Return messages

No error

```
{
  "success": True,
  "result": ...
}
```

Error

```
{
  "success": False,
  "result": "Error message here."
}
```

Protocols

The value `"priv"` will represent the client's personal private identity key.

Registration

1. Call `generatePrivateKey()` twice. Once for the identity key and once for SPK.
2. Save the private keys and get public keys for each.
3. Construct signature of spk with signature(`priv`, pub spk)
4. Initalise own storage to whatever is convenient.
5. Emit **`"register"`** to server with (username, pub key, pub spk, sig, own_storage)
6. Follow the login procedure as registration does not log you in.

Login

1. Emit login to server with (username).
2. Server responds with a challenge.
3. Call generateChallengeResponse(`priv`, challenge)
4. Emit login_challenge_response to server with (challenge response)

DM creation

1. Get usernames to make DM with (except yourself).
2. Get the key bundle of each username (except yourself). **`"get_user"`** -> (pub, spk, sig). This needs to fail if no spk is uploaded for a user.
3. Call `createGroupDM(priv, bundles)`. Fails if any spk is not genuine.
4. Emit usernames, messages and key tree to (**`"create_dm"`**)
5. Server responds with `dm_id` or failure
6. (`sharedKey, dm_id`) needs to be saved for use (can be used immediately)

On the recipient's side, they get a notification (**`"x3dh_notification"`**) -> (sender, pub, spk, ek, key_tree, position, dm_id). This will not be sent to the initiator.

1. Verify the keys received (ik, spk)
2. Retrieve private spk associated with the public spk received
3. Call `recvGroupDM(priv, pub, priv spk, ek, tree, position)`
4. (`sharedKey, dm_id`) needs to be saved for use (can be used immediately)

These notification events are also sent on initial login if there are any pending requests.

Message sending

1. Get `dm_id, message`
2. Get schedule and delete relative times (zero if not)
3. Encode message as hex string with TextEncoder and bytesToHex
4. Encrypt the message with the shared key for this dm
5. Construct signature of encrypted message with signature(`priv`, message)
6. Emit **`"send_message"`** to server with (`dm_id, message, signature, schedule, delete`)

All users that are part of the dm will get a notification (**`"message_notification"`**) **including** the sender.

1. Verify signature of message with verify(pub, message, signature)
2. Decrypt the message with the shared key for this dm (can fail if something is incorrect)
3. Decode message to string with hexToBytes and TextDecoder

Scheduled messages will be notified one minute before the scheduled send (if the message was scheduled more than one minute into the future). This is to give a direct notification to the user.

Adding reactions

1. Get message_id, reaction
2. Encode reaction as hex string with TextEncoder and bytesToHex
3. Encrypt the reaction with the shared key for this dm
4. Construct signature of encrypted reaction with signature(`priv`, reaction)
5. Emit **`"add_reaction"`** to server with (`reaction, signature, message_id`)

All users that are part of the dm will get a notification (**`"message_notification"`**) **including** the sender.

4. Verify signature of reaction with verify(pub, reaction, signature)
5. Decrypt the reaction with the shared key for this dm (can fail if something is incorrect)
6. Decode reaction to string with hexToBytes and TextDecoder

Friending

1. Get username of target.
2. Emit username to **`"send_friend_request"`** event.

On the recipient's side, they get a notification (**`"friend_request_notification"`**) -> sender

1. The get_friend_requests event can be used to get a list of all incoming friend requests on demand.
2. Emit a username and accept boolean to **`"ack_friend_request"`** event.

The acknowledgement of a friend request generates a notification to the sender. There are no notifications for blocks.

Server events

login(username)
`returns: challenge`

login_challenge_response(response)
`returns: True`

register(username, public_key, spk, sig, own_storage)
This does not log in the user.
`returns: True`

username_exists(username)
`returns True/False`

get_user(username)
`returns: User object EXCEPT own_storage`

get_full_user()
`returns: User object for yourself`

get_user_list()
`returns: [User, User, ...] EXCEPT own_storage of each`

set_user(props)
`props: User object EXCEPT "username" & "public_key" (other keys are optional)`
Extra keys will be silently ignored. If "spk" is given, "sig" is mandatory.
`returns: True`

create_dm(usernames, messages, key_tree)
example:

```
{
  "usernames": ["Agent", "Smith", ...],
  "messages": [x3dh, x3dh, ...],
  "key_tree": ["01...", "02...", ...]
}
usernames is a list of username strings
messages is a list of x3dh messages (from crypto.ts)
key_tree is a list of public keys (from crypto.ts)
returns: id of the new dm
```

get_dms()
`returns: [1, 2, ...] /* dm ids that this user is a part of */`

```
get_dm(id)
returns: DM object with "latest_message" and "scheduled_messages"
keys added.
"latest_message" is a Message object
"scheduled_messages" is a dictionary of scheduled message ids:
{
  1: {
      "message": "The answer is 42.", /* Encrypted */
      "signature": "0102...",
      "timestamp": "2023-07-22T01:58:59.123456" /* Estimated send
timestamp */
  },
  ...
}
```

```
set_dm(props)
props: "id", "name" (id is mandatory)
```
Extra keys will be silently ignored
```
returns True
```

```
leave_dm(id)
returns True
```

```
send_message(id, message, signature, schedule, delete)
```
Schedule/delete are relative times in seconds from now. Set schedule/delete to zero to disable that functionality. Delete timer starts *after* schedule.
```
returns: True
```

```
get_message(id)
returns: Message object
```

```
get_message_history(id, cursor, count)
```
Not inclusive of cursor.
```
returns: [Message, Message, ...]  sorted from the given dm id
```

```
get_pinned(id):
returns: [Message, Message, ...] sorted from the given dm id
```

```
set_message(props)
props: "id", "message", "signature", "pinned"
```
"id" key is mandatory.
Extra keys will be silently ignored. If "message" is given, "signature" is mandatory and must be your own message/signature.

cancel_scheduled_message`(dm_id, schedule_id)`
Call `get_dm` to get your scheduled message ids.
`returns: True`

add_reaction`(id, reaction, signature)`
Adds a reaction to the given message id.
`returns: id of the reaction`

remove_reaction`(id)`
Removes the reaction with the given id.
`returns: True`

ping_typing`(id)`
`returns True`

send_friend_request`(username)`
Unblocks target if applicable.
`returns True`

get_friend_requests`()`
```
returns ["Agent", "Smith", ...] /* list of usernames from incoming
friend requests */
```

get_outgoing_requests`()`
```
returns ["Agent", "Smith", ...] /* list of usernames from outgoing
friend requests */
```

ack_friend_request`(username, accept)`
Unblocks target if applicable. This is only possible if they sent a request before you blocked them.
`returns True`

unfriend`(username)`
`returns True`

get_friends`()`
```
returns ["Agent, "Smith", ...] /* List of usernames of friends */
```

block_user`(username)`
Unfriends target if applicable. Retract friend request if applicable.
`returns True`

unblock`(username)`
`returns True`

get_blocked()
returns ["Agent, "Smith", ...] /* List of blocked usernames */

join_call(id, uuid)
returns: dictionary of usernames and their uuid

leave_call(id)
returns: True

Notifications

"**profile_notification**" (any updates to a user get notified)
User object EXCEPT own_storage

"**dm_notification**" (name change, call list change, and users leaving)
DM object

"**typing_notification**"
{"id": 1, "username": "Joe"}

"**message_notification**" (new message, changes to message)
Message object

"**message_change_notification**" (changes to message)
Message object

"**message_delete_notification**" (message deleted)
Auto triggered when a self destruct timer expires.
Message object id

"**scheduled_message_sent_notification**" (scheduled message executed)
Auto triggered when a schedule timer expires.
{"dm_id": dm_id, "schedule_id": schedule_id}

"**scheduled_soon_notification**" (scheduled message to be sent in one minute)
Auto triggered.
{"dm_id": dm_id, "schedule_id": schedule_id}

"**x3dh_notification**" (invited to a new dm)
X3DH notification object

"friend_request_notification"
```
{"username": "Joe"}
```

"friend_request_accept_notification"
```
{"username": "Joe", "accept": true} /* Can be false */
```

"unfriend_notification"
```
{"username": "Joe"}
```

"user_joined_call"
```
{"id": 1, "username": "Joe"}
```

"user_left_call"
```
{"id": 1, "username": "Joe"}
```

Objects

User object
```
{
  "username": "Joe",
  "public_key": "01...",
  "spk": "02...", /* Can be null */
  "sig": "03...", /* sig(spk, private_key) */
  "status": "offline", /* Default is online at register/login time */
  "biography": "The answer is always 42.", /* Default is empty string
*/
  "profile_picture": "", /* Default is empty string */
  "own_storage": "" /* Default is empty string */
}
```

DM object
```
{
  "id": 1,
  "users": ["Agent", "Smith", ...], /* List of username strings */
  "name": "Simulacra and Simulation", /* Default value is null */
  "created_at": "2023-07-22T01:58:59.123456",
  "users_in_call": ["Agent", "Smith", ...]
}
```

Message object
```
{
  "id": 1,
  "dm_id": 1, /* The dm that this message is from */
  "sender": "Joe",
  "message": "Why is the answer 42?", /* Encrypted */
  "signature": "01...",
  "timestamp": "2023-07-22T01:58:59.123456",
  "delete_timestamp": "2023-08-22T01:58:59.123456",
  "pinned": true, /* Default value is false */
  "reactions": [
     {
     "id": 1,
     "sender": "Bob",
     "reaction": "01...", /* Encrypted */
     "signature": "02..."
     },
     ...
  ]
}
```

X3DH notification object

```
{
  "sender": "Joe Blogs",
  "ik": "01...", /* This is the ik of the sender */
  "spk": "02...",
  "ek": "03...",
  "key_tree": ["04...", ...],
  "position": 3,
  "id": 1 /* The dm id it is related to */
}
```