



POSSIBILITY SPACE • FULL USE CASE

03

The conversation nobody can prove happened

A private messenger where the network can't see who is talking, when, or that a message was sent at all.

BUILD STATE

Yours to take

COMPLEXITY



Vision

End-to-end encryption made the contents of a message unreadable and left everything around them in the open. Message someone through an ordinary secure app and the app still routes it through central servers, and those servers see who you are, who you contacted, and exactly when. That record of contact, rather than anything written in the messages, is often the part that can be used against a person. It is far smaller than the messages themselves, and far easier to store and act on.

A journalist and a confidential source never share a secret in plaintext; every message between them is encrypted. A hostile government or a compromised network provider can still see that these two identities spoke, repeatedly, at particular hours, in the days before a story broke, without reading a word of it. The relationship itself is the evidence: for the source, the encryption worked perfectly and protected nothing that mattered.

Some of that exposure sits off the network entirely. Devices are seized at borders every day, often while still unlocked and with an app running in memory, and no change of transport fixes what is on a phone in someone else's hands. What changes is what survives the device. With no account and no server-side history, there is no contact list held elsewhere to subpoena afterwards, and no log that outlasts the hardware.

On the network side, Swarm closes the gap by removing the server. No central point carries your conversation from you to someone else; a message is a piece of encrypted content sitting in a shared network, looking exactly like every other piece of content in it. An observer watching that network sees data moving, the same data that carries photos, backups, and web pages, with nothing to mark which of it is a message or who it is for. From the outside there is no record that the two of you spoke, so neither of you can be made to prove it.

That is a stronger property than unreadable contents. The *existence* of the conversation is private too, and that is the part a source or an organiser actually needs.

Architecture

Swarm reaches metadata privacy by treating messaging as storage rather than delivery. One participant writes content; another reads it. Nothing is ever sent to anybody, so there is no delivery event for an observer to catch. Every mechanism below works toward the same end: making a private conversation look, to the network, like ordinary storage activity.

Content addressing and chunks. Everything on Swarm is split into fixed-size chunks addressed by the hash of their contents and scattered across the network. A chunk carrying an encrypted message is byte-for-byte indistinguishable from a chunk of a photo or a database backup. Without the right key, it is not identifiable as a message and cannot be linked to any other chunk: there is no envelope, header, or recipient field to read.

PSS and Trojan chunks (shipped). PSS lets one node inject an encrypted “Trojan chunk” so that it routes into the recipient's neighbourhood and is picked up when their node next syncs, even if the recipient was offline when it was sent. The sender needs only the recipient's public key and a short prefix of their address; the network handles it as ordinary content. This is how the first handshake and the “you have a message” signal travel without ever creating a connection between the two parties. One deployment note: receiving a Trojan chunk requires a full node, since light nodes do not sync and store chunks and PSS never reaches them. First contact arrives only as a Trojan chunk, so anyone

who wants to be reachable by a stranger runs a full node. Feeds carry a conversation once it is open; they cannot open one.

Single Owner Chunks and Feeds (shipped). A Single Owner Chunk is an address in the network reserved to one Ethereum keypair; a Feed is a series of these chunks at predictable addresses, giving a person a stable, append-only, signed timeline that others can find and follow. Each user keeps an *outbox feed*: they encrypt a message, sign it, and post it as the next entry. The recipient, who knows where to look, reads the entry and decrypts it. Making each successive entry land at an address only the two participants can predict, so that an observer cannot even count the messages, follows naturally from this design, but it is not a shipped feature today. Treat it as a design goal rather than a promise.

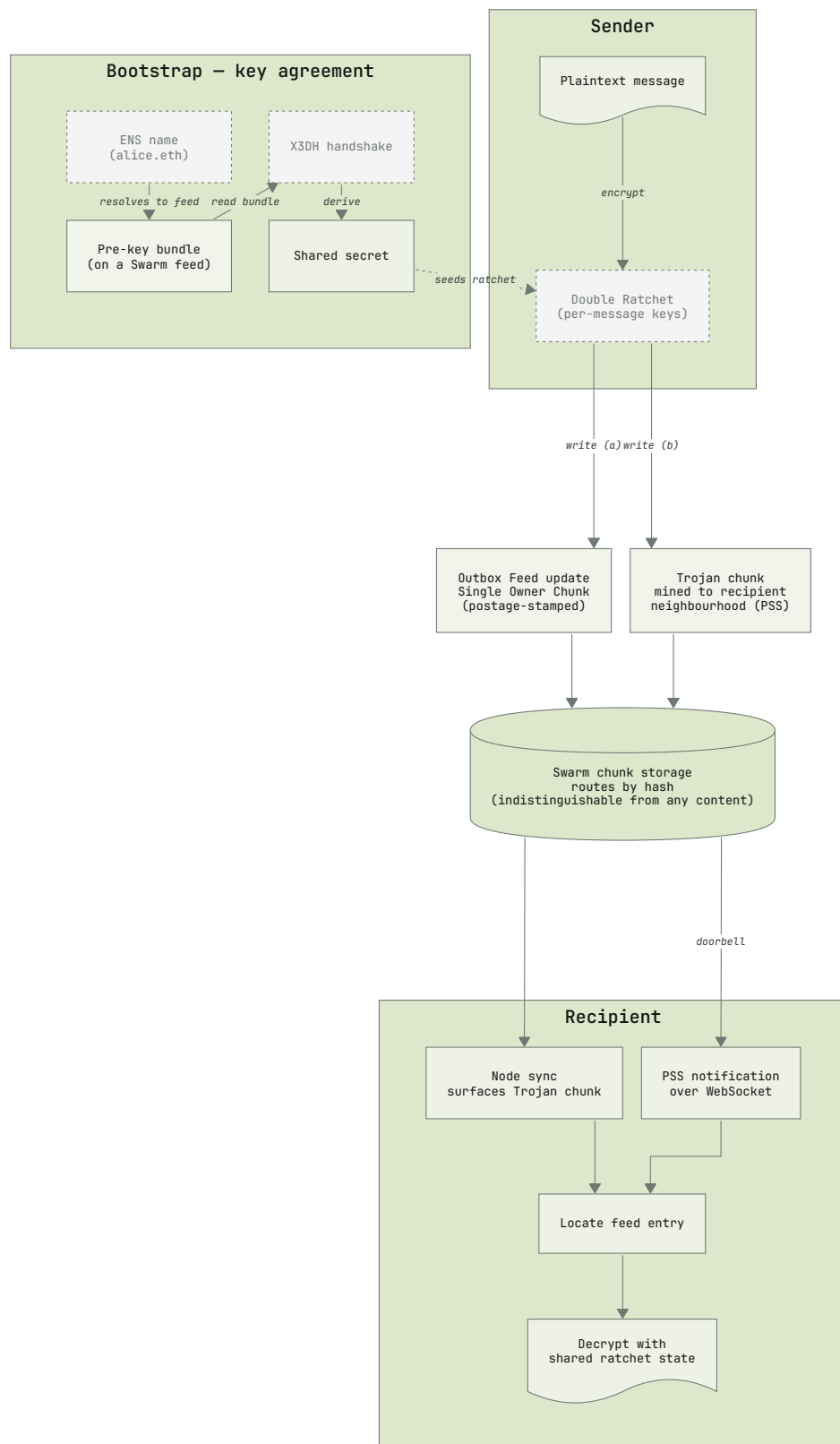
Postage stamps (shipped). Every write to Swarm (each feed update, each Trojan chunk) is paid for with a postage stamp, which is what keeps the data resident on the network with no server underwriting it.

Three pieces of this system are deliberately **not** Swarm's own, and the design leans on them rather than reinventing them: **ENS** (Ethereum Name Service) lets a human-readable name like `alice.eth` point at someone's keys and feeds; **X3DH** and the **Double Ratchet** are Signal's proven key-agreement and per-message-key protocols. Swarm supplies the private transport and storage; these supply the cryptographic core.

PRIMITIVES

Content addressing Single Owner Chunks Feeds PSS / Trojan chunks Postage stamps

EXTERNAL DEPENDENCIES ENS X3DH Double Ratchet



Data flow. Plaintext is encrypted with the Double Ratchet and written two ways, as an outbox Feed update and as a PSS Trojan chunk, both entering Swarm's chunk storage as ordinary content. The recipient is nudged by a PSS notification, locates the feed entry, and decrypts. A separate bootstrap lane derives the shared secret. To the network, none of it is marked as a message.

Development

Signal's cryptography stays as it is; what changes is the transport underneath it. Signal's servers do two jobs: distribute pre-key bundles and relay messages. Both jobs move onto Swarm, and the client-side crypto barely changes.

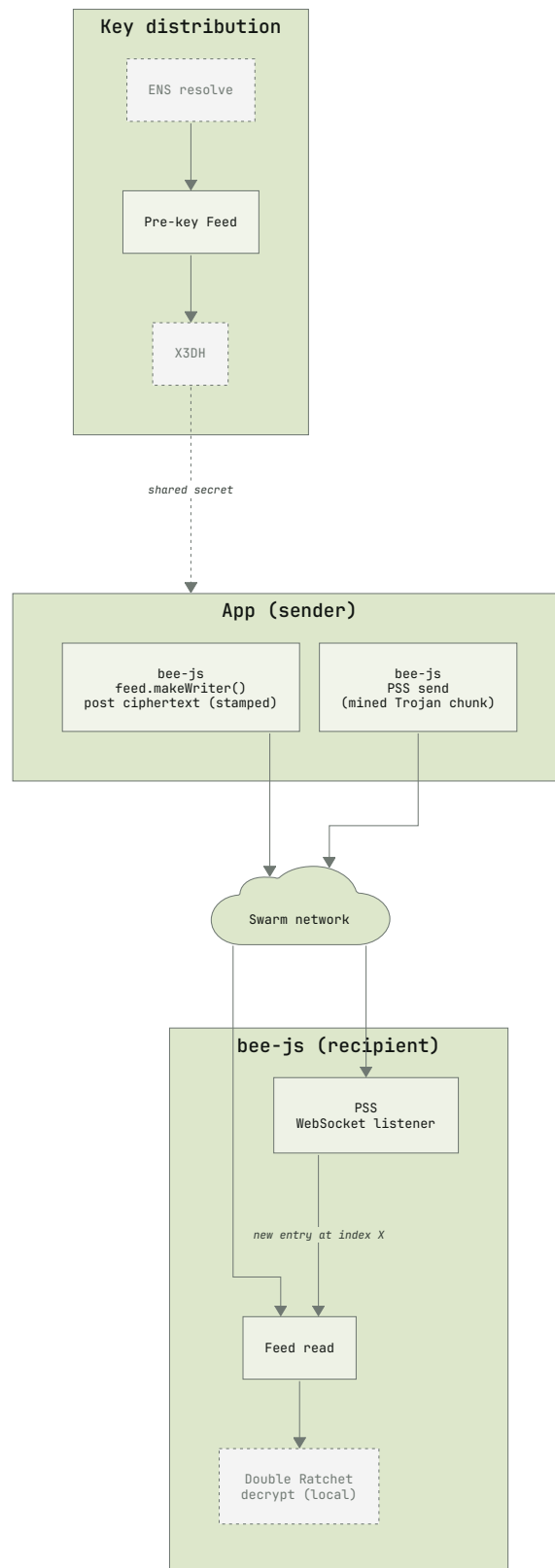
Bootstrap the conversation. Publish each user's pre-key bundle (identity key, signed pre-key, one-time keys) to a Swarm **Feed**, optionally addressed by an **ENS** name so a correspondent can find it as `alice.eth` instead of a raw hash. (The bundle must stay openly fetchable: anyone initiating first contact needs it, and their keys can't be known in advance.) To start a chat, the initiator reads the bundle from the feed and runs the **X3DH** handshake (*external: Signal spec*) locally to derive a shared secret. Deliver that very first handshake message with **PSS**: wrap it in a Trojan chunk mined to the recipient's neighbourhood so it lands with no metadata even while they're offline. One deployment note: this first handshake arrives only as a Trojan chunk, so the recipient picks it up the next time their node syncs, which takes a full node. Because X3DH is asynchronous, timing doesn't matter: the chunk stays in the network until they sync it.

Run the conversation. Move to the **Double Ratchet** (*external: Signal spec*) for per-message keys, giving forward and future secrecy. Post each ciphertext as the next entry in a per-recipient **outbox feed** (a signed, postage-stamped Single Owner Chunk). For near-real-time delivery, fire a **PSS** Trojan chunk ping, a "new entry at index X" doorbell, which their node receives over a WebSocket; the message body itself is fetched from the feed.

Harden it. Metadata privacy is only as strong as your indexing. Sequential feed indices are predictable and let an observer count traffic. Deriving each next index from shared ratchet state, so that only the two participants can find it, is the goal; nothing off the shelf does this today, so budget design and review time for it. Decide early how each participant hears about new messages: listen for the PSS ping for real-time delivery, or poll the outbox feed on an interval. The message body lives on the feed either way.

TOOLS & REFERENCES

- Bee documentation — <https://docs.ethswarm.org> (PSS, Feeds, postage stamps)
- bee-js library — <https://bee-js.ethswarm.org>
- PSS — <https://docs.ethswarm.org/docs/develop/tools-and-features/pss/>
- Feeds — <https://docs.ethswarm.org/docs/develop/tools-and-features/feeds/>
- **External dependencies (not Swarm primitives):** Signal X3DH specification · Signal Double Ratchet specification · ENS (ens.domains)



Component assembly. The app drives bee-js: a feed writer posts postage-stamped ciphertext and a PSS send mines a Trojan chunk into the Swarm network; the recipient's bee-js reads the feed, listens on a PSS WebSocket, and decrypts locally with the Double Ratchet. A key-distribution branch resolves ENS to a pre-key Feed feeding X3DH. External protocol boxes are drawn dashed to mark them non-Swarm.