room is responsible for creating the keypair.
    - The m.room.create event is signed with the room id to stop people making new rooms which look like old ones. After this point, the private key is never needed again[2].
- Define a **user_room_key**, which is yet another ed25519 public key.
    - It looks like `^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw`.
    - Homeservers are responsible for making up user keys for their users. They can (and should) use a different key in each room for each user.
    - This user_room_key is used where we currently use an mxid in the DAG: 'sender', m.room.member, m.room.power_levels. 'creator' is removed.
    - Events are **signed by the user_room_key of the sender instead of the server's key**.
    - If a user leaves and rejoins a room, they should use the same user_room_key (unless a server admin has manually removed the old mapping). This makes ban evasion harder. (It's up to server owners to ensure this rule is followed - servers which don't respect it and allow a serial abuser to evade bans by issuing different user_room_keys are likely to suffer whole-server bans.)
- Invite and join events include:
    - `mxid_mapping`: field which gives the user's `@user:server` mxid and which must be signed by the server in question, and the signature must be verified before the mapping is considered valid.
    - `user_mapping`: contains the user_key giving your True Identity, and signed by that key. The signature must be verified by receiving homeservers for it to be considered a valid invite/join event for a vNext room.

[2] although we might think about letting its use confer some sort of founder semantics.

- Event IDs need replacing; this is handled in MSC1659.
- We get rid of m.room.aliases events and just use the directory properly.

Examples

m.room.create:


```
{
  "type": "m.room.create",
  "state_key": "",
  "room_id": "!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI",
  "event_id": "$Riw5upWofaD4MNGM7bbZIj3bf+Th3fW/tklH4+6+VOg",
  "sender": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "content": {},
  "origin_server_ts": 1459891964497,
  "prev_events": [],
  "prev_state": [],
  "auth_events": [],
  "signatures": {
    "!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI": "<...>",
    "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw": "<...>"
  },
  "hashes":{"sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"}
}
```

m.room.member:


```
{
  "type": "m.room.member",
  "state_key": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "room_id": "!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI",
  "event_id": "$k21EhS3j8lhwqTi5NMTUH04oFyvR/1ujBGSWbW27aDs",
  "sender": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "content": {
    "membership": "join",
    "avatar_url": "...",
    "displayname": "...",
    "mxid_mapping": {
      "user_room_key": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
      "user_id": "@richvdh:matrix.org",
      "signatures": {
        "matrix.org": { "ed25519:a_zrXW": "<...>" }
      }
    },
    "user_mapping": {
      "user_key": "~1:dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh",
      "user_room_key": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
      "signatures": {
        "~1:dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh": "<...>"
      }
    }
  },
  "origin_server_ts": 1489597048772,
  "prev_events": [
    ["$Riw5upWofaD4MNGM7bbZIj3bf+Th3fW/tklH4+6+VOg", {
       "sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"
    }]
  ],
  "prev_state": [],
  "auth_events": [
    ["$Riw5upWofaD4MNGM7bbZIj3bf+Th3fW/tklH4+6+VOg", {
       "sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"
    }]
  ],
  "signatures": {
    "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw": "<...>"
  },
```

```
  "hashes":{"sha256":"KLx4Alfa0QzOihSmUMZ1WQj5QdWnbMHwmqKxmYO8hJE"}
}
```

Handling the mxid mapping

When a server joins a room, it will be presented with a bunch of m.room.member events which may claim mappings onto mxids. There are three reasons we need to know whether the mapping is valid:

- For clients, to help track users between rooms and to correlate to presence
- To authorise other servers to do backfill requests, etc.
- For outgoing messages, knowing which servers to send to.

None of these are *particularly* urgent (they all degrade fairly gracefully in the case that a mapping is missed). A lot of the slowness in joining rooms currently comes from having to pull server keys so that event signatures can be verified during the join process, so it would be nice to be able to consider the join complete, and verify the mxid_mappings in the background.

Implementation

We make an attempt to verify the sender before sending events to our clients.

When a new invite/join event turns up for a room you are already in (either via federation push, or because we pulled it via /event/xxx due to missing auth_events/prev_events), we make an attempt to verify its mxid mapping before persisting it into our db.

- If the sig is wrong, we can reject the event.
- If we can't get the key (with a shortish timeout), we handle it as normal (and schedule a retry for later).
- This should mean that in the majority of cases, we'll have a verified mxid by the time we send the event to a client.

Typically we only care about the most recent mxid mapping[3] for each user_room_key; when we see another we can cancel any pending verification of any previous mapping[4]. Any previously-verified mapping should remain in place until another mapping becomes available.

When we backfill, we do the same thing, although in many cases we'll already have active mappings for the users in question, so we can ignore any received that way. By only honouring the most recent mapping, we gain the correct semantics for account portability: authorisation for backfill depends on the *current* location of the user, rather than wherever they were in the past.

---

[3] or more accurately, the one in the current room state.
[4] if a user/server spams out mappings so quickly that none of them ever complete, that is their own loss.

When we join a new room, for now we do the same thing (ie, make an attempt to verify the mxid mappings in the join events we receive, and time them out quickly). In future we might optimise this so the the mappings are verified lazily.

For each user_room_key, we therefore have:

- zero or one verified mxid mappings.
- zero or one incomplete mxid mappings.

We extend the CS API to include a `verified_sender_mxid` field on any events sent via /sync or /messages where we know the user's current **verified** mxid.

- This is included in the interests of helping simple clients do the right thing most of the time - but it is annoying and dangerous because it will **sometimes** be missing. Still, we don't want to either (a) hold up all traffic in the room while we wait for a verification which may never succeed; (b) hold back *some* events while we do a verification for a sender; (c) require that all clients always have to wait for an asynchronous verification and match them up.

We also add a **new** field to the /sync response which tells clients about mxid mappings as they are resolved.

Question: should we remove unverified mxid mappings from join events before serving them to the clients, to stop client developers relying on it and breaking everything?

Sending invites

We have a bootstrapping problem for invites in that, until a user joins a room, we don't know their user_room_key.

Also: invite events are supposed to be signed by the invitee, so that other members of the room can be sure that they have actually received a copy of the event.

The current invite dance is:

- inviting server builds a complete invite event, and signs it
- inviting server sends a copy to invited server, along with some (unsigned) state about the room: name, avatar, inviting user's join event
- invited server checks that request came from server of inviting user
- invited server also signs the event, and tells the user about it
- inviting server adds the (double-signed) event to the DAG and sends it to the rest of the federation (including the invited server, if it was already in the room)

We could change this to:

- inviting server builds a partial invite event

- inviting server PUTs to `/_matrix/federation/v2/invite/<event_id>/<target_mxid>` on invited server
- invited server checks that request came from server of inviting user
- invited server adds:
    - user_room_key in state_key
    - mxid attestation
    - signature
- invited server tells the user about it, and returns the completed event to inviting server
- inviting server adds its signature to the complete event
- inviting server adds the (double-signed) event to the DAG and sends it to the rest of the federation (including the invited server, if it was already in the room)

Problems

~~Joining by room id (for example, when handling permalinks): this has never been a thing that ought to work, but sort of does, until now. How can we make permalinks continue to work? Ideally we want to include a (list of?) potential server names in the permalink.~~ Now resolved (I think) by MSC1704, at the expense of ruining the future anonymity of server names. My inclination is to descope server anonymity in permalinks for now, and revisit the problem in the future.

What if somebody does a join with an inappropriate avatar/displayname? If we redact their join, we'll redact their identity assertion too :/

Other things we should fix while we are breaking federation

- Change state-resolution rules to reduce resets, as per https://docs.google.com/document/d/1L2cr8djdpOFXJgqGTf3gUrk-YGBYf--rP8Nw6mYYAu8/edit#heading=h.bqv066j03e2i
- Spurious [200, $data] wrapping in v1/send_join, v1/make_join, v1/invite (https://github.com/matrix-org/synapse/issues/1383)
- ...

Other things that might fall out nicely

- Fixes broken backfill due to changed signing keys (#3121)
- Case sensitive mxid comparison problems?
- Opens a path to killing off perspectives in favour of just asking servers what their keys are (via TLS, with trust coming from X.509 certificates),
- Helping with the domain reuse problem
- Making coffee?
- Nicer way of validating redactions by comparing the user_rom_key of the message and the redaction which addresses the suboptimal solution introduced by MSC1659

- …

Stuff we might want to avoid designing out

- Ability to migrate users between servers by changing their mapping assertions
- Ability to support alternative identity mapping assertions rather than being strictly mxid->user_key (e.g. 3pid->user-key too). This could help decentralised identity mappings in general in future, and possibly unify 3pid invites with normal invites? It could also support discovering servers by key rather than DNS (e.g. via DHT), which would be useful for p2p in future.

Key management

- If a private user key gets lost, they can just start using a new one and announce a new mapping; however this may require an update to the power_levels to give rights to the new user.
- If a private user key is compromised, then again we start using a new one and announce a new mapping, so that new events from the old key wouldn't look like they came from that user. Again it may need power_levels updates to remove power from the old user_key, but I think that is fair: you give away the keys to your privileged account, you have to expect some cleanup. Ideally we would have a way of revoking the old key properly, but this can be deferred for now.

Migration

- We're changing room ID, so we have no choice but to find a way to set up a "room forward" from the old room to the new one.
- Who gets to set up the forward? Presumably an admin (or room creator)
- Need to mark the old room as closed for business, providing an OOB way for users to rejoin the new room if they want? (to avoid needlessly pulling stale users into the new room)
- How do we reference history from the new room to the old one? Just have a link in m.room.create to point to the previous room that clients can expose as a link to be followed when wading through scrollback? Smart clients (or servers?) could potentially do search operations over all the rooms grouped together in this manner?
- How do we recreate the power levels in the new room for users from the old one?
    - Presumably the admin/owner who set up the forward can get their power recreated, given they're creating the new room - and can also invite other trusted users and set power levels for them? In other words, the whole migration can be

performed manually by the admin’s client, almost like it’d happen in a purely social process?

Appendix: earlier versions of this proposal

Proposal v1

- Everything that follows applies to "version 2 rooms". You can only join a v2 room via `/_matrix/federation/v2/make_join`, which is like `v1/make_join` but includes a 'room_version' key in the response.
    - Attempts to join v2 rooms via the v1 api get a 4xx response.
    - There is no problem joining a v1 room via the v2 API.
- Room IDs become (unpadded urlsafe-base64ed) ed25519 public keys. They look like: `!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI`.
    - The m.room.create event is signed with the room id to stop people making new rooms which look like old ones. After this point, the private key is never needed again[5].
- Define a **user_key**, which is an (unpadded urlsafe-base64ed) ed25519 public key. It looks like `~Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw`. Homeservers are responsible for making up user keys for their users. They can (and should) use a different key in each room for each user.
    - This user_key is used where we currently use an mxid in the DAG: 'sender', m.room.member, m.room.power_levels. 'creator' is removed.
    - Events are **signed by the user_key instead of the server's key**.
    - If a user leaves and rejoins a room, they should use the same user_key (unless a server admin has manually removed the old mapping). This makes ban evasion harder.
- Invite and join events include an mxid_mapping field which gives the user's `@user:server` mxid and which must be signed by the server in question, and the signature must be verified before the mapping is considered valid.
- Event IDs need replacing somehow. See below for options.
- We get rid of m.room.aliases events and just use the directory properly.

Examples

m.room.create:


```
{
  "type": "m.room.create",
  "state_key": "",
  "room_id": "!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI",
  "event_id": "$Riw5upWofaD4MNGM7bbZIj3bf+Th3fW/tklH4+6+VOg",
  "sender": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "content": {},
```


[5] although we might think about letting its use confer some sort of founder semantics.

```
  "origin_server_ts": 1459891964497,
  "prev_events": [],
  "prev_state": [],
  "auth_events": [],
  "signatures": {
    "!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI": "<...>",
    "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw": "<...>"
  },
  "hashes":{"sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"}
}
```


m.room.member:


```
{
  "type": "m.room.member",
  "state_key": "*1:Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "room_id": "!1:Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI",
  "event_id": "$14895970491mAUoX:*1:Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "content": {
    "membership": "join",
    "avatar_url": "...",
    "displayname": "...",
    "mxid_mapping": {
      "user_key": "*1:Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
      "user_id": "@richvdh:matrix.org",
      "signatures": {
        "matrix.org": { "ed25519:a_zrXW": "<...>" }
      }
    }
  },
  "origin_server_ts": 1489597048772,
  "prev_events":[
    ["$15244790326087712Topvy:*1:Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw", {
       "sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"
    }]
  ],
  "prev_state":[],
  "auth_events": [
    ["$15244790326087712Topvy:*1:Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw", {
       "sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"
    }]
```

```
  ],
  "signatures": {
    "*1:Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw": "<...>"
  },
  "hashes":{"sha256":"KLx4Alfa0QzOihSmUMZ1WQj5QdWnbMHwmqKxmYO8hJE"}
}
```

Proposal v2: your identifier is now a key

- Join via via `/_matrix/federation/v2/make_join` as before
- Room IDs are a public key as before
- Your One True Identity in matrix also becomes a public key (eg `~1:dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh`).
    - These are divorced from rooms so probably need versioning - hence the `1:`.
    - Homeservers are responsible for making up keys for their users.
    - These keys are used in group memberships.
- Define a **user_room_key**, which is an (unpadded urlsafe-base64ed) ed25519 public key. It looks like `^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw`.
    - Homeservers are responsible for making up user keys for their users. They can (and should) use a different key in each room for each user.
    - This user_room_key is used where we currently use an mxid in the DAG: 'sender', m.room.member, m.room.power_levels. 'creator' is removed.
    - Events are **signed by the user_room_key instead of the server's key**.
    - If a user leaves and rejoins a room, they should use the same user_key (unless a server admin has manually removed the old mapping). This makes ban evasion harder.
- Invite and join events include two extra fields:
    - `user_mapping`: contains the user_key giving your True Identity, and signed by that key
    - `homeserver`: contains the server_name of the server where you live.
- We provide a lookup API, so you can still reach me at @richard:sw1v.org by sending a lookup to sw1v.org which responds with my user_key.
- Event IDs still need redesigning, in the same way as before
- Room alias events still need removing, in the same way as before

This is much nicer, as it means we don't have to mess around with any asynchronous fetching keys and verifying signatures.

Problems

- If you accidentally delete your server you'll no longer be able to simply create a new one and "magically" be rejoined into the rooms that you were in
    - This is largely a feature, though we'll need a way to communicate back to users that their PM is now going into /dev/null
- For this to work, everything that we currently do outside of rooms at the mxid level (presence, e2e device key lists, m.direct maps) would have to be switched to use user_keys instead. It's going to be very hard to make that work in the continued presence of v1 rooms - and even then, the migration is going to be a pita for clients.

Examples

m.room.create:


```
{
  "type": "m.room.create",
  "state_key": "",
  "room_id": "!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI",
  "event_id": "$Riw5upWofaD4MNGM7bbZIj3bf+Th3fW/tklH4+6+VOg",
  "sender": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "content": {},
  "origin_server_ts": 1459891964497,
  "prev_events": [],
  "prev_state": [],
  "auth_events": [],
  "signatures": {
    "!1:Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI": "<...>",
    "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw": "<...>"
  },
  "hashes":{"sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"}
}
```

m.room.member:


```
{
  "type": "m.room.member",
  "state_key": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "room_id": "!Sr_Vj3FIqyQ2WjJ9fWpUXRdz6fX4oFAjKrDmu198PnI",
  "event_id": "$k21EhS3j8lhwqTi5NMTUH04oFyvR/1ujBGSWbW27aDs",
  "sender": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
  "content": {
    "membership": "join",
    "avatar_url": "...",
    "displayname": "...",
    "homeservers": [ "sw1v.org" ],
    "user_mapping": {
      "user_key": "~1:dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh",
      "user_room_key": "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw",
      "signatures": {
        "~1:dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh": "<...>"
      }
    }
  },
  "origin_server_ts": 1489597048772,
  "prev_events":[
    ["$Riw5upWofaD4MNGM7bbZIj3bf+Th3fW/tklH4+6+VOg", {
       "sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"
    }]
  ],
  "prev_state":[],
  "auth_events": [
    ["$Riw5upWofaD4MNGM7bbZIj3bf+Th3fW/tklH4+6+VOg", {
       "sha256":"3ASU57dV3hr3yE9SxhsWEGBJdTho777S8ompkJTh/Uo"
    }]
  ],
  "signatures": {
    "^Noi6WqcDj0QmPxCNQqgezwTlBKrfqehY1u2FyWP9uYw": "<...>"
  },
  "hashes":{"sha256":"KLx4Alfa0QzOihSmUMZ1WQj5QdWnbMHwmqKxmYO8hJE"}
}
```