DChat

A messenger with key-pair identity, an on-chain identity registry, a staked relay network, and settlement inside the conversation

Whitepaper, version 1.0
31 July 2026

dchat.online

Contents

1. Introduction
2. System model
3. Identity
4. Direct messages
5. Group messages
6. Transport
7. Media
8. Real-time media
9. The client
10. The chain
11. The SDK
12. Verification and integrity
13. Security properties
14. Comparison
15. Roadmap
16. Conclusion
Appendix A. Parameter reference
Appendix B. Document status

Abstract

DChat is an end-to-end encrypted messenger in which the account is a secp256k1 key pair rather than a phone number, message transport is a store-and-forward relay network run by staked independent operators rather than a company's servers, and the registry that maps a human-readable username to an account is a public Cosmos SDK blockchain rather than a private directory. A non-custodial wallet is built into the same client and derives from the same seed, so a payment is something you send inside a conversation rather than in another application.

Direct messages use secp256k1 ECDH, HKDF-SHA256 and ChaCha20-Poly1305. Group messages use per-member sender keys distributed over the pairwise channel, so a group is end-to-end encrypted with no server ever holding a group key. Chat keys and sender keys carry generation numbers and rotate automatically after 10,000 messages or 30 days, which bounds the value of any single key compromise. One-to-one voice and video calls are peer-to-peer and end-to-end encrypted. Undelivered ciphertext is held by relay nodes for at most 14 days and no operator holds a key that would decrypt it.

Around that core the client ships the feature surface people actually expect from a messenger: groups with roles and invite approval, polls, checklists, live location, video notes, stickers, message translation, albums, disappearing messages, multi-device linking by QR, and group calls for up to 25 participants.

This document specifies the system as implemented, at the level of detail required to reimplement it or to attack it. Section 13 states which security properties DChat provides and which it does not, in the same manner as comparable published documentation from other private messengers. DChat has not been independently audited.

1. Introduction

Every private messenger makes three decisions, and almost every interesting difference between them follows from those three.

What is an account? Signal, WhatsApp and Telegram bind an account to a phone number. That is convenient for contact discovery and unhelpful for anyone whose threat model includes the entity that issued the number. DChat binds an account to a key pair generated on the device from a BIP-39 seed. There is no username-and-password credential anywhere in the system, so there is nothing to phish, nothing to reset, and no account database to leak.

Who runs the servers? Signal runs its own. Matrix federates. DChat distributes message storage across service nodes that anyone can operate, held honest by a stake that is slashed for downtime and by a peer-probing uptime quorum. Losing any node loses nothing.

What happens to the money? Most messengers answer "nothing, that is a different application". DChat treats settlement as part of the conversation. The wallet derives from the same seed as the identity, funds are non-custodial, and sending value is a message type.

DChat's distinctive contribution is the combination. Key-pair identity and staked relay storage are not new on their own. Pairing them with a real on-chain identity registry, with usernames, DIDs, verification attestations and moderation state, and with in-conversation settlement, and then shipping all of it inside a client with a full consumer feature set, is.

1.1 What this document covers

Sections 2 to 7 specify identity, message encryption, groups and transport. Sections 8 to 10 cover media, real-time calls and the client. Section 11 covers the chain, the token and node economics. Section 12 covers the SDK. Section 13 states the security properties. Section 14 compares DChat to Session and Signal. Appendix A is a single table of every parameter in the system.

1.2 Notation

`||` denotes concatenation. Multi-byte integers are big-endian. Base64 is RFC 4648 with padding. Byte lengths are exact and enforced at runtime. Where a parameter has a value in the shipped code, that value is given rather than a range.

2. System model

2.1 Components

Clients. The application on a user's device. It generates and holds every private key, performs all encryption and decryption, owns the local message database, and is the only component that ever sees plaintext. Clients are built on the DChat SDK.

Service nodes. Independently operated servers running a fork of CometBFT into which a store-and-forward relay called `swarmchat` has been added. A node participates in chain consensus and, separately, accepts client connections over gRPC, persists ciphertext for recipients who are offline, and gossips that ciphertext to its peers. Registration requires a stake; downtime is slashed.

The chain. A Cosmos SDK v0.50.13 application on CometBFT v0.38.17, carrying the identity registry, profiles, service-node staking and rewards, and client-reward accounting. It does not carry message content.

Auxiliary services. A middleware HTTP API that proxies IPFS storage and issues short-lived TURN credentials, a coturn deployment for NAT traversal, and a Janus selective forwarding unit for group calls.

2.2 Design goals

1. No party other than the intended recipients can read message content, including every server operator in the path.
2. No credential exists that any operator could be compelled to produce that would decrypt traffic.
3. Account creation requires no phone number, email address or other third-party identifier.
4. The loss of any single service node loses no messages and locks no user out.
5. The user holds the funds, always.
6. A single key compromise has a bounded blast radius.

2.3 Trust assumptions

The client is trusted completely: it holds the keys. Platform secure storage (Android Keystore, iOS Keychain) is trusted to protect the device master key. Service nodes are trusted for availability only, not for confidentiality or integrity of content; a node that returns altered ciphertext produces an authentication failure rather than a wrong plaintext. The chain is trusted for registry integrity under the usual Byzantine fault tolerance assumption. The middleware and TURN services are trusted for availability and see only ciphertext.

3. Identity

3.1 The account is a key pair

There is no username and password credential anywhere in DChat. The account is a secp256k1 key pair generated on the device at first launch. A username is chosen at the same time and registered on the chain, bound to that key pair, but the username is a label; the key pair is the account, and the username can be looked up by anyone. See Section 3.7.

The corollary, and the client states it during onboarding rather than burying it: if the recovery phrase is lost and no device holding the key survives, no one can restore the account.

3.2 The seed

Recovery phrases are BIP-39 mnemonics over the English wordlist. The implementation supports 12 and 24 words and the client generates 24 by default, giving 256 bits of entropy. Every generated mnemonic is round-tripped through derivation before it is displayed, and a mnemonic that fails to re-derive is discarded rather than shown, so a user is never handed a phrase that does not work. Leaving the phrase screen requires an explicit confirmation.

3.3 Derivation

```
seed     = BIP39-Seed(mnemonic)
master   = BIP32-Master(seed)
identity = master / 44' / 118' / 0' / 0 / 0
```

Coin type 118 is the registered Cosmos SLIP-44 index. The curve is secp256k1. Private keys are 32 bytes; public keys are stored and transmitted in 33-byte compressed form.

3.4 Address

```
address = bech32("dc", RIPEMD160(SHA256(pubkey_compressed)))
```

Addresses are of the form `dc1...`. Group identifiers share the `dc` prefix and are validated against it before any key operation.

3.5 Signatures

ECDSA over secp256k1, deterministic per RFC 6979, with SHA-256 pre-hashing. Signatures are emitted as a raw 64-byte `R || S` pair rather than DER, matching Cosmos ADR-036 for offline verification. Every signature is verified against its own public key immediately after generation and generation fails closed if that check does not pass. Verification accepts both the 64-byte raw form and 70 to 72-byte DER.

3.6 Key hierarchy

Five key types exist and are never interchanged.

Key	Curve	Origin	Purpose	Rotation
Identity key	secp256k1	Seed, at `m/44'/118'/0'/0/0`	Transaction and ADR-036 signing, wallet	Never
Chat key	secp256k1	Generated, independent of the seed	ECDH key agreement for direct messages	10,000 messages or 30 days

Key	Curve	Origin	Purpose	Rotation
Sender key	symmetric	32 CSPRNG bytes, one per member per group	Group message encryption	Same policy, plus on membership change
Account rotating key	secp256k1	Generated, shared across the user's own devices	Second-layer account operations	Every 7 days, 72-hour grace
IPNS signing key	Ed25519	HKDF from the identity seed	Signing IPNS records only	Never

Separating the chat key from the identity key is deliberate. It means the key that encrypts conversations can rotate freely without touching the key that holds funds, and a chat key discloses nothing about the account that owns it.

3.7 The on-chain registry

Registering a username submits `MsgRegisterIdentity` to the identity module, which enforces uniqueness through a `username -> DID` index, creates or fetches the account, generates a DID, and stores the record. The chat public key is stored separately, keyed by wallet address, so a correspondent can fetch it before the first message and no key-exchange server is needed.

The module also supports `ReportUser`, `SetVerification` (a verifier-signed attestation with a rotatable set of Ed25519 verifier keys), `UpdateChatKey`, and a governance-gated `PurgeIdentity` that erases the identity record, the username reservation, the chat public key, and the corresponding profile and client-reward records.

This is a genuine registry rather than a hash placed on a chain for marketing reasons. It gives globally unique, censorship-resistant names that resolve without asking any company, verification badges that anyone can check independently, and a moderation record that no single operator controls.

The trade-off is inherent to putting a registry on a public ledger, and it must be stated plainly because a user cannot opt out of it. **A username is required at account creation:** the client will not leave the Create Account screen until a username passes its availability check, so every account has one and every account is therefore in the public registry. Chain state is public, so that username resolves publicly to a DID, to a wallet address, and from there to the account's full public transaction history. A DChat username is a public directory entry with a public financial history attached. A user who needs their social identity unlinkable from their transactions must use a username unconnected to that identity, and must treat the wallet attached to it as public. This is a consequence of the design rather than a setting, and Section 15 lists optional registration as a change worth making.

4. Direct messages

4.1 Key agreement

Direct messages use static-static ECDH between the sender's chat private key and the recipient's chat public key. This is the `ecdh` mode on the wire.

```
P      = ECDH-secp256k1(sk_sender_chat, pk_recipient_chat)
shared = X(P)                      // 32-byte x-coordinate, prefix stripped
```

The shared secret is never used as a key directly. It is only ever input keying material for HKDF.

4.2 Key derivation

```
salt = SHA-256("dchat-salt-v1")
info = "dchat-message-encryption-v1"
K    = HKDF-SHA256(ikm = shared, salt, info, L = 32)
```

Domain separation across purposes is carried by distinct context strings, each isolating its own key: `dchat-config-v1`, `dchat-sync-v1`, `dchat-pairing-ecies-v1`, `dchat-ipns-v1`, `dchat-notes-media-v1`, `dchat-acct-cfg-secret-v1`, and `dchat-local-db-v1`. A key derived for one context cannot decrypt another context's data.

4.3 Authenticated encryption

```
nonce   = CSPRNG(12)
ct, tag = ChaCha20-Poly1305-Encrypt(K, nonce, plaintext)
```

Key 256 bits, nonce 96 bits, tag 128 bits, per RFC 8439. A fresh nonce is drawn from the platform CSPRNG for every message. Nonces are not counter-derived, so there is no cross-device counter state that can desynchronise between a phone and a linked browser. The implementation is BouncyCastle's managed `ChaCha20Poly1305`, chosen because it behaves identically on Android and in WebAssembly.

4.4 Wire envelope

```
{
  "ciphertext":       "<base64>",
  "iv":               "<base64, 12 bytes>",
  "authTag":          "<base64, 16 bytes>",
  "senderPublicKey":  "<hex, 33 bytes compressed>",
  "keyGeneration":    0,
  "chat_id":          "<string>",
  "encryption_mode":  "ecdh" | "sender_key" | "ecdh_bootstrap" | "none",
  "sender_did":       "<string>"
}
```

The `ciphertext` field is the confidential payload. The remaining fields are routing and key-selection material and are visible to a relay, which is what allows a relay to deliver a message it cannot read. Integrators should therefore treat `chat_id` and `sender_did` as public values.

On decryption failure the client retries against every historical key generation it holds, then falls back to the set of alternate recipient chat public keys, which is what allows a user's own linked devices to decrypt messages that user sent.

4.5 Key rotation

Chat keys and sender keys carry a generation number and rotate automatically. Rotation fires when **any** threshold is crossed.

Policy	Messages	Age
Default	10,000	30 days
Conservative	5,000	14 days
Relaxed	20,000	60 days

Emergency rotation can be triggered manually at any time for a key believed to be compromised. The `keyGeneration` field lets a recipient select the correct historical key, so rotation never renders already-received history unreadable, which is the failure mode that makes naive key rotation unusable in practice.

Automatic rotation is the mechanism that bounds the blast radius of a key compromise, and it is a property several comparable messengers with key-pair identity do not implement at all. It is coarse-grained forward secrecy rather than the per-message forward secrecy of a Double Ratchet, and Section 13 states the difference precisely.

The message counter persists every 50 messages rather than on every send, so a hard process kill can lose at most 49 increments and delay a rotation by at most 0.5 percent of the threshold. The age-based cap is unaffected.

5. Group messages

5.1 Sender keys, and why groups are end-to-end encrypted

Groups use the N-sender-keys model. Every member holds their own symmetric sending key for the group, rather than the group sharing one key that must be reissued wholesale. A sender key is 32 bytes drawn directly from the platform CSPRNG and is not derived from the identity key, so it discloses nothing about the account that owns it.

Each member's sender key is delivered to every other member individually, wrapped in the pairwise ECDH channel of Section 4. No server ever handles a sender key in the clear, and no server can construct one. A group message is therefore end-to-end encrypted in the full sense: the ciphertext leaves the sender's device encrypted, the relay forwards material it cannot read, and only members holding the sender key can decrypt.

A message to the group is encrypted once under the sender's own sender key using ChaCha20-Poly1305 with the parameters of Section 4.3. Recipients select the decryption key by the sender DID and key generation carried in the envelope.

5.2 Membership and administration

Group membership, roles, name and photo changes propagate as unicast control messages to each member, ordered by a client-side `StateVersion`, with admin approval carried in `ApprovingAdmins` and `ConsensusReached` fields.

No group state is written to the chain. That is a deliberate privacy choice: a chain-visible membership list would be a permanent public record of who is in which group. Holding group state client-side keeps the social graph of a group off every server and off the ledger. The cost is that group state is a replicated client-side structure rather than a consensus object, so clients that receive control messages in different orders can briefly hold divergent views.

For a member added while no other member is online, an Ed25519-signed IPNS group-authority pointer allows the new member to fetch the latest admin-signed membership snapshot without a live peer.

The model is two roles, Member and Admin, with the creator becoming an Admin. Invite links generate join requests that an admin approves. Groups hold up to 100 members. New members cannot read history from before they joined, because they do not hold the earlier sender keys.

5.3 Removal

When a member is removed or leaves, every remaining member's sender key is rotated and redistributed to the reduced membership, so the departed member cannot decrypt anything that follows. Historical keys are retained locally, so already-received messages stay readable.

Two implementation notes matter to anyone building a third-party client on the SDK. The SDK exposes the rotation call and the DChat application invokes it on removal and on leave; a client that omits the call leaves departed members able to decrypt until the routine policy rotation fires. And rotation iterates members individually and tolerates per-member failure, so an interrupted rotation can leave members on different generations until it is retried.

6. Transport

6.1 The relay network

Message transport is not the blockchain. It is `swarmchat`, a store-and-forward relay implemented as an addition to a vendored CometBFT fork: a peer-to-peer reactor between nodes on channel `0x50`, plus a gRPC front door for clients on port 9091. Every service node runs both, so the same staking, slashing and uptime machinery that secures consensus also secures message delivery.

No message content is written to consensus state, is part of the application hash, or is voted on by a validator.

6.2 Client transport

Native clients open a bidirectional gRPC stream. WebAssembly clients use gRPC-Web server streaming. Acknowledgements are signed unary calls, coalesced and batched every 250 milliseconds rather than one signed call per message. The local send queue retries three times. Reconnection uses a fixed backoff of 3, 5, 10, 30 and 60 seconds.

Authentication is one of two forms: an ADR-036 wallet signature over a canonical payload bound with a timestamp and a nonce, which makes a captured request non-replayable, or a wallet-signed device certificate that vouches for a device key, used by linked devices that never hold the seed.

6.3 Persistence and retention

Each node keeps a local LevelDB pool holding opaque ciphertext.

Class	Retention
Regular messages	14 days
Configuration messages, including key material	30 days
Namespaced ConfigSync entries	Persistent, last writer wins per namespace
Ephemeral types: call signalling, typing, presence	Never persisted

A per-user cap of 10,000 messages applies, oldest evicted first, with a cleanup sweep every hour.

Retention exists so that a message sent to someone whose phone is off still arrives when they turn it on. What a relay holds is ciphertext, for a bounded window, under no key any operator possesses.

6.4 Replication

A message is persisted locally with a per-recipient monotonic sequence number, delivered immediately to any live device streams belonging to the recipient, and then forwarded to every connected peer node. Every node holds every message until its receiver drains it.

The property this buys is unusually strong availability: a client can connect to any service node in the network and find its mail, and no node is special or assigned. Pending messages are retried to peers that have not confirmed receipt every 60 seconds for up to 30 minutes, and a peer that reconnects receives the full pending backlog. Duplicate suppression uses an in-memory seen-set with a 45-minute lifetime.

Because storage and bandwidth per node scale with total network traffic rather than with a node's share of it, moving to sharded assignment is the principal scaling item on the roadmap.

6.5 Delivery states

`MessageStatus` takes the values Sent, Delivered, Read, Received and Retrying. One tick means the relay accepted the message. Two ticks mean the recipient's device acknowledged it. Read receipts are themselves ordinary encrypted direct messages sent back to the sender, batched per sender as a high-water-mark receipt rather than one receipt per message.

6.6 Multiple devices

Delivery watermarks are per device: an acknowledgement advances only the acknowledging device's watermark rather than deleting the message, so a user's other devices still receive it. Clients drain by delta using the monotonic sequence number, with a sentinel that skips the pool for a fresh or recovered device.

Devices are provisioned by QR. The seed-holding device exports an encrypted provisioning bundle, registers the new device in an IPNS-backed device registry, and triggers an archive export so the new device can bootstrap history. Revoking a device tombstones it in the registry and rotates keys, so a revoked device cannot decrypt anything sent afterwards.

7. Media

Media bytes are base64-encoded, encrypted with the same ECDH and ChaCha20-Poly1305 envelope used for text, and the resulting JSON is signed with the identity key. The signed blob is uploaded to IPFS through a middleware HTTP API whose requests carry ADR-036 signature headers; the SDK never contacts a raw IPFS gateway. Integrity is checked with a SHA-256 of the media, and the upload request carries a SHA-256 of the payload as signed custom data, so a substituted blob is detected rather than decrypted.

Thumbnails up to 150 KB travel inline as base64 rather than being uploaded, so a chat list renders without a network round trip per row. The client caps images at 50 MB. Avatars use AES-256-GCM.

The content hash of an encrypted blob is public, which is what content addressing means. The content is not.

8. Real-time media

8.1 One-to-one calls

One-to-one voice and video calls use WebRTC through SIPSorcery. The media path is peer-to-peer where the network permits and falls back to a relay when it does not. DTLS-SRTP terminates on the two participating devices, so **one-to-one call media is end-to-end encrypted**. Signalling travels as ephemeral relay messages that are never persisted.

ICE servers are issued as short-lived credentials by the middleware using the coturn REST pattern, with a public STUN fallback if the middleware is unreachable.

As with any peer-to-peer WebRTC call in any application, a direct call reveals your IP address to the person you are calling and to the TURN server. Users in high-risk situations should account for that.

8.2 Group calls

Group calls route through a Janus VideoRoom selective forwarding unit supporting up to 25 publishers per room. Media is encrypted with DTLS-SRTP between each participant and the SFU, which decrypts each incoming stream and re-encrypts it outward to the other participants. This is the standard SFU architecture and it is what makes many-party video calls feasible on mobile hardware at all.

The consequence is that **group calls are encrypted in transit rather than end-to-end**: the SFU is inside the trust boundary for group call media, while group *messages* remain fully end-to-end encrypted under sender keys. Users who need end-to-end confidentiality for a live conversation should use a one-to-one call. Adding frame-level encryption to remove the SFU from the trust boundary is on the roadmap in Section 15.

9. The client

9.1 Data at rest

The message and contact database is SQLCipher-encrypted SQLite. The 32-byte database key lives in platform secure storage. On first run it is either a preserved legacy key, an HKDF-SHA256 derivation from the device private key with context `dchat-local-db-v1`, or a fresh CSPRNG key when no device identity yet exists.

Key material held by the SDK is separately wrapped under a device master key from platform secure storage:

```
device_key = CSPRNG(32)
K_at_rest  = PBKDF2-HMAC-SHA256(device_key, salt, iterations, 32)
blob       = nonce(12) || tag(16) || ciphertext
```

PBKDF2 runs at 250,000 iterations on native platforms. In WebAssembly it runs at 10,000, because WASM is single-threaded and the higher count blocks the event loop for two to three seconds; the input there is a 256-bit random device key rather than a human password, so the KDF is not performing brute-force resistance work.

The device master key is stored twice, primary and backup, because losing it orphans every blob it wraps. Android Auto Backup is disabled for the application, because Keystore-wrapped material cannot survive a cloud restore and allowing it would produce blobs that decrypt nowhere.

9.2 Feature surface

DChat is a full consumer messenger, not a minimal privacy tool. It ships replies, edits, delete for me and delete for everyone, forwarding, pinning including per-side pinning in one-to-one chats, starring, ten animated reactions, voice messages, round video notes, albums of two to ten items, files, polls with optional multiple answers, checklists, live location with a duration, contact cards, message translation, in-chat search, and per-chat drafts. Disappearing messages offer off, 24 hours, 7 days, 1 month and 3 months, applying to new messages.

Privacy controls cover who may contact you, who may add you to groups including a per-contact allow list, blocking, and a biometric app lock with configurable timeout. Message delivery has three modes trading battery against latency: push notifications, battery saving, and a persistent connection.

The client targets Android from API 24, with iOS, Mac Catalyst and Windows build targets. Five languages are selectable: English, Arabic, Persian, Kurdish Sorani and Pashto, with right-to-left layout support throughout.

9.3 Backup and restore

Restoring from the recovery phrase restores the identity key, and therefore the account and the wallet, immediately and without contacting anyone.

Message history is restored separately from an encrypted backup: either an archive located through an IPNS pointer derived from the seed, or a local backup file the user selects. The backup carries messages, contacts and optionally media, and nothing downloads automatically. Keeping history out of the seed is deliberate: a seed that reconstructed message history would make a written-down phrase equivalent to the entire archive.

10. The chain

10.1 Purpose

The chain is a public ledger for identity and stake. Its jobs are to make usernames globally unique and resolvable without a central directory, to hold verification attestations and moderation state, to make service-node operation stake-backed and slashable, to account for client rewards, and to settle payments.

10.2 Modules

Module	Function
`identity`	Username registry, DID, chat public key directory, verification, reports, bans, governance-gated purge
`profile`	Optional profile record bound to a DID
`servicenode`	Registration, staking, heartbeats, uptime proofs, epoch rewards, downtime slashing
`clientreward`	Message-activity rewards with proof-of-humanity gating, referrals, streaks
`grpc`	The client-facing gRPC surface, including the chat service

10.3 Token and fees

The base denomination is `udchat` with 6 decimals, so one DCHAT is 1,000,000 udchat. The client displays "DChat".

Nodes enforce a minimum gas price of 0.0001 udchat. The client offers three tiers, 0, 0.025 and 0.04 udchat per unit of gas, presented as Slow, Normal and Fast. Gas is simulated then multiplied by 1.4 as a safety margin, and a sequence mismatch triggers one automatic re-sign and retry.

Messaging costs nothing. Fees apply only to on-chain transactions: payments, staking and identity registration.

10.4 Service node economics

Parameter	Value
Minimum stake	10,000,000 DCHAT
Operator's own minimum share	One quarter of the minimum stake
Maximum co-stakers per node	10
Heartbeat interval	300 seconds
Unbonding period	7 days
Reward epoch	Every 100 blocks
Payable after	4 blocks
Daily pool fraction	0.151
Downtime slashing	5 percent
Jail duration	720 blocks
Initial decommission credit	240 blocks, about 2 hours
Maximum decommission credit	2,880 blocks, about 48 hours

Rewards are split evenly across payable active nodes each epoch, then divided among a node's co-stakers in proportion to stake after the operator takes a fee in basis points. Rewards accrue to a pending balance and are claimed explicitly.

Uptime is verified by nodes probing each other rather than by self-report. A quorum of 10 verifiers is derived deterministically from the current block's application hash and 7 must agree. Probes run every 720 blocks with a 4-second timeout. A verifier signs `SHA256(subject || height || result)` with its registered secp256k1 key, and the chain rejects a proof from a node outside the derived quorum or one that has already submitted for that subject and height. An operator cannot vouch for their own node, cannot choose who probes it, and cannot predict the quorum in advance.

10.5 Client rewards

A separate module rewards users for message activity, gated by proof of humanity through World ID, BrightID or Gitcoin. A nullifier is bound on first claim and is immutable thereafter.

Parameter	Value
Epoch	100 blocks
Daily cap	100 DCHAT
Bootstrap cap, first 14 days after verification	10 DCHAT per day
Pool daily emission	0.0274 percent of pool balance
Counted messages per sender-recipient pair	20 per epoch, 100 per day
Unacknowledged event expiry	14 days
Claim maturity hold	7 days
Claims per day	1
Referral payout	5 DCHAT, maximum 50 referrals, 14-day hold
Streak minimum	5 messages per day
Streak multiplier	1 + min(days x 0.02, 0.50), capped at 1.50x at 25 days

Per-edge caps, proof-of-humanity gating and the maturity hold exist to make farming uneconomic rather than merely discouraged. Reward computation runs in the end-blocker at epoch boundaries and is idempotent, and identifiers are generated deterministically from epoch, height and counter rather than from a random source, because randomness would cause a validator application-hash mismatch.

11. The SDK

The DChat SDK is a .NET library targeting `net10.0` with Android, iOS, Mac Catalyst and Windows heads. It provides identity and key management, the message envelope and its encryption, transport with reconnection and acknowledgement batching, groups and sender keys, contacts, media, the wallet and staking, device provisioning, and the IPNS-backed device registry. It uses NBitcoin for BIP-39 and BIP-32, BouncyCastle for ChaCha20-Poly1305 and Ed25519, and gRPC natively with gRPC-Web on WebAssembly.

It does not contain the WebRTC media pipeline: call signalling types and TURN credential fetching live in the SDK, while camera capture, encoding and the media path live in the client application.

The protocol is documented at parameter level in the published cryptographic specification, at a level sufficient to reimplement it or to attack it. The SDK itself is not currently published as open source; source-level engagement is available through the enterprise programme.

Two responsibilities the SDK does not carry for an integrator: post-removal sender-key rotation is invoked by the client rather than enforced by the SDK, and only the `ciphertext` field of the envelope is confidential.

12. Verification and integrity

Several properties of the system are checkable by a third party without trusting any statement in this document.

The chain is public: anyone can run a node, query the identity registry, verify that a verification badge carries a valid verifier signature, audit service-node stakes and slashing events, and confirm that no message content appears in state. Reward distribution is deterministic and recomputable from chain data. Uptime proofs are signed and quorum-bound, so a false uptime record requires collusion by 7 of a quorum of 10 that the prover did not choose.

Message authenticity does not rest on transport trust: every envelope is authenticated with a Poly1305 tag under a key the relay does not hold, so a relay that alters, forges or replays ciphertext produces a decryption failure rather than an accepted message.

13. Security properties

This section states what DChat protects and what it does not, in the manner of comparable published documentation from other private messengers.

13.1 Adversaries and outcomes

Adversary	What they get	What they do not get
Passive network observer	Ciphertext, envelope routing fields, timing	Message content, call content, keys
Service node operator	Ciphertext for the retention window, connecting IP, envelope routing fields	Message content, sender keys, any decryption key
Full server breach	Ciphertext and routing metadata	Any key that decrypts anything
Legal demand on an operator	Retained ciphertext within its window, connection metadata, public chain state	Plaintext, which the operator cannot produce or derive
Stolen device, app lock on	An encrypted SQLCipher database and Keystore-wrapped key material	Message history, keys
Malicious correspondent	Everything you sent them	Anything you did not

13.2 Properties provided

1. Confidentiality of direct and group message content against every party except the intended recipients.
2. Integrity and authenticity of message content through AEAD and signature verification.
3. End-to-end confidentiality of one-to-one call media.
4. Automatic key rotation bounding the blast radius of any key compromise to a window of 10,000 messages or 30 days, with immediate manual rotation available.
5. Forward-secure group removal: a departed member is cut off by sender-key rotation rather than by policy.
6. No phone number, email address or other real-world identifier at account creation.
7. No operator-held credential that could be compelled to decrypt traffic.
8. Non-custodial funds.
9. Availability under the loss of any individual service node.
10. Encryption at rest for both message history and key material on the device.

13.3 Properties not provided

1. **Per-message forward secrecy and post-compromise security.** Key agreement is static-static ECDH with automatic generation rotation rather than a Double Ratchet. Compromise of a chat key exposes that rotation window rather than a single message, and the session does not self-heal.
2. **Metadata protection.** Envelope routing fields are visible to relays, so a relay operator can observe communication patterns. DChat is pseudonymous rather than anonymous.
3. **Traffic-analysis resistance.** There is no padding, cover traffic or delay mixing, so message timing and approximate size are observable.
4. **IP protection from service nodes.** There is no onion-routing layer. Users who need IP protection should use a VPN or Tor, both of which work normally with DChat.
5. **End-to-end encrypted group calls.** The SFU is inside the trust boundary for group call media. Group messages are unaffected and remain end-to-end encrypted.
6. **IP privacy on one-to-one calls,** which are peer-to-peer by design.
7. **Consensus-backed group state.** Group membership is client-asserted, which keeps it off every server at the cost of strict ordering guarantees.
8. **Registry privacy.** Every account has a username, because the client requires one at creation, and every username is public on the chain along with the address it resolves to. There is no unregistered mode today. Choose a username that is not linkable to your real-world identity.
9. **An independent audit.** The primitives are standard and used in their intended modes, and the full specification is published, but no external cryptographic review has been completed. Commissioning one is the first item on the roadmap.

14. Comparison

14.1 Against Session

Session is the closest comparable system: key-pair identity from a mnemonic, message storage on staked service nodes, and a 14-day retention window. The differences are material.

	DChat	Session
Forward secrecy	Chat keys and sender keys rotate automatically every 10,000 messages or 30 days	No perfect forward secrecy at present, with re-implementation planned for a future protocol revision
Group messages	End-to-end encrypted, per-member sender keys, up to 100 members	End-to-end encrypted, up to 100 members
Large public channels	Not offered	Communities, encrypted in transit to the hosting server rather than end-to-end
Group calls	Up to 25 participants	Not available
One-to-one calls	Available, peer-to-peer, end-to-end encrypted	Available in beta, off by default, peer-to-peer, IP shared with the call partner and a foundation-operated TURN server
Usernames	Registered on-chain, free, with verification attestations	A separate name service, historically purchased

	DChat	Session
Payments	Native, on-chain, non-custodial, inside the conversation	Not offered
Attachment size	50 MB	10 MB
Feature surface	Polls, checklists, live location, video notes, stickers, translation, albums, per-side pinning	Core messaging with disappearing messages and reactions
IP protection from storage nodes	None at present	Onion requests
Source availability	Specification published, SDK not published	Fully open source
Independent audit	Not yet	Audited by Quarkslab

The honest summary: Session leads on routing privacy and on transparency, both of which are real advantages and neither of which this document disputes. DChat leads on the cryptographic property that matters most for a key-pair messenger without a ratchet, which is automatic key rotation, and it leads substantially on capability: group calls, in-conversation settlement, a first-class identity registry, and a feature set built for everyday use rather than for a minimal privacy tool.

14.2 Against Signal

Signal remains the reference point for message confidentiality. X3DH and the Double Ratchet provide per-message forward secrecy and post-compromise security, sealed sender hides the sender from the server, and the protocol has been audited and formally analysed repeatedly. DChat does not claim parity on ratcheting, and Section 13.3 says so.

Signal binds an account to a phone number and runs its own centralised infrastructure. DChat requires no identifier at all, distributes storage across independently staked operators, and holds funds non-custodially. These are different products for different threat models, and a user who needs a ratchet above all else should use Signal.

14.3 Against federated systems

Matrix federates rather than decentralising: homeserver operators hold room state and see substantial metadata. DChat holds group state entirely client-side, which removes that metadata sink at the cost of strict ordering. Status pairs a chain with a messenger as DChat does, and faces the same trade-off in making the identity graph public in exchange for removing the central directory.

15. Roadmap

1. **Independent security audit.** External review of the protocol and implementation.
2. **A ratcheting protocol.** X3DH-style agreement with a Double Ratchet, replacing static-static ECDH, delivered as a versioned migration.
3. **End-to-end encrypted group calls.** Frame-level encryption keyed from the existing sender-key infrastructure, removing the SFU from the trust boundary.
4. **Envelope metadata minimisation.** Recipient-blinded addressing so routing fields no longer expose communication patterns to relays.

5. **Sharded storage assignment.** Decoupling per-node cost from total network traffic.
6. **Decentralising auxiliary services.** IPFS middleware, TURN and the SFU.
7. **Optional username registration.** Allowing an account to exist without a public registry entry, for users who do not need a resolvable name.
8. **Post-quantum key agreement,** in step with the ratchet work.

16. Conclusion

DChat is a working messenger in which the account is a key pair, the servers are staked and independently operated, group messages are end-to-end encrypted under per-member sender keys, encryption keys rotate automatically, usernames resolve through a public registry rather than a company, and money moves inside the conversation. It ships this with a consumer feature set most privacy messengers do not attempt, and with a cryptographic specification published at a level of detail that allows the design to be checked rather than believed.

It is not finished. It has no ratchet, it does not hide communication patterns from relays, its group calls trust a forwarding unit, and it has not yet been audited. Section 13 lists these in full, and Section 15 says what is being done about them.

Publishing both halves is the point. A messenger asks for more trust than most software, and the only durable basis for that trust is a specification precise enough to be tested and a list of limitations honest enough to be useful.

Appendix A. Parameter reference

Parameter	Value
Mnemonic	BIP-39 English, 12 or 24 words, 24 by default
Derivation path	`m/44'/118'/0'/0/0`
Identity curve	secp256k1
Address encoding	bech32, HRP `dc`
Address hash	RIPEMD160(SHA256(compressed public key))
Signature	ECDSA secp256k1, RFC 6979, SHA-256 pre-hash, raw 64-byte R‖S
Direct message key agreement	Static-static secp256k1 ECDH
KDF	HKDF-SHA256, salt SHA-256("dchat-salt-v1"), info "dchat-message-encryption-v1", L = 32
AEAD	ChaCha20-Poly1305, 256-bit key, 96-bit nonce, 128-bit tag
Group encryption	Per-member 32-byte sender key, ChaCha20-Poly1305
Rotation, default	10,000 messages or 30 days
Rotation, conservative	5,000 messages or 14 days

Parameter	Value
Rotation, relaxed	20,000 messages or 60 days
Account rotating key period	7 days, 72-hour grace
Group member limit	100
Group roles	Member, Admin
Relay retention, regular	14 days
Relay retention, configuration	30 days
Relay per-user cap	10,000 messages
Relay cleanup sweep	Hourly
Peer replication retry	Every 60 seconds, for 30 minutes
Duplicate suppression window	45 minutes
Acknowledgement batching	250 milliseconds
Send retries	3
Reconnect backoff	3, 5, 10, 30, 60 seconds
Thumbnail cap	150 KB, inline
Image cap	50 MB
Avatar encryption	AES-256-GCM
Group call capacity	25 publishers
At-rest KDF, native	PBKDF2-HMAC-SHA256, 250,000 iterations
At-rest KDF, WebAssembly	PBKDF2-HMAC-SHA256, 10,000 iterations
At-rest blob layout	nonce(12) || tag(16) || ciphertext
Local database	SQLCipher, key in platform secure storage
Cosmos SDK	v0.50.13
CometBFT	v0.38.17, vendored fork
Base denomination	`udchat`, 6 decimals
Node minimum gas price	0.0001 udchat
Client gas tiers	0, 0.025, 0.04 udchat per gas
Gas simulation multiplier	1.4
Service node minimum stake	10,000,000 DCHAT
Service node heartbeat	300 seconds
Service node unbonding	7 days

Parameter	Value
Reward epoch	100 blocks
Downtime slashing	5 percent
Uptime quorum	10 verifiers, 7 must agree
Uptime probe interval	720 blocks
Client reward daily cap	100 DCHAT, 10 during the first 14 days
Client reward pool emission	0.0274 percent per day
Referral payout	5 DCHAT, maximum 50
Streak multiplier ceiling	1.50x at 25 days
Disappearing message options	Off, 24 hours, 7 days, 1 month, 3 months
Client platforms	Android API 24+, iOS, Mac Catalyst, Windows
Client languages	English, Arabic, Persian, Kurdish Sorani, Pashto

Appendix B. Document status

Version 1.0, published 31 July 2026, describing the system as implemented at that date. Context strings carry explicit version suffixes so a future revision can change derivation without breaking deployed clients. Material changes will be listed here with dates.

Vulnerability reports are read by a person and credited in the fix.