Package ‘mx.client’

September 11, 2026

Type Package

Title Stateful Matrix Client Helpers

Version 0.2.1

Date 2026-09-11

Description Stateful helpers for building 'Matrix' (<https://matrix.org>) chat clients in R. Builds on the low-level 'mx.api' Client-Server API bindings, adding local configuration persistence, room resolution, sync cursor handling, sync-event extraction, invite acceptance, a conservative Markdown-to-HTML converter for formatted messages, and 'Olm'/'Megolm' end-to-end encryption orchestration over the optional 'mx.crypto' package.

License Apache License (>= 2)

URL https://github.com/cornball-ai/mx.client

BugReports https://github.com/cornball-ai/mx.client/issues

Depends R (>= 4.0)

Imports jsonlite, mx.api (>= 0.3.0.2), stats, tools, utils

Suggests mx.crypto (>= 0.2.2), simplermarkdown, tinytest

VignetteBuilder simplermarkdown

Encoding UTF-8

NeedsCompilation no

Author Troy Hernandez [aut, cre] (ORCID: <https://orcid.org/0009-0005-4248-604X>), cornball.ai [cph]

Maintainer Troy Hernandez <troy@cornball.ai>

Repository CRAN

Date/Publication 2026-09-11 16:50:02 UTC

Contents

`mx.client-package` *Stateful Matrix Client Helpers*

Description

Stateful helpers for building 'Matrix' (<https://matrix.org>) chat clients in R. Builds on the low-level 'mx.api' Client-Server API bindings, adding local configuration persistence, room resolution, sync cursor handling, sync-event extraction, invite acceptance, a conservative Markdown-to-HTML converter for formatted messages, and 'Olm'/'Megolm' end-to-end encryption orchestration over the optional 'mx.crypto' package.

Package Content

Index of help topics:

```
mx.client-package       Stateful Matrix Client Helpers
mx_accept_invites       Accept pending Matrix room invites
mx_client_config_path   Path to a Matrix client config file
mx_client_configure     Configure and save a Matrix client
mx_client_from_config   Wrap a list as an mx.client config
mx_client_legacy_config_path
                        Legacy Matrix config path for an application
mx_client_load          Load a Matrix client config
mx_client_relogin       Re-login with stored credentials and refresh
                        the saved token
mx_client_save          Save a Matrix client config
```

```
mx_client_session       Build an mx.api session from a client config
mx_crypto_account       Load or create this client's Olm account
mx_crypto_account_save
                        Persist an Olm account to the store
mx_crypto_claim_otks    Claim a one-time key for each device
mx_crypto_cross_signing_bootstrap
                        Bootstrap and sign this Matrix device's
                        cross-signing identity
mx_crypto_cross_signing_load
                        Load locally persisted Matrix cross-signing
                        keys
mx_crypto_decrypt_event
                        Decrypt an m.room.encrypted event (Megolm)
mx_crypto_device_keys   Build a signed device_keys object for upload
mx_crypto_encrypt_event
                        Encrypt event content for a room (Megolm)
mx_crypto_encrypt_for_devices
                        Encrypt an event for an encrypted room's
                        devices
mx_crypto_handle_to_device
                        Decrypt an inbound Olm to-device payload
mx_crypto_inbound_session
                        Build an inbound Megolm session from a shared
                        room key
mx_crypto_known_devices
                        List the devices (and identity keys) of some
                        users
mx_crypto_mark_key_requests_sent
                        Mark successfully transmitted room-key requests
mx_crypto_process_sync
                        Process a sync response: store room keys,
                        decrypt room events
mx_crypto_publish_keys
                        Publish this device's identity and one-time
                        keys
mx_crypto_room_key_payload
                        Encrypt a Megolm room key to one device as a
                        to-device payload
mx_crypto_send_key_requests
                        Send queued Matrix room-key requests or
                        cancellations
mx_crypto_sessions_load
                        Load a session set from the crypto store
mx_crypto_sessions_new
                        Create an empty E2EE session set
mx_crypto_sessions_save
                        Persist a session set to the crypto store
mx_crypto_store_dir     Directory holding this client's encryption
```

```
                        state
mx_crypto_user_trust    Check a directional Matrix identity
                        verification
mx_crypto_verify_user   Verify another Matrix user's independently
                        authenticated identity
mx_extract_invite_records
                        Extract pending invites with the member who
                        sent them
mx_extract_invites      Extract pending invite room ids from a sync
                        response
mx_extract_media_events
                        Extract media message events from a sync
                        response
mx_extract_reaction_verdict
                        Extract a reaction approval verdict from sync
                        events
mx_extract_reactions    Extract reaction events from a sync response
mx_extract_text_events
                        Extract text message events from a sync
                        response
mx_markdown_to_html     Convert a conservative markdown subset to
                        Matrix custom HTML
mx_pill_mentions        Turn textual @mentions into Matrix pills in
                        formatted HTML
mx_resolve_room         Resolve a room id, name, or default room
mx_room_encrypted       Is a room end-to-end encrypted?
mx_room_lookup_by_name
                        Look up a joined room by display name
mx_sas_accept           Accept an incoming SAS verification request
mx_sas_cancel           Cancel a SAS transaction without granting trust
mx_sas_confirm          Confirm or reject the displayed Matrix SAS
mx_sas_console          Compare a Matrix SAS through a trusted
                        interactive console
mx_sas_from_request     Create a SAS transaction from an incoming
                        verification request
mx_sas_outgoing         Read or acknowledge queued SAS protocol
                        messages
mx_sas_receive          Receive one standard Matrix SAS protocol event
mx_sas_record_trust     Record trust after a human-confirmed,
                        authenticated SAS exchange
mx_sas_session          Create an in-memory Matrix SAS transaction
mx_sas_start            Begin the SAS key agreement after request
                        negotiation
mx_sas_status           Inspect a Matrix SAS verification transaction
mx_send_encrypted       Send an end-to-end encrypted message to a room
mx_send_media           Send a media file to a Matrix room
mx_send_table           Send tabular data to a Matrix room
mx_send_text            Send plain text to a Matrix room
```

mx_set_displayname	Set the account's profile display name
mx_sync_update	Sync once and update the stored cursor
mx_table_html	Render tabular data as Matrix custom HTML
mx_verify_console	Verify a Matrix client interactively from an R console
mx_with_relogin	Run a client operation, re-logging in once on an expired token
print.mx_client_config	Print a Matrix client config

Maintainer

Troy Hernandez <troy@cornball.ai>

Author(s)

Troy Hernandez [aut, cre] (ORCID: <https://orcid.org/0009-0005-4248-604X>), cornball.ai [cph]

---

mx_accept_invites *Accept pending Matrix room invites*

Description

Accept pending Matrix room invites

Usage

```
mx_accept_invites(client, invites)
```

Arguments

client	Matrix client config.
invites	Character vector of room ids.

Value

Character vector of joined room ids.

Examples

```
## Not run:
# Needs a live homeserver session.
client <- mx_client_load("myapp")
res <- mx_sync_update(client)
mx_accept_invites(res$client, mx_extract_invites(res$sync))

## End(Not run)
```

`mx_client_configure` *Configure and save a Matrix client*

Description

Logs in with **mx.api**, joins or records the target room, and writes a reusable local config. Extra fields are merged into the saved config so applications can persist their own defaults without reimplementing login.

Usage

```
mx_client_configure(
  server,
  user,
  password,
  room,
  app = "mx.client",
  path = NULL,
  device_id = NULL,
  extra = list()
)
```

Arguments

`server`	Character. Homeserver base URL.
`user`	Character. Matrix user localpart or full MXID.
`password`	Character. Account password.
`room`	Character. Room ID or alias to join.
`app`	Character. Application namespace.
`path`	Character or NULL. Explicit destination path.
`device_id`	Character or NULL. Existing device id to reuse.
`extra`	Named list. Additional fields to save.

Value

Saved `"mx_client_config"`, invisibly.

Examples

```
## Not run:
# Needs a live homeserver and account credentials.
mx_client_configure("https://matrix.example.org", "bot", "secret",
                    room = "#general:example.org", app = "myapp")

## End(Not run)
```

mx_client_config_path *Path to a Matrix client config file*

Description

Resolves the config path for an application that uses mx.client. The default environment variable is derived from app; for example, app = "corteza" honors CORTEZA_MATRIX_CONFIG.

Usage

```
mx_client_config_path(app = "mx.client", env_var = NULL)
```

Arguments

app	Character. Application namespace for tools::R_user_dir().
env_var	Character or NULL. Override environment variable name.

Value

Character path.

Examples

```
mx_client_config_path("myapp")
```

mx_client_from_config *Wrap a list as an mx.client config*

Description

Wrap a list as an mx.client config

Usage

```
mx_client_from_config(cfg, path = NULL, app = NULL)
```

Arguments

cfg	Named list.
path	Character or NULL. Source/sink path for saves.
app	Character or NULL. Application namespace.

Value

An object of class "mx_client_config".

Examples

```
cfg <- mx_client_from_config(list(server = "https://matrix.example.org",
                                  token = "syt_example",
                                  user_id = "@bot:example.org",
                                  device_id = "DEVICEID"))
class(cfg)
```

`mx_client_legacy_config_path` *Legacy Matrix config path for an application*

Description

Currently only `app = "corteza"` has a historical path: `~/.corteza/matrix.json`.

Usage

```
mx_client_legacy_config_path(app = "mx.client")
```

Arguments

`app`	Character. Application namespace.

Value

Character path or NULL.

Examples

```
mx_client_legacy_config_path("corteza")
```

`mx_client_load` *Load a Matrix client config*

Description

Reads a JSON config. If `path` or the derived environment variable is explicit, that path is authoritative. Otherwise `legacy_path` is used as a compatibility fallback when present.

Usage

```
mx_client_load(
  app = "mx.client",
  path = NULL,
  legacy_path = mx_client_legacy_config_path(app),
  env_var = NULL
)
```

Arguments

app	Character. Application namespace.
path	Character or NULL. Explicit config path.
legacy_path	Character or NULL. Backward-compatible fallback path.
env_var	Character or NULL. Override environment variable name.

Value

An "mx_client_config" object.

Examples

```
path <- file.path(tempdir(), "matrix.json")
cfg <- mx_client_from_config(list(server = "https://matrix.example.org",
                                  token = "syt_example",
                                  user_id = "@bot:example.org",
                                  device_id = "DEVICEID"))
mx_client_save(cfg, path = path)
mx_client_load(path = path)$user_id
unlink(path)
```

---

mx_client_relogin *Re-login with stored credentials and refresh the saved token*

Description

Uses the password persisted in the client config to obtain a fresh access token for the *same* device (reusing client$device_id, so an E2EE device identity survives the refresh), then saves the updated config. Typical use is recovering from an invalidated token; see mx_with_relogin for the catch-and-retry wrapper.

Usage

```
mx_client_relogin(client, save = TRUE)
```

Arguments

client	Matrix client config with password.
save	Logical. Persist the refreshed config (default TRUE).

Value

The refreshed "mx_client_config".

Examples

```
## Not run:
# Needs a live homeserver and a stored password.
client <- mx_client_relogin(mx_client_load("myapp"))

## End(Not run)
```

`mx_client_save` *Save a Matrix client config*

Description

Writes JSON with mode 0600.

Usage

```
mx_client_save(client, app = NULL, path = NULL)
```

Arguments

`client`	Named list or `"mx_client_config"`.
`app`	Character or NULL. Application namespace.
`path`	Character or NULL. Destination path.

Value

The saved config, invisibly.

Examples

```
path <- file.path(tempdir(), "matrix.json")
mx_client_save(list(server = "https://matrix.example.org",
                    token = "syt_example",
                    user_id = "@bot:example.org",
                    device_id = "DEVICEID"),
               path = path)
unlink(path)
```

mx_client_session *Build an mx.api session from a client config*

Description

Build an mx.api session from a client config

Usage

```
mx_client_session(client)
```

Arguments

client	Named list or "mx_client_config" with server, token, user_id, and device_id.

Value

An "mx_session" from **mx.api**.

Examples

```
s <- mx_client_session(list(server = "https://matrix.example.org",
                            token = "syt_example",
                            user_id = "@bot:example.org",
                            device_id = "DEVICEID"))
class(s)
```

mx_crypto_account *Load or create this client's Olm account*

Description

Unpickles account.pickle from the store, or mints a fresh account and persists it. The account holds the device's long-lived Curve25519/Ed25519 identity keys. Reads raw pickles and version-1 JSON envelopes; unknown envelope versions are rejected. Loading never rewrites the file, and saving continues to write raw encrypted pickles for compatibility with older clients.

Usage

```
mx_crypto_account(store_dir)
```

Arguments

store_dir	Character. Crypto store directory.

Value

An mx.crypto account handle.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
  store <- mx_crypto_store_dir("myapp", path = tempfile())
  acct <- mx_crypto_account(store)
  unlink(store, recursive = TRUE)
}
```

`mx_crypto_account_save` *Persist an Olm account to the store*

Description

Writes the raw encrypted account pickle, preserving compatibility with older clients. The encryption key and device identity are unchanged.

Usage

```
mx_crypto_account_save(account, store_dir)
```

Arguments

`account`	An mx.crypto account handle.
`store_dir`	Character. Crypto store directory.

Value

The pickle path, invisibly.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
  store <- mx_crypto_store_dir("myapp", path = tempfile())
  acct <- mx_crypto_account(store)
  mx_crypto_account_save(acct, store)
  unlink(store, recursive = TRUE)
}
```

`mx_crypto_claim_otks` *Claim a one-time key for each device*

Description

Calls `/keys/claim` and attaches the claimed key to each device as `$otk`, ready for `mx_crypto_encrypt_for_devices()`.

Usage

```
mx_crypto_claim_otks(client, devices, strict = FALSE)
```

Arguments

`client`	Matrix client config.
`devices`	List of devices from `mx_crypto_known_devices()`.
`strict`	Logical. Treat a `failures` map as an error rather than a warning.

Details

Each claimed key's signature is checked against the device's Ed25519 key, which must itself have come from a verified `device_keys` (that is, from `mx_crypto_known_devices()`). A key that fails is dropped with a warning and its device comes back with no `otk`.

`/keys/claim` carries the same `failures` map as `/keys/query`, and it is handled the same way: warned about by default, an error under `strict`.

Value

The devices with an `otk` field added where one was claimed and verified.

Examples

```
## Not run:
devs <- mx_crypto_claim_otks(client, mx_crypto_known_devices(client, uid))

## End(Not run)
```

`mx_crypto_cross_signing_bootstrap` *Bootstrap and sign this Matrix device's cross-signing identity*

Description

Creates master, self-signing and user-signing keys only when neither the local crypto store nor homeserver already has an identity. Existing local keys are reused; a server/local mismatch aborts instead of resetting the identity. The current device signs the master key and the self-signing key signs the current device. Valid signatures already returned by the homeserver are not re-uploaded.

Usage

```
mx_crypto_cross_signing_bootstrap(
  client,
  device_account,
  store_dir,
  password = NULL,
  auth = NULL
)
```

Arguments

`client`	Matrix client config.
`device_account`	This device's persisted Olm account.
`store_dir`	Character. Crypto store directory.
`password`	Character or NULL. Account password used only for UIA.
`auth`	Completed Matrix UIA object or NULL.

Value

Public master, self-signing and user-signing key ids, invisibly.

Examples

```
## Not run:
# Requires a live Matrix account and this device's existing crypto store.
# Stop other processes using the device first. Never substitute a new store.
client <- mx_client_load(path = Sys.getenv("MATRIX_CONFIG"))
store <- Sys.getenv("MATRIX_CRYPTO_STORE")
stopifnot(nzchar(store), file.exists(file.path(store, "account.pickle")))
account <- mx_crypto_account(store)
public <- mx_crypto_cross_signing_bootstrap(client, account, store,
    password = Sys.getenv("MATRIX_ACCOUNT_PASSWORD"))
# The encrypted private keys remain local; the return value contains key ids.
public$master

## End(Not run)
```

mx_crypto_cross_signing_load

Load locally persisted Matrix cross-signing keys

Description

Returns NULL when this crypto store has never bootstrapped a cross-signing identity. A present but malformed store is an error: generating over it would silently reset the user's Matrix identity.

Usage

```
mx_crypto_cross_signing_load(store_dir)
```

Arguments

store_dir	Character. Crypto store directory.

Value

A list containing master, self-signing and user-signing key handles, or NULL.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    # A missing store returns NULL and does not create a replacement identity.
    unused_store <- tempfile("unused-cross-signing-")
    stopifnot(is.null(mx_crypto_cross_signing_load(unused_store)),
        !file.exists(unused_store))
}
```

mx_crypto_decrypt_event

Decrypt an m.room.encrypted event (Megolm)

Description

Decrypt an m.room.encrypted event (Megolm)

Usage

```
mx_crypto_decrypt_event(inbound_session, encrypted)
```

Arguments

inbound_session	An inbound Megolm session for the event's session_id.
encrypted	The m.room.encrypted event content.

Value

The decrypted event payload (a parsed list).

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    account <- mx.crypto::mxc_account_new()
    sender <- mx.crypto::mxc_account_identity_keys(account)$curve25519
    megolm_out <- mx.crypto::mxc_megolm_outbound_new()
    key <- mx.crypto::mxc_megolm_outbound_info(megolm_out)$session_key
    inb <- mx_crypto_inbound_session(key)
    enc <- mx_crypto_encrypt_event(megolm_out,
        list(msgtype = "m.text", body = "Hello"),
        "!room:example.org", sender, "ALICE")
    ev <- mx_crypto_decrypt_event(inb, enc)
    stopifnot(identical(ev$content$body, "Hello"))
}
```

---

mx_crypto_device_keys *Build a signed device_keys object for upload*

Description

Produces the `device_keys` structure `/keys/upload` expects: the device's public identity keys plus an Ed25519 signature over their canonical JSON. Hand the result to `mx.api::mx_keys_upload()`.

Usage

```
mx_crypto_device_keys(account, user_id, device_id)
```

Arguments

`account`	An mx.crypto account handle.
`user_id`	Character. Full Matrix user id.
`device_id`	Character. This device's id.

Value

A named list ready to upload.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
  store <- mx_crypto_store_dir("myapp", path = tempfile())
  acct <- mx_crypto_account(store)
  dk <- mx_crypto_device_keys(acct, "@bot:example.org", "DEVICEID")
  unlink(store, recursive = TRUE)
}
```

```
## Not run:
# Uploading needs a live homeserver session:
mx.api::mx_keys_upload(session, device_keys = dk)

## End(Not run)
```

`mx_crypto_encrypt_event` *Encrypt event content for a room (Megolm)*

Description

Returns the `m.room.encrypted` content to send as the event body.

Usage

```
mx_crypto_encrypt_event(
  megolm_out,
  content,
  room_id,
  sender_curve25519,
  device_id,
  event_type = "m.room.message"
)
```

Arguments

`megolm_out`	An outbound Megolm session.
`content`	Named list. The plaintext event content (e.g. an `m.room.message`).
`room_id`	Character. Room id.
`sender_curve25519`	Character. This device's Curve25519 key.
`device_id`	Character. This device's id.
`event_type`	Inner Matrix event type. Defaults to m.room.message; verification replies use their m.key.verification.* event type.

Value

A named list: `m.room.encrypted` content.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    account <- mx.crypto::mxc_account_new()
    sender <- mx.crypto::mxc_account_identity_keys(account)$curve25519
    megolm_out <- mx.crypto::mxc_megolm_outbound_new()
    enc <- mx_crypto_encrypt_event(megolm_out,
        list(msgtype = "m.text", body = "Hello"),
        "!room:example.org", sender, "ALICE")
    stopifnot(identical(enc$algorithm, "m.megolm.v1.aes-sha2"))
}
```

---

mx_crypto_encrypt_for_devices

Encrypt an event for an encrypted room's devices

Description

Ensures an outbound Megolm session for the room, shares its key with any recipient device that has not received it yet (establishing an Olm session, claiming a one-time key when needed), and encrypts the event. Returns the to-device payloads and the `m.room.encrypted` event; the caller sends them with `mx.api::mx_send_to_device()` and `mx.api::mx_send()`.

Usage

```
mx_crypto_encrypt_for_devices(
  account,
  sessions,
  room_id,
  content,
  sender_curve25519,
  device_id,
  recipients = list(),
  sender_user_id = NULL,
  event_type = "m.room.message"
)
```

Arguments

`account`	An mx.crypto account handle.
`sessions`	A session set.
`room_id`	Character room id.
`content`	Named list. Plaintext event content.
`sender_curve25519`	Character. This device's Curve25519 key.
`device_id`	Character. This device's id.

recipients	List of recipient devices, each a list with user_id, device_id, curve25519, ed25519, and (only needed to open a new Olm session) otk, a claimed one-time key. Use mx_crypto_known_devices(), which returns exactly this shape for devices whose keys verified.
sender_user_id	Character. This user's Matrix id. Required to build a spec-conformant Olm payload that the recipient can attribute.
event_type	Inner Matrix event type, defaulting to m.room.message.

Value

List with to_device (per-device payloads), event (the m.room.encrypted content), and the updated sessions.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
  acct <- mx.crypto::mxc_account_new()
  out <- mx_crypto_encrypt_for_devices(
    acct, mx_crypto_sessions_new(), "!r:ex",
    list(msgtype = "m.text", body = "hi"),
    mx.crypto::mxc_account_identity_keys(acct)$curve25519, "DEV",
    recipients = list(), sender_user_id = "@me:ex")
  names(out)
}
```

---

mx_crypto_handle_to_device

Decrypt an inbound Olm to-device payload

Description

Accepts an m.room.encrypted to-device content addressed to this device and returns the decrypted event. When it is an m.room_key, the caller builds an inbound Megolm session from content$session_key with mx_crypto_inbound_session().

Usage

```
mx_crypto_handle_to_device(
  account,
  my_curve25519,
  content,
  self_id = NULL,
  self_ed25519 = NULL,
  devices = NULL,
  olm_sessions = list()
)
```

Arguments

account	An mx.crypto account handle.
my_curve25519	Character. This device's Curve25519 key.
content	The to-device m.room.encrypted content.
self_id	Character or NULL. This user's Matrix id. When NULL the recipient user-id check is skipped; pass it whenever it is known.
self_ed25519	Character or NULL. This device's Ed25519 key. Defaults to the account's own key.
devices	List of verified devices from mx_crypto_known_devices(), or NULL. Used to tie the payload's claimed sender to a device whose keys were verified. The result carries sender_bound, which is FALSE when no list is supplied or nothing matches; an unbound sender identity is a claim, not a fact, because anyone can open an Olm session to this device.
olm_sessions	List of existing Olm session handles for this sender, whether locally or remotely initiated. Both normal and prekey messages try these sessions first; successful decryption advances the handle in place. With no matching session, only a prekey message can open one. Use mx_crypto_process_sync() to retain newly created sessions and persist the returned state with mx_crypto_sessions_save().

Details

The decrypted payload is checked against this device's identity before it is returned: a message that does not name us as recipient is dropped, because decrypting successfully only proves it was encrypted to our key, not that it was meant for us here.

Value

The decrypted event (a parsed list) with a sender_bound flag, or NULL if it was not for us, could not be decrypted, or failed the recipient checks. Undecryptable messages produce a warning.

Examples

```
## Not run:
ev <- mx_crypto_handle_to_device(acct, my_curve, td_content,
                                 self_id = "@me:example.org",
                                 devices = mx_crypto_known_devices(cl, uid))
if (identical(ev$type, "m.room_key") && isTRUE(ev$sender_bound)) {
  inb <- mx_crypto_inbound_session(ev$content$session_key)
}

## End(Not run)
```

mx_crypto_inbound_session

Build an inbound Megolm session from a shared room key

Description

Build an inbound Megolm session from a shared room key

Usage

```
mx_crypto_inbound_session(session_key)
```

Arguments

session_key	Character. The session_key from an m.room_key event.

Value

An inbound Megolm session.

Examples

```
## Not run:
inb <- mx_crypto_inbound_session(ev$content$session_key)

## End(Not run)
```

mx_crypto_known_devices

List the devices (and identity keys) of some users

Description

Queries /keys/query and flattens the result to a list of devices.

Usage

```
mx_crypto_known_devices(
  client,
  user_ids,
  strict = FALSE,
  self_master_key = NULL
)
```

Arguments

client	Matrix client config.
user_ids	Character vector of Matrix user ids.
strict	Logical. Treat a failures map as an error rather than a warning.
self_master_key	Character or NULL. Trusted local cross-signing master public key for client$user_id. This user's devices are cross-signed only when their verified chain matches this key. NULL leaves this user's devices not cross-signed; other users' chains are checked for internal consistency only.

Details

Devices are returned only if their Ed25519 self-signature verifies against the identity the homeserver claims for them. A device that fails is dropped with a warning naming it, and the remaining devices are still returned: one bad device must not make a room unusable.

/keys/query answers 200 with a failures map when it could not reach a server, and returns whatever it did manage. That is not the same as a user having no devices, so it is never silent: by default it warns, and with strict = TRUE it is an error. Encrypting to a user whose devices could not be listed is how a message ends up readable by nobody, so the send path asks for strict.

Value

List of verified devices, each list(user_id, device_id, curve25519, ed25519, cross_signed, master_key, identity_verified). identity_verified additionally requires the locally pinned master for this user, or its authenticated user-signing signature on another user's master, followed by that user's self-signing/device chain. This metadata does not change recipient policy or the existing device-level meaning of sender_verified. master_key is the server-reported key from a valid chain, even if it fails the pin.

Examples

```
## Not run:
mx_crypto_known_devices(client, "@bob:example.org")

## End(Not run)
```

---

mx_crypto_mark_key_requests_sent

Mark successfully transmitted room-key requests

Description

Updates durable request records after transport succeeds. Requests remain queued until marked, allowing a failed send to retry on a later poll even after the original encrypted event has left the sync timeline.

Usage

```
mx_crypto_mark_key_requests_sent(sessions, requests)
```

Arguments

sessions	An E2EE session set.
requests	Request descriptors sent successfully.

Value

The updated session set.

Examples

```
sessions <- mx_crypto_sessions_new()
request <- list(request_id = "example-request", sent = FALSE)
sessions$key_requests[["example-session"]] <- request
sessions <- mx_crypto_mark_key_requests_sent(sessions, list(request))
stopifnot(isTRUE(sessions$key_requests[["example-session"]]$sent))
# This only updates memory. The caller saves sessions after successful transport.
```

---

mx_crypto_process_sync

Process a sync response: store room keys, decrypt room events

---

Description

Handles inbound to-device `m.room.encrypted` (Olm) messages, storing direct `m.room_key` events and requested `m.forwarded_room_key` events as inbound Megolm sessions. It then decrypts timeline events whose session is known and queues stable, retryable `m.room_key_request` messages for missing sessions. The caller sends those requests to this user's other devices.

Usage

```
mx_crypto_process_sync(
  account,
  sessions,
  sync_resp,
  self_curve25519,
  self_id = NULL,
  devices = NULL,
  self_device_id = NULL
)
```

Arguments

account	An mx.crypto account handle.
sessions	A session set.
sync_resp	Parsed /sync response.
self_curve25519	Character. This device's Curve25519 key.
self_id	Character or NULL. This user's Matrix id, for is_self tagging and as the recipient of key requests.
devices	List of verified devices from mx_crypto_known_devices(), or NULL. Room keys arrive over Olm carrying a claimed sender, and anyone who can reach this device can send one, so the claim is only worth something once it is matched against a device whose device_keys verified. Without this list peers' decrypted events report sender_verified = FALSE: they still decrypt, but nothing attests to who sent them. Own echoes can be verified against the locally retained outbound session. Forwarded keys require this user's cross-signed devices, queried with a trusted local self_master_key; they never verify the original sender.
self_device_id	Character or NULL. This device id. Both this and self_id are required to create room-key requests.

Value

List with events (decrypted, normalized), updated verification_events (original verification envelopes, separated from chat messages; no handshake or network side effect is performed), sessions, unsent key_requests, matching key_request_cancellations, and incoming_key_requests for a policy-aware sharing layer to inspect.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
  acct <- mx.crypto::mxc_account_new()
  res <- mx_crypto_process_sync(acct, mx_crypto_sessions_new(),
    list(to_device = list(events = list()), rooms = list(join = list())),
    mx.crypto::mxc_account_identity_keys(acct)$curve25519)
  length(res$events)
}
```

---

mx_crypto_publish_keys

Publish this device's identity and one-time keys

Description

Builds and signs the device keys and a batch of one-time keys, uploads them with mx.api::mx_keys_upload(), marks them published, and persists the account. Call once after login and again to replenish one-time keys.

Usage

```
mx_crypto_publish_keys(client, account, store_dir, n_otks = 50L)
```

Arguments

client	Matrix client config (needs user_id, device_id).
account	An mx.crypto account handle.
store_dir	Character. Crypto store directory.
n_otks	Integer. Number of one-time keys to publish.

Value

The /keys/upload response, invisibly.

Examples

```
## Not run:
acct <- mx_crypto_account(mx_crypto_store_dir("corteza"))
mx_crypto_publish_keys(mx_client_load(app = "corteza"), acct,
                       mx_crypto_store_dir("corteza"))

## End(Not run)
```

---

mx_crypto_room_key_payload

Encrypt a Megolm room key to one device as a to-device payload

---

Description

Wraps the outbound Megolm session's key in an m.room_key event, Olm-encrypts it to the recipient device, and returns the m.room.encrypted to-device content to hand to mx.api::mx_send_to_device().

Usage

```
mx_crypto_room_key_payload(
  olm_session,
  sender_curve25519,
  recipient_curve25519,
  room_id,
  megolm_out,
  sender_user_id,
  sender_ed25519,
  recipient_user_id,
  recipient_ed25519
)
```

Arguments

olm_session	An outbound Olm session (mx.crypto::mxc_olm_create_outbound()).
sender_curve25519	Character. This device's Curve25519 key.
recipient_curve25519	Character. Target device's Curve25519 key.
room_id	Character. Room the key is for.
megolm_out	An outbound Megolm session.
sender_user_id	Character. This user's Matrix id.
sender_ed25519	Character. This device's Ed25519 key.
recipient_user_id	Character. Target user's Matrix id.
recipient_ed25519	Character. Target device's Ed25519 key.

Value

A named list: the to-device m.room.encrypted content.

Examples

```
## Not run:
content <- mx_crypto_room_key_payload(olm, my_curve, their_curve,
                                      "!room:ex", megolm_out,
                                      "@me:ex", my_ed, "@them:ex", their_ed)

## End(Not run)
```

---

mx_crypto_send_key_requests

Send queued Matrix room-key requests or cancellations

Description

Sends each descriptor returned by [mx_crypto_process_sync()] as an unencrypted m.room_key_request to every device of this user. The receiving clients apply their own verified-device sharing policy.

Usage

```
mx_crypto_send_key_requests(client, requests)
```

Arguments

client	Matrix client config.
requests	A list of request/cancellation descriptors returned by [mx_crypto_process_sync()].

Value

The endpoint responses, invisibly.

Examples

```
## Not run:
mx_crypto_send_key_requests(client, result$key_requests)

## End(Not run)
```

`mx_crypto_sessions_load`

Load a session set from the crypto store

Description

Accepts version 1 and unversioned legacy stores. Unknown or malformed schema versions are rejected before unpickling, without rewriting the file.

Usage

```
mx_crypto_sessions_load(store_dir)
```

Arguments

`store_dir`	Character. Crypto store directory.

Value

A session set (empty if nothing is stored yet).

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
  s <- mx_crypto_sessions_load(file.path(tempfile(), "crypto"))
}
```

mx_crypto_sessions_new

Create an empty E2EE session set

Description

Create an empty E2EE session set

Usage

```
mx_crypto_sessions_new()
```

Value

A session set: named lists `olm`, `olm_in`, `megolm_out`, `megolm_in`, and `key_requests`.

Examples

```
s <- mx_crypto_sessions_new()
names(s)
```

mx_crypto_sessions_save

Persist a session set to the crypto store

Description

Pickles every live session (encrypted at rest with the store key) into `sessions.json`. Reload with `mx_crypto_sessions_load()`. New files declare schema version 1; unversioned legacy files remain readable.

Usage

```
mx_crypto_sessions_save(sessions, store_dir)
```

Arguments

`sessions`	A session set.
`store_dir`	Character. Crypto store directory.

Value

The path written, invisibly.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
  dir <- file.path(tempfile(), "crypto")
  mx_crypto_sessions_save(mx_crypto_sessions_new(), dir)
}
```

`mx_crypto_store_dir` *Directory holding this client's encryption state*

Description

The crypto store keeps the pickled Olm account, the 32-byte key that encrypts those pickles at rest, and (later) per-peer Olm and per-room Megolm sessions. It lives beside the JSON config under `tools::R_user_dir()`.

Usage

```
mx_crypto_store_dir(app = "mx.client", path = NULL)
```

Arguments

`app`	Character. Application namespace.
`path`	Character or NULL. Explicit store directory.

Value

Character directory path.

Examples

```
mx_crypto_store_dir("myapp", path = tempfile())
```

`mx_crypto_user_trust` *Check a directional Matrix identity verification*

Description

Reads the homeserver's signature on another user's master key and verifies it through this user's locally pinned master and user-signing keys. Both identities must match their expected keys. This does not upload signatures, initialize a store, run sync, or modify encryption sessions.

Usage

```
mx_crypto_user_trust(client, store_dir, user_id, master_key)
```

Arguments

client	Matrix client config for the account whose trust is checked.
store_dir	Character. That account's existing cross-signing store.
user_id	Character. The other user's full Matrix id.
master_key	Character. The other user's full, unpadded base64 master public key, authenticated through a trusted independent channel. Do not obtain this pin solely from the homeserver response being checked.

Value

A list with user_id, master_key, signer_user_id, signer_master_key, and verified. FALSE means the expected identities were found but the directional trust signature was absent or invalid. Key mismatches error.

Examples

```
## Not run:
mx_crypto_user_trust(client, existing_store, "@peer:example.org", peer_pin)

## End(Not run)
```

---

mx_crypto_verify_user *Verify another Matrix user's independently authenticated identity*

Description

Signs the other user's pinned master key with this account's existing user-signing key. Uploads only a missing or invalid signature, then queries it back and verifies it before reporting success. No private keys are sent. A successful upload followed by a failed read-back may already have recorded trust; rerunning with the same authenticated key is safe.

Usage

```
mx_crypto_verify_user(client, store_dir, user_id, master_key)
```

Arguments

client	Matrix client config for the account granting trust.
store_dir	Character. That account's existing cross-signing store.
user_id	Character. The other user's full Matrix id.
master_key	Character. The other user's full master public key, authenticated independently of the homeserver being queried.

Details

This establishes one direction only. For mutual verification, preflight both accounts with `mx_crypto_user_trust()`, then call this function once as each account using the other account's independently checked pin. The two uploads are not atomic. No SAS handshake or history-key sharing is performed, and devices still need their own valid self-signing chains.

Value

The same status list as `mx_crypto_user_trust()`, with verified TRUE, invisibly. An error is raised if read-back does not confirm trust.

Examples

```
## Not run:
mx_crypto_verify_user(client, existing_store, "@peer:example.org", peer_pin)

## End(Not run)
```

`mx_extract_invites` *Extract pending invite room ids from a sync response*

Description

Extract pending invite room ids from a sync response

Usage

```
mx_extract_invites(sync_resp)
```

Arguments

`sync_resp`	Parsed `/sync` response.

Value

Character vector of invited room ids.

Examples

```
sync_resp <- list(rooms = list(invite = list("!inv:example.org" = list())))
mx_extract_invites(sync_resp)
```

`mx_extract_invite_records` *Extract pending invites with the member who sent them*

Description

The fuller form of `mx_extract_invites`, which reports only room ids. An invite's sender is the whole of whether it should be accepted – auto-joining anyone's invite hands a stranger a session with whatever the client can do – and a caller that wants to decide had to walk `invite_state` itself to find out.

Usage

```
mx_extract_invite_records(sync_resp, self_id = NULL)
```

Arguments

`sync_resp`	Parsed `/sync` response.
`self_id`	Current user's Matrix id, whose membership event names the inviter. NULL reports every invite with `inviter` NA.

Details

The sender comes from the stripped state the homeserver sends alongside an invite: the `m.room.member` event whose `state_key` is `self_id` and whose membership is `"invite"`. That is not always present, so `inviter` is NA when it is missing rather than guessed at, and a caller gating on it can tell "nobody I trust" from "I could not tell".

Stripped state carries no reliable `origin_server_ts`, so there is no timestamp here. An invite is a standing state, not an event at a moment.

Value

List of records, each `list(room_id, inviter)`.

Examples

```
sync_resp <- list(rooms = list(invite = list(`!inv:example.org` = list(
    invite_state = list(events = list(list(type = "m.room.member",
        state_key = "@bot:example.org", sender = "@ann:example.org",
        content = list(membership = "invite")))))))
mx_extract_invite_records(sync_resp, self_id = "@bot:example.org")
```

mx_extract_media_events *Extract media message events from a sync response*

Description

Walks joined-room timeline events and returns normalized media records: images, files, audio, and video sent as `m.room.message` events. Self events are retained and tagged with `is_self`, as in `mx_extract_text_events`.

Usage

```
mx_extract_media_events(
  sync_resp,
  self_id,
  msgtypes = c("m.image", "m.file", "m.audio", "m.video")
)
```

Arguments

`sync_resp`	Parsed `/sync` response.
`self_id`	Current user's Matrix id.
`msgtypes`	Character vector of media message types to include.

Details

Cleartext media carries its content address in `content$url`; end-to-end encrypted media carries a `content$file` object (address, key material, sha256) instead. `url` is filled from whichever is present, so a consumer that only wants to know where the bytes live reads one field. `file` rides verbatim, the `relates_to` treatment: decrypting is the caller's business, and an extractor inventing a partial view of key material would help no one.

`m.sticker` is its own event type, not an `m.room.message` msgtype, and is not reported here.

Value

List of normalized records, each carrying `room_id`, `event_id`, `sender`, `is_self`, `body` (the filename, or the caption when `filename` is set), `filename` (or NULL), `msgtype`, `ts` (or NULL), `url` (the mxc address, from `content$url` or `content$file$url`), `mime` (or NULL), `size` (or NULL), `sha256` (from the encrypted file object; cleartext media carries no hash, so NULL there), `encrypted`, `file` (the `content$file` object verbatim, or NULL), `mentions`, and `relates_to`.

An event with no address in either place is skipped: there is nothing a consumer could ever fetch, so reporting it would hand out a record whose one job is impossible.

Examples

```
sync_resp <- list(rooms = list(join = list("!room:example.org" = list(
    timeline = list(events = list(list(type = "m.room.message",
        event_id = "$1", sender = "@alice:example.org",
        content = list(msgtype = "m.image", body = "plot.png",
            url = "mxc://example.org/abc",
            info = list(mimetype = "image/png", size = 1024)))))))))
mx_extract_media_events(sync_resp, self_id = "@bot:example.org")
```

`mx_extract_reactions` *Extract reaction events from a sync response*

Description

Walks joined-room timelines and returns every `m.reaction` that annotates another event. Self reactions are retained and tagged with `is_self`, the same way `mx_extract_text_events` treats the client's own messages: a consumer that wants only other people's reactions filters on the flag, and one tracking its own (which reactions it has already placed) needs them.

Usage

```
mx_extract_reactions(sync_resp, self_id)
```

Arguments

`sync_resp`	Parsed `/sync` response.
`self_id`	Current user's Matrix id.

Details

This is the general form of `mx_extract_reaction_verdict`, which answers one approve/deny question about one event. That function bakes in which keys mean yes and which mean no, and stays for callers who want that; this one reports keys and leaves their meaning alone.

Only additions are reported. Removing a reaction is an `m.room.redaction` of the `m.reaction` event, which is not an `m.reaction` and so does not appear here – a consumer that has to notice un-reactions needs to read redactions itself.

Value

List of records, each carrying `room_id`, `event_id` (the reaction's own id, not its target), `sender`, `is_self`, `target_event_id` (the annotated event), `key` (the reaction text, usually an emoji), and `ts` (`origin_server_ts` in milliseconds, or NULL when the server omits it). An `m.reaction` carrying no annotation relation, no target, or no key is not a reaction to anything and is skipped.

Examples

```
sync_resp <- list(rooms = list(join = list("!room:example.org" = list(
    timeline = list(events = list(list(type = "m.reaction",
        event_id = "$r1", sender = "@alice:example.org",
        content = list("m.relates_to" = list(rel_type = "m.annotation",
            event_id = "$msg", key = "+1")))))))))
mx_extract_reactions(sync_resp, self_id = "@bot:example.org")
```

`mx_extract_reaction_verdict` *Extract a reaction approval verdict from sync events*

Description

Scans a room timeline for a reaction on `target_event_id` from someone other than `self_id`. Returns TRUE for approval keys, FALSE for denial keys, or NULL when no verdict is present.

Usage

```
mx_extract_reaction_verdict(
  sync_resp,
  room_id,
  self_id,
  target_event_id,
  approve_keys = NULL,
  deny_keys = NULL
)
```

Arguments

`sync_resp`	Parsed `/sync` response.
`room_id`	Character room id.
`self_id`	Current user's Matrix id.
`target_event_id`	Event id being reacted to.
`approve_keys`	Character vector of reaction keys read as approval. `NULL` (default) uses thumbs-up (U+1F44D), check-mark (U+2705), and `"y"`/`"yes"`/`"ok"`.
`deny_keys`	Character vector of reaction keys read as denial. `NULL` (default) uses thumbs-down (U+1F44E), cross-mark (U+274C), and `"n"`/`"no"`/`"nope"`.

Value

TRUE, FALSE, or NULL.

Examples

```
sync_resp <- list(rooms = list(join = list("!room:example.org" = list(
    timeline = list(events = list(list(type = "m.reaction",
        sender = "@alice:example.org",
        content = list("m.relates_to" = list(rel_type = "m.annotation",
            event_id = "$msg", key = "yes")))))))))
mx_extract_reaction_verdict(sync_resp, "!room:example.org",
                            self_id = "@bot:example.org",
                            target_event_id = "$msg")
```

---

mx_extract_text_events

Extract text message events from a sync response

Description

Walks joined-room timeline events and returns normalized text-message records. Self events are retained and tagged with `is_self`.

Usage

```
mx_extract_text_events(sync_resp, self_id, msgtypes = "m.text")
```

Arguments

`sync_resp`	Parsed `/sync` response.
`self_id`	Current user's Matrix id.
`msgtypes`	Character vector of message types to include.

Value

List of normalized event records, each carrying `room_id`, `event_id`, `sender`, `is_self`, `body`, `msgtype`, `ts` (the event's `origin_server_ts`, in milliseconds since the epoch, or NULL when the server omits it), `mentions`, and `relates_to`.

`relates_to` is the event's `m.relates_to` content, verbatim, or NULL. It is what distinguishes a threaded reply (`rel_type` of `m.thread`) from a rich reply (an `m.in_reply_to` with no `rel_type`) from an edit (`m.replace`), and dropping it left every caller unable to tell any of them from an ordinary message. Passed through rather than interpreted: which of those a caller cares about is its own business.

Examples

```
sync_resp <- list(rooms = list(join = list("!room:example.org" = list(
    timeline = list(events = list(list(type = "m.room.message",
        event_id = "$1", sender = "@alice:example.org",
        content = list(msgtype = "m.text", body = "hello")))))))))
mx_extract_text_events(sync_resp, self_id = "@bot:example.org")
```

mx_markdown_to_html *Convert a conservative markdown subset to Matrix custom HTML*

Description

Supports headings, bullets, numbered lists, fenced code blocks, inline code, bold, simple underscore emphasis, and GitHub-style pipe tables.

Usage

```
mx_markdown_to_html(text)
```

Arguments

text	Character markdown body.

Value

Character HTML suitable for m.room.message formatted_body.

Examples

```
mx_markdown_to_html("# Status\n- built\n- checked\n\nShip `0.1.0` **soon**")
```

mx_pill_mentions *Turn textual @mentions into Matrix pills in formatted HTML*

Description

Turn textual @mentions into Matrix pills in formatted HTML

Usage

```
mx_pill_mentions(html, user_ids)
```

Arguments

html	Character HTML (e.g. from mx_markdown_to_html).
user_ids	Character Matrix user ids, such as "@jorge:example.org".

Value

HTML with textual @localpart occurrences replaced by matrix.to links. Unmatched user ids leave the HTML unchanged; they can still be placed in m.mentions by mx_send_text().

Examples

```
mx_pill_mentions("<p>ping @jorge</p>", "@jorge:example.org")
```

`mx_resolve_room` *Resolve a room id, name, or default room*

Description

Resolution order: literal room IDs beginning with `!`, a supplied `room_cache` name-to-id map, joined-room display-name lookup, then the config's `room_id` fallback.

Usage

```
mx_resolve_room(
  client,
  room = NULL,
  room_cache = NULL,
  fallback = TRUE,
  details = FALSE,
  quiet = FALSE
)
```

Arguments

`client`	Matrix client config.
`room`	Character or NULL.
`room_cache`	Named list or character vector mapping names to ids.
`fallback`	Logical. Use `client$room_id` when lookup misses.
`details`	Logical. Return source metadata instead of just the id.
`quiet`	Logical. Suppress fallback message.

Value

Character room id, or a list when `details = TRUE`.

Examples

```
client <- list(room_id = "!default:example.org")
# Literal ids and cache hits resolve without a server round-trip:
mx_resolve_room(client, "!abc:example.org")
mx_resolve_room(client, "general",
                room_cache = list(general = "!gen:example.org"))
mx_resolve_room(client) # NULL room falls back to the config default
```

mx_room_encrypted *Is a room end-to-end encrypted?*

Description

Resolves the room (by name, id, or the config default) and reads its `m.room.encryption` state. Needs mx.api >= 0.3.0.

Usage

```
mx_room_encrypted(client, room = NULL, room_cache = NULL)
```

Arguments

client	Matrix client config.
room	Character room id/name or NULL for the default room.
room_cache	Optional room name-to-id cache.

Value

TRUE when the room advertises an encryption algorithm, FALSE otherwise.

Examples

```
## Not run:
if (mx_room_encrypted(client, "secret plans")) {
    # use mx_send_encrypted() instead of mx_send_text()
}

## End(Not run)
```

mx_room_lookup_by_name *Look up a joined room by display name*

Description

Look up a joined room by display name

Usage

```
mx_room_lookup_by_name(client, name)
```

Arguments

client	Matrix client config.
name	Character room name.

Value

Room id, or NULL when no joined room has that name.

Examples

```
## Not run:
# Needs a live homeserver session.
mx_room_lookup_by_name(mx_client_load("myapp"), "general")

## End(Not run)
```

`mx_sas_accept` *Accept an incoming SAS verification request*

Description

Accept an incoming SAS verification request

Usage

```
mx_sas_accept(sas, now = Sys.time())
```

Arguments

`sas`	An in-memory SAS transaction.
`now`	Current time.

Value

The transaction, invisibly. A ready event is queued, not sent.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    mx_sas_accept(bob)
    stopifnot(identical(mx_sas_status(bob)$phase, "ready"))
}
```

mx_sas_cancel *Cancel a SAS transaction without granting trust*

Description

Cancel a SAS transaction without granting trust

Usage

```
mx_sas_cancel(sas, code = "m.user")
```

Arguments

sas	An in-memory SAS transaction.
code	Matrix cancellation code.

Value

The transaction, invisibly. At most one cancellation is queued.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    mx_sas_cancel(bob)
    stopifnot(identical(mx_sas_status(bob)$cancel_code, "m.user"),
        !mx_sas_status(bob)$local_trust_recorded)
}
```

mx_sas_confirm *Confirm or reject the displayed Matrix SAS*

Description

Call only after the human compares the display through a trusted channel. TRUE queues MACs for this device and its master key. Both a valid peer MAC and local confirmation are required before the phase becomes verified. This alone does not upload cross-signing signatures.

Usage

```
mx_sas_confirm(sas, matches, now = Sys.time())
```

Arguments

sas	An in-memory SAS transaction displaying a SAS.
matches	One explicit TRUE or FALSE, without a default.
now	Current time.

Value

The transaction, invisibly.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_receive", package = "mx.client", echo = FALSE)
    # Demonstrate declining the comparison. No identity trust is granted.
    mx_sas_confirm(alice, matches = FALSE)
    stopifnot(identical(mx_sas_status(alice)$cancel_code, "m.mismatched_sas"),
        !mx_sas_status(alice)$local_trust_recorded)
}
```

---

mx_sas_console *Compare a Matrix SAS through a trusted interactive console*

Description

Drives one transaction using the application's existing event consumer. No second sync loop is created by this function. The human must explicitly type yes after comparing all seven emoji or all three numbers with the peer. Empty input, no, or cancellation grants no trust. Do not connect the input callback to an LLM or a Matrix room.

Usage

```
mx_sas_console(sas, receive, send, complete, input = sas_console_read)
```

Arguments

sas	An in-memory transaction from mx_sas_from_request().
receive	Function with no arguments returning a list of original Matrix events from the sole event consumer. It should wait briefly.
send	Function accepting one outgoing envelope. It must throw on failure and use the envelope's id for idempotent retries.
complete	Function accepting sas, recording and checking durable trust with mx_sas_record_trust(), outside the crypto commit window.
input	Function taking a prompt. The default requires an interactive console; replacement is intended for a trusted human UI or isolated tests.

Value

A status list, invisibly. Interrupts cancel and attempt notification.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    # Isolated refusal example. Real input must come from a trusted human UI.
    sent <- new.env(parent = emptyenv())
    sent$events <- list()
    result <- mx_sas_console(bob,
        receive = function() list(),
        send = function(event) {
            sent$events[[length(sent$events) + 1L]] <- event
        },
        complete = function(sas) stop("No trust should be recorded"),
        input = function(prompt) "no")
    stopifnot(identical(result$phase, "cancelled"),
        !result$local_trust_recorded, length(sent$events) == 1L,
        identical(sent$events[[1]]$type, "m.key.verification.cancel"))
}
```

---

mx_sas_from_request *Create a SAS transaction from an incoming verification request*

Description

Validates the request's age, recipient, device and method, then reads a fixed key snapshot. No sync, transport, new identity, or trust upload occurs. Requests are accepted only when the operator calls mx_sas_accept().

Usage

```
mx_sas_from_request(client, store_dir, event, now = Sys.time())
```

Arguments

client	This device's Matrix client config.
store_dir	This device's existing crypto store.
event	Original request envelope from the existing event consumer.
now	Current time.

Value

An in-memory SAS transaction, or NULL for an irrelevant/expired request.

Examples

```
# Ordinary messages are ignored without reading a store or contacting a server.
client <- mx_client_from_config(list(user_id = "@alice:example.org",
    device_id = "ALICE"))
event <- list(type = "m.room.message", sender = "@bob:example.org",
    content = list(msgtype = "m.text", body = "Hello"))
unused_store <- tempfile("unused-crypto-store-")
stopifnot(is.null(mx_sas_from_request(client, unused_store, event)),
    !file.exists(unused_store))
# See mx_verify_console() for receiving real requests using an existing store.
```

`mx_sas_outgoing` *Read or acknowledge queued SAS protocol messages*

Description

Send outside any cryptographic state commit window. Acknowledge an id only after transport succeeds. Retries retain the same id and payload. Never display SAS codes in the Matrix conversation being verified.

Usage

```
mx_sas_outgoing(sas, acknowledge = character())
```

Arguments

`sas`	An in-memory SAS transaction.
`acknowledge`	Character vector of successfully sent queue ids.

Value

A list of pending envelopes, containing type, content, destination, and a stable transport id. No private key material is included.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    mx_sas_accept(bob)
    pending <- mx_sas_outgoing(bob)
    stopifnot(length(pending) == 1L,
        identical(mx_sas_outgoing(bob), pending))
    # Deliver locally, then acknowledge. Real transports acknowledge on success.
    item <- pending[[1]]
    mx_sas_receive(alice, list(sender = "@bob:example.org",
        type = item$type, content = item$content))
    mx_sas_outgoing(bob, acknowledge = item$id)
    stopifnot(length(mx_sas_outgoing(bob)) == 0L)
}
```

`mx_sas_receive` *Receive one standard Matrix SAS protocol event*

Description

Ignores other senders, rooms, devices, and transaction ids. Invalid messages in this transaction cancel it. Duplicate identical messages do not repeat operations; conflicting repeats cancel. NULL checks timeouts. Feed decrypted original event type and content, not flattened chat text. No transport or persistent trust operation occurs here.

Usage

```
mx_sas_receive(sas, event = NULL, now = Sys.time())
```

Arguments

`sas`	An in-memory SAS transaction.
`event`	A Matrix event envelope with type, sender, content, and room_id for in-room verification, or NULL.
`now`	Current time.

Value

The transaction, invisibly.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    # Deliver queued envelopes between the two synthetic, in-memory peers.
    relay <- function(from, to) {
        for (item in mx_sas_outgoing(from)) {
            mx_sas_receive(to, list(type = item$type, content = item$content,
                sender = mx_sas_status(from)$user_id, room_id = item$room_id))
            mx_sas_outgoing(from, acknowledge = item$id)
        }
    }
    mx_sas_accept(bob)
    for (i in seq_len(3)) {
        relay(bob, alice)
        relay(alice, bob)
    }
    alice_status <- mx_sas_status(alice)
    bob_status <- mx_sas_status(bob)
    stopifnot(identical(alice_status$phase, "sas"),
        identical(bob_status$phase, "sas"),
        length(alice_status$decimal) == 3L,
        identical(alice_status$decimal, bob_status$decimal),
```

```
        !alice_status$confirmed, !alice_status$local_trust_recorded)
    # A real exchange still requires a human comparison on trusted displays.
}
```

`mx_sas_record_trust` *Record trust after a human-confirmed, authenticated SAS exchange*

Description

Rechecks the fixed identity snapshot, signs the peer master with this user's user-signing key (or its own peer device with its self-signing key), and verifies server read-back. Only then is done queued. A peer's done acknowledgement does not expose or prove its private user-signing key.

Usage

```
mx_sas_record_trust(sas, client, store_dir, now = Sys.time())
```

Arguments

`sas`	A SAS transaction with both human confirmation and valid peer MACs.
`client`	This device's Matrix client config.
`store_dir`	This device's existing cross-signing store.
`now`	Current time.

Value

The transaction, invisibly. Errors leave completion retryable.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    client <- mx_client_from_config(list(user_id = "@alice:example.org",
        device_id = "ALICE"))
    unused_store <- tempfile("unused-crypto-store-")
    # An unconfirmed exchange cannot grant trust or access the crypto store.
    try(mx_sas_record_trust(alice, client, unused_store))
    stopifnot(!mx_sas_status(alice)$local_trust_recorded,
        !file.exists(unused_store))
}
# mx_verify_console() calls this after human confirmation and valid peer MACs.
```

`mx_sas_session` *Create an in-memory Matrix SAS transaction*

Description

Holds a fixed snapshot of both parties' device and master public keys. Feed this object events from the application's existing event consumer; this function does not start sync, send messages, or record trust. Ephemeral state cannot be persisted. Restart interrupted transactions with a new request id and new session. Only modern SAS algorithms are used.

Usage

```
mx_sas_session(
  user_id,
  device_id,
  keys,
  peer_user_id,
  peer_device_id,
  peer_keys,
  transaction_id,
  room_id = NULL,
  initiator = FALSE,
  now = Sys.time()
)
```

Arguments

`user_id`	This account's Matrix user id.
`device_id`	This account's device id.
`keys`	Named list of this device's Ed25519 key and locally trusted master key. Names are ed25519:device_id and ed25519:master_public_key.
`peer_user_id`	The selected peer user id.
`peer_device_id`	The selected peer device id.
`peer_keys`	Named list containing the peer device and master public keys fetched and validated before the handshake. SAS authenticates this exact snapshot, not later replacements from a homeserver.
`transaction_id`	Initial request event id for room verification, or the initial to-device request's transaction_id.
`room_id`	Room id, or NULL for to-device verification.
`initiator`	TRUE if this device sent the initial request.
`now`	Current time. Injectable for deterministic timeout tests.

Value

An opaque in-memory transaction. Use mx_sas_status() for display.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    # Synthetic identities for a local example: no server or store is used.
    make_keys <- function(device) {
        public <- replicate(2, mx.crypto::mxc_signing_key_public(
            mx.crypto::mxc_signing_key_new()))
        stats::setNames(as.list(public),
            paste0("ed25519:", c(device, public[2])))
    }
    alice_keys <- make_keys("ALICE")
    bob_keys <- make_keys("BOB")
    alice <- mx_sas_session("@alice:example.org", "ALICE", alice_keys,
        "@bob:example.org", "BOB", bob_keys, "example-request",
        initiator = TRUE)
    bob <- mx_sas_session("@bob:example.org", "BOB", bob_keys,
        "@alice:example.org", "ALICE", alice_keys, "example-request")
    stopifnot(identical(mx_sas_status(alice)$phase, "requested"))
}
```

---

mx_sas_start *Begin the SAS key agreement after request negotiation*

Description

Begin the SAS key agreement after request negotiation

Usage

```
mx_sas_start(sas, now = Sys.time())
```

Arguments

sas	An in-memory SAS transaction in the ready phase.
now	Current time.

Value

The transaction, invisibly. A start event is queued, not sent.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    mx_sas_accept(bob)
    mx_sas_start(bob)
    stopifnot(identical(mx_sas_status(bob)$phase, "started"))
}
```

mx_sas_status *Inspect a Matrix SAS verification transaction*

Description

Display codes only on the operator's trusted console, never in the Matrix conversation being verified. A verified SAS is distinct from a recorded local trust signature and from the peer's completion acknowledgement.

Usage

```
mx_sas_status(sas)
```

Arguments

sas	An in-memory SAS transaction.

Value

A list with identities, phase, comparison values, local confirmation, peer MAC validity, local trust status, peer completion, and cancellation code. cancel_detail identifies a locally diagnosed missing peer master proof.

Examples

```
if (requireNamespace("mx.crypto", quietly = TRUE)) {
    example("mx_sas_session", package = "mx.client", echo = FALSE)
    status <- mx_sas_status(alice)
    stopifnot(identical(status$phase, "requested"),
        !status$confirmed, !status$local_trust_recorded)
}
```

mx_send_encrypted *Send an end-to-end encrypted message to a room*

Description

Discovers the room members' devices (unless `recipients` is given), claims one-time keys for any without an Olm session, shares the room key over to-device, encrypts the content with Megolm, sends the `m.room.encrypted` event, and persists the updated sessions.

Usage

```
mx_send_encrypted(
  client,
  account,
  sessions,
  room_id,
  content,
  store_dir,
  recipients = NULL,
  member_ids = NULL
)
```

Arguments

client	Matrix client config.
account	An mx.crypto account handle.
sessions	A session set (see mx_crypto_sessions_new()).
room_id	Character room id.
content	Named list. Plaintext event content.
store_dir	Character. Crypto store directory.
recipients	List of recipient devices, or NULL to discover them from member_ids. An empty list is an error: an m.room.encrypted event whose room key was shared with nobody is unreadable by everyone, and returning its event id would report that as a successful send. Discovery finding nobody is allowed, since a room whose other members have no devices is a real room.
member_ids	Character vector of room member user ids (used when recipients is NULL).

Value

List with event_id and the updated sessions.

Examples

```
## Not run:
res <- mx_send_encrypted(client, acct, sessions, "!r:ex",
  list(msgtype = "m.text", body = "secret"), store,
  member_ids = "@bob:example.org")

## End(Not run)
```

`mx_send_media` *Send a media file to a Matrix room*

Description

Client-layer wrapper over `mx.api::mx_send_media()`: resolves the room by name (or falls back to the config's default room), builds the session from the client config, and uploads + posts in one call. The msgtype is derived from the file's MIME type unless given.

Usage

```
mx_send_media(
  client,
  path,
  room = NULL,
  body = basename(path),
  msgtype = NULL,
  content_type = NULL,
  info = list(),
  room_cache = NULL,
  dry_run = FALSE
)
```

Arguments

`client`	Matrix client config.
`path`	Character. Path to the file to upload.
`room`	Character room id/name or NULL for the default room.
`body`	Character. Message body / filename shown by clients.
`msgtype`	Character or NULL. NULL derives it from the MIME type.
`content_type`	Character or NULL. MIME type override for files whose extension guesses wrong (tempfiles, odd extensions); NULL guesses from the extension.
`info`	List. Extra fields merged into the media `info`.
`room_cache`	Optional room name-to-id cache.
`dry_run`	Logical. Print instead of uploading/sending.

Details

If you attach mx.api and mx.client together, namespace-qualify – the two packages export an `mx_send_media` each (session-first there, client-first here).

Value

Event id, or NULL on dry-run.

Examples

```
client <- list(room_id = "!default:example.org")
png <- file.path(tempdir(), "plot.png")
file.create(png)
mx_send_media(client, png, dry_run = TRUE)
unlink(png)
```

`mx_send_table` *Send tabular data to a Matrix room*

Description

Sends a plain-text fallback body plus Matrix custom HTML table in `formatted_body`. This bypasses Markdown entirely.

Usage

```
mx_send_table(
  client,
  x,
  room = NULL,
  header = TRUE,
  title = NULL,
  room_cache = NULL,
  dry_run = FALSE
)
```

Arguments

`client`	Matrix client config.
`x`	A data frame, matrix, or list coercible to a data frame.
`room`	Character room id/name or NULL for the default room.
`header`	Logical. Include a header row using column names.
`title`	Optional text prepended to the plain fallback body.
`room_cache`	Optional room name-to-id cache.
`dry_run`	Logical. Print instead of sending.

Value

Event id, or NULL on dry-run.

Examples

```
client <- list(room_id = "!default:example.org")
mx_send_table(client, data.frame(A = 1, B = 2), dry_run = TRUE)
```

mx_send_text *Send plain text to a Matrix room*

Description

Send plain text to a Matrix room

Usage

```
mx_send_text(
  client,
  text,
  room = NULL,
  msgtype = "m.text",
  room_cache = NULL,
  dry_run = FALSE,
  markdown = FALSE,
  mentions = NULL,
  thread = NULL
)
```

Arguments

client	Matrix client config.
text	Character message body.
room	Character room id/name or NULL for the default room.
msgtype	Character Matrix message type.
room_cache	Optional room name-to-id cache.
dry_run	Logical. Print instead of sending.
markdown	Logical. If TRUE, include Matrix custom HTML derived from a conservative markdown subset.
mentions	Character vector of Matrix user ids to mention (e.g. "@jorge:cornball.ai"). Each id is added to the event's m.mentions (so the user is notified) and any textual @localpart in the body becomes a matrix.to pill in the HTML. Implies an HTML formatted body even when markdown is FALSE – pills only render from HTML.
thread	Event id of a thread root, or NULL for an ordinary message. The event is sent as a threaded reply (m.relates_to with rel_type "m.thread"). It also carries the reply fallback the spec asks for – is_falling_back with an m.in_reply_to pointing at the root – so a client that does not implement threads renders the message as a reply to the root rather than as a loose message in the room. Without it those clients show a threaded conversation as unattached chatter.

Value

Event id, or NULL on dry-run.

Examples

```
client <- list(room_id = "!default:example.org")
mx_send_text(client, "release is out", dry_run = TRUE)
## Not run:
# A real send needs a live homeserver session:
client <- mx_client_load("myapp")
mx_send_text(client, "release is out", markdown = TRUE,
             mentions = "@jorge:example.org")

## End(Not run)
```

`mx_set_displayname` *Set the account's profile display name*

Description

Client-level wrapper over `mx.api::mx_set_displayname()`: builds the session from the client config and retries once through `mx_with_relogin` when the homeserver rejects a rotated token. The display name is account-global and shows in the sender line of every message the account posts.

Usage

```
mx_set_displayname(client, name, save = TRUE)
```

Arguments

`client`	Matrix client config.
`name`	Character. New display name.
`save`	Logical. On a relogin, persist the refreshed token to the client's config path (see `mx_client_relogin`).

Value

TRUE, invisibly.

Examples

```
## Not run:
# A real rename needs a live homeserver session:
client <- mx_client_load("myapp")
mx_set_displayname(client, "mybot (maintenance)")

## End(Not run)
```

`mx_sync_update` *Sync once and update the stored cursor*

Description

Calls `mx.api::mx_sync()` using `client$sync_token`, stores the returned `next_batch` in a returned client object, and optionally saves it back to disk.

Usage

```
mx_sync_update(
  client,
  timeout = 0L,
  filter = NULL,
  save = TRUE,
  path = NULL,
  app = NULL
)
```

Arguments

`client`	Matrix client config.
`timeout`	Integer long-poll timeout in milliseconds.
`filter`	Character or NULL. Matrix sync filter.
`save`	Logical. Persist the updated client config.
`path`	Character or NULL. Save destination.
`app`	Character or NULL. Application namespace for default saves.

Value

List with `sync`, `client`, and `first_run`.

Examples

```
## Not run:
# Needs a live homeserver session.
client <- mx_client_load("myapp")
res <- mx_sync_update(client, timeout = 30000L)
events <- mx_extract_text_events(res$sync, client$user_id)

## End(Not run)
```

mx_table_html *Render tabular data as Matrix custom HTML*

Description

Produces the conservative table shape rendered by Matrix clients such as FluffyChat 2.6.0+: a bare <table> containing <tr>, <th>, and <td> nodes. No CSS, colspan, rowspan, or custom attributes are emitted.

Usage

```
mx_table_html(x, header = TRUE)
```

Arguments

x	A data frame, matrix, or list coercible to a data frame.
header	Logical. Include a header row using column names.

Value

Character HTML suitable for Matrix formatted_body.

Examples

```
mx_table_html(data.frame(A = 1:2, B = c("x", "y")))
```

mx_verify_console *Verify a Matrix client interactively from an R console*

Description

Owns the device's existing sync cursor and crypto store while waiting for a verification request from the selected user and room. In the other client, choose Start verification, then compare the emoji or numbers in its UI with this trusted console. No private client database, desktop keyring, or remote user's private signing key is read.

Usage

```
mx_verify_console(
  client,
  store_dir,
  peer_user_id,
  room_id = NULL,
  exclusive = FALSE,
  timeout = 120,
  on_messages = verification_print_messages
)
```

Arguments

client	Client loaded with mx_client_load() from its existing config.
store_dir	This device's existing, initialized crypto store.
peer_user_id	Full Matrix id of the person being verified.
room_id	Exact room id, or NULL for to-device requests.
exclusive	Must explicitly be TRUE after stopping other consumers of this device and store. FALSE refuses before any network or store access.
timeout	Maximum seconds to wait for an initial request, 1 to 600.
on_messages	Function receiving ordinary messages read while the console owns sync. The default reports only their count, keeping untrusted chat text out of the verification prompts; content is returned in messages.

Details

Stop any bot or other process using this device before calling. The exclusive argument is an explicit operator assertion, not a process lock. Restart that process only after this call returns. Ordinary messages read during this session go to on_messages and are returned for review; a bot will not automatically process messages whose cursor the console advanced. Existing event-loop hosts can instead use mx_sas_from_request() and mx_sas_console() with their own receive/send callbacks. Each attempt reloads the saved cursor and credentials, refusing a changed server, user, or device identity rather than replaying the caller's old cursor.

Value

Invisibly, a list with status, updated client, and ordinary messages.

Examples

```
## Not run:
# First stop the service using this exact device and crypto store.
client <- mx_client_load(path = config_path)
result <- mx_verify_console(client, existing_store,
    "@peer:example.org", "!room:example.org", exclusive = TRUE)
# In the peer's Matrix app, choose Start verification.
# Restart the service after the console returns.

## End(Not run)
```

---

mx_with_relogin *Run a client operation, re-logging in once on an expired token*

Description

Calls `fn(client)`; if it fails with the server's invalid-token error (`M_UNKNOWN_TOKEN`, signalled as a classed condition by mx.api >= 0.3.0), re-logs in via `mx_client_relogin` and retries once with the refreshed client. Any other error propagates.

Usage

```
mx_with_relogin(client, fn, save = TRUE)
```

Arguments

client	Matrix client config with password.
fn	Function taking a client config.
save	Logical. Persist the refreshed config on relogin.

Value

fn's return value.

Examples

```
## Not run:
mx_with_relogin(client, function(cl) {
    mx_send_text(cl, "still here after a token rotation")
})

## End(Not run)
```

---

print.mx_client_config

Print a Matrix client config

Description

Prints the config field by field with credentials masked, so that an interactive session, a screenshot, or a pasted bug report does not leak the access token or password. Secret fields show whether they are set (<hidden>) or empty (<unset>) without showing the value. Use unclass(x) or str(unclass(x)) when the raw credentials are genuinely needed.

Usage

```
## S3 method for class 'mx_client_config'
print(x, ...)
```

Arguments

x	An mx_client_config, as returned by mx_client_load() or mx_client_from_config().
...	Ignored.

Value

x, invisibly.

Examples

```
cfg <- mx_client_from_config(list(server = "https://matrix.example.org",
                                  token = "syt_secret_value",
                                  user_id = "@bot:example.org",
                                  device_id = "DEVICEID"))
cfg
```

Index