Unlink Protocol Whitepaper V0.5

Date: December 2025
Author: Unlink Foundation
Contact: research@unlink.foundation

Table of content

1. Abstract

Unlink is a decentralized, pseudonymous messaging and data transmission protocol designed for high throughput, unlinkability, and forward secrecy. It enables secure exchange of encrypted payloads over the HyperRelay network without revealing sender–recipient relationships or relying on persistent identities, accounts, or global coordination. The protocol operates entirely at the application layer, using short-lived view keys, ephemeral encryption contexts, and stateless filter-tag matching on top of a content-agnostic relay fabric.

While optimized for private messaging, Unlink generalizes to any compact data exchange requiring anonymity, integrity, and efficient spam mitigation. It supports near real-time communication and is designed to scale to billions of messages per day, relying only on deterministic client-side cryptography and economically regulated relay infrastructure.

Fees and relay costs are settled via anonymous prepaid credits, breaking the link between funding and message transmission while preserving a fully stateless and permissionless design.

2. Motivation & Goals

Across the world, regulatory and commercial pressures are eroding the viability of private digital communication. Platforms face growing demands to monitor user activity, enforce scanning rules, and retain metadata that can be used to map social relationships. Services that cannot meet these requirements risk being restricted, delisted, or made inaccessible in entire regions. As a result, millions of people could lose reliable channels for speaking, organizing, and sharing information safely.

2.1 Problems with Existing Models

- **Centralized intermediaries:** Reliance on a single organization or company to operate relays or servers creates chokepoints for surveillance, control, and coercion. Even when the client software is open source, the entity running the infrastructure can be compelled by governments to insert backdoors or apply selective censorship.
- **Metadata exposure**: Most protocols leak identifiers (e.g., phone numbers, user handles) and allow correlation of senders, recipients, and usage patterns.
- **Spam and Sybil resistance**: Permissionless systems must defend against abuse, but few do so without trusted parties.
- **Scalability**: High-throughput messaging with privacy and decentralization is typically a trade-off; few systems can scale without compromising one or more of these.

2.2 Protocol Goals

1. **Unlinkability**
   Messages must not be linkable to the sender, recipient, or other messages, even when inspected in bulk.
2. **Minimal metadata leakage**
   The protocol must not expose identifiers, timestamps, or message routing data.
3. **Decentralization**
   There must be no need for trusted servers, relays, or intermediaries. All verification and matching should be client-side or trustless.
   The entire codebase must remain open source, and the infrastructure must be sustained through the economic model itself, ensuring that anyone can operate it, earn from it, and prevent shutdown by centralized pressure.
4. **Scalability**
   The protocol must support billions of messages per day, using on-chain infrastructure without introducing centralized bottlenecks.
5. **Forward secrecy and ephemeral identity**
   View keys and filters should evolve over time, making historical correlation computationally or economically infeasible.
6. **Quantum resistance**
   As ciphertext could be permanently stored by any party, encryption must remain secure against future quantum adversaries.
7. **Composable cost-based resistance**
   Resistance to spam, Sybils, and passive crawling should be economically enforced, not permissioned or heuristically gated.
8. **Message-agnostic design**
   Although messaging is the first use case, the protocol should generalize to any type of small, forward-secret data exchange.

3. Design Principles

Unlink is an application-layer protocol that provides strong privacy and unlinkability without relying on trusted infrastructure. It assumes that all transport and relay components are observable and potentially adversarial, and therefore places all security-critical logic at the client edge.

Messages are broadcast as indistinguishable encrypted envelopes tagged only with short, pseudorandom prefixes. Only the intended recipient can recognize and decrypt a message, and no observer can link messages to a sender, recipient, or conversation. Each message also carries the information required to derive the next encryption context, allowing conversations to progress without persistent identifiers or coordination.

These properties define the core design of Unlink: opaque transport, receiver-only discovery, and forward-evolving security state. The protocol is guided by the following principles:

3.1 Minimal On-Chain State

Unlink does not introduce protocol-level accounts, contact lists, or globally visible conversation state. Each message is self-contained at the application layer and is carried over HyperRelay as an opaque frame or blob. Because the underlying network layer maintains no per-user or per-conversation state, Unlink can operate without relying on off-chain metadata, global indexes, or trusted intermediaries. All security-relevant state—keys, authentication context, and message progression—is maintained exclusively by clients.

3.2 End-to-End Encryption

Unlink enforces strong end-to-end cryptographic guarantees using a hybrid construction executed entirely on the client. Each message derives its encryption keys from ephemeral Curve25519 ECDH shared secrets and, once established, a post-quantum Kyber shared secret, as specified in Section 6.3.

This construction provides per-message forward secrecy and long-term confidentiality against future quantum adversaries. All encryption and decryption occur exclusively on client devices; the protocol never exposes keys or plaintext to the network layer.

3.3 Sender Unlinkability

Messages are not attributable to persistent sender identities at the transport or network layer. Unlink does not use on-chain accounts, signatures, or authenticated sender addresses. Message submission costs are paid using anonymous prepaid credits, decoupling message transmission from funding identity and preventing linkage between payment activity and communication behavior.

3.4 Receiver Unlinkability

Only the intended receiver can recognize messages addressed to them. Unlink relies on HyperRelay's tag- and hint-based delivery model and does not introduce any additional addressing or routing layer. Message discovery is performed through a two-stage, stateless process that maps directly onto HyperRelay primitives.

HyperRelay tag (prefix, up to 40 bits)

Each Unlink message is emitted under a HyperRelay tag derived from a receiver-specific discovery root and the current epoch. Receiving clients subscribe to the corresponding tag prefixes at the HyperRelay layer and receive only candidate message entries whose tags match. This stage acts as a bandwidth prefilter and may be handled by untrusted relay nodes or third-party infrastructure without revealing recipient identity or message acceptance.

HyperRelay hint (shared-secret confirmation)
Each Unlink message carries a HyperRelay hint that embeds the sender's ephemeral public key, a cryptographic confirmation hint, and a one-byte metadata field. Upon receiving a tag-matched hint, the receiver derives the message's shared secret as defined in Section 6.3. Only if the comparison succeeds does the receiver proceed. From the metadata byte the receiver determines whether to fetch the corresponding full HyperRelay frame or ignore the message.

This design ensures that relays and intermediaries observe only pseudorandom tags and opaque hints, while all message acceptance, decryption, and retrieval decisions remain private and entirely client-side.

3.5 Deterministic Parsing
Every message is self-contained and follows a strict binary layout. Clients can parse, decrypt, and verify a message with only their private view key. There is no dynamic deserialization or schema negotiation. This makes parsing efficient, stable, and secure across time.

3.6 Performance-Oriented
Unlink is designed to operate efficiently on top of a high-throughput, fixed-frame relay network. By embedding a deterministic, fixed-size message format inside HyperRelay frames, Unlink ensures that message discovery and rejection incur a bounded and predictable processing cost before any decryption is attempted. Message filtering relies on short prefix comparisons and lightweight cryptographic checks before any decryption is attempted. This allows Unlink to scale with the underlying relay capacity and remain usable on both high-performance systems and constrained devices, without imposing additional load or coordination requirements on the network layer.

4. System Overview

Unlink is a messaging protocol that decentralizes trust, identity, and verification, built as an application layer on top of the HyperRelay network. It does not rely on centralized servers, trusted intermediaries, or persistent identities. The system combines client-side cryptography, compact routing metadata, and stateless discovery with a content-agnostic relay fabric that transports opaque frames and hints.

HyperRelay provides fixed-size frame transport, tag-based delivery, and economically regulated capacity. Unlink defines how encrypted messages, discovery information, and forward-secret state transitions are encoded within this transport, without imposing additional trust or coordination requirements on the network layer.

4.1 Participants

Sender

Constructs an Unlink message by deriving shared secret material, encrypting the payload, and emitting it as a HyperRelay frame under a receiver-derived tag. The sender includes the necessary hint data to allow the receiver to privately recognize the message and decide whether to retrieve it.

Receiver

Subscribes to relevant HyperRelay tag prefixes, processes incoming hints, derives shared secrets using private view keys, and selectively fetches and decrypts full frames. The receiver rotates keys and discovery parameters to maintain forward secrecy and unlinkability.

Relay Node (HyperRelay)

Operates the network layer by accepting frames, propagating them to peers, and delivering tag-matched hints or frames to connected clients. Nodes do not interpret Unlink semantics, decrypt payloads, or learn sender or receiver identities. Their behavior is constrained economically by stake-based capacity rules.

Payment Interface

Unlink message submission costs are paid using anonymous prepaid credits redeemed with a HyperRelay operator. Payment and funding are decoupled from message content and identity, and are not visible at the protocol level.

4.2 Trust Assumptions

No network component is trusted to preserve privacy, correctness, or availability.
Only holders of the correct private view keys can recognize, fetch, and decrypt messages.
Relay nodes observe only pseudorandom tags, opaque hints, and encrypted payloads, and cannot link messages to identities or conversations.
Post-quantum security and forward secrecy depend on correct client-side handshake execution and key rotation.

4.3 Message Lifecycle

Derive

The sender derives shared secret material, constructs the payload, and emits a HyperRelay message under a receiver-derived tag. The message includes hint data that allows the receiver to privately recognize the event and determine subsequent action.

Discover

The receiver receives tag-matched hints from the HyperRelay network and applies cryptographic confirmation using shared-secret derivation.

Act
If the cryptographic hint matches, the receiver interprets the embedded metadata to decide whether to fetch the corresponding HyperRelay frame, process the event based on the hint alone, or ignore it entirely. Hint-only events may encode state transitions or signals that do not require payload retrieval.

Retrieve and Decrypt (optional)
If indicated by the metadata, the receiver fetches the corresponding HyperRelay frame and decrypts the payload using ECDH- and Kyber-derived secrets.

Advance
The receiver updates local conversation state, including view keys and discovery parameters, and may send a reply under the next derived tag.

The protocol is state-progressing yet network-stateless: all identity, authentication, and conversation state exists exclusively on client devices, and no persistent addresses or global accounts are ever exposed to the network.

5. Core Concepts

Unlink defines a minimal, scalable, and secure messaging primitive with a strict separation between sender, receiver, and relay roles. It combines hybrid encryption, anonymous addressing, and forward-secret state transitions to enable private, pseudonymous messaging and data transmission over a content-agnostic relay network.

5.1 Identity and Key Structure

Each user generates a 24-word mnemonic seed from which a long-term view key pair (Curve25519) is deterministically derived. The public part of this key pair serves as the user's long-term public view key and is used by senders to address initial messages.

The mnemonic may be stored for recovery. The protocol does not rely on network-level identities, accounts, or registrations, and no persistent identifiers are exposed to relays.

5.2 Handshake and Key Exchange

Conversation setup begins with a handshake protocol, initiated by a sender who knows the receiver's long-term view public key. The handshake uses ephemeral Curve25519 ECDH for immediate forward secrecy and simultaneously establishes post-quantum protection by exchanging Kyber material. It completes in two standard messages, an invitation carrying a Kyber public key and a confirmation carrying the corresponding ciphertext, after which both sides share a Kyber secret used in subsequent communication. Each non-handshake message is encrypted in two layers:

- **Inner Layer (Kyber):** A quantum-resistant shared secret established during conversation setup. It can be renewed on request at any point in the conversation to add forward secrecy beyond the initial exchange.
- **Outer Layer (Ephemeral ECDH):** A unique ephemeral Curve25519 key generated per message and combined with the receiver's view public key to derive a fresh symmetric key. This enforces forward secrecy at the message level and keeps decryption efficient.

Messages are only decryptable by the intended receiver and expose no metadata. Ciphertexts are indistinguishable from random noise.

5.3 Discovery Roots

A discovery root is a value shared between peers and used to derive epoch-specific tag spaces (Tag Space ID's) for message routing and as contextual partial input to encryption key derivation. Discovery roots are derived directly from a keypair's private component and are distributed explicitly by clients to intended peers or subscribers. Distinct keypairs produce unlinkable discovery roots, ensuring isolation across relationships and distributions.

$\text{discovery\_root}_i = \text{BLAKE3}(\text{“discovery\_root”} \parallel \text{private\_key} \parallel i)$

Clients may derive one or more discovery roots from the same keypair and share selected roots with peers to enable message discovery.

5.4 Filter Tags and Message Discovery

Receivers discover messages through HyperRelay tag prefixes derived from discovery roots and rotated per epoch. The number of meaningful bits is configurable, allowing clients to trade bandwidth for anonymity: shorter tags increase collision cover, while longer tags reduce noise.

Clients derive only the tags required for active conversations and may add decoys for deniability. Matching is stateless and does not require registration. Relays assist with prefiltering by delivering tag-matched hints, but cannot link tags or messages to recipient identities.

5.5 Stateless Matching

Unlink introduces no protocol-level state at the network layer. Clients can receive and process messages without accounts or persistent identifiers. Because discovery relies solely on rotating tags and cryptographic confirmation, infrastructure can scale horizontally and remain untrusted.

5.6 Forward Secrecy and Required Key Rotation

Each message includes a new view public key for the next step in the conversation, enforcing mandatory forward secrecy. Encryption keys advance with every message, and view keys must never be reused once rotated. Compromise of a single key does not reveal past or future messages. All conversation state is tracked and persisted locally by clients.

6. Envelope and Payload Transmission

All Unlink messages are transported using fixed-size HyperRelay frames. Each frame carries an opaque, encrypted payload and a compact set of routing and discovery fields defined by the HyperRelay network. Unlink defines how application data is framed, padded, encrypted, fragmented, and encoded within this transport, without exposing message type, structure, or semantics to the network layer.

All message classes, including handshakes and control messages, use the same encrypted payload format and are indistinguishable at the transport level.

6.1 DataFrame Structure (Inner Payload)

Each Unlink message contains an encrypted DataFrame that holds all application-layer state and content. The DataFrame plaintext is constructed, padded to a fixed size, and then encrypted before being embedded as the payload of a HyperRelay frame.

The plaintext DataFrame consists of:

- **Length Prefix (2 bytes)**: Size of the actual plaintext payload.
- **Conversation ID (16 bytes)**: Uniquely identifies the conversation thread.
- **Sender Proof (32 bytes)**: HMAC that authenticates the sender without linking identities.
- **Next View Public Key (32 bytes)**: Enables forward-secret progression for replies.
- **Current View Public Key (32 bytes)**: Identifies the current message sender.
- **Payload (variable)**: Message content, metadata, or application-specific data. The payload always begins with a fixed-size fragment
- header, followed by a fragment body, as specified in Section 6.2.

The DataFrame is padded with random bytes to a fixed size before encryption. The entire DataFrame, including headers and payload, is fully encrypted.

6.2 Message Fragmentation and fragment Header

Unlink treats each HyperRelay frame as carrying exactly one encrypted DataFrame fragment. Larger logical messages are split into multiple fragment that share a common identifier and are reassembled client-side after decryption.

The Payload field of the DataFrame (Section 6.1) is structured as:

- **Fragment Header**: Fixed-size metadata describing how this fragment relates to a larger logical message.
- **Fragment Body**: A slice of application data (e.g., part of a Kyber public key, ciphertext, or text message), followed by random padding to fill the DataFrame.

6.2.1 Fragment Header Layout

Each fragment begins with the following header:

- **Version (1 byte)**
  Indicates the fragment header format version. The initial version is 1.
- **Message ID (8 bytes)**
  A 64-bit random identifier generated by the sender for a single logical message. All fragments belonging to the same logical message must share the same Message ID.
- **Fragment Index (2 bytes, big-endian unsigned)**
  Zero-based index of this fragment within the logical message.
- **Total Fragments (2 bytes, big-endian unsigned)**
  Declares the total number of fragments for this logical message.
- **Message Type (1 byte)**
  Identifies the semantic type of the logical message carried across one or more fragments.

The value space is partitioned into two regions:

0–127: Protocol-Reserved Types
Values in this range are defined and versioned by the Unlink protocol specification.
Current assignments:

- 0: Handshake Request
- 1: Handshake Confirmation
- 2: Standard Text Message
- 3: Credit transfer

128–255: Application-Custom Types
These values are available to applications and extensions.

- **Client ID (8 bytes)**
  An opaque per-device or per-instance identifier that allows senders to direct a message to specific clients without affecting security. A value of zero can be used to indicate a broadcast to all clients sharing the same view key. Client ID is a delivery hint rather than an access control mechanism.

The remaining bytes of the payload after the fragment header are the Fragment Body.

6.2.2 Fragment Encoding and Transport Rules

Senders:

1. Construct the logical message body.
2. Compute its total length and split it into consecutive slices of at most MAX_CHUNK_BODY bytes.
3. Generate a fresh random 8-byte Message ID for this logical message.
4. For each slice i in [0, TotalFragments − 1], create one chunk with:
   - Message ID set to the generated value.
   - Fragment Index = i.
   - Total Fragments equal to the total number of slices.
   - Message Type set appropriately (e.g., 0 for handshake request).
   - Client ID set to the intended audience (or a reserved broadcast value).
   - Chunk Body equal to the slice (plus padding to fill the frame).

Receivers:

1. For each decrypted frame, parse the Chunk Header and Chunk Body.
2. Group chunks by (Conversation ID, Message ID, Client ID) and buffer them until all fragments 0..TotalFragments − 1 have been received.
3. When all fragments are present, concatenate Chunk Bodies in ascending Fragment Index order to reconstruct the logical message body.
4. Pass the reconstructed body and Message Type to the appropriate higher-level handler (e.g., handshake handler in Section 7, text message parser, etc.).

Fragments may arrive out of order. To avoid resource exhaustion, clients discard partial messages that remain incomplete for too long or if conflicting fragments with the same index appear.
If two fragments with the same (Message ID, Fragment Index) contain different data, the entire logical message is discarded.

6.3 Hybrid Encryption Scheme

Unlink uses a two-layer hybrid encryption model that combines an ephemeral Curve25519 ECDH secret with an optional Kyber shared secret. The ECDH secret is always present; the Kyber secret becomes available after a successful handshake and strengthens the scheme against future quantum adversaries.

For each message, the sender derives:

- a 32-byte ECDH secret from an ephemeral Curve25519 keypair and the receiver's current view key
- an optional 32-byte Kyber secret (established during the handshake)

In addition, each message is bound to its current tag space via a fixed-length Tag Space ID (Section 8.2).

These values are treated as key material inputs to a BLAKE3-based key derivation step with simple domain separation:
- the inner (post-quantum) layer derives its encryption key from the concatenation of the Kyber secret, ECDH secret and the Tag Space ID
- the outer layer derives its encryption key from the ECDH secret and the Tag Space ID

Both layers use separate BLAKE3 labels so their keys cannot collide or be confused. The output of each derivation is used as the symmetric key for a ChaCha20-Poly1305 encryption pass.

The inner layer, when present, encrypts the padded DataFrame first. The resulting ciphertext is then encrypted again under the outer layer using the ephemeral ECDH key. If no Kyber secret is available, only the outer layer is applied.

This structure provides message-level forward secrecy through ephemeral ECDH and long-term post-quantum confidentiality through the Kyber component. Encryption keys are derived with explicit inclusion of the Tag Space ID, binding ciphertexts to their routing context and preventing correlation or reuse across different tag spaces, even when carrying identical plaintext.

6.4 Envelope Layout (Outer Structure)

Each Unlink message is embedded in a HyperRelay frame with fixed size. The frame includes:
- HyperRelay Tag - Derived from the receiver's discovery root and current epoch
- HyperRelay Hint - Contains:
  - Sender's ephemeral public key
  - Cryptographic confirmation hint
  - 1-byte metadata field indicating receiver action (e.g. fetch frame, hint-only event)

  The entire hint is encrypted under the same key material as the outer encryption layer, derived from the ephemeral ECDH secret, optional Kyber shared secret, and the current Tag Space ID.
- Encrypted Payload - The encrypted DataFrame, padded to fill the frame.

6.5 Shared Secret Hint

Upon receiving a tag-matched hint, the receiver derives a shared ECDH secret using the sender's ephemeral public key and its own private view key. A 4-byte cryptographic confirmation value is derived from this shared secret and compared verbatim against the confirmation hint carried in the message. Only an exact 32-bit match causes the client to consider the frame potentially relevant.

The same shared secret is then used to decrypt the final byte of the hint, yielding a one-byte encrypted semantics field. This byte is recoverable only by a receiver in possession of the correct shared secret and is computationally indistinguishable from random data to all relays and passive observers.

The confirmation hint is used exclusively for exact-match filtering and is never interpreted. The decrypted semantics byte is interpreted only after a successful confirmation match and serves solely as advisory client-side guidance. It is not authoritative and does not affect message correctness, authenticity, or protocol state.

6.6 Encrypted Semantics Byte

Each Unlink message carries a single encrypted semantics byte as part of the HyperRelay hint. This byte is XOR-masked with a key byte derived from the same master secret as the confirmation hint. The transformation is intentionally minimal and length-preserving: one input byte produces one output byte with no expansion.

The semantics byte provides lightweight, advisory metadata that helps a client make local decisions before fetching or fully decrypting the payload. It is explicitly non-authoritative and must never be treated as a source of truth for message validity, authenticity, or application state.

Typical uses include fetch prioritization, client-side scheduling, rough estimation of retrieval cost (for example, expected fragment count), and coarse message classification such as distinguishing ordinary messages from status signals or application-defined extensions.

All definitive interpretation of message type and meaning must occur only after full payload retrieval, decryption, and verification using sender proofs and conversation state.

Indicative, Not Authoritative

The semantics byte is explicitly advisory:

- Clients must not derive persistent state, security decisions, or correctness assumptions from it.
- Corrupted, missing, or misleading semantics must not cause incorrect application behavior.
- Clients may ignore the semantics byte entirely without affecting protocol correctness.

6.7 Padding and Size Uniformity

All encrypted payloads are padded to a fixed size before transmission. This prevents inference of message type, size, or fragmentation pattern and ensures that all Unlink messages are indistinguishable at the transport level.

6.8 Anti-Spam Mechanism

Unlink enforces a minimal, economic anti-spam mechanism using anonymous prepaid credits. Credits are embedded inside encrypted handshake request payloads as transfer bundles and are evaluated exclusively by the receiver.

A sender initiating a handshake may include a transfer bundle containing a denomination, expiry timestamp, redeem endpoint, and an opaque operator-signed proof (Appendix B). After decryption,

the receiver may accept or ignore the handshake based on locally defined thresholds. All filtering is client-side and requires no on-chain state, accounts, or centralized lists.

Credits are required only for initial contact. Once a conversation state is established, subsequent messages proceed without additional credits.

Credit evaluation allows receivers to set their own economic spam threshold. Thresholds may vary by receiver and context, reflecting differing tolerance for unsolicited contact.

Credits are anonymous bearer instruments obtained off-path. Relays and observers cannot associate credit usage with sender identities or funding sources.

7. Message Types and Lifecycle Semantics

Unlink makes a strict distinction between transport and semantics. All messages use the same envelope, encryption scheme, padding rules, and fragmentation logic as defined in Chapter 6. There are no transport-level message classes. Differences between messages exist only at the semantic layer, after decryption and reassembly.

A logical message may span one or more fixed-size frames. Frames are reassembled client-side, after which the resulting byte sequence is interpreted according to its message type. Message types affect only semantic processing and do not alter transport behavior or observability.

7.1 Logical Messages and Fragment Reassembly

Each frame carries exactly one fragment of a logical message. Fragments are identified by a random 64-bit Message ID and include their fragment index and total fragment count. Fragments may arrive out of order or be duplicated. Receivers buffer fragments and reassemble the logical message only when all fragments are present. If conflicting fragments are detected, or if reassembly does not complete within a local timeout, the message is discarded.

Once reassembled, the logical message body is processed exactly once. Deduplication is purely client-side and based on Message ID. The protocol does not require acknowledgements or retries at the transport layer.

7.2 Protocol-Reserved Message Types

The Unlink protocol reserves a range of message types for core functionality. These types are interpreted uniformly across implementations and define cryptographic state transitions.

Type 0 denotes a Handshake Request.
Type 1 denotes a Handshake Confirmation.
Type 2 denotes a Standard Message payload.
Type 3 and higher reserved values may be used for protocol-level status updates or future extensions.

Values above 127 are application-defined and have no meaning to the base protocol beyond correct framing and authentication.

7.3 Handshake Semantics

The handshake establishes a shared post-quantum secret between two parties and binds future messages to that secret. It is implemented entirely as ordinary messages and does not introduce any special transport behavior.

7.3.1 Handshake Request (messageType = 0)

A handshake request initiates a conversation and introduces post-quantum key material.
The payload conveys the sender's Kyber public key and an optional introductory message. Upon successful decryption, the receiver performs Kyber encapsulation and prepares a confirmation response.

The handshake request is transported as an ordinary message and is indistinguishable from other traffic at the transport layer. The canonical payload encoding is defined in Appendix C.

7.3.2 Handshake Confirmation (messageType = 1)

A handshake confirmation completes the post-quantum key exchange. The payload contains the Kyber ciphertext generated for the initiator, allowing both parties to derive a shared post-quantum secret.

After decryption and decapsulation, the derived Kyber secret is combined with the existing Curve25519 shared secret to form key material for subsequent messages. Payload encoding is defined in Appendix C.

7.4 Status and Control Messages

Status and control messages may be conveyed as hint-only updates that do not require retrieval or decryption of a full HyperRelay frame. Such updates are permitted only after the post-quantum handshake has completed and both parties have established a shared Kyber secret for the conversation. This ensures that a client never emits hint-only status or reaction signals in response to messages from peers with whom no authenticated conversation state exists.

For each successfully decrypted post-handshake message m, both parties derive a 32-byte per-message secret S_m:
$S_m = BLAKE3("unlink/message\_secret" \| Kyber\_secret \| ECDH\_secret\_m)$

where Kyber_secret is the established post-quantum shared secret for the conversation and ECDH_secret_m is the message's ephemeral Curve25519 shared secret. S_m is never transmitted and is undefined for pre-handshake messages.

A hint-only status/control update is encoded entirely inside the 37-byte HyperRelay hint field as:
- 8 bytes: receipt_id
- 13 bytes: payload_enc
- 16 bytes: tag

Receivers must attempt to interpret incoming hints as status/control hints first by checking the receipt_id against the local outbound-message receipt table. If no match exists, the hint must be processed as a standard message hint as defined in Section 6.5.
The decrypted status payload format and semantic field definitions are specified in Appendix G.

The receipt identifier is derived from S_m and a monotonic counter ctr:
$receipt\_id = Trunc64(BLAKE3("unlink/hint/rid" \| S_m \| ctr))$

Payload encryption is a length-preserving XOR with a derived keystream:
$ks = BLAKE3("unlink/hint/ks" \| S_m \| receipt\_id)[0..12]$
$payload\_enc = payload \text{ XOR } ks[0..12]$

Authenticity and integrity are enforced with a mandatory 128-bit authentication tag:
$tag = Trunc128(BLAKE3\_KEYED\_HASH(key=S_m, input="unlink/hint/mac" \| receipt\_id \| payload\_enc))$

Receipt identifiers are used exclusively as non-authoritative local lookup hints and are not relied upon for authenticity or integrity. Implementations must ensure that receipt identifiers are never reused for the same message by strictly advancing the associated counter, as reuse may introduce linkability between status updates. Only parties who successfully decrypted the referenced message and derived S_m can generate or verify valid status hints.

Status hints are replay-protected using a bounded, receiver-maintained counter window scoped to the referenced message. Counters outside the acceptance window or previously processed values must be rejected.

Status and control updates do not alter transport behavior or observability. Implementations may emit a status hint as a hint-only update or may optionally retrieve and transmit a full frame with application-defined or random payload data. In all cases, status handling occurs exclusively at the semantic layer and must not introduce observable transport-level differences.

7.5 State Progression and Rekeying

Every successfully processed message advances local cryptographic state. Each message carries a next view public key, which becomes the active sender key for subsequent communication. This enforces forward secrecy and prevents correlation across messages.

Clients may optionally establish new post-quantum key material at any point by sending a message that carries fresh Kyber parameters. Such rekeying uses ordinary message types and does not alter transport behavior or observability.

8. Filter Tag System

Unlink uses compact, prefix-based filter tags mapped directly onto the HyperRelay tag space to enable scalable message discovery without exposing recipient identities or relationship structure. Filter tags are short, pseudorandom prefixes derived from secret discovery roots and rotated over time. Relays and intermediaries observe only opaque tag prefixes and cannot infer their origin or meaning.

Senders addressing a specific recipient do not choose tags arbitrarily. Instead, they derive tags from discovery roots explicitly provided by the recipient for that relationship. Only the recipient can derive the correct tag set for a given epoch and determine which messages may belong to them.

8.1 Discovery Surface

Each client maintains a fixed-size discovery surface consisting of a set of opaque discovery roots. Discovery roots are secret values used to derive filter tags and have no protocol-visible semantics. How roots are assigned to contacts, channels, or other relationships is an implementation detail.

Each client must maintain 32 discovery roots per epoch. If fewer are used for active relationships, remaining slots must be filled with decoy roots. If a client uses fewer roots for active relationships, remaining slots are filled with decoy roots. As a result, all clients expose an identical-sized discovery surface per epoch, independent of contact count or usage patterns.

8.2 Prefix Derivation and Epoch Rotation

Discovery is anchored to HyperRelay epoch derivation:
epoch_e = floor(unix_timestamp_seconds / 600)

Where unix_timestamp_seconds is derived from network-observed time consistent with HyperRelay.

Tag Space ID

For each discovery root R and epoch e, the client derives a Tag Space ID:
tag_space_id = BLAKE3(R || e)

The Tag Space ID uniquely identifies the routing context for that root during the given epoch and is used to bind encryption to the tag space.

The HyperRelay tag is derived by taking the leftmost L bits of tag_space_id, left-aligned and zero-padded into a fixed 5-byte field. The prefix length L is selected by the client within the supported range (1–40 bits), allowing a trade-off between anonymity set size and matching overhead.

Filter Tag Serialization

Each filter tag is serialized as:
1 byte: prefix length L, selected by the client based on expected global traffic density.
Each client may choose its own prefix length within the supported range (1–40 bits), trading anonymity set size against matching overhead.

- **5 bytes:** tag data = the leftmost L bits of tag_value_e, left-aligned and zero-padded

To ensure reliable discovery across epoch boundaries, clients must follow tags derived from:

- All 32 tags for the **current** epoch e
- All 32 tags for the **previous** epoch e − 1

Messages addressed under either epoch are considered valid. This dual-epoch subscription prevents message loss due to propagation delay or epoch transition while preserving a fixed observable footprint.

8.3 Sender Routing

Senders derive filter tags exclusively from discovery roots provided by recipients or publishers. For first contact, the discovery root is the recipient's long-term public view key; subsequent roots may be rotated by the recipient. Senders never choose arbitrary prefixes and cannot route messages without possession of a valid discovery root.

Direct Routing

For direct or small-group relationships, the recipient reveals exactly one discovery root to the sender. The sender:

- derives the Tag Space ID from that root
- places all messages for that relationship under the derived tag

At any given time, each relationship is bound to a single discovery root. The recipient may rotate or replace the root at an epoch boundary by revealing a new root; such changes are opaque to the network.

Broadcast and Multicast Routing

For broadcast-style distribution, a publisher exposes a multicast keypair.
Subscribers deterministically derive the same discovery root set from this keypair and select exactly one root to follow for that distribution. The selected root is added to the subscriber's discovery surface for both epoch e and e − 1.

When publishing a message, the sender encrypts the payload once and addresses it under a small, fixed subset of the associated discovery roots (e.g., one per index or per audience cluster). The same encrypted payload may therefore appear under multiple filter tags.

Subscribers following different roots observe the same ciphertext under different prefixes. Relays and intermediaries see only independent, pseudorandom tags and cannot infer audience membership, size, or overlap.

Publisher Authenticity

Multicast confidentiality does not imply publisher authenticity. Any party in possession of the multicast key material can construct valid ciphertexts for the channel and attempt to impersonate the publisher. Clients must verify the sender proof to verify authenticity. (Section 9)

8.4 Trust and Root Exposure

Clients can bound information leakage about their discovery surface by controlling which roots they reveal and when. A simple pattern is:

- **Low-trust bucket:**
  New or unverified contacts are initially assigned to a designated subset of personal roots that the user is comfortable exposing more broadly (a "low-trust bucket").
- **High-trust bucket:**
  After a local trust decision, the recipient may move a relationship to another personal root and start using that root from the next epoch onward.

The specific policies (e.g. which roots constitute the low-trust bucket) are client-defined. The protocol only requires that each relationship is bound to a single root at any given time.

8.5 Decoys and Traffic Shaping

To prevent inference of discovery surface size or user activity, clients maintain a fixed discovery footprint per epoch. Each client tracks a protocol-defined number of discovery roots per epoch; any unused slots are filled with decoy roots that produce synthetic filter tags.

Clients may further inject decoy retrievals and vary query patterns (for example, fetching entire blocks or additional candidate frames) to obscure real activity. These measures ensure that both the number of active tags and the observable traffic envelope remain uniform across users, independent of contact count or usage.

8.6 Privacy Properties

The filter tag system provides:

- Fixed discovery surface: Observers see a constant number of active filter tags per client per epoch (and for the previous epoch), independent of the number of contacts, channels, or message volume.
- Per-root isolation: Disclosure or compromise of a single discovery root affects only the tag spaces derived from that root and does not expose other relationships.
- Epoch unlinkability: Filter tags are derived from discovery roots and epoch identifiers. Without knowledge of the underlying root, tags from different epochs cannot be linked.
- Multicast anonymity: Subscribers to the same distribution typically follow different tag spaces. Relays and intermediaries cannot infer audience membership, size, overlap, or message equivalence from observed tags.

Together, these properties enable scalable, private message discovery while keeping routing infrastructure fully untrusted and observationally uniform.

9. Authentication and Integrity

Unlink ensures message authenticity and integrity without wallet signatures, on-chain identity, or interactive channels. Authentication is handled entirely off-chain through a deterministic HMAC-based sender proof. The proof is derived from the Curve25519 ECDH shared secret between the sender's view private key and the receiver's view public key, and does not expose sender metadata.

9.1 Sender Proof Construction

Every Unlink message carries a cryptographic sender proof that allows the receiver to authenticate the sender using only the message contents and key material included inside it.

The sender computes:
shared_secret = ECDH(sender_view_priv, receiver_view_pub)
hmac_key = SHA-256(shared_secret)

The HMAC-SHA256 is computed over a deterministic challenge containing, in order:

1. A fixed domain separation prefix (to prevent cross-protocol misuse)
2. the conversation ID
3. the receiver's view key
4. the next view key for forward secrecy
5. a hash of the plaintext message content

This binds the proof to the message, the expected state transition, and the recipient's view key. Only a party holding the sender's view private key can derive the correct shared secret and produce a valid proof.

9.2 Verification

After decryption, the receiver recomputes the ECDH shared secret:
shared_secret = ECDH(receiver_view_priv, sender_view_pub)
hmac_key = SHA-256(shared_secret)

The receiver reconstructs the challenge and verifies the HMAC. No additional session state, out-of-band metadata, or on-chain identity is required.
Verification confirms:

- The message was created by someone in possession of the correct sender private view key
- The message has not been modified in transit
- The forward key rotation (via next view key) is legitimate

9.3 Forward Secrecy Enforcement

Each message mandates the inclusion of a **next view public key**, rotated for every transmission. The sender HMAC covers this next key to ensure its authenticity and prevent rollback attacks.
This mechanism ensures:

- Compromise of a single key reveals no past or future messages
- Recipients can detect replay or impersonation by validating expected key progression

9.4 Threat Resilience

Unlink's authentication layer is designed to resist a range of adversarial models:

Passive observers (e.g., relays, indexers, or network monitors)
Cannot correlate messages to identities or relationships. Messages carry no stable identifiers, and sender authentication material is derived from ephemeral shared secrets and tag-space–bound context, making proofs unlinkable across messages or epochs.

Active matchers or malicious clients
Cannot forge valid messages without possession of the appropriate private view key or, in the case of multicast, a valid publisher signing key. Injected or modified messages fail sender-proof or signature verification and are discarded by receivers.

Key compromise and IP-level correlation
Compromise of a private view key enables future impersonation but does not allow decryption of past messages or forgery of prior traffic due to mandatory key rotation and forward secrecy. Network-layer metadata (e.g., IP addresses) remains outside the protocol threat model and must be mitigated by clients through appropriate transport-layer anonymity measures.

Together, these mechanisms provide cryptographic authenticity and strong sender deniability, even in adversarial environments or under surveillance.

10. Security & Privacy Model

Unlink is designed to minimize metadata leakage, resist deanonymization attempts, and preserve the privacy and integrity of all messages, even in the face of advanced adversaries.
It achieves this through a layered set of security guarantees, outlined below.

10.1 End-to-End Confidentiality

- Every message is encrypted in two layers, one using Curve25519 ECDH for forward secrecy and one using a shared Kyber-derived secret for post-quantum resistance. Encryption keys are additionally bound to the current tag space, ensuring ciphertexts are unlinkable across routing contexts even when carrying identical plaintext.
- The only visible message content on-chain is opaque ciphertext. There are no addresses, public keys, or plaintext metadata exposed.

10.2 Forward Secrecy

- Each message uses a new ephemeral ECDH keypair.
- A new view public key is included in every message payload.
- Clients must rotate view keys, tag spaces (via epoch progression), and shared secrets accordingly.

This ensures that the compromise of a single key does not reveal past messages, nor allow linking across sessions.

10.3 Post-Quantum Resistance

- The protocol incorporates Kyber to protect against future quantum attacks.
- A shared secret is established via Kyber during the handshake phase and used as part of the encryption key for all follow-up messages.
- Even if **Curve25519** were broken by a quantum adversary:
  - The handshake messages and conversation links may become visible.
  - However, **all follow-up messages remain encrypted** and secure due to Kyber-based encryption.
- Clients can rotate Kyber secrets periodically to improve forward secrecy over long sessions, further limiting the blast radius of any key compromise.

10.4 Unlinkability

- No addresses, account identifiers, or sender/receiver metadata appear on-chain.
- Filter tags are compact pseudorandom values derived from shared discovery roots and epoch identifiers. They define tag spaces used for routing and are bound into encryption key derivation, preventing linkage across epochs or routing contexts.
- These tags are indistinguishable from random data and do not leak identity information.
- Ephemeral keys, shuffled tag lists, and optional decoys make it infeasible to correlate two messages to the same user.
- Payments are unlinkable through the integration of one-time anonymous prepaid credits rather than wallet transfers, breaking any link between funding identity and communication activity.

10.5 Passive Adversary Resistance

- Observers, or relayers cannot infer which user a message belongs to.
- Shared secret hints only reduce bandwidth requirements, they do not compromise privacy.
- Users may optionally fetch decoy transactions or entire blocks to obscure retrieval patterns.
- HyperRelay hints, including cryptographic confirmation material and receiver-action metadata, are encrypted and do not reveal whether a message was accepted, ignored, or fully retrieved.

10.6 Active Adversary Resistance

- HMAC sender proofs allow receivers to detect spoofed messages.
- Matching via filter tags is read-only; an adversary cannot force processing or cause resource exhaustion.
- Optional programmable minimum fees discourage denial-of-service attacks via spam.

10.7 Key Compromise Resilience

- Because messages rotate both view keys and Kyber secrets, past traffic remains secure if a current key is compromised.
- Compromise of a view key does not, by itself, reveal message contents. An attacker would also need access to the encrypted message payloads and either the correct filter tag parameters or a way to identify which messages to attempt decryption against.
- Under incorrect implementation or adversarial matcher setups, IP-level metadata may be linked to a user's public view key. However, the protocol itself does not expose or rely on any network-layer identifiers.

10.8 Network-Layer Privacy

Unlink assumes that clients route both message submission and message retrieval through privacy-preserving network channels. In practice, the recommended approach is to use a fresh Tor circuit for every message post and every message fetch. This prevents HyperRelay nodes or local network observers from linking multiple actions to the same user or correlating them with a stable network identifier.

11. Credits

Unlink uses anonymous, prepaid credits as the only fee mechanism for message submission and related on-chain operations. Credits act as unlinkable bearer instruments: they are purchased or obtained off-path, redeemed by whoever holds them, and never tied on-chain to a funding source or user identity. This preserves sender unlinkability while giving relay operators a simple, predictable revenue model.

11.1 Credit Proof and Transfer Bundle

Each relay operator issues its own prepaid credits. At the core is a credit proof: an operator-signed, opaque value that authorizes one unit of service. The operator defines the internal structure of the credit proof and the verification rule for that proof.

The protocol distinguishes three related representations:

1. **Credit proof (operator-level object)**
   - An opaque, operator-signed value.
   - Only the issuing operator can validate it.
   - No structure is assumed by the protocol.
2. **Transfer bundle (user-to-user representation)**
   When a credit is sent from one user to another, it is wrapped in a transfer bundle so the receiver knows what it is and where it can be redeemed. The transfer bundle contains:
   - redeem endpoint (how to reach the issuing operator),
   - denomination or usage class,
   - expiry timestamp,
   - credit proof.

This bundle is not placed on-chain; it is used for off-path transfer and storage.

11.2 Redemption and Spend Semantics

To redeem a credit, the client provides the issuing operator with a transfer bundle and the Unlink envelope to be submitted. The operator must verify the credit proof according to its internal rule set, check the expiry timestamp, and enforce single use. If valid, the operator submits the envelope on-chain and marks the proof as spent.

11.3 Issuer Trust and Credit Acceptance

Credit proofs are validated only by the issuing operator, so their practical value depends on the recipient's confidence that the operator will honor redemptions. Clients therefore maintain a local trust policy: credits from unknown issuers are treated as having no effective value for spam-fee thresholds and place incoming handshakes in a low-trust or quarantine state. An issuer becomes trusted once the user accepts the contact or successfully redeems a credit with that operator. Issuers that refuse or fail to redeem are removed from the trusted set, and future credits from them are ignored. This lightweight trust model keeps credit-based filtering decentralized while preventing malicious or unreliable issuers from influencing recipient decisions.

11.4 Scope, and In-Protocol Usage

Credits are scoped to a single operator and are never validated or interpreted by any other party. Users may hold credits from multiple operators simultaneously, and relay selection strategies are entirely client-side.

Within the protocol, credits appear only inside encrypted message payloads. They are typically included in handshake requests to satisfy the receiver's anti-spam requirement but may also be returned or forwarded in any subsequent message. No outer envelope field indicates the presence or absence of a credit, preserving indistinguishability and unlinkability across all message types.

12. Comparison to Existing Systems

In section 12.1 is a condensed scorecard covering the key attributes that matter most for users: privacy strength, metadata exposure, unlinkability, post-quantum readiness, scalability, and regulatory resilience. These attributes determine real-world safety; if a messenger leaks metadata or linkability, users can be profiled, mapped, or targeted even when messages are encrypted. Weak scalability or regulatory pressure also degrade reliability and can force operators to weaken security.

12.1 Overview

Messenger	Privacy	Metadata resilience	Linkability resilience	Post Quantum Secure	Scalability	Regulatory Resilience	Trust required
Signal	Medium	Medium	Low	Y	High	Low	Medium
Telegram	Low	Very low	None	N	High	Very low	Very high
WhatsApp	Medium	Low	Low	N	High	Very low	Very high
Matrix	Low	Low	Low	P	Medium	Low	High
Keet	Medium	Low	Low	N	Poor	Strong	Medium
Nostr DM	Low	Very low	None	N	Medium	Low	Medium
Session	Medium	Medium	Medium	N	Medium	Medium	Medium
Waku	Low	Low	None	N	Moderate	Low	High
Briar	High	High	High	N	Low	High	Low
Tox	Medium	Medium	Medium	N	Limited	Medium	Medium
Wire	Medium	Low	Low	N	High	Low	High

Threema	Medium	Low	Medium	No	Good	Medium	High
Cwtch	High	High	High	Partial	Limited	High	Low
Unlink	**Very high**	**Very high**	**Very high**	**Yes**	**High**	**Very high**	**None**

12.2 What Competitors Do Well

Signal - Industry-leading cryptography and UX, but centralized and metadata-exposed, making it vulnerable to coercion.
WhatsApp - Massive global reach and frictionless onboarding, but fully dependent on Meta and entirely metadata-visible.
Telegram - Extremely fast with rich features and huge channels, but operators have full visibility into content and metadata.
Matrix - Open, extensible federation good for communities, but home servers expose metadata and cannot withstand scanning mandates.
Keet / Holepunch - Strong P2P performance with no central servers, but IP metadata and frequent relay fallbacks break anonymity.
Session - Number-free onboarding and onion routing, but static IDs and storage servers leak metadata.
Nostr DM - Open, censorship-resistant protocol, but DMs are tied to public keys and relays can log traffic.
Cwtch / Briar -Strong metadata resistance, but not scalable for mainstream or high-throughput use.
Waku / XMTP - Developer-friendly Web3 messaging, but identity is tied to wallets and metadata resistance is weak.

12.3 Comparison

Unlink advances beyond existing systems on the properties required to operate privately under coercive or scanning-mandated environments.

No metadata surfaces
The protocol has no phone numbers, no accounts, no inboxes, no routing tables, and no server-side logs. Messages are fixed-size ciphertext envelopes with no observable patterns.

No reliance on operator trust
Unlink assumes every relay could be malicious. Privacy holds even if infrastructure is fully adversarial, because relayers receive no linkable information.

Resilience under scanning mandates
There are no central operators with data or keys that can be compelled to weaken privacy. The design removes coercion points rather than hardening them.

Post-quantum protection
Hybrid Curve25519 + Kyber ensures forward secrecy today and protection against future quantum attacks. This is essential because ciphertext is stored on-chain.

Privacy by architecture
All trust is isolated to the client. Key handling, encryption, matching, and verification occur exclusively locally. Infrastructure learns nothing about users or relationships.

Economic spam resistance
Prepaid anonymous credits act as decentralized spam filters. When A contacts B for the first time, the message includes a small "spam fee." B sets a minimum threshold; anything below it is rejected locally before display. Even a minimal required fee makes large-scale spam economically unattractive, without accounts, captchas, or centralized moderation.

Stateless, horizontally scalable architecture
No global state or user accounts. Relayers operate like commodity nodes: accept encrypted payloads, forward them, and scale linearly by adding more operators.

13. Appendices

A. Filter Tag Serialization

Filter tags are 6-byte values used for efficient message filtering. Their structure:

- Byte 0: Bit length (1–40), number of meaningful prefix bits.
- Bytes 1–5: Tag data, left-aligned, zero-padded prefix of BLAKE3(discovery_root ‖ epoch).

Each client exposes a fixed set of tags derived from its discovery roots across the current and previous epoch. Tags are public and appear in the message envelope.

B. Transfer-Bundle Format (Canonical Definition)

The following fields appear inside the bundle:

- **2 bytes — Proof Length**
- **P bytes — Credit Proof (opaque)**
  Operator-signed blob. The operator's verification key fully defines what is valid.
- **1 byte — Denomination**
  Operator-defined integer or class.
- **8 bytes — Expiry Timestamp**
  Unix seconds or block-height based.
- **1 byte — Redeem Endpoint Length**
- **E bytes — Redeem Endpoint**

A transfer-bundle is treated as an **opaque object** outside the credit subsystem.
Clients attach it where needed; operators validate it only by verifying the embedded credit proof.

C. Message Layout Reference

HyperRelay Frame Envelope

- HyperRelay Tag - Derived from the receiver's discovery root and current epoch
- HyperRelay Hint - Contains:
  - Sender's ephemeral public key
  - Cryptographic confirmation hint
  - 1-byte metadata field indicating receiver action (e.g. fetch frame, hint-only event)

  The entire hint is encrypted under the same key material as the outer encryption layer, derived from the ephemeral ECDH secret, optional Kyber shared secret, and the current Tag Space ID.
- Encrypted Payload - The encrypted DataFrame, padded to fill the frame.

Encrypted DataFrame

- 2 bytes: Message length prefix.
- 16 bytes: Conversation ID.
- 32 bytes: Sender proof (HMAC).
- 32 bytes: Next view public key.
- 32 bytes: Current view public key.
- 16 bytes: Next discovery root (optional, only at handshake or trust update).
- N bytes: DataFrame payload:
  - Fragment Header (fixed size; see Section 6.2.1)
  - Fragment Body (slice of the logical message body; may be complete or partial)
- Padding: Random bytes to reach 983 bytes

Payload Layouts

The logical message body is defined over the concatenation of all Fragment Bodies belonging to the same (Conversation ID, Message ID, Message Type, Client ID) as described in Section 6.2.2. Once all fragments have been reassembled in order of Fragment Index, the resulting byte sequence is interpreted according to Message Type.

1. Text Message Payload
   - 1 byte: Version — Must be 1.
   - 2 bytes: Message Length — Big-endian uint16, specifies the number of bytes in the message.
   - N bytes: Message Content — UTF-8 encoded text.
   - 2 bytes — Transfer-Bundle Length
   - X bytes — Transfer-Bundle (if length > 0)
2. Handshake Request Payload
   - 1 byte: Version — Must be 1.
   - 1 byte: Message Length — Length in bytes of the introductory message (max 255).
   - 1568 bytes: Kyber Public Key — Sender's long-term PQ key for key exchange.
   - ≤255 bytes: Intro Message — UTF-8 encoded, human-readable invitation.
   - 2 bytes — Transfer-Bundle Length
   - X bytes — Transfer-Bundle (if length > 0)
3. Handshake Confirmation Payload
   - 1 byte: Version — Must be 1.
   - 2 bytes: Ciphertext Length — Big-endian uint16.
   - N bytes: Kyber Ciphertext — Output of post-quantum encapsulation.

D. Protocol Constants

- Ciphertext size: 983 bytes
- Filter tag size: 6 bytes (1-byte length prefix + 5 bytes tag data)
- Hint size: 4 bytes
- Encrypted semantics byte: 1 byte
- Ephemeral key size: 32 bytes
- Conversation ID: 16 bytes
- View key: 32 bytes (Curve25519)
- Kyber public key: 1568 bytes
- HMAC sender proof: 32 bytes
- Max message size (plaintext): determined by remaining bytes after headers and padding

E. Cryptographic Primitives

Unlink uses a hybrid encryption model combining conventional and post-quantum primitives:

- ECDH: Curve25519 for ephemeral Diffie-Hellman key exchange (outer layer encryption).
- Kyber1024: Post-quantum key encapsulation mechanism, used to derive a second shared secret for long-term resistance.
- ChaCha20-Poly1305: Symmetric encryption for both inner and outer layers.
- BLAKE3: Hashing function used for filter tag derivation and shared secret hinting.
- HMAC-SHA256: Used to compute sender proofs, binding the sender identity to the message and recipient.

All cryptographic operations are deterministic and versioned for future upgrades. Only minimal, proven algorithms are used to ensure performance, forward secrecy, and resilience against cryptographic deprecation.

F. Encrypted Semantics Byte Layout

The decrypted semantics byte is interpreted as a compact bitfield:
Major Type (Bits 7–4)

Defines the high-level message class. Values are advisory and application-extensible.

Example assignments:

- 0x0 — Standard message
- 0x1 — Status or receipt message
- 0x2 — Media or attachment indicator
- 0x3 — Session or key update
- 0x4 — Presence, typing, or ephemeral signal
- 0x5–0xE — Application-defined
- 0xF — Reserved

Size Class (Bits 3–1)

Provides a coarse bucket indicating the approximate number of fragments associated with the logical message. Buckets represent ranges rather than exact counts to avoid metadata leakage.

Suggested mapping:

- 000 — Single fragment
- 001 — 2–3 fragments
- 010 — 4–7 fragments
- 011 — 8–15 fragments
- 100 — 16–31 fragments
- 101 — 32–63 fragments
- 110 — 64–127 fragments
- 111 — 128+ fragments

Clients may use this information to estimate retrieval cost but must not assume exact sizes.

Fetch Recommendation (Bit 0)

- 0 — Fetch may be deferred
- 1 — Immediate fetch recommended

This bit is purely advisory and does not guarantee message urgency or importance.

Appendix G. Status Hint Payload Format

This appendix defines the canonical payload format for hint-only status and control updates.

G.1 Status Payload (13 bytes)

After authentication and decryption, the status hint payload has the following fixed layout:

- Byte 0 — Version
  Must be set to 1. Other values are reserved for future versions and must be ignored by current implementations.
- Byte 1 — Type
  Semantic type of the status update.
    - 0–127: protocol-reserved
    - 128–255: application-defined
- Byte 2 — Counter (ctr)
  Monotonic counter scoped to the referenced message's status stream. Used exclusively for replay and out-of-order protection.
- Bytes 3–6 — Timestamp (timestamp_sec)
  Unsigned 32-bit integer representing seconds. Encoding is implementation-defined (big-endian or little-endian) but must be consistent within an implementation.
- Bytes 7–12 — Data
  Six bytes of application-defined parameters. Interpretation depends on the type field.

G.2 Semantics and Security Considerations

The timestamp and data fields are advisory and must not be treated as authoritative for security or correctness decisions. Authenticity and acceptance are determined solely by successful authentication and replay-window validation.

Replay protection is enforced using the ctr field with a bounded, receiver-defined acceptance window. Implementations must reject duplicate or out-of-window counters.

Status hints are interpreted entirely at the semantic layer. They do not alter transport behavior, framing, padding, or observability, and they introduce no additional protocol-visible message classes.