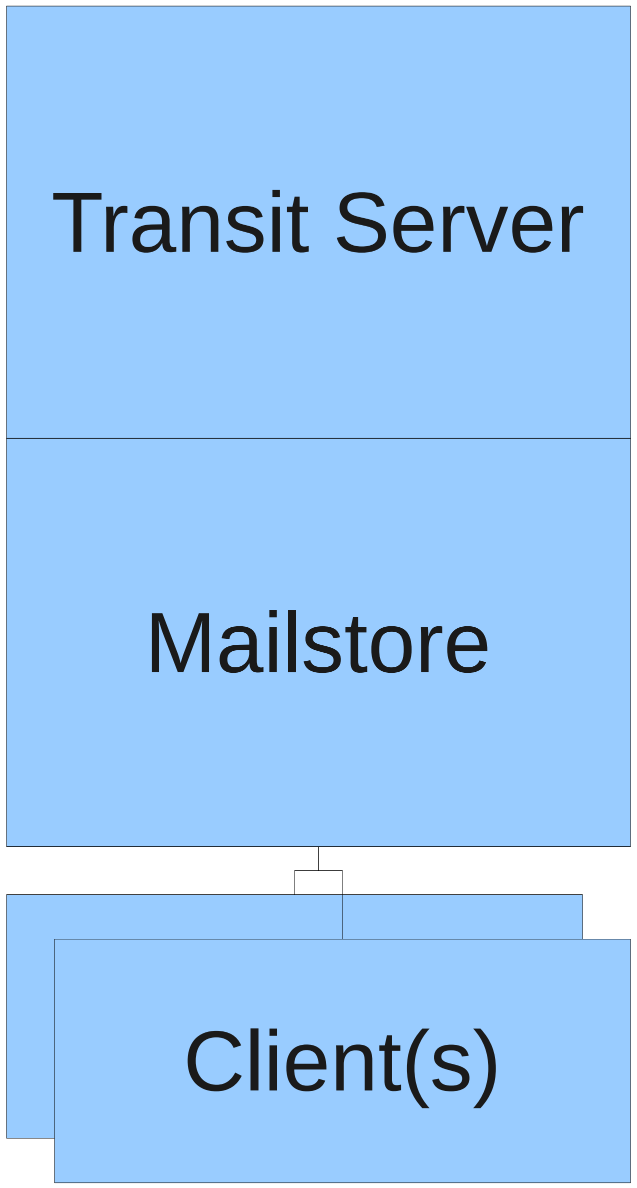
Deuxdrop.

Deuxdrop: Many Things

- Mobile Messaging UX
  - Foot-in-the-door
  - Simpler start, not intended to only be IM-ish.
- Federated Messaging Experiment:
  - Strong Identity
  - Conversation-centric
  - Layered crypto: graduated trust of servers
- Messaging Client and Server Infrastructure
  - Pure JS (save for crypto libs)
  - Attempts to be multi-device aware

Federated Overview

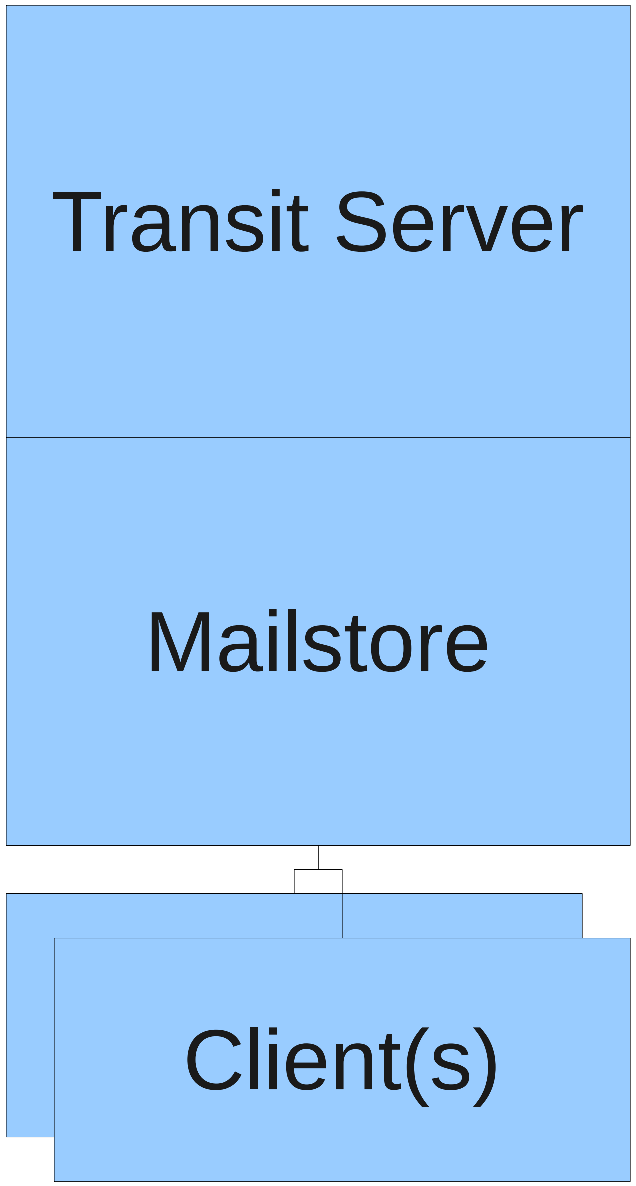


- The unit of federation.
- Receives messages.
- Sends messages.
- Conversation maestro
- Sees traffic, some routing info

- Authoritative message store for the user.
- Sees message envelopes, unless you want it to see more (FTS).

- Is the user.
- Sees / can read everything.

Federated Overview (2)



•High Availability.
•Needs to be publicly reachable.
•Anyone can run one.

•Only needs to be as available as the clients need.
•Dial-out-only.
•Anyone can run one.

UX Reqs: Identity / Connections

- Friending idiom: Mutual establishment of contact relationship required to communicate/ (initiate communication).
- Unsolicited contact requests are a thing, and can include message text, but they live in their own UI universe. They don't show up in your inbox as spam.
- Exact flow still needs to be iterated.

Crypto: Currently using DJB nacl

- Boxing: (gen keypair: 0.1ms, box: ~0.1ms, unbox: ~0.1ms)
  - Generates the **same** shared secret from one user's public key and another user's private key.
  - Requires nonce.
  - Authenticates and encrypts message.
- Secret Boxing:
  - Uses shared key, authenticates and encrypts.
- Public-key signatures (gen: ~4.4ms, sign: ~4.4ms, verify: ~11.5ms)
- Secret key message authentication

Crypto: Every day user keys

- Envelope Box Key: Used to encrypt envelope data sent to the user. Provided to the mailstore.
- Body Box Key: Used to encrypt body data sent to the user. Could be provided to the mailstore.
- “Announce” Signing Key: Signs messages sent to conversations or any case where individually boxed messages are not viable. Kept just to the clients.
- Tell Key: Authorship key for boxed messages (to servers, other users, etc.) Kept just to the clients.

UX Goals: Conversations

- Be able to add someone to a conversation at any time.
- The added person gets the backlog of the conversation with the same fidelity as if they had been involved in the conversation from the get-go.
- Received/read watermarks.
- Participants only need be “friends” with the conversation participant who invites them.

Crypto: Conversations (1)

- Two secret-keys generated: envelope, body
- Each user gets boxed copies of the keys (encrypted with the respective key) in their 'welcome' message.
- The 'fanout' role of the transit server only knows the participants. It sees encrypted payloads. It wraps the blobs saying when it saw the message and who it appears to be from.
- Participants send messages to the transit server to send a message to the conversation.

Crypto: Conversations (2)

- Because of the approved/mutual contact requirement and the non-requirement for all participants in a conversation to be contacts, you can end up in a conversation hosted by a non-contact.
- So when you invite someone to a conversation:
  - you: “yo, authorize this conv and let me know”
  - their transit server: “authorized!”
  - your transit server to conversation transit server: “add this person”

Crypto: Conversations (3)

- You furnish an “other person identity” when you add someone to a conversation. It is:
  - Their signed self identity:
    - Crypto keys, authorization chain from their root, etc.
    - Transit server self-identity (keys + url)
    - Their PoCo (Portable Contacts) Rep (“Lord James Jimmington the Eighth, earl of Shoesville”)
  - Your Poco Rep/changes for them (“Pompous Jim”)
  - Signed by you.
- Allows us to display: “Andrew's friend Pompous Jim” (SDSI/SPKI/pet names style.)

Crypto: Conversations (4)

- Conversation identified by the public key of a freshly generated signing keypair when started.
  - Allows unambiguous 'ownership' of the conversation and expression of the appropriate transit server to use.
  - Could be used for conversation migration if the user changes servers.

Mailstore Goals

- Provide user-agency, filtering, convenience:
  - Be aware of active devices. You pocket doesn't need to vibrate if the tablet/laptop on your lap is telling you the same thing.
  - Unified notification state across all devices (with take-back); you don't need to read stale notifications on N-1 devices.
  - Your phone doesn't need to know about everything going on in the universe unless you want it to.
- Efficient/cheap disk I/O characteristics.

Mailstore/Client Interaction

- Client subscribes to some subset of all possible message data. (Currently all).
- Mailstore queues changes for clients as they happen. If the queue overflows because the client falls too far behind (currently impossible), the client will start from scratch.
- Communication is simple req/ack that goes realtime when caught up. Transport is currently CurveCP-ish crypto over Websockets for persistent TCP conn for power efficiency goals.

Clients and Trust

- No blind trust of the mailstore; an IMAP server can define the client's reality, a mailstore cannot.
- State to be shared amongst clients sent 2 ways:
  - Secret-boxed when the mailstore does not need to know. Useful for: data/metadata that does not need to be indexed or where the indexing can be done on an opaque value. (The data is still associated with a specific context.)
  - Authenticated (HMAC) when the mailstore does need to see inside.
- Clients always 'replay' operations, even locally; replication is not a separate code-path.

Client Overview: UI

- UI logic interfaces via JS API (as opposed to REST or message passing as API.)
- UI window/thread communicates with 'client daemon' window/thread via postMessage-style bridge.
- The UI issues queries whose results are updated as changes happen until the queries are killed.
- UI actions are assumed to succeed (apart from signup). Errors to be reported via their own awareness/interaction required channel.

Fin.

- Extra slides follow.
- Unit tests on:
  - http://clicky.visophyte.org/examples/arbpl-loggest/20110914/

Comparisons in Brief

- Wave:
  - The wave server is in control/sees everything.
  - Wave has operational transform magic, mutable messages, complex threading.
- XMPP/Jabber (core-wise):
  - transport layer; we could build on top of it.
  - Aka: much more instant-messaging client biased.
  - Pub/sub XEP not really comparable

Key Hierarchy

- Not actually trying to reinvent the wheel, just wanted to make sure the code would be aware of the possibility of multiple keys / etc.
- Root signing key is the all-powerful key for the identity.
  - It only is used to authorize long-term signing keys.
  - Can be kept offline in a deep mine-shaft.
- Long-term signing keys authorize keys for specific purposes.

Client Overview: 'Daemon'

- Data stored using IndexedDB
- Simple queries on known views with pre-built indices.