

End-to-end encrypted messaging on a relay you run

PDG Global Limited, Hong Kong

This paper describes protocol version 3, used by rChat 2.0.0, the current release of the relay and the iOS app. It states what rChat protects, what the relay knows about you, and what is on the roadmap.

Abstract

rChat is end-to-end encrypted messaging that you run yourself. It has two parts: a relay server that you deploy on a host you control, and an iOS app that connects to it. Messages are encrypted on the phone with keys agreed through a hybrid handshake, X25519 together with the post-quantum ML-KEM-768, and advanced by the Double Ratchet, using XChaCha20-Poly1305. The relay routes ciphertext between phones and holds no message keys.

This paper describes protocol version 3, the current release. It covers the cryptography, how keys are authenticated, how conversations start, how later conversations are paired without naming who they are between, what the relay learns and stores, how the relay protects itself, and where rChat still falls short. Where a property holds only at the relay's default settings, the paper says so.

1. What rChat is, and is not

rChat is software, not a service. There is no rChat cloud, no sign-up and no subscription. A relay is operated by whoever deploys it, on a server they choose. PDG Global Limited, which develops rChat and publishes it under the AGPL-3.0-or-later licence, does not run relays for users and holds no accounts or messages. In this paper, "the relay" means whichever relay a user connects to, usually one they or their organisation run.

Accounts are key pairs. An identity is an Ed25519 signing key, an X25519 agreement key and an ML-KEM-768 key, generated on the phone; the ML-KEM key is derived from the same seed, so one backup restores all three. There are no usernames, phone numbers or passwords. Contacts are added by scanning or pasting a contact code that carries their public keys.

Joining a relay takes one scan. The relay's operator prints an invite, a link shown as a QR code that carries the relay's address and its identity. Scanning it sets up the app for that relay and pins the relay's identity at once (§5.3).

Jurisdiction. PDG Global Limited is based in Hong Kong, outside the jurisdiction of the Five Eyes intelligence alliance. Because rChat is self-hosted, the law that governs a relay is the law where its operator runs it. Message content is protected by end-to-end encryption, which does not depend on any jurisdiction.

Clients. The iOS app is the only client today. A native Android client is planned. Both are built on one shared Rust core that the app calls through a C interface.

2. Goals and status

Goal	Status in protocol v3
Confidentiality of message content from the relay and the network	Provided.
Confidentiality against recorded traffic and a future quantum computer	Provided for every session and every message request: the handshake and knocks are hybrid X25519 + ML-KEM-768 (§7, §6.2). The TLS link to the relay uses a hybrid post-quantum key exchange too (§12.1). Not provided for authentication, which stays classical (§4, §12.1).
Integrity of message content	Provided: every message is authenticated together with its header and both identities (§8).
Forward secrecy of messages	Provided by the Double Ratchet. Not provided for first-contact requests and their intros (§6.3).

Goal	Status in protocol v3
Authentication of contacts' keys	Provided for contacts added from their code, and checkable for any contact with a safety number (§9). A contact first reached through a request is trusted on first use until checked.
Authentication of the relay	Trust on first use of the relay's identity key, or an identity entered by the user (§5.3).
Minimal metadata at the relay	Partial. Nothing about who talks to whom is logged or written to disk by default. Conversations after the first are paired by rotating blinded tokens (§6.4), but a running relay still holds routing metadata in memory (§3, §11).
Resistance to traffic analysis	Partial. Sizes are padded to buckets. An opt-in steady mode reduces what timing reveals while the app is open; otherwise timing stays visible (§10).
Availability against a hostile relay	Not provided. The relay can refuse, drop, delay or reorder traffic.
Deniability	Not claimed.

3. What the relay knows about you

The relay is the one server in rChat that sees traffic. This is what it has and what it does not, at its default settings.

The relay never has	The relay sees while running
Your identity. No name, phone number, email address, contact list or sign-up. An account is a public key made on the phone.	Each connection's IP address and account key. Held in memory only, never written to disk. Client IP addresses are kept out of the logs unless the operator turns on <code>log_client_ips</code> (off by default).

The relay never has

Message content. Only encrypted packets, each padded to a size bucket. The relay holds no message keys.

The relay sees while running

Which two connections share an open conversation, while it is open. Later conversations between the same two people are paired by a token that changes every 10 minutes and at every reconnect, but while a conversation is open the relay's memory still links its two members, and someone watching the relay live sees two accounts pair at the same moment. Once both have been gone about ten minutes, nothing it keeps names the pair.

A lasting record of who talks to whom. Its lookup tables are kept in memory and keyed with a random secret that changes on every restart. Held messages are in memory only unless the operator turns on encrypted persistence, and nothing about who talks to whom is written to its logs.

First requests: who is asking to reach whom. On first contact, after a lost phone or a key change, and at most once a day as a fallback, an ask names its target. It is held for up to ten minutes. A request waiting for someone who is offline keeps a keyed pair tag until it is delivered, for at most 72 hours.

When messages flow, and their rough size, from the size bucket.

In short: the relay sees keys, not people, and keeps no lasting record. To link keys to IP addresses or to each other, whoever runs it would have to modify it or read its memory while it runs. That is why rChat is built for you to run your own relay.

Two operator settings change this table, and both are off by default. `log_client_ips` adds client addresses to debug log lines. Encrypted queue persistence writes held messages to disk, encrypted, with their key on a memory filesystem; the table key is then kept with the queue until the machine reboots. §11 gives the details.

4. Cryptographic primitives

Use	Primitive
Key agreement	X25519 (RFC 7748). Every Diffie–Hellman output must be contributory; low-order keys are refused.
Post-quantum key encapsulation	ML-KEM-768 (FIPS 203), in a hybrid with X25519, for sessions and knocks. Public keys pass the FIPS 203 check.
Identity signatures	Ed25519 (RFC 8032), verified strictly; weak keys are refused.
Message and envelope encryption	XChaCha20-Poly1305, 256-bit key, 192-bit nonce
Key derivation	HKDF-SHA256 (RFC 5869)
Ratchet chain keys, request authentication	HMAC-SHA256
Channel binding	TLS 1.3 exporter (RFC 8446 §7.5), label <code>EXPORTER-rchat-conn-auth-v1</code>
Safety numbers	SHA-512, 5200 iterations per party
Relay identity pin	SHA-256 of the relay’s Ed25519 key
Relay routing tags, log tags, address hashing	Keyed BLAKE3 with random per-process keys
Transport	TLS 1.3 only (RFC 8446): rustls with the ring provider on the relay, BoringSSL in the app; key exchange X25519MLKEM768 (hybrid, IANA 0x11EC), falling back to X25519; resumption tickets, never 0-RTT

5. Identities, key bundles and connection proof

5.1 Hello and version

A client opens one WebSocket connection to its relay, inside TLS 1.3, and sends a `hello` that lists protocol version 3. A client that does not offer version 3 is refused with “client too old; update rChat”; version 3 is a hard cutover from version 2. The relay answers with a fresh 32-byte challenge and its own signed identity (§5.3).

5.2 Signed bundles, bound to the TLS session

Each account publishes a key bundle: its Ed25519 identity key, its X25519 agreement key and its 1184-byte ML-KEM-768 public key. There is no separate signed pre-key, and one-time pre-keys are not used; each session instead mixes in fresh ephemeral keys from both sides (§7).

- **Bundle signature.** The identity key signs `rchat/bundle/v3 || identity key || agreement key || ML-KEM key`. Anyone who receives the bundle checks this signature. Because the identity key signs the ML-KEM key, the ML-KEM key cannot change without the identity changing, and safety numbers stay computed over the identity and agreement keys.
- **Connection proof.** Every publish also carries a signature over the relay’s challenge, the bundle and a 32-byte **channel binding**: the TLS exporter value of the session the publish travels on. A proof captured on one TLS session fails on any other, so it cannot be replayed or relayed through a second connection.
- **What the relay does with it.** The relay checks both signatures before it maps the identity to the connection. It only routes to, and delivers queued items to, a connection that proved the identity, and a connection can never switch to another identity. A failed proof closes the connection.

Because both ends bind to the TLS session, **TLS must terminate at the relay**. A TLS-terminating proxy or CDN in front of it gives the two ends different sessions, and every connection fails. Only a TCP (layer-4) passthrough is supported (§12.1).

5.3 The relay’s identity

The relay has its own long-term Ed25519 identity, separate from its TLS certificate. In every `hello` reply it signs the channel binding, the challenge, its key and a version number. The app pins the relay per origin (`scheme://host:port`) as `rk:1:` followed by the SHA-256 of the key.

- **Trust on first use.** By default the first connection to a relay pins whatever identity it presents. An attacker present at that first connection is pinned instead. The user can close that gap by entering the identity the operator publishes and turning on **Require a saved server identity**.
- **After pinning,** a different identity stops the connection: no fallback, no prompt to continue. A middlebox with a mis-issued certificate, or a DNS hijack after the first connection, cannot produce the signature.
- **Invites.** `rchat-server --print-invite` prints a link, `rchat://relay?u=...&pin=...`, and draws it as a QR code. Scanning it (or pasting or opening it) shows the relay's host and asks to confirm, then saves the address and pins the relay's identity at once, which closes the first-use gap. An invite never replaces another relay without a confirmation naming both hosts, and a different identity for the same host needs a separate, two-step confirmation. An invite contains the relay's secret path, so it lets the holder reach the relay; it never lets anyone read messages. Typing the address by hand remains possible.
- **Rotation.** An operator can replace the key with `--rotate-relay-identity`. The old key signs a handover to the new key at version + 1, and clients pinned to the old key follow it. The previous key is deleted, not archived. If the key file is lost instead, the relay refuses to start, and a new key is refused by every client that pinned the old one.
- Release builds of the app connect only over `wss://`, except to a test relay on the same device. Over plain `ws://` there is no channel binding and pinning protects nothing.

6. Conversations start by mutual intent

6.1 Asks and circuits

Two phones talk over a **circuit**, a pair of routing entries on the relay. A circuit forms only when **both** accounts have asked for it. Each ask carries a fresh X25519 ephemeral key, a fresh ML-KEM-768 ephemeral key, and the asker's identity signature over both. A first ask (`CircuitAsk`) names the target identity and carries a sealed, fixed-size knock (§6.2); a later one is blinded and names nobody (§6.4). Both kinds are the same size on the wire.

- If the other account has an outstanding ask and is connected, the relay creates the circuit and sends each side the other's verified bundle and both asks' ephemeral keys and signatures, unmodified. The earlier asker becomes the initiator.

- Otherwise the relay keeps the ask for up to ten minutes and delivers the knock. It sends nothing back to the asker, whether the target is online or not.
- A member closing the circuit (on block, delete or unpin) tears down both sides.

6.2 Knocks

A knock is how one account tells another that it wants to talk. It is always exactly **5120 bytes**. It has its own hybrid envelope: the sender encapsulates a secret to the recipient's static ML-KEM-768 key (which it knows only from the recipient's contact code or an earlier session, never from the relay), and the envelope key combines that secret with an X25519 exchange. A recording plus a future quantum computer alone cannot open a knock. Inside are the sender's bundle, the hour it was issued, a MAC, a flag saying whether the sender holds no session with the recipient and, optionally, an intro. The MAC is keyed from a static-static Diffie-Hellman of the two agreement keys mixed with the encapsulated secret, so the recipient could have made it too: it proves nothing to a third party. The encapsulated secret does not authenticate the sender, since anyone can encapsulate to the recipient; sender authentication stays classical.

The recipient accepts a knock only if the bundle signature and the MAC verify and the issue hour is inside the knock's lifetime. What happens next depends on the sender:

- a blocked sender: dropped;
- a contact whose keys are pinned and match: the app asks back, and the circuit forms;
- a pinned contact with different keys: shown as a key mismatch, not accepted;
- anyone else: a **message request**.

6.3 Message requests and intros

A request appears in a quiet "Requests" row. Until the user accepts, nothing goes back to the sender: no session, no receipt and no reply, so someone who has your code cannot tell whether you are online. Accept, delete and block all happen on the phone.

A request may carry one short first message, an **intro**: text only, up to 1 KiB, with control and direction-changing characters removed. The intro is encrypted with a key derived from the static agreement keys of both accounts and the encapsulated secret. **Intros and knocks are not forward-secret**: anyone who recorded a knock and later obtains the recipient's long-term secrets can read its intro.

6.4 Blinded pairing

After two people's first session, both phones derive a **pair secret** from it. From then on their asks name no one: each carries a token, an HMAC of the pair secret over the UTC day, a counter and the current 10-minute slot. Both phones advance the counter after every pairing and reset it each day, so the token changes every 10 minutes and at every reconnect. This needs the two phones' clocks within 10 minutes of each other, which connecting already ensures: the relay refuses a phone whose clock is more than 5 minutes off its own. The relay keeps only a keyed hash of the token and pairs two asks with the same token from two different accounts. A blinded circuit is never resumed by its id: every reconnect pairs afresh under the next token, with a new handshake. While a conversation is open, the relay's memory still links its two members (as it did before). Once both have been gone about 10 minutes, nothing it keeps names the pair; waiting blind asks name no one; and no token or circuit id links one connection period to the next.

- **Healing.** Around midnight a phone tries the neighbouring day's token once. Two phones one counter apart heal by asking on two counters at once; further apart, they meet again at the next midnight.
- **The identity path remains** for first contact, after a phone lost its sessions, after a key rotation, and as a fallback: one identity ask only after a full day of unmatched blind asks, at most once per pair per day. A relay that withholds matches, or withholds a contact's knock and delivers it later, can force at most one such target-naming ask per pair per day. Every identity re-pair is noted on both phones.
- **Limits.** While a conversation is open, the relay still routes it between two known connections, each tagged with its account, and someone watching live sees two accounts pair at the same moment. A contact can tell that you have reconnected, because their next packet on the old circuit is refused.

7. Session setup: X3DH version 3

When a circuit forms, each phone:

1. refuses the peer if it is not accepted or is blocked;
2. checks that its own X25519 and ML-KEM ephemeral keys in the relay's message match one of its recent asks to that peer (used once), so a substituted or replayed setup

message is refused;

3. checks the peer's bundle signature, compares the peer's agreement key (and, when its contact code supplied one, its ML-KEM key) with the pinned ones in constant time, and refuses an agreement key already pinned for another identity;
4. **before any ML-KEM operation**, checks the peer's signature over its own ephemeral keys under the peer's pinned identity key. Without this, a relay could hand the initiator an ML-KEM key of its own as the responder's and decapsulate the post-quantum half;
5. computes four Diffie-Hellman values, all of which must be contributory (**IK** is each identity's X25519 agreement key, **EK** the ephemeral keys; **i** initiator, **r** responder). The agreement key takes the place of version 2's signed pre-key, which always equalled it, so **DH2** and **DH3** are the same value; the layout is kept from version 2:
 - **DH1 = DH(IK_i, IK_r)**
 - **DH2 = DH(EK_i, IK_r)**
 - **DH3 = DH(EK_i, IK_r)**
 - **DH4 = DH(EK_i, EK_r)**
6. on the initiator, encapsulates to the responder's ephemeral ML-KEM key, giving a 1088-byte ciphertext **ct** and a shared secret **ss_kem**;
7. derives **SK = HKDF-SHA256(salt = 32 zero bytes, ikm = DH1 || DH2 || DH3 || DH4 || ss_kem, info = "rchat/x3dh/v3" || Ed25519 identity key of the initiator || that of the responder || SHA-256(responder's ML-KEM ephemeral key || ct))**.

The initiator's first packet on the circuit is a **session-init**: **ct**, a 32-byte confirmation tag (an HMAC of **ct** under a key derived from **SK**) and its first ratchet message, real or a dummy. The responder decapsulates, derives the same **SK**, checks the tag and only then installs the session. A failure installs nothing. The ephemeral ML-KEM secrets are wiped right after use, so the post-quantum half is forward-secret.

Both sides contribute fresh ephemeral keys, and **both Ed25519 identity keys are bound into the derivation**, so a session key belongs to exactly one pair of identities. An attacker must break both X25519 and ML-KEM-768 to recover it. Authentication remains Ed25519, as in Signal's PQXDH: a future quantum computer could forge signatures, but could not read traffic recorded today. Because every session is keyed afresh, envelopes recorded under an older session do not decrypt in a new one.

A bundle with a bad signature is treated as a transient failure, because a hostile relay could forge one: the app closes the circuit and asks again with backoff. A pin mismatch is sticky: the contact shows “Keys don’t match” and the app does not ask again until the code is scanned again.

8. Messages

8.1 The Double Ratchet

After X3DH, both sides run the Double Ratchet.

- **Root chain.** `HKDF-SHA256(salt = root key, ikm = DH output, info = "RChat ratchet")` yields a new root key and chain key, whenever a new ratchet key arrives from the other side.
- **Chain keys.** `message key = HMAC-SHA256(chain key, 0x01)`, `next chain key = HMAC-SHA256(chain key, 0x02)`.
- **Encryption.** XChaCha20-Poly1305 under the message key, with a fresh random nonce.
- **Header.** 40 bytes: the sender’s ratchet public key, the message number and the previous chain length.
- **Out-of-order delivery.** Up to 100 skipped message keys per chain, and at most 1000 in all. A skipped key expires after 72 hours.

Sessions are kept on the phone, sealed under a key derived from the app’s database key and excluded from backups, so an app restart loses nothing except messages the relay was handing over at the moment the app was killed. Messages in flight when a connection drops are re-sent, byte for byte, on the next session. If a phone loses its sessions (a reinstall, or a restore onto a new phone), the other side learns it inside the new session, never from the relay, and resends what could not be read: automatically to the same, unchanged keys, and never after a key change. Automatic resending needs a signature from a key held in the contact’s phone hardware, so a stolen account key alone can’t trigger it.

8.2 Associated data

Every ratchet message is encrypted with `AAD = header || IK_sender || IK_recipient`. The header is authenticated, and a message only decrypts for the pair of identities it was written for.

8.3 Sealed relay payloads

Every payload the relay forwards or queues is sealed to the recipient. The sender makes a fresh X25519 key for each payload, derives an envelope key from it and the recipient's agreement key, and encrypts the **sender's identity key** together with the ratchet message or knock. The relay sees a message ID, the recipient and the ciphertext.

This is what lets the offline queue file items under the recipient alone (§11.2). It does not hide the sender from the relay in general: on a live circuit the relay knows both connections, and it knows which connection handed it a queued item at the time. This envelope uses X25519 only; the post-quantum protection is in the session itself and in knocks, which have their own hybrid envelope (§6.2).

A message that names the wrong sender inside the envelope fails to decrypt and is dropped. It is never attributed to anyone.

8.4 Receipts

Delivered and read receipts are ratchet messages, end-to-end encrypted like any other message. Only accepted contacts get receipts; a request never produces one. "Delivered" means the recipient's app has stored the message, not only that its phone received it. The relay's own acknowledgement means only that it has accepted the message: it has handed it to the recipient's connection, or holds it for the recipient's next connection. It never means delivered.

8.5 Media, files and disappearing messages

- **Photos** are re-encoded by default (HEIC, or JPEG as a fallback, 1600 pixels on the long side, about 150 KB) with all metadata removed: EXIF, GPS, camera details and dates. **Send as file** sends the original bytes and name instead, after a warning that the original includes any location and camera details. The **File** choice sends any document, up to just under 4 MiB. Voice notes are AAC at 64 kbps, up to 60 seconds.

- **Received files** are stored encrypted like other media, opened only through the system share sheet after a tap, and never previewed automatically except images, which go through the existing safe decoder. File names are cleaned on both ends: no path separators or look-alikes, no control or invisible characters, no leading dots.
- **Saving** a photo to Photos uses add-only permission, so rChat can never read the photo library. A notice says that saved copies leave rChat's encrypted storage and follow the phone's Photos and iCloud settings.
- **Disappearing messages.** Either person can set a conversation's timer: off, 5 minutes, 1 hour, 1 day, 1 week or 4 weeks. The change travels inside the session, and every message carries the timer in force when it was sent, which the receiver honours. The countdown starts when the relay accepts a sent message, and when a received message is first displayed (at the latest 7 days after receipt). Each phone deletes the message, its media file and its notification when it next runs. The other person can still screenshot or save it, saved copies are not deleted, and copies waiting at the relay expire within 72 hours.

9. Contacts and verification

Pinning is accepting. When the user scans or pastes a contact's code, or accepts a request, the app pins that contact's identity, agreement and ML-KEM keys and will not talk to them under different keys. A contact code is about 1.7 KB, because it carries the ML-KEM key and the identity's signature over all three keys; carrying the key in the code means the relay never has to be asked for it, which would reveal interest. Contact Info shows where the keys came from: "Keys from their code" or "Keys from their first message". A contact first reached through a request is trusted on first use: rChat cannot tell who is behind that key until the user checks it.

Safety numbers. For each contact the app computes a **60-digit** number from both people's pinned identity and agreement keys: 30 digits per person, from 5200 iterations of SHA-512, joined in a fixed order. The numbers match only if each phone pinned the other's real keys. Compared in full in person or on a call where the two recognise each other, this gives the same assurance as scanning the code in person. The app never shows only part of the number, because a short prefix could be ground out by an attacker.

- **QR comparison.** One phone shows the safety number as a QR code and the other scans it; the app compares the numbers and offers **Mark as compared** when they match. Anyone who knows both people's contact codes can compute the same number, so a photo of that screen shows that the two are contacts.
- **Mark as compared** is removed if the contact's keys later change. If it clears a key-mismatch warning, it clears only the warning: the pin is unchanged, and different keys from the relay raise the mismatch again.

Contact key rotation. A user can replace their identity with **Rotate keys**, once a day. The old identity key signs a link to the new keys at version + 1. Contacts in the plain pinned state follow it automatically; a contact showing a mismatch, blocked or not pinned never moves silently. The pin moves only when the new identity itself knocks, which proves it holds the new keys. Contacts see that the keys were updated, and any "compared" mark is dropped.

Whoever holds the old identity secret can sign a rotation, so **rotation does not recover a stolen account**: a thief could move contacts to keys they control. Each contact accepts only the first valid link for a version and at most one silent move a day, so a later competing rotation shows up as a request. Users who suspect theft should use **Reset Identity**, which starts over without contacts. Messages queued at the relay for the old keys are dropped; the sender's app shows how many may not have arrived and offers **Resend**.

10. Traffic shaping

- **Steady traffic** (opt-in, new in version 3). While the app is open and connected, the phone sends one packet every slot (0.5 seconds by default; 1 or 2 seconds), each exactly 1 KiB inside, to a conversation chosen at random among the open ones. Messages, receipts and session notices ride in the slots when they fit; otherwise the slot carries a dummy that the other side drops. To the network and the relay, a slot carrying a short message looks the same as a slot carrying a dummy. Its limits, stated in the app too:
 - it helps fully only when **both** people use it: a contact without it answers real messages with receipts at their own times, which shows which slots were real;
 - after a reconnect, real messages that were in flight are re-sent outside the rhythm, which shows afterwards which earlier slots carried them;

- photos, files, voice notes and texts over about 1 KB go outside the stream, as do requests and the start of a session;
 - a message waiting too long jumps the queue, the one timing leak inside the stream; a message waits on average about the slot length times the number of open conversations;
 - it costs about 10 MB an hour sent at 0.5 s (5 MB at 1 s, 2.6 MB at 2 s), plus up to as much received from each contact who uses it and has a conversation open with you; it does not hide that you are online, and it stops when the app is in the background.
- **Mandatory padding.** Every ratchet plaintext, messages and receipts alike, is padded to a size bucket: 256 B, 1 KiB, 4 KiB, 16 KiB, then multiples of 64 KiB. Padding hides the exact size, not the bucket. A receipt is the same size as a short message.
 - **Fixed-size requests.** Every knock is exactly 5120 bytes, with or without an intro, and every ask, named or blinded, is the same size.
 - **Receipts.** Sent 0.5 to 3 seconds after the message and batched per contact (in steady mode they wait for a slot instead). The relay can still place a read to within a few seconds.
 - **Keepalives** are jittered on both sides.
 - **Extra traffic protection** (off by default; replaced by steady traffic while that is on). While the app is open, it sends small dummy messages at random times, only to contacts heard from in the last minute, and each dummy is answered like a real message. It uses more data and battery and makes the timing of real messages only somewhat harder to read. Because dummies are sent only while the app is open, people with open conversations can tell when the user has rChat open.

These measures reduce what timing reveals; they do not remove it. A relay that sees both ends of a conversation can still correlate bursts, above all when only one side uses steady traffic.

11. What the relay stores

11.1 Logs

At the default level the relay logs no identity keys, IP addresses, message content or request paths. Where a log line must tell events apart it uses a short keyed hash that changes at every restart, and no log line carries two identity-derived values. An operator can add client IP addresses to `debug` lines for abuse handling; this is off by default.

11.2 The offline queue

Messages and requests for someone who is offline are held:

- **in memory by default**, nothing written to disk;
- filed under the **recipient only**, keyed by a hash that changes at every restart, with the sender's identity inside the sealed payload. Each record holds the payload, one enqueue time and whether it is a knock;
- for at most **72 hours**, which is both the default and the most an operator can set (the shortest is 1 hour), and deleted once delivered.

An operator can opt in to **encrypted persistence**, so held items survive a restart. The queue file is encrypted, and the relay refuses to start unless its 32-byte key is on a memory filesystem (tmpfs, ramfs or mfs), on a different filesystem from the data, owned by the service user and readable by it alone. The key is then lost when the machine reboots. An operator can override the memory-filesystem check; such a relay no longer has that property.

11.3 What the relay still learns

While it is running, the relay sees:

- which accounts are connected, when, and from which IP address, and each connected account's identity key;
- which two connections share an open conversation, and the size class, timing and direction of each packet between them, including the moment a request is accepted. Later conversations are paired by rotating blinded tokens, but while a conversation is open each side's routing entry carries that account's tag, so the relay's memory links the two members as before, and two accounts pairing at the same moment is visible to anyone watching live; once both have been gone about ten minutes, nothing it keeps names the pair;

- on first contact, after a lost phone or a key change, and at most once a day per pair as a fallback, which account is trying to reach which, for up to ten minutes; a request waiting for someone offline keeps a keyed pair tag until delivered, for at most 72 hours;
- for queued items, the recipient, the count, the sizes and the time each was queued;
- when a receipt is likely sent, from its timing.

It does not write any of this to its logs. Conversation state is discarded within ten minutes of both people disconnecting; a blinded conversation's routing record goes sooner, at once when its last member leaves and within ten minutes after either leaves. This is a reduction, not protection: anyone who operates, seizes or gains root on a running relay can read the connected accounts and which of them share open conversations from its memory.

Rotation is linkable. The relay receives no rotation data and stores no link between old and new identities, but it can usually link them, because the new identity contacts the same people straight away.

A relay that replays a recorded knock within its lifetime can probe whether the recipient has accepted the sender, by watching for an automatic ask back.

12. Transport and relay hardening

12.1 Transport

The relay speaks TLS 1.3 only, with no TLS 1.2, no 0-RTT and no plaintext mode outside local development. It can obtain and renew its own Let's Encrypt certificate (ACME TLS-ALPN-01 on port 443) or use a static certificate.

- **Post-quantum key exchange.** Connections between an rChat client and an rChat relay use the hybrid post-quantum key exchange X25519MLKEM768 (IANA group 0x11EC, draft-ietf-tls-ecdhe-mlkem), which combines ML-KEM-768 with X25519 so the TLS session keys stay secret unless both are broken; this applies to full and resumed TLS 1.3 handshakes alike, since every resumption runs a fresh hybrid exchange. It protects what the TLS link carries (the relay-visible metadata such as hello, key-bundle publish, circuit requests and sealed-envelope headers) against an adversary who records traffic today and decrypts it later with a quantum computer. Only the key exchange is post-quantum: the relay's TLS certificate signature, its Ed25519 relay identity and the client's **rk:1:** pin

of it, and the Ed25519 connection proofs remain classical. That is sufficient against record-now-decrypt-later, because a signature only has to resist forgery at the moment of the handshake and cannot be forged retroactively for a recorded session, but it does not protect against an attacker who already has a quantum computer at connection time and could impersonate the relay; moving authentication to post-quantum signatures is separate, future work. Clients and relays that do not offer the hybrid group fall back to classical X25519 (the relay also accepts P-256 and P-384); rChat clients offer only X25519MLKEM768 and X25519.

- **Resumption.** The relay issues TLS 1.3 resumption tickets, sealed with keys held only in memory that rotate every six hours, so a returning app resumes instead of running a full handshake. The app keeps at most two tickets for its pinned relay, each used once, in memory only; they are never written to disk. Resumption never skips rChat's own checks: every connection still proves the account's identity over that connection's TLS exporter, and the relay signs its identity over it. A presented ticket is visible on the network and can link two connections of one app made up to 12 hours apart.
- **Decoys.** Only a WebSocket upgrade to the operator's secret path reaches the relay; every other request gets a decoy website. Each relay's decoy is its own: the built-in page is chosen from the relay's secret identity key and its host name, so it can't be predicted from the host name and pin, and `--render-decoy` writes the same complete invented site (webmail, storage, photos and the like) as static files, with no external assets and login forms that never submit. The decoy keeps a casual scan from seeing what runs on the port; a determined prober can tell it apart from a real web server. rChat's traffic is not disguised as a web browser's.

Supported layouts: the relay owns port 443, or a TCP passthrough on the same host (HAProxy with `send-proxy-v2`, or Caddy with the caddy-l4 plugin) forwards raw bytes to it with a PROXY protocol v2 header. TLS-terminating reverse proxies and CDNs are not supported, and neither is nginx's stream module, which sends only PROXY protocol version 1.

12.2 Resource limits

Every limit has a default, and the relay refuses to start on a missing or out-of-range value.

- **Connections.** Caps on total connections, per IPv4 address or IPv6 /64, and per IPv6 /48. A new connection must prove an identity within 15 seconds by default. Hellos and publishes are budgeted per connection.

- **Rate and size.** A token bucket throttles each connection's inbound bytes; oversized frames close the connection.
- **Circuits and asks.** Caps per connection, per identity across reconnects, and relay-wide.
- **Per-source fairness.** A source is an IPv4 address or an IPv6 /48. Each source has a fair share of pending asks, circuits, the request store (at most 1/16) and the 24-hour quota of new requests. When a table is full, a lighter source displaces the heaviest source's entries rather than being refused. Behind a passthrough, these shares need the PROXY v2 header; without it they are off.
- **Request store.** Requests have their own 16 MiB budget, never charged to messages: at most 200 per recipient, one per pair, and 100 new ones per asker a day.
- **Memory budgets.** All outbound queues together are capped at 1 GiB and all WebSocket buffers at 512 MiB; when a budget is spent, the heaviest connection of the heaviest source is closed. A connection that accepts nothing for 30 seconds while a message waits is closed.
- **Panic containment.** The relay is built with overflow checks and unwinding panics, so a fault in one connection closes only that connection and the relay keeps serving everyone else. Core dumps are disabled.

12.3 Availability limits

These limits concern someone disrupting delivery. None of them helps anyone identify a user or read a message.

- **Request floods.** A sustained flood of requests from many sources can crowd older requests out of a full request store. Each source holds at most 1/16 of the store and 100 new requests a day, and requests never use the message budget.
- **Shared addresses.** Many users behind one carrier-grade NAT address share one fair share, so a busy shared address can be the one that is slowed first. Operators can raise the per-source limits for such users.
- **Slow-consumer harassment.** An accepted contact can send as fast as the rate limit allows. A recipient that cannot keep up is reconnected, and its messages wait safely in the offline queue.

13. Endpoints

Alerts while the app is closed. There is no notification server and nothing is sent to Apple. iOS wakes rChat from time to time, when it decides to (at best about every 15 minutes, often less); the app then connects, collects what the relay holds, stores it, posts local alerts and disconnects, without opening any conversation. Alerts show the sender's name only by default ("Name only"; "Nothing" and "Name and message" are the other choices), and a request always shows "New message request". With "Name and message", iOS keeps the text in its notification list, outside rChat's encrypted storage, until the chat is opened or the alert cleared. There are no alerts before the first unlock after a restart, after the app is force-quit until it is opened again, or for muted contacts. The relay sees such a check as a short connection with no conversations.

If a phone is compromised, the messages and keys on it are exposed. The app keeps its identity secret in the iOS Keychain, readable only on that device after first unlock, encrypts its local message store with a key held in the Keychain, and excludes its databases and media from device backups. Someone who seizes the unlocked phone, or copies the app's files together with its database key, also gets the current session keys: they can read messages still queued at the relay under those keys, and future messages until both sides complete a ratchet step. Keys already used or expired are deleted, and should not be recoverable, because iOS gives each file its own key and discards it when the file is replaced. It hides its content in the app switcher, blanks QR codes and safety numbers while the screen is recorded, refuses third-party keyboards, and checks incoming media before showing it.

14. Roadmap

Shipped in protocol version 3 (rChat 2.0.0), from the previous roadmap:

- *Post-quantum hybrid key agreement:* X25519 with ML-KEM-768 for every session and every request (§6.2, §7). Authentication is still classical.
- *Steady traffic:* opt-in same-size sending at a fixed rhythm while the app is open (§10). It reduces what timing reveals; it helps fully only when both people use it.
- *Rotating pair tokens:* later conversations are paired by blinded tokens that change daily and at every reconnect (§6.4). Someone watching a relay live can still see two accounts pair.

Planned work, in no particular order and with no dates.

Metadata

- *Shorter-lived relay state*: a blinded mailbox, so that a conversation's routing record does not outlive a member's connection.
- *Faster re-pairing* when a contact drops before answering the first message: today that can take a few minutes.

Cryptography

- *Key transparency*: a public, checkable log of account keys. Today safety numbers give the same check by hand.

Features

- *Voice and video calls*, inside the same end-to-end sessions. Steady traffic cannot cover a call, and waking a closed iPhone app for an incoming call needs Apple's VoIP notifications and a server to send them, which would be a deliberate decision with its own metadata cost.

Availability

- *Proof of work for requests*: a small cost per request, to blunt floods from many sources.

Assurance

- *Reproducible builds*: anyone could check that a published binary matches the source.

15. Not claimed

rChat does not claim that the relay doesn't know who talks to whom, that it has no metadata, that it is zero-knowledge or anonymous, that timing is invisible, that conversations cannot be linked, or that the sender is hidden from the relay ("sealed sender" in the sense other messengers use the term). It does not claim post-quantum authentication, that requests or intros are forward-secret, that rotation recovers a stolen account, that cover or steady traffic makes messaging hard to detect, that alerts arrive as soon as a message is sent, that disappearing messages cannot be kept by the other person, that its traffic looks like a web

browser's, or that messages queued for you are safe if your device's keys are stolen. It does not use "private" or "verified" as a status for a conversation, a contact or a server.

References

1. M. Marlinspike, T. Perrin. *The X3DH Key Agreement Protocol*. Signal, 2016.
2. T. Perrin, M. Marlinspike. *The Double Ratchet Algorithm*. Signal, 2016.
3. A. Langley, M. Hamburg, S. Turner. *Elliptic Curves for Security*. RFC 7748, 2016.
4. S. Josefsson, I. Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032, 2017.
5. Y. Nir, A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439, 2018.
6. S. Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305*. IETF draft.
7. H. Krawczyk, P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869, 2010.
8. E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446, 2018.
9. R. B. Shoemaker. *Automated Certificate Management Environment (ACME) TLS Application-Layer Protocol Negotiation (ALPN) Challenge Extension*. RFC 8737, 2020.
10. W. Tarreau. *The PROXY protocol, versions 1 and 2*. HAProxy Technologies.
11. NIST. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. FIPS 203, 2024.
12. E. Kret, R. Schmidt. *The PQXDH Key Agreement Protocol*. Signal, 2023.
13. K. Kwiatkowski, P. Kampanakis, B. Westerbaan, D. Stebila. *Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3*. IETF draft-ietf-tls-ecdhe-mlkem.