

The Spojiti White Paper

Version 2.0 · 22 October 2026 · CognitionX Logic India Pvt Limited. The full technical description of Spojiti: identity without identifiers, the post-quantum session protocol, sealed media, anonymous entitlements, Echoes, the infrastructure, the threat model and what our servers can and cannot see.

Abstract

Spojiti is an end-to-end encrypted messenger designed around a single constraint: the service must be unable to identify its users. No user record has a field for a phone number, e-mail or name. A user who opts in to discovery publishes a blind-signature token of their number; because the relay holds the signing key, someone operating the relay who already knows a number can test whether it is registered. A match gives the finder no account: a hello is a knock sealed to the owner's discovery key, and it reaches the owner only if the finder's number is in the owner's own address book. An identity is an X25519 agreement key and an Ed25519 signing key, both derived on the device from a twelve-word BIP-39 recovery phrase; the relay knows the account by a random UUID. Sessions use X3DH, extended with an ML-KEM-1024 encapsulation when both devices pass the post-quantum self-test. The account-level signed prekey is checked under Ed25519 and, where the peer has published one, ML-DSA-65; each per-device prekey is signed under the account's Ed25519 key. When a session was established with the post-quantum leg against a current (ML-KEM-1024) prekey, every DH ratchet step also mixes in a fresh ML-KEM-768 secret. Media is sealed per item under AES-256-GCM. Paid tiers are unlocked with RSA blind-signature tokens, so the issuer cannot link a token to the purchase that paid for it. The app presents the token when it connects, so the relay knows a connected account's tier while it is connected; it stores no account-to-tier record. Public communities ("Echoes") run on per-community pseudonyms behind blind attunement tokens, from tables that hold no account data; a box of their own is designed and not yet running. This paper describes each construction, the infrastructure that carries them, and the residual metadata, what the relay necessarily learns, with the mitigations for each item.

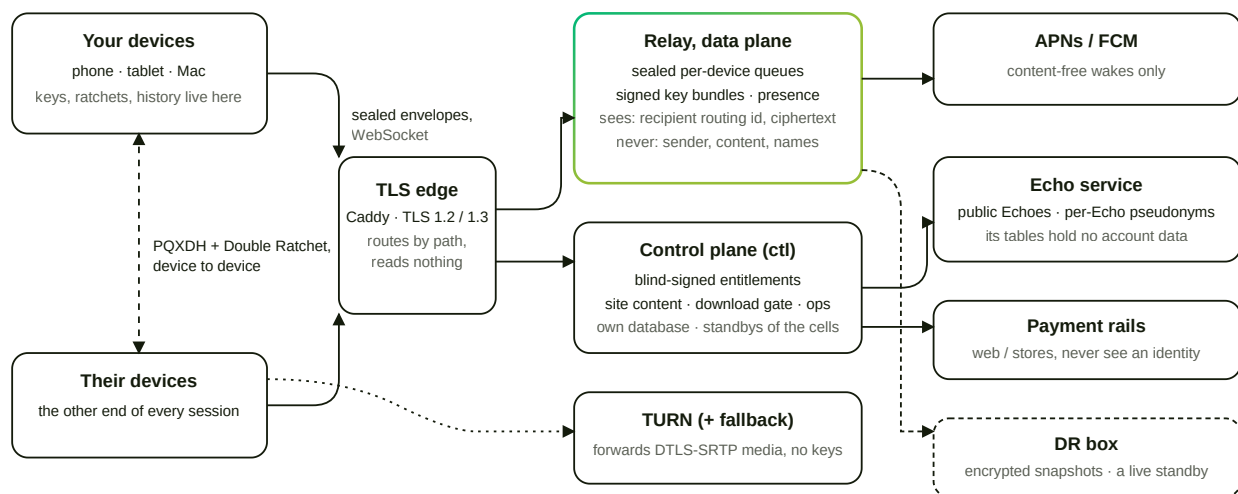
Preface: how to read this paper

This document describes what Spojiti does, in enough detail for a competent third party to evaluate it without our help. It is written against the application and relay source as they stand for build 391 (October 2026). Every construction below is stated the way the code implements it. Where something is a design that has not shipped, or a limitation we have chosen to live with, the text says so in the same paragraph rather than in a footnote. A claim we could not check against the source has been left out.

Spojiti is built by CognitionX Logic India Pvt Limited. The cryptographic and transport core of the product is published as source under the AGPL v3 as Spojit Private Messenger Core (the `spojiti-core-app` and `spojiti-core-relay` repositories), assembled from the same files that ship in the product by a generator that refuses to drift; the full product is not open source. An independent audit of the full product is in progress; this paper is the specification that audit works from, and its findings will be published unredacted with our fixes.

Architecture at a glance

Everything a user's device talks to, and what each part can and cannot see. Solid lines carry sealed data the far end cannot read; the dashed line is the one plaintext exchange, and it happens on the device, not with us.



The boundary that matters is the dashed line on the left: the keys that open a conversation exist only on the two ends of it. Everything to the right of the TLS edge handles ciphertext, routing identifiers and tokens that were designed to be safe in our hands: the relay has no payment record, and the standby copies and encrypted backups of its databases hold the keys to no conversation.

1 · Identity without identifiers

1.1 The design constraint

There is no name, phone or e-mail field for users. A test fails if any table or column is added without being declared and justified in the published data inventory. Everything else in this paper follows from this one decision: if the service cannot know who a user is, it cannot be made to say.

1.2 The recovery phrase and the key tree

An identity is created on the device, offline, by generating a twelve-word BIP-39 recovery phrase. The phrase never leaves the device in readable form; optional linking and recovery features carry it encrypted under keys we do not hold. From it the app derives, deterministically and using HKDF-SHA-256:

Account identity key	Ed25519 for signing and X25519 for key agreement. This is the identity. It is recreated on any device from the phrase, which is why an identity can be restored without us and why we cannot restore one for you; from build 422 the relay also asks for the account's Spojit PIN before it lets a new device sign in (1.5).
Post-quantum signing key	ML-DSA-65 (FIPS 204, security level 3). Signs, beside Ed25519, the account's classical signed prekey and its post-quantum prekey; when both people have one, the safety number covers it too.
Device key	Ed25519, generated per install and deliberately not derived from the phrase, so that one device's record can be removed without touching the identity.
Device signed prekey	X25519, from the per-install device seed; published in the relay's device registry with the account identity's Ed25519 signature over it (from build 383). A peer adopts a device's prekey only when that signature verifies; every supported build publishes one, and an unsigned device prekey is refused.
Post-quantum prekey	ML-KEM-1024 (FIPS 203, security level 5; the parameter set Signal's PQXDH and Apple's PQ3 use for the same long-lived key). Published beside the classical prekey, signed twice: Ed25519 and ML-DSA-65.

The device identifier is `base64url(SHA-256(device_public_key)[0..9])`, twelve characters derived from a per-install random key; no hardware or advertising identifier is used to identify a device to us. Device records are signed by the account identity key over a domain-separated string, and from build 383 so is each device's prekey. Peers check the prekey signature before encrypting to a device, so a device a hostile relay invented, a prekey it swapped, or a prekey sent without a signature is refused.

1.3 Handles, pairing and the safety number

The relay routes to a random account identifier, not to anything a person chose. Two people pair by scanning a QR code, by opening a shared invite link, or by pasting an Account ID; a **:Handle** is a display name, not a lookup key, and the relay keeps no directory of them. Each pairing produces a 60-digit safety number from the two identity public keys, the two account ids and, when both have one, the two ML-DSA-65 signing keys, so it is identical on both devices and changes if either identity key changes. Comparing it over any other channel detects an interception of the session.

1.4 What this means for account recovery

If the recovery phrase is lost and no optional recovery was set up (a passkey, trusted contacts or another linked device), the identity is lost. There is no reset e-mail, no support ticket that can restore it and no back door for us to open: every copy of the secret that defines the identity is on the user's own devices or sealed under keys only the user, or the contacts they chose, can open. We say this plainly in the app at the moment the phrase is shown, and again here.

A lost DEVICE is a different case, and the phrase answers it too. The app derives a lost-device code from the phrase (120 bits, its own HKDF label), so every device of the identity and any phone restored from the 12 words shows the same code, and registers only its SHA-256 with the relay. With the account id and that code, the owner reports the loss on our support page; the control plane hashes the code and asks the cells, and only the cell that holds the account can match it. Every device record that existed at that moment is quarantined at once: its connection is closed, it is refused until the report fires, and every device of the account is told. The erase fires 48 hours later, so a report someone else made with a copied code can be cancelled from any of the owner's devices, with the Spojiti PIN; after such a cancel the support page cannot report the account again for 7 days, and three wrong codes lock its reports for 30 minutes, doubling. A report made in the app from a device that has been on the account for 72 hours or more acts at once. A marked device then tries to back up its chats, sealed (1.5), and erases itself; since build 383 it erases only when the report carries a proof that only a holder of the code can make (HMAC of the account id under the code), which the relay passes on but cannot originate. If the backup cannot be made, the device erases without it. A device set up again afterwards has a new device id and is never touched; a device kept offline never learns, which no remote erase can change, and the erase is carried out by our app.

1.5 Account control: the Spojiti PIN

The phrase is 128 bits: guessing one is out of reach for any computer, and that does not change. What the phrase cannot do on its own is tell a copy from the owner. From build 422 a copied phrase is no longer the whole account. Each account has a six-digit Spojiti PIN, the same PIN

that opens the app on the owner's devices. The app stretches it with Argon2id (19 MiB, two passes) under a salt derived from the phrase and turns it, with a second phrase-derived key, into a 32-byte verifier; the relay stores only SHA-256 of that verifier, which cannot be searched for a PIN without the phrase. A new device signs in with a signature from the phrase's key and the verifier. The signature is checked first, so only a holder of the phrase can spend a try; three wrong PINs lock sign-in for 30 minutes, each further lock doubling to 24 hours, and every device of the account is told of each wrong try. That is at most about 1,100 guesses a year against a million values. The check sits on the relay because a limit inside the app stops nobody: whoever holds the phrase can run their own copy of the app.

Backups are sealed under HKDF of the phrase's backup key and a 32-byte share the relay makes when the first PIN is set and hands out only with the right PIN, so a backup made after that needs both. A device that joined the account less than 72 hours ago cannot remove another device, delete the account, change the PIN or file an immediate lost report; a relay-signed device token, issued only to a device that signs with its own key, says which device is asking. Every device added, replaced, removed or blocked, every wrong PIN or lock, every PIN change or reset and every lost report is recorded for 30 days and told to every device at once, by the live connection and a content-free wake. From a device that has been on the account 72 hours or more, "Not me" does five things in one step: the named device is told to erase itself, its id is blocked from the account for 30 days, every sign-in of the account is revoked, the PIN changes and the backup share changes, so the intruder holds neither the new PIN nor the key to later backups. A forgotten PIN costs neither a wait nor the backups where it can be helped. On a device still signed in, Face ID or a fingerprint (never the phone's passcode) or the phrase opens the app and a new PIN is chosen before anything else; the share is kept. On a new phone, the owner's passkey, the one that can already seal the phrase, yields under its own label a 32-byte recovery value whose hash the relay keeps; the phrase's signature and that value set a new PIN in one step, counted with the PIN's own tries, and the share is kept. With no device and no passkey, the phrase starts a reset that every device is told of and can cancel; after 7 days it sets a new PIN with a new share, and the share from before is kept and handed back 30 days after the reset started, unless a device that was on the account before the reset keeps it closed. Until then the phone sends nothing to Spojiti.cloud, which keeps one backup chain, so the older backups are still there on the day.

Limits we state: an account that has not yet set its PIN behaves as before until its app sets one (the app requires it from build 422); a phrase and a copy of the relay database together could be searched for the PIN, a pair that is the whole account in any design; backups made before the account had a PIN remain under the phrase alone until the next full backup replaces them; and the erase after "Not me" is carried out by our app, while the block, the revocation and the new PIN hold whatever the other device runs.

2 · The session protocol

2.1 Session establishment: X3DH with a post-quantum leg

A session between two devices is established with the X3DH key agreement over X25519, extended by one ML-KEM leg to the construction generally known as PQXDH. The initiator fetches the responder's published bundle (classical signed prekey and post-quantum prekey), verifies both signatures, and computes:

```
SK_ec = X3DH(IK_a, EK_a, IK_b, SPK_b)           // classical
(ct, SS_pq) = ML-KEM.Encaps(EK_b)              // post-quantum
SK      = HKDF-SHA-256(SK_ec || SS_pq, info = "spojiti/pqxdh/v1")
```

The first message from the initiator is a prekey message carrying the X3DH parameters and the KEM ciphertext, so the responder derives the same root with no extra round trip. For sessions established with the post-quantum leg, an adversary must break both X25519 and ML-KEM to recover the root key.: adopting the post-quantum primitive cannot make the session weaker than the classical one it extends, even if ML-KEM is later found wanting.

2.2 Hybrid signatures on the account's signed prekeys

A peer refuses an account prekey whose Ed25519 signature fails and, once it knows the peer's ML-DSA-65 key, one whose ML-DSA-65 signature fails. From build 383 it also refuses an unsigned prekey from an account that has ever shown it a signing key, so a relay cannot strip a signature to slip one in; only an account that never published a signing key is accepted unsigned. Neither major messenger authenticates keys post-quantum today; both say so in print. Applying to the account's prekey signatures the same hybrid rule that PQXDH applies to key agreement means an attacker who breaks one scheme still cannot forge the pair.

2.3 The Double Ratchet, and the KEM ratchet inside it

Messages are protected by a Double Ratchet written from the Signal specification on X25519, HKDF-SHA-256, HMAC-SHA-256 and AES-256-GCM, with no invented cipher or key exchange; unlike the specification, message AEADs bind the ratchet header but not the two identity keys. Every message body has its own key from a forward-ratcheting chain, so a stolen key cannot decrypt earlier messages, and each round trip performs a fresh Diffie-Hellman so the session heals after a key leak; the short lock-screen preview, when it is on (the default), travels beside it under a long-lived per-contact key with no forward secrecy. Out-of-order and dropped messages are handled with a bounded store of skipped message keys.

When a session was established with the PQXDH leg (both devices passed the on-device NIST self-test and the peer published a signed ML-KEM prekey), every ratchet step also folds in a fresh ML-KEM-768 shared secret. The DH ratchet becomes a DH + KEM ratchet: the per-message keys of every chain of that session stay quantum-safe, and a device state stolen today heals against a quantum adversary exactly as it heals against a classical one. This gives post-quantum forward secrecy and post-compromise security inside the session, not only at its establishment.

2.4 Per-device sessions

Sessions are keyed per device, not per account. A per-account ratchet forks if two devices of one account both advance it and silently breaks decryption; per-device sessions prevent that. The relay serves per-device prekey bundles on demand so that any device can open a session to every device of a peer without a peer-to-peer device list.

2.5 The envelope and the franking commitment

The sealed payload is a versioned envelope. Inside it, every message carries a franking commitment: a 32-byte per-message key and an HMAC-SHA-256 of the exact pre-envelope bytes under that key. The sender also binds that commitment to itself and to the recipient under a key shared with Spojiti's report key, so when a recipient reports messages, Spojiti can verify that the reported person really sent them. Nobody else can check that binding, so a report proves nothing to anyone outside Spojiti.

2.6 Sealed sender: the relay learns the recipient and nothing else

The ratchet ciphertext is wrapped once more before it leaves the device, encrypted to the recipient's two long-term keys: the X25519 identity key and the ML-KEM-1024 prekey.

```
eph      = fresh X25519 keypair
dh       = X25519(eph.sk, IK_recipient)
(ct, ss) = ML-KEM-1024.Encaps(EK_recipient)
key      = HKDF-SHA-256(dh || ss, info = "spojiti/sealed/v1" || to || eph.pk || ct)
body     = AES-256-GCM(key, nonce, {from, inner}, aad = "spojiti/sealed/v1" ||
to)
wire     = "ss1." || base64(eph.pk || ct || nonce || body || tag)
ss2      = the same seal with "spojiti/sealed/v2" labels and a binary body,
          0x02 || varint(len from) || from || packed inner; sent to every device
          that announces it
preview  = optional, after "~": AES-256-GCM(per-contact preview key, text,
          aad = "spojiti/pv3" || SHA-256(sealed part))
```

Any device of the recipient can open it, because both keys derive from the twelve words; nobody else can; and a quantum adversary who recorded the relay's traffic cannot unwrap the sender later. "Harvest now, de-anonymise later" is the same attack as harvest-now-decrypt-later, aimed at metadata, and for the sender of a sealed envelope it is closed the same way; the recipient and the timing stay visible to whoever recorded them. The sender field inside the wrapper is a routing hint for the recipient's session store, not an authentication: the inner ratchet's authentication is what proves who wrote, exactly as before. Consequently an envelope on the sealed lane is stored with no sender: the relay sees an envelope addressed to the recipient's random account id (and, for a copy meant for one device, that device's id), its size, when it arrived and whether to wake a phone for a message or a call, and it forwards or queues it. The lane carries no plan token (B383), so nothing on the frame ties it to an account. Two limits remain: envelopes to and from the official :Spojiti account travel on the signed-in connection with the sender's account id, because its daemon does not open sealed envelopes, and the network address and timing of the two connections can still be correlated.

2.7 The post-quantum activation gate

No post-quantum material is derived, published or accepted on a device until the ML-KEM and ML-DSA implementations have passed known-answer tests at runtime on that device. The vectors are copied verbatim from NIST's ACVP repository, and the implementation carries an explicit prohibition on substituting invented vectors, on the reasoning that a test passing against made-up numbers proves nothing while looking like assurance. The gate fails closed: if the vectors are absent or the self-test fails, no post-quantum key is derived or published on that device, and the degradation is logged and shown in Settings. Because every envelope between people must be sealed, and sealing needs those keys, such a device holds its messages on the phone rather than send them weaker, and only the chat with the official :Spojiti account keeps moving until the self-test passes.

2.8 Groups

No group record exists on the relay. Each group message is sent as a batch of individually sealed copies, so the relay can see which accounts receive copies together. Messages are delivered by pairwise fan-out over the existing per-device sessions: one individually sealed copy per member device. We deliberately rejected a shared group key, which would cost per-member forward secrecy and turn removal into a rekey problem. Pairwise fan-out costs bandwidth linear in membership, an accepted trade at a cap of 6 members in a group created on Free or Basic and 256 from Pro. Membership changes are carried end to end and signed by an admin; the relay cannot inject one.

2.9 Multi-device

A new device is linked by a QR exchange with no password and no server-held secret. The new device mints a rendezvous identifier and an X25519 key pair and shows the identifier and the public key in the QR code (from build 393; before it, the QR carried the key itself); the primary seals the identity payload to that public key and uploads it, and the relay keeps the first upload, refusing any other for that identifier; the new device fetches it, as often as a bad network needs, and the relay deletes the record 20 minutes after upload. The relay holds ciphertext for which it has no key. Outgoing messages fan out to the sender's own other devices as well, so devices stay consistent without a server-side mirror. Linking a device copies no history. History moves only by the optional, end-to-end-sealed switch-phone transfer or encrypted backup, which we can store but not open. A limitation we state rather than hide: provisioning shares the identity secret, so a linked device is a full holder of the identity. Per-device delegated credentials with cryptographic revocation are designed and not shipped. From build 422 a linked or restored device also needs the account's Spojit PIN to sign in, every device is told when one joins, and a device the owner does not recognise can be erased and blocked with "Not me" (1.5).

3 · Media, files and calls

3.1 Sealed blobs

Every image, file, voice note and video is encrypted on the sending device with a fresh random AES-256-GCM key per blob, producing `nonce(12) || ciphertext || tag(16)`. The key travels inside the ratchet message to the recipient; only the ciphertext is uploaded to the media store. Media is therefore exactly as server-blind as text. The identical sealed bytes go to the local disk and to the relay: Media is stored on the device as ciphertext and opened into memory for display; short-lived plaintext scratch files are used for transcription and are deleted afterwards.

3.2 View-once and expiry

A view-once item is opened into memory, shown once, and its local copy is discarded. The media store holds ciphertext with an expiry; encrypted disaster-recovery copies age out within hours.

3.3 Calls

Voice and video calls use WebRTC with DTLS-SRTP media encryption, peer to peer where the network permits. Call signalling travels inside the Double Ratchet. The relay learns that a call is ringing or has stopped ringing (so it can wake the phone), but no call parameters. When a direct path is not available, media is relayed through TURN servers we operate, with a fallback TURN

on a separate box in a separate country. TURN credentials are short-lived, minted by the relay using coturn's REST scheme (an HMAC over an expiry-scoped username with a one-hour lifetime), and a TURN relay forwards DTLS-SRTP media it cannot decrypt; it does see the network addresses of both ends of the call. Call recording, where the plan allows it, requires the other party's explicit consent in the call and is indicated on both screens for its whole duration.

4 · Anonymous entitlements and payments

4.1 The problem

Every feature is free, and a pack raises one limit for a year. A pack has to be provable to the relay without linking a payment, which necessarily identifies a person, to a messaging identity, which must never be identified. A flag on an account row saying which packs it holds would be exactly that link.

4.2 Blind signatures

Spojiti uses RSA blind signatures (RSABSSA, the randomised variant) to sever the two. The device builds a 64-byte token message, a 32-byte prefix followed by a 32-byte random nonce, so no two tokens ever share a message even for the same pack and epoch. It blinds the message and presents the blinded value together with proof of payment. The issuer signs what it cannot read, under a key that names only the pack and the month it expires. The device unblinds the signature and holds a bearer token that proves "this pack was paid for, until this month" and nothing else. The relay stores no account-to-pack record. Because a device presents its tokens when it connects, the relay knows which packs a connected account holds for the life of that connection, and nothing about which payment produced them.

4.3 Rails

Five packs, Basic, Pro, Plus, Max and Ultra, each raise the limits for a year, and a higher pack includes every pack below it, so a device holds at most one pack token at a time: the highest unexpired one. Prices are set per price group (rupee, dollar, pound and euro groups; a country's income category picks its group, and the price book can move any country) and shown in the buyer's own currency.

Three rails carry a purchase, and the relay is in the same position on every one of them: it can check a token, and it cannot say who bought it. On iPhone the App Store sells the pack as a product named by its price group; the device presents the store transaction with a blinded token message, the relay maps the product to its pack and signs once per transaction. On Android, Mac and Windows, and on the website, a payment processor takes the card or net-banking payment (the processor learns what a card payment normally reveals about the buyer) and the

relay issues a bearer pack code that names no account; the app adds the code in one tap, and a code bought on the website is pasted in. Redeeming a code blind-signs the pack's token in the same transaction as the code is spent, idempotent on the blinded value, so the relay sees that some code was spent and never the token it became: the pack a device later presents cannot be matched to the code that paid for it, nor either to an account.

Promotions use the same token. A free-pack offer or a referral reward is a claim counted per hashed account id until the offer's limit, signed blind like a purchase; a promo code is applied once per hashed account id, and the relay keeps how many times a code was used, never by whom. A percentage discount changes only the amount the processor charges on the web rail; the store rails carry no discount.

5 • Echoes

Echoes are Spojit's communities. They exist in two forms, and the difference is the point.

5.1 Private Echoes

A private Echo is a sealed group as described in section 2.8: a client-side construct, delivered by pairwise fan-out, invisible to the relay as a group, with admin-signed membership. Nothing about a private Echo exists on any server except sealed envelopes in transit.

5.2 Public Echoes

A public Echo is readable by anyone with the app and posted to by members; today the relay serves it, from tables of its own, and a box of its own is designed and not yet running. Every member acts under a per-Echo pseudonym derived on the device; the same person is a different pseudonym in every public Echo, and no pseudonym is linkable to an account identity. Standing to post is proved with a blind attunement token, issued the same way as an entitlement in section 4, so the Echo service can check that a poster is entitled to post without knowing who they are. The public Echo tables keep no account data. Posting rights are proved with blind tokens; an operator able to see both the Echo service and the main relay could still correlate an entitlement token used in both. A review queue, link hygiene rules, standing and badges, and operating valves for rate and reach exist to keep public spaces usable, and they operate on pseudonyms and content, never on people.

6 · Infrastructure

6.1 What we run and where

The relay is written in Elixir on Phoenix and speaks WebSocket over TLS (1.2 or 1.3), behind Caddy. We run it on virtual servers rented from a hosting provider in two countries, not on a hyperscaler's managed platform. The data plane (the relay) and the control plane (entitlement issuance, the download gate, the operations console's backend) are separate services with separate databases, so that the process that carries messages holds nothing the control plane needs and vice versa. Relay nodes form a cell over a private wire and a node is a role, not a copy: a second node in the same cell shares the work without duplicating the data. An account lives in exactly one cell, and a directory on the control plane records which; to count accounts, the control-plane box also keeps a standby copy of each cell's database. The control plane shares no cluster with any cell, so when it must hand something to an app (a code the website asked for, a lost-device report) it signs one small instruction with its Ed25519 key, domain-separated from every other use of that key, and posts it to each cell; a cell accepts it only under that public key, only once and only before it expires.

6.2 Push without content

Push notifications carry no content and no sender. They are a wake: the platform's push service is told that a device should connect, and the device fetches its sealed queue over its own connection. Apple and Google learn that a device was woken and whether for a message or a call, with an opaque number that pairs a ring with its cancel or, on Android, names the waiting envelope so the phone can show it once, and nothing else.

6.3 Backups and recovery

Database snapshots are encrypted on the box (AES-256-CBC, PBKDF2 passphrase) before being shipped to a separate disaster-recovery box every thirty minutes, with bounded retention at both ends; the same box also runs a live standby of a cell's database, and every cell's database has a live standby on a separate box. Restore drills are scripted and run against the snapshots, not assumed. The disaster-recovery box also holds the source repository's remote; the fallback TURN runs on a box of its own. A snapshot contains everything the relay database holds (envelopes awaiting delivery, device registries, key bundles, push tokens, signing keys) and, separately, encrypted media. No readable message content.

6.4 Abuse resistance without identity

A messenger that knows no one cannot ban a person; it can only make abuse expensive and unrewarding. Registration requires a proof of work. Every route is rate limited in fixed windows. A

forced-upgrade gate at the socket handshake retires builds with known defects. Every message carries a franking commitment bound to its sender and recipient; reporting a :Handle can send us the whole chat or the messages the reporter selects (at most 2,000, words only); theirs are marked verified only when the binding checks and everything else is kept as unverified context, for 90 days, in a reports store on Spojiti.cloud in the reporter's region rather than on the relay, and none of it proves anything to anyone else. Public Echoes have a review queue and operating valves. Nothing is posted to our support board or roadmap from the website (a ticket, feedback, an idea or a reply) without a six-digit code sent to the poster's own official chat. The code request is deleted within a day; a verified post keeps the random account id that verified it, visible only to our staff. None of these needs to know who anyone is.

6.5 The operations console

An operations console exists for the company's own staff. It manages website content, plans, download limits and fleet health, and it is served from a separate host with its own authentication and audit trail. It has no view of message content; it sees support posts, payment records and aggregate counters.

6.6 Spojiti Private Messenger Core is open source

The cryptographic and transport core (identity derivation, PQXDH, the Double Ratchet with the KEM ratchet, sealed sender, media sealing, blind entitlements on the client; accounts-without-accounts, key bundles, sealed queues and the no-PII test on the relay) is published as the Spojiti Private Messenger Core under the AGPL v3. Its crypto and transport files are copied verbatim from the product by a generator whose check fails on drift; its relay's delivery, channels and schema are a smaller stand-in for the production relay. The full product is not open source.

7 · Threat model and residual metadata

7.1 Adversaries we design against

Network observer	Passive or active observation of traffic in transit. Protection: content confidentiality and integrity; no plaintext ever transits; TLS 1.2 or 1.3 outside, the ratchet inside.
Compromised relay, or us under compulsion	Full read and write access to our own servers and databases. Protection: content and conversation history are unavailable, and so is the sender of an envelope on the sealed lane. Envelopes to and from the official :Spojiti account, and correlation by network address and timing, can still reveal who spoke to whom. The operator can deny service; it cannot read it.
Device thief	Physical possession of a locked or unlocked device. Protection: local ciphertext at rest under SQLCipher (AES-256) keyed from the recovery phrase; key material in the platform keychain or keystore; App Lock with biometrics; the Quiet switch, a tap pattern the owner records on any screen of the app, which switches to a real, working Spojiti containing only the conversations the owner chose. Both open as complete apps, the second one working offline; nothing in the app says which one is showing or whether a pattern has been recorded, and the switch is made under observation without a word or a screen. It defends against a person demanding the phone be unlocked in front of them; it does not defend against forensic analysis of the device, and the app says so on the screen where it is set up.
Future quantum adversary	Records ciphertext today to decrypt later. Protection: hybrid post-quantum key agreement at establishment, a KEM ratchet inside every session except the classical one with the official :Spojiti account, post-quantum signatures on the account's signed prekeys, and a post-quantum sealed-sender wrapper, so that for sealed envelopes the sender cannot be recovered later either.

Legal compulsion

A valid order to produce user data. Response: no name, number, address book or conversation to produce. The design goal is that compliance and disclosure are the same empty set, and section 7.2 lists exactly what would be in it.

7.2 What the relay sees, and what it never sees

Message and media content	No. Opaque ciphertext; deleted once every device of the account has collected it, or on expiry.
Who wrote a message	Not for sealed envelopes. They are sealed to the recipient with a post-quantum wrapper (section 2.6) and stored with no sender. Every envelope between people is sealed; only envelopes to and from the official :Spojiti account carry the sender's account id until delivered.
Group existence or membership	No record. Groups do not exist server-side: nothing stored names a group or its members. A group message is sent as one batch of individually sealed copies, so while it carries that batch the relay can see which accounts receive copies together (section 2.8).
Real-world identity	Never. There is no field for it.
Payment ↔ identity link	No. Blind signatures sever the purchase from the token.
The recipient's routing identifier	Yes, until delivery. The recipient's random account id, required to route, stored with each envelope until every device it was for has collected it or the 120-hour retention window ends; never archived.
Presence	Yes. A per-device online flag, ephemeral with the socket. The relay tells any account that asks by account id whether it is online, unless the user hides their online status from anyone; then it answers no one, and contacts learn it only from the user, sealed. The time of each account's last connection is kept (next row).
Push token	Yes. Used only for content-free wakes.
Connection address and times	The address while connected; one time kept. A relay necessarily sees the address a socket comes from, and stores none: no per-account address history exists in the data model. It keeps one time per account, its last connection, for device cleanup and active-account counts, and hands it to no one. Network-level anonymity is out of scope (7.3).

7.3 What we do not claim

We are not a mixnet. An observer at the network layer can see that a device contacts our relay, and when. Packet timing and size are not padded in this version, so traffic analysis of an observed connection is possible. A recipient's routing identifier and the address that connection came from coexist on the relay while the connection is open; sealed sender removes the sender from that picture, not the recipient.

A compromised endpoint is total. No messenger protects a person whose device is under an adversary's control, and we do not claim to.

Deletion is cooperative. "Delete for everyone" depends on a well-behaved peer client and is never described as cryptographic.

A linked device holds the identity. Section 2.9. Revocable per-device credentials are designed and not shipped.

The audit is in progress, not complete. The constructions above have been checked by us against our own source. Independent verification is what an audit is for, and until it is published this paper is the company's account, not a third party's.

Appendix A · Primitives and parameters

Recovery phrase	BIP-39, 12 words, generated on device
Key derivation	BIP-39 seed (PBKDF2-HMAC-SHA-512), then HKDF-SHA-256 with domain-separated info strings; HMAC-SHA-256 chain keys
Classical agreement / signing	X25519 / Ed25519
Post-quantum agreement	ML-KEM-1024 (prekey and sealed sender), ML-KEM-768 (ratchet step)
Post-quantum signing	ML-DSA-65 beside Ed25519 on the account's two prekeys (both must verify)
Message and media AEAD	AES-256-GCM, 12-byte nonce, 16-byte tag
Session	X3DH → PQXDH; Signal Double Ratchet with a DH + KEM ratchet step
Local store	SQLCipher (AES-256), key derived from the recovery phrase
Entitlements	RSABSSA, randomised variant, 64-byte token message, for paid plans; the welcome trial is a relay-signed Ed25519 token
Transport	WebSocket over TLS 1.2 or 1.3; WebRTC with DTLS-SRTP; coturn REST credentials, 1-hour TTL
Self-test gate	NIST ACVP known-answer tests for ML-KEM and ML-DSA at runtime, fail-closed to classical

Appendix B · Revision history

2.0 · 22 October 2026

Spojiti Packs (4, rewritten): five packs, Basic to Ultra, a year each, a higher pack including the lower; priced per price group in the buyer's currency; bought through the App Store on iPhone and through a card or net-banking checkout as a bearer pack code everywhere else, redeemed with a blind signature in the same transaction; promotions and referrals as blind-signed claims. Spoj, the currency, removed. The group cap on groups you create (2.8).

1.9 · 17 October 2026

No tiers (4, rewritten): every feature is free, and a pack raises one limit for a year. Packs are paid for in Spoj, Spojiti's own currency, on a bearer code the relay knows only by its hash; buying a pack blind-signs its token in the same transaction as the debit, so no pack can be traced to the code that paid for it. The group cap with the Everything pack (3.2).

1.7 · 16 October 2026

A forgotten PIN (1.5): a new one at once with the face or fingerprint on a device still signed in, or with the owner's passkey on a new phone, the backups kept; and after a 7-day reset, the backups from before it handed back after 30 days unless a device from before the reset keeps them closed.

1.6 · 16 October 2026

Account control (1.5, new): the six-digit Spojiti PIN the relay checks at sign-in, three tries then 30 minutes doubling, backups sealed under the phrase and a share released only with the PIN, the 72 hours before a new device can act on the account, every change told to every device, and "Not me". The lost-device erasure as it runs since builds 383 and 420: an owner's proof, a 48-hour quarantine for a report from the website, cancelled with the PIN (1.4). Linking by an X25519 key in the QR (2.9).

1.3 · 13 October 2026	Every sentence read again against the code: sealed sender between people with no fallback (every supported build is 388 or later; the official :Spojiti account is the one exception, and it is classical), every device prekey signed by its account and an unsigned one refused, post-quantum sessions for every chat between people including chats begun before post-quantum, verifiable abuse reports (2.5, 6.4), ML-DNA-65 named on the prekeys each account publishes rather than on every key, and the boxes as they run today.
1.2 · 11 October 2026	Corrections, and one change of design: four sentences that had lost words in 1.1 are whole again (1.4, 2.7, 2.8, 7.1), and stray punctuation is gone from the abstract, 1.3 and 6.4; the group cap is the shipping one, 64 members on the free plan and 256 on Premium (2.8); the table of what the relay sees says, as 2.8 already did, that it can see which accounts receive copies of a group message together (7.2). Discovery is mutual: a match no longer opens to an account, and a hello reaches someone only if each has the other's number (abstract). The PDF is generated from this page, so the two cannot differ.
1.1 · 30 September 2026	Cells and the signed control-plane-to-cell road (6.1); the lost-device erasure (1.4); the website posting code (6.4); backups every thirty minutes, live standbys, and where the fallback TURN runs (6.3).
1.0 · September 2026	First public edition, written against build 321; architecture overview added in the same week. Supersedes the internal protocol draft of August 2026 (which predated the ML-KEM-1024 prekey, hybrid ML-DNA-65 signatures, the KEM ratchet and sealed sender).