Secret: Architecture Overview

Secure, decentralized messaging with built-in crypto and community-powered infrastructure.

Version: 1.0
Date: October 2025
Secret Foundation
https://secretapp.io

Table of Contents

---

1. Purpose and Scope

This document explains how Secret works end to end at the architectural level. It covers the system model (actors and trust assumptions), identity and onboarding with mnemonic-based accounts and multi-device lifecycle, cryptographic foundations (MLS for communities and groups; X3DH + Double Ratchet for direct messages; Sphinx routing), the data model (identity ledger, community/group ledger, content manifests), messaging flows, networking and transport, storage and delivery, decentralized/zero-trust infrastructure, role responsibilities, and privacy guarantees.

The intended readers are independent security reviewers, prospective host operators, researchers, and technical readers evaluating Secret's design and privacy properties.

This is an architecture-level description: it explains behaviors and guarantees while intentionally abstracting away byte-level message formats, API signatures, database schemas, and UI details.

2. Core Principles

The design of Secret is guided by the following principles:

- **End-to-End Encryption.** All content is encrypted on the sender's device and decrypted on the recipient's device. Infrastructure never sees plaintext.

- **Metadata Minimization.** Routing information is concealed using onion-style relay routing. Hosts and relays learn as little as possible about participants, paths, and timing.

- **Anonymous Onboarding.** Accounts are created without PII; a mnemonic is the primary credential for sign-up and sign-in.

- **User Sovereignty.** Users own identities and keys; storage and replication targets are user-selectable and portable.

- **Zero-Trust Infrastructure.** Servers are untrusted; integrity and authenticity follow from client-side verification of signed, append-only state. Clients verify identity and community state (ledgers, device sets, prekeys, group epochs) and enforce authorization locally; servers never enforce plaintext ACLs.

- **Multi-Device by Design.** Devices are explicitly enrolled and revocable; state synchronization remains end to end.

- **Transport Agnosticism.** Operates over multiple transports (e.g., TCP, QUIC, WebSockets, WebRTC) with secure peer connections, stream multiplexing, multi-addressing, NAT traversal, and circuit relays. Transport choice does not affect end-to-end security or message semantics.

- **Resilience.** Works under poor connectivity; supports offline composition, delayed delivery, and opportunistic synchronization.

- **Open Participation.** Anyone can run Content and Relay Hosts; Authority Hosts are decentralized and accountable.

- **No Data Collection.** Hosts store and forward only opaque ciphertext and minimal operational signals; user data is not collected or monetized.

3. System Model

3.1. Actors

- **Client.** Root of trust. Holds mnemonic-derived keys; encrypts/decrypts and signs; verifies identity and community state; establishes DM and group sessions; manages devices, backups, and offline/interval sync. Clients can also enable lightweight hosting or relaying when desired.

- **Authority Host.** Distributes decentralized identity state (public keys, identity timelines) and device prekeys; clients never trust responses blindly and instead verify signed identity chains and cross-check across authorities. Operators stake to participate and can be slashed for misbehavior; authorities replicate among themselves for availability.

- **Content Host.** Stores and serves only ciphertext—messages, ledger segments, and media chunks—addressed by opaque identifiers; acts as delivery/availability layer when clients are offline. Hosts are untrusted and verified client-side; both verified (staked, slashable, listed) and unverified (anyone can run) modes exist; hosting can be enabled directly from the client or deployed standalone.

- **Relay Host.** Forwards fixed-size, onion-encapsulated packets along single- or multi-hop paths. Each relay learns only its immediate predecessor and next hop, peels its own layer of encryption, and gains no information about endpoints, path length, or payload. Relays can be enabled in-client or deployed standalone, and are designed to remain lightweight and content-agnostic.

- **Foundation Host.** Provides non-sensitive public services: push-notification fan-out, search/indexing for explicitly public data, and the marketplaces for visual assets and dApps; also receives abuse reports/feedback. It has no access to plaintext and is optional for the protocol to function.

3.2. Trust and Threat Assumptions

- **Trust boundary.** Clients are the root of trust. Authority, Content, Relay, and Foundation Hosts are untrusted for confidentiality and integrity.

- **Adversaries.** Global passive observer; active network attacker (drop/modify/replay); malicious infrastructure; targeted attacker; compromised endpoint (plaintext on that device is out of scope for E2E).

- **Confidentiality.** End-to-end encryption for DMs (Double Ratchet) and groups (MLS). Hosts/relays never see plaintext; media is chunk-encrypted.

- **Integrity and authenticity.** Identity events, device sets, prekeys, and MLS commits/epochs are signed and verified on clients. Clients cross-check multiple authorities and fail closed on inconsistencies.

- **Metadata privacy.** Onion-style relay routing aims to prevent linking sender and recipient; hosts handle opaque identifiers only.

- **Availability.** Multiple hosts and relay paths; local queueing and delayed delivery during outages; replication is client-driven.

- **Assumptions.** Secure on-device keystore and quality randomness; loosely synchronized clocks; networks may be lossy or censored.

- **Failure policy.** On verification failure or equivocation suspicion, halt sending in the affected scope, continue safe reads where possible, and prompt remediation (re-fetch, session reset, or device revocation).

3.3. Communication Topology

- **Client → Relay → Host.** Clients communicate with Authority and Content Hosts directly or via one or more Relay Hosts. Onion-style encapsulation prevents relays and hosts from linking sender to recipient.

- **Multi-transport substrate.** The system is transport agnostic. Transport choice does not change end-to-end security or message semantics.

- **Relay chaining.** Single-hop or multi-hop paths are supported. Each relay sees only its adjacent hops; paths can be rotated to reduce correlation.

- **Host access.** Authority Hosts serve identity heads and prekeys; Content Hosts store ciphertext messages, ledger fragments, and media chunks addressed by opaque identifiers.

- **Push notifications.** Foundation Hosts deliver metadata-minimal push notifications containing pull hints only. Clients fetch ciphertext after interaction.

- **Peer opportunities.** When available, clients may sync directly with peers while preserving the same end-to-end guarantees.

- **Replication.** Clients can use multiple Authority and Content Hosts concurrently for availability and censorship resistance. Consistency is enforced client-side by verification across views.

- **Offline and delay-tolerant operation.** When no route is available, clients queue locally and deliver once any viable path reappears.

4. Identity and Onboarding

4.1. Anonymous Account Creation

- **No PII.** Accounts are created without phone numbers, emails, or any other PII. Creation works fully offline until the first sync.

- **Mnemonic as credential.** The client generates a high-entropy mnemonic locally; it is the user's primary credential for future sign-in and recovery.

- **Deterministic identity.** From the mnemonic, the client derives an identity binding (used only to anchor the identity) and the initial long-term keys needed to operate.

- **Initial device.** The enrolling device becomes the first active device for the identity and records a signed identity-creation event that peers can later verify.

- **Optional display data.** Any user-facing display elements (e.g., avatar or label) are optional and can be set or changed later without affecting cryptographic identity.

- **No server secrets.** No secrets or recovery material are uploaded to infrastructure. Hosts only see public, signed state after the first sync.

4.2. Recovery and Account Portability

- **Single source of recovery.** The mnemonic is the only recovery mechanism. No server-stored recovery data or custodial keys exist.

- **Sign-in on new devices.** Entering the mnemonic on a new device deterministically re-derives the same identity and device-capable keys; the device then enrolls via a signed identity event.

- **Portability.** The same mnemonic can be used with external wallets. Messaging and wallet key trees remain isolated and non-interchangeable.

- **User guidance.** Clients encourage secure storage of the mnemonic and warn that Secret cannot recover it on the user's behalf.

- **Offline-friendly.** Account import works offline; the device synchronizes public state once a network path is available.

4.3. Key Derivation and Domain Separation

- **Root derivation.** The mnemonic is processed with a memory-hard key derivation step to produce a Master Key. All long-term keys are derived from this Master Key on the client.

- **Isolated trees.** The Master Key deterministically derives independent key trees for identity binding, messaging (signing/encryption), and wallet use. Trees use distinct, application-scoped labels/paths and are non-interchangeable.

- **Identity binding.** A dedicated derivation produces an identity-binding key; its hash forms the Identity ID used only to bind the identity to the mnemonic.

- **Operational separation.** Wallet keys (if used) are derived in a separate tree. Using the same mnemonic in external wallets does not expose messaging keys.

- **No server involvement.** All derivations occur locally; no seed material or derivation inputs are uploaded.

4.4. Device Lifecycle and Multi-Device

- **Enrollment.** Adding a device creates a signed identity event, binding the new device key to the identity. Bootstrap can use QR transfer between existing and new devices or re-derivation with the mnemonic.

- **Authorization to enroll.** Existing devices must explicitly approve new-device enrollment, or the user must prove control of the mnemonic. Hosts do not grant enrollment permissions.

- **Consistency.** All devices compute the same identity head and device set from the verified identity ledger. If any device observes divergence (missing events, stale heads), clients pause sensitive actions until re-synced.

- **Revocation.** Removing or suspending a device emits a signed identity event. Direct-message sessions involving that device are reset.

- **Recovery paths.** A lost or compromised device is removed via a signed identity event from another device or by re-deriving keys with the mnemonic on a replacement device, then rejoining state.

- **Minimal disclosure.** Device labels and metadata aren't exposed and do not affect cryptographic identity. Only the presence of valid device keys matters for authorization decisions.

4.5. Local Storage, Backup, and OS Keystore Protection

- **Key custody.** Private keys and other sensitive material are sealed by the platform's secure keystore where available. The client never writes unsealed key material to disk.

- **Encrypted state.** All local data (messages, media chunks, manifests) is stored inside an encrypted database; only verified public state (e.g., identity/community ledger snapshots) may be kept outside. Clearing caches does not affect recoverability.

- **Backup export.** Users may create encrypted backups of client state (e.g., account data, local settings, recent message indexes). Backups are produced and decrypted only on the client; no server copy exists.

- **Backup restore.** Restoring requires the mnemonic and the backup's decryption material. If the backup is unavailable, the mnemonic alone re-derives identity and

devices; history resynchronizes from peers/hosts as permitted.

- **Compromise response.** On suspected device compromise, users remove the device via a signed identity event and rotate sessions/epochs; local data on the compromised device is treated as exposed.

- **Minimal telemetry.** The client does not emit plaintext content or identifiers. Operational metrics, if any, are local-only or anonymized and optional.

5. Cryptographic Foundations

5.1. Group Messaging — Message Layer Security

Secret uses MLS (RFC 9420) for group messaging. This profile defines how clients maintain consistent, authenticated group state under untrusted infrastructure.

Goals

- Confidentiality, integrity, and authentication for groups up to N members.
- Forward secrecy and post-compromise security across membership changes.
- Efficient rekeying on joins/leaves (TreeKEM, O(log N)).
- Convergent, consistent group state across clients and devices.

Protocol Profile

- **Protocol:** MLS 1.0 (RFC 9420).
- **Credentials:** Basic credentials bound to member identity.
- **Extensions:** Ratchet Tree.
- **Lifetimes & rotation:** KeyPackages have a finite validity window; application secrets rotate per epoch/commit.

Group Lifecycle

1. **Creation.** The creator initializes a `GroupID` and issues **Welcome** messages; all members start at `epoch = 0`.

2. **Proposals.** Members emit **Add/Remove/Update** proposals; proposals are disseminated by the delivery service and may be batched.

3. **Commit.** A designated **committer** sends a **Commit** that advances the epoch; all members deterministically update trees and secrets.

Consistency and Authenticity

- **State authentication.** Each epoch change is signed by the committer; the **confirmation tag** authenticates the new state.
- **Tree validation.** Clients validate parent hashes and the full ratchet tree on every epoch.
- **Credential visibility.** Member credential updates (e.g., key rotation) are visible in the tree and audited locally.

Application Message Protection

- **AEAD & AAD.** Messages are protected with the suite's AEAD; associated data binds `{GroupID, epoch, contentType}`.
- **Ordering & replay.** MLS sequence numbers support out-of-order delivery; replays are rejected via the Secret Tree ratchet.
- **Padding.** Traffic shaping/padding is handled at the routing layer.

Membership Changes & Security Properties

- **Forward secrecy.** Prior epochs become unreachable after an Update/Commit.
- **Post-compromise security.** A compromised sender that updates their leaf regains secrecy in the next epoch.
- **Add/Remove correctness.** Removed members cannot compute future epochs; new members cannot read past epochs.
- **Untrusted servers.** Servers may drop/reorder but cannot forge commits or read content.

State and Hygiene

- **Per-group state.** Ratchet tree, transcript hash, sender-data secrets, Secret Tree, and the current epoch's key schedule.
- **Key erasure.** Old epoch material is erased after the replay window; only long-term credentials and current MLS state persist.

Failure Handling

- **Malicious members.** Inconsistent commits are rejected by tree validation; signed artifacts allow attribution.

- **Flood control.** Rate-limit proposals; enforce one active committer per epoch or a defined queueing policy.

Security Summary

By fixing MLS 1.0 behavior, validating the tree each epoch, and strictly handling membership changes, Secret achieves group confidentiality, integrity, forward secrecy, post-compromise security, and convergent state—even with untrusted servers.

5.2. Direct Messaging — X3DH and Double Ratchet

Secret uses an X3DH-style asynchronous setup followed by the Double Ratchet for ongoing 1:1 messaging.

Goals

- Confidentiality and integrity of messages and attachments
- Authentication of long-term identity
- Forward secrecy and post-compromise security
- Asynchronous delivery (no simultaneous online requirement)
- Correct multi-device behavior (safe add/remove, per-device sessions)

Protocol Profile and Scope

- **Setup + transport.** X3DH for initial authenticated key agreement; Double Ratchet thereafter.

- **Metadata privacy.** Transport-layer hardening (Sphinx) is handled separately; this section describes the content layer.

Deviation from stock X3DH (OPKs disabled)

We do not employ One-Time Pre-Keys (OPKs). In a decentralized, untrusted directory there is no reliable, globally authoritative mechanism to atomically mark OPKs as “consumed.” When recipients are offline, replicas cannot be updated consistently, so the “one-time” property is unenforceable. Consequently, our initiation uses the 3DH variant with IK + SPK only.

Security impact. The absence of OPKs removes the additional one-time secrecy at session start; authenticity and confidentiality are still provided by X25519 (3DH) with Ed25519 authentication, and forward secrecy/post-compromise security are obtained immediately after the first Double-Ratchet step.

Mitigations. We compensate with (i) frequent SPK rotation, (ii) an explicit identity-binding record in the first application message, and (iii) prominent key-change UX (TOFU or verification). This preserves strong practical security under our decentralized trust model while avoiding brittle OPK semantics.

Roles and Keys

- **Identity key (IK).** Long-term signing identity; also used to bind the X3DH handshake to the recipient's identity.

- **Signed prekey (SPK).** Per-device key that is signed by the device's IK and periodically rotated.

- **Per-device independence.** Each recipient device maintains its own IK/SPK and ratchet state; the sender maintains one session per recipient device.

Session Establishment (X3DH)

- **Inputs.** Initiator fetches `{recipient IK, recipient SPK}` and generates an ephemeral key.

- **Key schedule.** The handshake mixes multiple DH values into a root key and initial chain keys.

- **Authentication binding.** The initiator cryptographically binds the session to the recipient's identity; the first application message includes an identity-binding record sufficient for the recipient to verify sender identity and device.

Message Protection (Double Ratchet)

- **AEAD + AAD binding.** Each message is AEAD-protected with associated data that binds the conversation context, sender device, ratchet position, coarse time, content type, and padding signal.

- **Header encryption.** Per-message headers are encrypted under keys derived from the chain to reduce traffic correlation.

- **Reordering and replay.** A bounded skipped-key window tolerates out-of-order delivery; duplicate nonces/positions are rejected per conversation.

- **Retransmission.** The last message can be explicitly re-sent upon request without weakening secrecy.

Multi-Device Semantics

- **Per-device fan-out.** The sender encrypts a distinct message copy for each recipient device's session.

- **Device add/remove.** Adding a device publishes fresh identity/prekey material; a new X3DH+DR session is established to that device. Removing a device updates the identity view, and clients stop sending to it.

Attachments

- **Separate content keys.** Each attachment uses a fresh content-encryption key; the message carries only a manifest with an authenticated locator and integrity hash.

- **Integrity on fetch.** Recipients verify the manifest hash on download before decryption.

Rotation and Lifetimes

- **Prekey rotation.** Signed prekeys rotate on a fixed schedule; initiators refuse stale or unsigned material.

- **Ratchet cadence.** Chain keys advance per message; old keys are erased after use.

Failure Handling

- **DoS surfaces.** Directory/delivery abuse is mitigated by rate limits at the delivery layer and bounded skipped-key windows per conversation.

- **Suspicious state.** On identity/prekey inconsistencies or device revocation, clients reset sessions and halt sends until the new state is verified.

Security Summary

X3DH provides asynchronous mutual authentication and a fresh shared secret at session start; the Double Ratchet delivers forward secrecy and post-compromise security thereafter. AEAD with contextual associated data binds each message to its conversation and device; per-device fan-out avoids shared sender keys; and identity verification detects active substitution—even with untrusted infrastructure.

5.3. Onion Routing — Sphinx Packet Format

Secret uses Sphinx-style onion routing to harden transport metadata. Content confidentiality remains at the message layer.

Goals

- **Route unlinkability.** Relays cannot link sender ↔ receiver; each sees only its adjacent hops.
- **Header privacy.** Path length, final destination, and per-hop secrets are hidden from relays.
- **Payload secrecy & integrity.** End-to-end protected; relays cannot read or alter content.
- **Replay resistance.** Duplicate packets are detected and dropped at relays and endpoints.
- **Traffic-analysis friction.** Fixed-size packets, padding, optional delays, and cover traffic.

Roles

- **Client (sender/receiver).** Constructs Sphinx packets; verifies end-to-end integrity.
- **Relay.** Stateless per-packet processing; peels one layer and forwards.

Relay and Path Selection

- **Relay set.** Chosen from publicly reachable hosts advertising relay capability.
- **Path length.** Policy allows 0–5 relays.

- **Diversity.** Avoid repeat relays on a single path.

- **Weighted random.** Prefer higher-reliability/bandwidth relays subject to diversity constraints.

- **Churn.** Rebuild paths on a fixed cadence (e.g., ~minute-scale) to reduce correlation.

Packet Classes and Size Policy

- **Two constant-size classes.** An “interactive” class (small) for chat/control and a “bulk” class (large) for file/media.

- **Why two.** Small packets keep latency low; large packets amortize onion overhead for bulk transfer.

- **Leakage bound.** At most a one-bit signal (“interactive vs bulk”). Within each class, size is constant on the wire.

- **Mitigations.** Cover traffic in both classes, per-circuit rate shaping, and a threshold policy that sends tiny files over the small class.

Cryptographic Operations (per packet)

- **Per-hop header.** Derived from a sender–relay secret; carries next-hop info and a per-hop MAC over that hop’s header segment, plus encrypted control bits (drop/forward/deliver/SURB).

- **End-to-end payload.** A single AEAD protects the inner payload; only endpoints hold the content key.

- **Replay defense.** Nonces/tags and relay-local caches detect duplicates without retaining long-lived identifiers.

Reply Paths (Single-Use Reply Blocks)

- **Purpose.** Allow replies without revealing the receiver’s identity or path.

- **Creation.** The receiver precomputes reply tokens bound to its chosen return path and shares them inside an end-to-end message.

- **Use.** The sender attaches a token; the exit relay learns only the final hop of the return path.

Mixing and Timing

- **Delays.** Relays add small randomized delays (e.g., Poisson) to blur timing.
- **Cover traffic.** Clients (and optionally relays) inject dummies at a low, randomized rate.
- **Batching.** Relays flush in small batches to reduce fine-grained timing leakage.

Link-Layer Protection (between hops)

- **Encrypted channels.** Neighboring hops use an authenticated, encrypted session to prevent on-path observers from fingerprinting hop-to-hop headers.
- **Separation of concerns.** This protects the link only; onion layers and end-to-end payload security remain independent.

Failure Handling

- **Relay behavior.** On invalid MAC, unreachable next hop, or malformed header, drop silently.
- **Client behavior.** On timeout or loss, rebuild a fresh path and retransmit with new ephemerals; temporarily down-weight failing relays.

DoS and Abuse Controls

- **Relay-side.** Constant-time parsing, cheap MAC checks before expensive work, connection-level rate limits.
- **Client-side.** Cap outstanding packets per path; exponential backoff on loss.

Logging and Telemetry

- **Minimal by design.** No per-packet logs or stable identifiers; only coarse aggregates (uptime, bytes, queue depth) with short retention.

Security Summary

Layered headers and per-hop secrets ensure each relay sees only adjacent hops; the end-to-end payload remains opaque. Fixed packet sizes, path churn, randomized delays, and cover traffic raise the cost of traffic analysis. Even with some malicious relays, endpoints remain hidden unless an adversary controls both the first and last hop, and content confidentiality is preserved.

5.4. Verification and Key Transparency

How clients decide what keys and group states to trust—without trusting infrastructure.

- **Ledger-anchored identity.** Each identity is an append-only, signed timeline (key introductions/rotations; device add/remove). Clients reconstruct and verify this timeline locally to derive the effective device set.

- **Cross-authority checks.** Identity heads and snapshots fetched from multiple Authority Hosts are compared; any divergence (stale, forged, or equivocal views) causes clients to halt outgoing sends for affected peers until resolved.

- **Prekey provenance.** A device's signed prekey is accepted only if it chains to the verified device set derived from the identity timeline. Stale or unsigned material is rejected.

- **Group state alignment.** For communities/groups, clients verify MLS commits/transcripts and require that the resulting membership exactly matches the community/group ledger before sending under a new epoch.

- **Device consistency.** Messages must validate to a sender in the verified device set; messages from unknown or revoked devices are rejected and surfaced.

- **Audit and caching.** Clients cache verified ledger segments and transcript digests to prevent rollback/downgrade by servers and to accelerate re-verification.

- **Fail-closed policy.** On any verification failure or suspicion of equivocation, clients stop sending for the affected peer/group, continue safe reads where possible, and require a safe transition (session reset or epoch confirmation) before resuming.

5.5. Prekey Management and Publication

How devices advertise asynchronous key material for first contact and how clients consume it safely.

- **Directory model.** Each device publishes a signed prekey to Authority Host. Directories expose these bundles without PII. Authority Host do not transform or enforce policy; clients verify provenance.

- **No one-time prekeys.** Secret does not rely on server-managed one-time prekeys. Each device maintains a single signed prekey that rotates on a schedule. This avoids stale, “consumed-but-not-removed” entries in decentralized, untrusted infrastructure.

- **Rotation policy.** Clients rotate their signed prekey at a fixed cadence (implementation constant) or on explicit triggers (device compromise, key aging). Old prekeys remain fetchable only for a short grace window to complete in-flight handshakes, then expire.

- **Publication workflow.**

    - Device generates a fresh prekey and signs it with its current device identity key.
    - Device submits the bundle to Authority Host.
    - The identity timeline references the active prekey so verifiers can chain it to the device set.

- **Client consumption.** Initiators fetch `{recipient identity key, signed prekey}` and refuse material that is stale, unsigned, or not linked to the verified device set. If multiple prekeys are present, initiators prefer the newest valid one.

- **Exhaustion and fallback.** Because Secret does not use one-time prekeys, “exhaustion” does not occur. If the directory is unreachable, initiators defer or attempt peer-assisted retrieval once available; no plaintext fallback exists.

- **Privacy and rate control.**

    - Directories expose only public keys and freshness metadata; requests are content-agnostic and cacheable.
    - Authority Hosts apply anonymous, coarse rate limits and short retention of operational metrics.
    - Clients randomize fetch timing and may prefetch recent contacts to reduce correlation.

- **Revocation and compromise.** On suspected device compromise, the user rotates device keys (identity event) and publishes a new signed prekey. Peers reject the old prekey once the updated device set is observed.

- **Interoperability note.** If multiple authorities are used, clients fetch from several and cross-check results; mismatches cause a fail-closed posture until a consistent view is established.

5.6. Authentication and Deniability Model

What recipients can authenticate, what third parties can prove, and how Secret avoids creating publicly verifiable transcripts.

- **Recipient authentication.**
    - **Direct messages:** Recipients authenticate that a message was produced by a sender device in the verified device set for the peer's identity under the current Double Ratchet state.
    - **Groups:** Recipients authenticate that a message was sent by a current group member under the active MLS epoch and matches the validated transcript.

- **Third-party verifiability.**
    - Message ciphertexts and per-message MACs are not globally verifiable signatures; they are keyed and context-bound.
    - A recipient cannot, without revealing session secrets, produce a transferable proof that convinces an uninvolved third party that a specific sender authored a specific plaintext.

- **Deniability properties.**
    - **DMs:** The Double Ratchet's symmetric message authentication provides recipient assurance without creating a non-repudiable signature on plaintexts.
    - **Groups:** MLS application messages are AEAD-protected; commit operations are signed for group state correctness, not to produce public, message-level non-repudiation.

- **What is verifiable to third parties.**
    - **Identity and membership state:** Signed identity-ledger events and MLS commits/confirmations are publicly verifiable artifacts that prove “who was in the device set or group when,” not “who said what.”
    - **Operator behavior:** Misbehavior by Authority Hosts (e.g., equivocation) is provable via conflicting signed heads/checkpoints.

- **Binding and replay semantics.**
    - Messages are bound to their context (conversation/group, epoch/session, sender device) via associated data.
    - Replays and out-of-context ciphertexts are rejected and cannot be reused to create false attributions.

- **Compromise considerations.**
    - If a recipient voluntarily discloses session keys or plaintexts, they can create after-the-fact “evidence,” but this relies on secret disclosure and does not arise from protocol-level non-repudiation.
    - Device compromise prior to rotation can allow an attacker to forge messages within that window; forward secrecy and post-compromise recovery limit the scope.

5.7. Parameters, Suites, and Key Material Handling

Scope and Provider

- **Provider.** Libsodium.

- **Algorithms in use.** XChaCha20-Poly1305 (AEAD), Ed25519 (signatures), X25519 (key agreement), HKDF-SHA-256 (KDF/PRF), BLAKE2b and SHA-256 (hash), Argon2id (mnemonic → master key), AES-Key-Wrap (CEK/DEK wrapping only).

Algorithm Suites (registry)

- **Groups (MLS, RFC 9420).** MLS_128_DHKEMX25519_CHACHA20POLY1305_SHA256_Ed25519.

- **Direct Messages (X3DH + Double Ratchet).**
  DH: X25519 · AEAD: XChaCha20-Poly1305 · KDF/PRF: HKDF-SHA-256 · Signatures: Ed25519 · Hash: SHA-256.

- **Onion Transport (Sphinx).**
  Per-hop stream: XChaCha20 · Header MAC: keyed-BLAKE2b · End-to-end payload AEAD: XChaCha20-Poly1305.

Primitive Inventory (where each applies)

- **X25519:** X3DH handshake tuples; Sphinx per-hop secrets.

- **Ed25519:** Identity/device credentials; MLS commitments per RFC 9420; signed prekeys.

- **XChaCha20-Poly1305:** DM application messages and encrypted headers; MLS application payloads; attachments; Sphinx end-to-end payload.

- **XChaCha20 (stream):** Sphinx per-hop layer encryption.

- **BLAKE2b / SHA-256:** Header MACs, transcript/material hashing, identifiers.

- **HKDF-SHA-256:** Handshake key schedules; chain/header keys; envelope keys.

- **Argon2id:** Transform mnemonic into Master Key for local derivations.

- **AES-KW:** Optional wrapping of content keys (packaging/export only).

- **RNG:** OS CSPRNG via Libsodium for all randomness.

Key Generation and Identity Binding

- **On-device generation.** All long-term and ephemeral key pairs are generated locally.

- **Mnemonic → Master Key.** A memory-hard KDF (Argon2id) derives a Master Key from the mnemonic.

- **Domain separation.** Independent derivation trees for identity binding, messaging (signing/encryption), and optional wallet use; trees use distinct labels/paths and are non-interchangeable.

- **Identity ID.** Computed as a hash of a dedicated identity-binding derivation.

Lifetimes and Rotation

- **MLS KeyPackages:** Validity window is finite (implementation constant); refresh before expiry.
- **MLS application secrets:** Rotate on every Commit/epoch advance.
- **Signed prekeys (DM):** Rotate on a fixed schedule and immediately on compromise.
- **Double Ratchet:** Chain keys advance per message; used message keys are erased after use.
- **Replay windows:** Bounded skipped-key cache for DMs; MLS accepts within the active epoch's replay window only.

Associated Data (binding rules)

- **Direct messages:** `{conversationID, senderDeviceID, ratchetCounter, coarseTimestamp, contentType, paddingInfo}`.
- **Groups (MLS application):** `{GroupID, epoch, contentType}`.
- **Requirement:** Ciphertexts are rejected if associated data does not match the verified context.

Nonce and Counter Rules

- **DMs:** Nonces derived from ratchet position (per-direction counters); no reuse within a conversation.
- **MLS:** Per-message sequence numbers with transcript binding; no reuse within an epoch.
- **Cross-domain hygiene:** Keys/nonces/labels are never reused across DM, MLS, and Sphinx domains.

Storage, Backup, and Zeroization (cryptographic aspects)

- **At rest:** Long-term keys in sealed/secure storage; MLS group state in encrypted app storage; no plaintext user data on servers.

- **Backups:** Optional, end-to-end encrypted; mnemonic material is transformed via Argon2id for backup master data.

- **In memory:** Session/chain keys are zeroized immediately after use; sensitive buffers are cleared.

Sizes, Padding, and Limits (cryptographic policy)

- **Transport packets:** Fixed-size classes for Sphinx; all transport messages are padded to their class size.

- **Attachments:** Chunked; each chunk AEAD-protected; per-chunk integrity hashes validated on download.

- **Operational bounds:** Implementation constants set maximum group size, message-to-attachment threshold, and bounded per-sender windows.

Distribution, Transparency, and Revocation

- **Public material distribution:** Via untrusted directories or introduction channels; authenticity is end-to-end.

- **Revocation:** Signed identity events revoke devices/keys; recipients must retire affected sessions and adopt a fresh state before further sends.

Compromise and Recovery (crypto response)

- **Device compromise:** Revoke device keys; rotate signed prekeys; reset Double Ratchet; advance MLS epoch.

- **Identity compromise:** Re-provision identity; require re-verification; re-establish DM and MLS states.

- **Infrastructure compromise:** Content remains E2E protected; rotate relay paths and publish operator notice if applicable.

Implementation Safety

- **Constant-time.** Use constant-time primitives for MAC/AEAD and key comparisons.
- **Side-channel hygiene.** Avoid secret-dependent branches and timing; rely on vetted library implementations.
- **RNG.** All randomness originates from the OS CSPRNG; failure of this assumption cannot be remediated by infrastructure.

6. Data Model

6.1. Identity Ledger

A per-identity, append-only timeline of signed events that defines who the user is and which devices represent them.

- **Purpose.** Bind a mnemonic-derived identity to a stable identifier; enumerate active devices and their public keys; record rotations and revocations.
- **Event types.** Identity creation; device add/remove; key rotation; prekey publication; policy hints like retention preferences.
- **Verification.** Each event is signed by authorized keys from the prior state; clients reconstruct the chain locally and compute the current “effective” device set.
- **Anti-equivocation.** Clients fetch heads from multiple Authority Hosts and compare digests. Divergence results in a fail-closed posture for sends to that identity.
- **Privacy.** Events carry only public material and opaque references; no PII or plaintext content.
- **Durability.** The ledger is replicated by Authority Hosts; clients cache verified segments to prevent rollback.

6.2. Community and Group Ledger

An append-only record that captures community definitions, group creation, and membership/role changes that must align with MLS state.

- **Purpose.** Provide a public, auditable view of “who is in which group and with what role,” independent of transport and storage.

- **Scope.** Community metadata (opaque identifiers, role definitions), group creation, membership add/remove/ban, and role updates; optional governance actions (e.g., admin delegation).

- **Coupling to MLS.** Membership-changing events correspond to MLS operations. Clients do not send under a new epoch until the ledger change and the MLS commit both verify.

- **Authorization surface.** Clients derive authorization decisions from the latest verified ledger view (e.g., who may post, invite, or moderate). Servers do not enforce plaintext ACLs.

- **Privacy.** Names/titles need not be present in plaintext; identifiers are opaque. Ledger entries reveal only what is necessary to verify membership state.

- **Integrity & availability.** Events are signed by authorized members; Content Hosts replicate and serve heads; clients detect forks by digest comparison.

6.3. Content Manifests

Opaque, signed descriptors that let clients upload, locate, and verify encrypted payloads (messages, attachments, media chunks) without revealing content to hosts.

- **Purpose.** Decouple large or binary content from messages while preserving end-to-end integrity and confidentiality.

- **Use in messages.** Messages carry only the manifest and per-item content keys; hosts see ciphertext blobs addressed by opaque IDs.

- **Verification.** On download, clients validate hash before decryption; mismatches are discarded.

- **Chunking and retrying.** Large files are chunk-encrypted; clients fetch/retry chunks independently, enabling resilient, partial reassembly over lossy links.

7. Messaging Flows

7.1. Direct Messaging

Send (sender device → recipient devices):

1. **Resolve state.** Verify recipient identity head and device set; fetch active signed prekeys.
2. **Session ensure.** For each recipient device, ensure a session exists; if not, perform asynchronous setup and derive initial chain keys.
3. **Encrypt.** Construct ciphertext with AEAD and context AAD; encrypt per-device copy under that device's session.
4. **Route.** Wrap payload(s) for transport (onion packet) and enqueue to Content/Relay Hosts.
5. **Record.** Store the encrypted send locally and advance ratchet counters.

Receive (recipient device):

1. **Fetch.** Wake on push or poll; download ciphertext envelopes addressed to this device.
2. **Verify and decrypt.** Check context, counters, and authenticity; decrypt; apply skipped-key window if out-of-order.
3. **Acknowledgement.** Optionally emit an encrypted lightweight ack to aid peer retransmit logic.
4. **Persist.** Update local state; derive next message keys; zeroize used keys.

Multi-device semantics. Sender fan-outs one encrypted copy per recipient device. On device add/remove, the sender stops to removed devices and starts sessions to added devices.

Failure handling. If prekey/session inconsistencies are detected, reset the session before further sends. On delivery timeout, rebuild route and retransmit with fresh ephemerals.

7.2. Group Messaging

Send (member device → group):

1. **Resolve state.** Ensure local view of membership matches the latest verified group/community ledger; confirm current epoch.

2. **Encrypt.** Produce an application message bound to `{GroupID, epoch}`; protect with the current epoch's application secret.

3. **Route.** Submit ciphertext to Content/Relay Hosts; transport padding/paths follow routing policy.

4. **Record.** Update local transcript hash and per-sender state.

Receive (member device):

1. **Fetch.** Wake or poll; retrieve group ciphertexts for the current epoch.

2. **Verify and decrypt.** Validate transcript sequence and confirmation material; decrypt application payloads; reject replays.

3. **Persist.** Update local transcript and sender state.

Membership changes. Add/Remove/Update proposals are disseminated; a Commit advances the epoch. Devices do not send under a new epoch until both the ledger change and the MLS commit verify. Removed members cannot compute future epochs; new members cannot read past epochs.

Failure handling. Inconsistent trees or commits are rejected; clients halt sends until a coherent state is restored. Flooding is mitigated via proposal rate-limits and single-committer or queued-commit policy.

7.3. Media Upload and Retrieval

Upload (sender device):

1. **Prepare.** Split large payloads into chunks; generate fresh content keys; compute integrity hash.

2. **Encrypt.** Encrypt each chunk with its content key; create a signed manifest containing opaque locator and integrity material.

3. **Store.** Upload encrypted chunks to selected Content Host.

4. **Reference.** Send a message that carries only the manifest and per-item content keys (encrypted to recipients via DM/MLS).

Download (recipient device):

1. **Resolve.** Decrypt the manifest from the message; obtain an opaque locator.

2. **Fetch.** Download required chunks; retry independently on failure; resume partial downloads.

3. **Verify and decrypt.** Decrypt with content keys; assemble, validate hash;

Classes and routing. Interactive messages use the small, fixed-size packet class; bulk media uses the large class. Paths rotate on a cadence; relays add timing noise and may inject cover traffic.

Failure handling. If a Content Host is unreachable or data fails integrity checks, surface an error without exposing plaintext.

8. Networking and Transport

8.1. Transport Agnostic

- **Abstraction.** Operates over multiple transports (e.g., TCP, QUIC, WebSockets, WebRTC) without altering message semantics or security guarantees.

- **Secure channels.** Peer links use authenticated, encrypted sessions to protect hop-to-hop metadata from on-path observers.

- **Multiplexing.** Multiple logical streams share a single connection to reduce head-of-line blocking and handshake overhead.

- **Addressing & NAT traversal.** Multi-address reachability is supported; relays can proxy for clients behind NATs/firewalls.

- **Policy-driven choice.** Transport selection adapts to availability, latency, and censorship conditions without impacting end-to-end confidentiality or integrity.

8.2. Relay Routing

- **Relay role.** Relays forward onion-wrapped packets. Each sees only its adjacent hops and cannot decrypt payloads.

- **Path construction.** Clients choose 0–5 relays per path, avoid reuse on a single route, and periodically rebuild paths to reduce correlation.

- **Relay discovery.** Relays are advertised by publicly reachable hosts with relay capability; clients maintain a weighted set based on availability and performance.

- **Classes & shaping.** Interactive traffic uses a small fixed-size packet class; bulk uses a larger class. Relays may add small randomized delays and cover traffic.

- **Failure handling.** On timeout or drop, clients down-weight failing relays, rebuild a fresh path, and retransmit with new ephemerals.

- **Minimal observability.** Relays keep only coarse, short-lived aggregates (uptime/bytes/queue depth); no per-packet identifiers or plaintext.

8.3. Push Notifications

- **Wakeups, not content.** Push carries only metadata-minimal wakeups (e.g., “new items available” with a pull hint). Clients then fetch ciphertext over normal channels.

- **Separation.** Delivered by independent services and isolated from identity, keys, and content.

- **Privacy.** No participant identities, message headers, or counts appear in push; tokens are unlinkable across providers and can be rotated.

- **Resilience.** Clients degrade gracefully without push (polling/backoff). Wakeups are idempotent and safe to coalesce or drop.

- **Abuse controls.** Rate-limited fan-out and per-device backoff prevent notification storms; failures never cause plaintext exposure.

9. Storage and Delivery

9.1. Content and File Hosting

- **Opaque storage.** Hosts store ciphertext messages, ledger fragments, and encrypted media addressed by opaque identifiers. No plaintext indexing or inspection occurs.

- **Manifests.** Messages reference media via signed manifests that contain integrity information and opaque locators; hosts see only blobs and IDs.

- **Integrity on read.** Clients verify hashes after assembling chunks; mismatches are treated as corruption and discarded.

- **Resource controls.** Hosts apply size limits, rate shaping, and fairness policies uniformly without content awareness.

9.2. Delivery Semantics and Ordering

- **At-least-once delivery.** Messages may be delivered more than once; clients perform idempotent processing and deduplication using authenticated context.

- **Per-conversation ordering.**

    - **Direct messages:** Monotonic per-direction counters enforce order and freshness at the recipient.

    - **Groups:** Sequence numbers within each epoch provide ordered processing; out-of-order arrivals are tolerated within a bounded window.

- **No server reordering.** Hosts forward/carry opaque items; ordering is established and enforced only by clients.

- **Acks and retries.** Lightweight acknowledgments and bounded retransmission timers improve liveness without exposing content or metadata.

- **Size and class.** Small, fixed-size packets prioritize interactive latency; larger classes are used for media to amortize overhead.

9.3. Offline Operation and Synchronization

- **Queue-first design.** When disconnected, clients queue outbound items locally and attempt delivery when any path becomes available.

- **Catch-up on reconnect.** On wake or network change, clients fetch new ciphertext and reconcile state; integrity and replay checks prevent duplication or rollback.

- **Partial downloads.** Media is chunked; clients resume interrupted transfers and re-request missing chunks without exposing plaintext.

- **Device sync.** A user's devices exchange encrypted state to converge on the same view; servers never reconcile plaintext.

- **Degradation behavior.** Under sustained loss or censorship, clients expand backoff windows, rotate paths, and continue opportunistic sync without weakening security.

10. Decentralized and Zero-Trust Infrastructure

10.1. Distributed Nodes

- **Independence by default.** Authority, Content, and Relay Hosts are operated by independent parties; clients can use multiple of each concurrently.

- **Foundation-operated utility.** The Foundation Host is operated by Secret Foundation and provides non-sensitive utilities (e.g., metadata-minimal push wakeups, public catalogs). Its role is bounded and optional for core messaging.

- **Interchangeability.** Any Authority/Content/Relay Host can be replaced without impacting end-to-end security or user identity.

- **Redundancy.** Using several Authority/Content/Relay Hosts improves availability and censorship resistance; clients select and rotate based on reachability and performance.

- **Failure isolation.** Failure or misbehavior of any host—including the Foundation Host—does not expose plaintext; clients can migrate to alternatives where applicable.

10.2. Untrusted but Verifiable

- **Assumed untrusted.** All hosts, including the Foundation-operated one, are untrusted for confidentiality and integrity.

- **End-to-end assurances.** Only clients handle private keys and plaintext. Identity/device events, group membership, and commits are signed; messages are authenticated and encrypted.

- **Consistency checks.** Clients validate public state served by infrastructure and compare independent views where available; conflicts halt sensitive actions until resolved.

- **Opaque operations.** Storage and routing operate on opaque identifiers and ciphertext; hosts cannot interpret application semantics.

10.3. No Data Collection

- **Minimal surface.** Hosts retain only coarse operational aggregates (e.g., bytes, uptime, transient queue metrics). No participant IDs, message headers, or plaintext content are collected.

- **Short retention.** Aggregates are rotated frequently and are not linkable to individual users or conversations.

- **No profiling.** There is no indexing, classification, or monetization of user data. Storage keys and locators are opaque; routing packets are constant-size within their class.

- **User control.** Users choose Authority/Content/Relay Hosts and can revoke any at any time. Use of the Foundation Host is optional and never required for end-to-end confidentiality.

11. Role Responsibilities

11.1. Client

- Generates and holds mnemonic-derived keys; encrypts, decrypts, and signs.
- Builds and verifies identity and community/group state; enforces authorization locally.
- Establishes and maintains DM sessions and participates in group epochs.
- Packages content and manifests; selects storage/relay paths; handles retries and offline queueing.
- Manages device lifecycle (add, revoke, recover) and local backups.

11.2. Authority Host

- Serves public identity timelines (heads and segments) and device prekeys.
- Accepts published updates from clients; stores only public, signed material.
- Ensures availability and basic rate/fairness controls; does not mutate client state.
- Never accesses plaintext, private keys, or per-user identifiers beyond public keys/IDs.

11.3. Content Host

- Stores ciphertext messages, ledger fragments, and chunked media by opaque identifiers.
- Provides read/write endpoints with integrity-preserving semantics; supports replication and retention policies.
- Applies uniform size/rate limits and resource fairness without content awareness.
- Never accesses plaintext or enforces application-level access rules.

11.4. Relay Host

- Forwards onion-wrapped packets along configured paths; can operate single- or multi-hop.
- Implements lightweight timing noise, batching, and optional cover traffic.
- Enforces anonymous rate limits and drops malformed traffic without side effects.
- Sees only adjacent hops and packet class; never sees plaintext or stable user identifiers.

11.5. Foundation Host

- Run by Secret Foundation. Provides non-sensitive utility services that are optional for core messaging.
- Delivers metadata-minimal wakeups that contain only pull hints; no message content, headers, or participant identifiers.
- Serves signed, cacheable catalogs of public assets (e.g., stickers and other media) and other publicly verifiable resources. Artifacts are tamper-evident and do not require login or PII.
- Does not relay traffic, store ciphertext messages, or handle identity/device state. Never processes plaintext or cryptographic material.
- Retains only coarse, short-lived operational aggregates (e.g., uptime, bytes served). No per-user logs, stable identifiers, or profiling.
- Designed for high uptime and broad reach, but the system functions without it; clients can ignore or replace these utilities without impacting end-to-end security.

12. Privacy Guarantees

12.1. Security Properties

- **Content confidentiality.** Messages and attachments are end-to-end encrypted; infrastructure never sees plaintext.

- **Content integrity and authenticity.** Recipients verify that ciphertexts were produced by authorized devices or group members; tampering is detected and rejected.

- **Forward secrecy and post-compromise security.** Past messages remain protected after key compromise; security is re-established after device rotation or epoch advances.

- **Authorization at the edge.** Clients enforce who may send, read, invite, or moderate; servers cannot elevate privileges by altering stored data.

- **Consistency under untrusted infrastructure.** Identities, device sets, and group state are verified on clients; forged or equivocal views do not lead to unsafe sends.

- **Optional deniability.** Message authentication does not create publicly verifiable signatures on plaintext; recipients are convinced, third parties are not.

12.2. Metadata Privacy

- **Routing unlinkability.** Onion-style relay routing prevents a relay from linking sender to recipient; each relay sees only adjacent hops.

- **Opaque storage.** Hosts store ciphertext addressed by opaque identifiers; no participant lists, titles, or plaintext attributes are exposed.

- **Limited size signals.** Transport uses fixed-size packet classes; within a class, messages are indistinguishable by size.

- **Timing friction.** Path churn, small randomized delays, batching, and optional cover traffic reduce timing correlation.

- **Minimal operational data.** Hosts retain only coarse, short-lived aggregates (e.g., uptime, bytes, queue depth), not per-user or per-conversation logs.

12.3. Residual Risks and User Guidance

- **Endpoint compromise.** If a device is compromised, its local plaintext and keys are exposed. Users should revoke the device promptly and migrate to a new one.

- **Traffic analysis limits.** A powerful network observer may still extract patterns from timing and packet class usage, especially if controlling both first and last hops. Vary paths and enable cover features where available.

- **Contact confirmation gaps.** Without out-of-band verification, active substitution of identity material is harder but not impossible if a user accepts an attacker's keys. Prefer recognized contacts and verify key changes when surfaced.

- **Side-channel environment.** Weak randomness, insecure local storage, or compromised OS keystores can undermine guarantees; use modern platforms and keep devices updated.

- **Loss of mnemonic.** The mnemonic is the sole recovery mechanism; losing it leads to permanent account loss. Store it securely and offline.

- **Human factors.** Screenshots, cloud backups outside the app, or forwarding decrypted content fall outside the model and can leak data. Limit secondary copies and use encrypted backups only.

13. Conclusion

Secret's architecture provides end-to-end confidentiality, integrity, and strong metadata privacy while treating all infrastructure as untrusted. Identity and onboarding are anonymous and mnemonic-based, with domain-separated keys and explicit multi-device lifecycle. Direct messages use an asynchronous X3DH setup followed by the Double Ratchet; groups use MLS with efficient rekeying and robust consistency checks. Routing privacy is enforced with Sphinx-style onion packets and fixed-size transport classes. Content and media remain opaque to hosts, referenced by signed manifests and verified on download. Availability comes from client-driven replication, multiple relays/hosts, offline queuing, and delay-tolerant delivery.

The result is a system where clients own trust decisions, operators have narrow, verifiable roles, and even compromised infrastructure cannot access plaintext or rewrite authorization. Privacy is the default, and participation is open—without sacrificing security or resilience.

This document outlines the current architecture and design principles of the Secret platform. Implementers and reviewers should treat this text as the authoritative reference for trust assumptions, role boundaries, and security properties; any deviations must be documented with rationale and security impact.