Date: December 2025
Author: Joe, Unlink Protocol
Contact: joe@stampprotocol.org

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 a public blockchain (eg. Solana) without revealing sender-recipient relationships. The protocol operates without requiring global consensus or account linkage, instead leveraging short-lived view keys, ephemeral encryption contexts, and stateless filter tag matching.
While optimized for messaging, its architecture generalizes to any compact data exchange requiring anonymity, integrity, and efficient spam mitigation. Unlink supports near real-time communication and is designed to scale to over 100 million messages per day, with zero central dependencies.
Fees and relay costs are settled via anonymous prepaid credits, breaking the link between funding and usage while keeping the protocol stateless.

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.

Unlink provides infrastructure that remains resilient under these conditions: technically independent, resistant to coercive scanning requirements, and capable of operating even if centralized communication providers are limited or removed. By design, it preserves private communication as an open, permissionless capability rather than a privilege granted by intermediaries.

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 hundreds of millions 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 is permanently stored on-chain, 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

Imagine a giant public square where couriers, who do not know the senders, drop sealed letters into a single public pile. Each letter is written in a secret code, and even the address on the envelope is encoded. Only the intended recipient can recognize it as theirs, and the address changes with every message. Outsiders cannot link any two letters to the same person. Inside each letter is not just the message, but also the instructions for the new code to use in the next exchange.
This captures the core idea of Unlink: messages are indistinguishable from the outside, only recipients can discover and read their own, and conversations evolve securely without central coordination or exposed identities. Its architecture is guided by the following core principles:

3.1 Minimal On-Chain State

The protocol avoids unnecessary on-chain complexity. There are no user accounts, no contact lists, and no protocol-level conversations. All messages are encoded as standalone transactions. This stateless design enables horizontal scalability, simplifies auditing, and ensures that any Solana-compatible client can interpret and validate messages without relying on off-chain metadata or global state.

3.2 End-to-End Encryption

Unlink enforces strong cryptographic guarantees:

- **Ephemeral ECDH (Curve25519):** Used as the outer layer to ensure forward secrecy and low-latency decryption.
- **Post-Quantum Kyber Layer:** Used as the inner encryption layer to ensure quantum resistance against long-term ciphertext harvesting.
  All encryption is done client-side. The protocol never interacts with keys or plaintext.

3.3 Sender Unlinkability

Messages are unlinkable to sender identities on-chain. There are no on-chain signatures or authenticated accounts for message posting. Instead of paying fees directly, senders use anonymous, prepaid credits to cover costs. This removes any payment trail from the sender and further severs messaging activity from wallet identity.

3.4 Receiver Unlinkability

Only the receiver can recognize their own messages. There is no central delivery mechanism, clients must scan the public message pool to find what belongs to them. Discovery is two-step:

- **Filter tag (6 bytes):** Derived from discovery roots per epoch. Used as a bandwidth prefilter. Can be outsourced to untrusted matchers, who only forward candidate messages and learn nothing about identities.
- **Shared secret hint (4 bytes):** The client derives the ECDH secret from the message's ephemeral public key and its private view key, hashes it, and compares the first 4 bytes to the hint. Only on match does the client attempt decryption, saving CPU.

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 scale to 100 million+ messages per day without requiring centralized control or trusted infrastructure.

While capable of direct peer usage by high-performance clients, Unlink is *optimized for mobile environments* where background execution time, bandwidth, and compute are constrained. In these cases, clients rely on untrusted relayers that forward encrypted traffic without visibility into message contents, metadata, or recipient identity. These relayers are stateless and interchangeable, they merely assist with reachability, not trust or availability.

4. System Overview

Unlink is a decentralized protocol for pseudonymous message and data exchange, designed to operate entirely on-chain without relying on central servers or trusted intermediaries. The system revolves around lightweight encryption primitives, compact routing metadata, and minimal assumptions about client behavior or trust.

4.1 Participants

- **Sender**: Encrypts and broadcasts messages to the public (Solana) chain, including the necessary metadata to enable recipient-side matching.
- **Receiver**: Periodically scans or queries the chain for messages addressed to their current view key, decrypts matched content, and rotates keys for forward secrecy.
- **Matcher**: An optional third-party that receives a set of filter tags and assists in matching relevant messages for a receiver. Matchers cannot decrypt content or link tags to identities.
- **Payment & Transport Layer:** Unlink does not pay fees from user wallets. The sender redeems an anonymous prepaid credit with a relayer, which submits the Unlink transaction and covers all on-chain costs.

4.2 Trust Assumptions

- No party is trusted to preserve privacy or security.
- Only the holder of a private view key can recognize and decrypt matching messages.
- Even when relayers or matchers are used, they are blind to identities, message types, and contents.
- Post-quantum security depends on correct handshake participation and key rotation.

4.3 Message Lifecycle

1. **Derive**: The sender derives a shared secret, constructs the double-encrypted payload, and includes the new filter tag and view key.
2. **Match**: The receiver (or their matcher) compares incoming messages against a precomputed tag set.
3. **Deliver**: The ciphertext is decrypted using ECDH and Kyber-derived secrets. The new view key and tag are stored for future use.
4. **Reply**: The receiver sends a new message using the contact's filter tag and rotated keys, continuing the conversation in forward-secret form.

The protocol is state-progressing yet stateless: identity is preserved entirely through key rotation, and no on-chain addresses or accounts are ever revealed.

5. Core Concepts

Unlink defines a minimal, scalable, and secure messaging primitive with a clear separation between sender, receiver, and relayer roles. It combines hybrid encryption, anonymous addressing, and forward-secret state transitions to enable private, pseudonymous messaging and data transmission at scale.

5.1 Identity and Key Structure

Each user generates a 24-word mnemonic seed, from which the following keys are deterministically derived:

- **View Key Pair (Curve25519):** Used to derive shared secrets and receive messages.

For post-quantum security, each client also generates a **Kyber key pair**. This is created independently of the mnemonic, since it is not needed for the initial exchange.
The mnemonic can be stored to later recover the account. The protocol does not rely on on-chain identities. No accounts are registered or linked.

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. Handshake messages are indistinguishable from normal traffic and contain no identifiable metadata. Full details are described in Section 7.5.3 Dual-Layer Encryption
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.

5.4 Filter Tags and Message Discovery

Receivers discover messages through filter tags (section 8.1) published on-chain. Each tag is derived from a discovery root and rotates per epoch. The number of meaningful bits is configurable, allowing clients to trade bandwidth for anonymity, shorter tags produce more collisions and cover, longer tags reduce noise.
Clients generate only the tags required for their active conversations, but may add decoys to increase deniability. Matching is stateless and requires no on-chain registration. Relayers can assist with prefiltering by forwarding candidate messages, yet even with full access to tags and ciphertexts they cannot link them to a recipient identity.

5.5 Stateless Matching

Unlink is inherently stateless. Clients can receive messages without any on-chain identity or account, and the protocol is lightweight enough for constrained devices such as phones. Because matching relies only on rotating filter tags, not persistent state, backends can scale horizontally and remain untrusted.

5.6 Forward Secrecy and Required Key Rotation

Each message must include a new view public key for the next message, enforcing mandatory forward secrecy.

- Every message rotates the encryption key forward.
- A compromised key cannot decrypt past or future messages.
- Senders are never allowed to reuse view keys once rotated.

Clients must track and persist the conversation state locally.

6. Envelope and Payload Transmission

All Unlink messages are carried inside a compact, fixed-length on-chain envelope. Each envelope contains a fully encrypted 700-byte DataFrame that hides all payload structure, message type, and fragmentation details. This section describes how messages are framed, padded, encrypted, and segmented for transport. Specialized message flows such as handshakes are built using the same DataFrame format and are described later in Section 7.

6.1 DataFrame Structure (Inner Payload)

Messages begin as a structured data frame containing:

- **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 chunk header, followed by a chunk body, as specified in Section 6.2.

The DataFrame is padded with random bytes to exactly 700 bytes before encryption.
The entire Chunk Header and Chunk Body are contained within the DataFrame payload and are fully encrypted.

6.2 Message Fragmentation and Chunk Header

Unlink treats each on-chain message as a fixed-size transport frame carrying exactly one **chunk** of application data. Larger logical messages (for example, PQ handshakes) are split into multiple chunks that share a common identifier and are reassembled client-side after decryption.
The Payload field of the DataFrame (Section 6.1) is structured as:

- **Chunk Header**: Fixed-size metadata describing how this chunk relates to a larger logical message.
- **Chunk 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 Chunk Header Layout

Each chunk begins with the following header:

- **Version (1 byte)**
  Indicates the chunk 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 chunks.
  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 chunk header are the **Chunk Body**.

6.2.2 Fragmentation and Reassembly

Senders:

1. Construct the logical message body according to the Message Type (e.g., handshake request, handshake confirmation, or text message payload as defined in later sections).
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, Message Type, 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.

6.2.4 Single-Chunk Messages

Most ordinary text messages fit within a single chunk:

- Total Fragments = 1
- Fragment Index = 0

The receiver can process the message immediately.
Larger items, such as PQ handshake material, are split across multiple chunks but remain indistinguishable from single-chunk traffic on-chain thanks to the fixed envelope size and full DataFrame padding.

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)

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 and ECDH secrets
- the outer layer derives its encryption key from the ECDH secret alone

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. The nonce used for AEAD operations is shared between layers, but the keys are derived independently, ensuring clean separation and safe composition.

6.4 Envelope Layout (Outer Structure)

After encryption, the ciphertext is embedded in a fixed-length envelope for on-chain transmission:

- **Discriminator (1 byte)**: Indicates message type (e.g., message, fee update).
- **Ciphertext (700 bytes)**: The padded, double-encrypted DataFrame.
- **Ephemeral Public Key (32 bytes)**: Used to derive ECDH shared secret.
- **Shared Secret Hint (4 bytes)**: First bytes of hash of shared secret, enables matching.
- **Filter Tag (6 bytes)**: Compact identifier derived from the receiver's discovery root.

Total message size: 743 bytes
All fields are fixed-length, enabling uniformity and resistance to metadata inference.

6.5 Shared Secret Hint

Each message includes a 4-byte hint derived from the hash of the shared ECDH secret between sender and receiver. This hint allows clients to efficiently detect which messages are likely intended for them, without attempting full decryption.

- Upon encountering a message, the client derives the shared secret using the sender's ephemeral public key and its own private view key.
- It then hashes this secret and compares the first 4 bytes to the provided hint.
- A match indicates the message may be intended for the client, prompting full decryption.

Why 4 bytes?

- 4,294,967,296 total possibilities.
- At high throughput (hundreds of tx/s), the false positive rate per user remains negligible.
- Still efficient for mobile clients and relayers.

To obscure network behavior, clients can:

- Fetch full blocks instead of single transactions,
- Randomly include decoys in retrievals,
- Retrieve messages via Tor-based services or Tor-accessible RPCs to prevent network-level correlation.

6.6 Padding and Size Uniformity

To prevent analysis based on ciphertext length, all inner DataFrames are padded to a fixed 700 bytes:

- Prevents observers from inferring message size or type.
- Enforces uniformity across all message classes.

This consistency ensures messages, including handshakes, are indistinguishable on-chain.

6.7 Anti-Spam Mechanism

Unlink enforces a minimal, economic anti-spam mechanism using anonymous prepaid credits. These credits are included inside the handshake request payload as a transfer-bundle and are recovered by the recipient upon decryption. The recipient may reuse the credit, redeem it with the issuing operator, or discard it. Credits are anonymous bearer instruments; receiving a credit does not reveal sender identity.

Only *opening* messages require a fee. Established conversations proceed without additional credits.

Mechanism

- A sender initiating a handshake may include a credit by embedding a transfer-bundle in the handshake request payload.
- The bundle contains a denomination, expiry timestamp, redeem endpoint, and an opaque operator-signed proof (appendix B).
- After decrypting the handshake request, the receiver extracts the bundle and forwards it to an operator (typically the one indicated by the endpoint, or one they trust) for validation and rollover.

Receivers define a local minimum-fee threshold. A handshake request containing no credit or an insufficient denomination is ignored locally. This filtering happens entirely client-side and requires no on-chain state, no accounts, and no centralized lists.

Properties

- **Receiver-controlled:**
  Each receiver sets the minimum acceptable denomination for unsolicited contact. High-value targets may require higher amounts; ordinary users may allow low-cost or zero-cost contact.
- **Economic deterrence:**
  Unsolicited messages become costly because any handshake request that includes a credit transfers usable value to the target.
- **One-time requirement:**
  Credits are required only during the initial handshake. After two parties establish a conversation state, further messages require no credits beyond the relay's normal transaction cost.
- **Unlinkable and anonymous:**
  Credits are prepaid off-path and embedded as opaque proofs. Neither relays nor observers can associate credit usage with funding sources or user identities.

Local discovery exception

If a handshake is initiated through an offline or local channel (e.g., QR code, NFC, optical transfer), the initiating party may omit the credit bundle. In such cases, the receiver treats the request as trusted local contact, and no anti-spam fee is enforced.

Known-contact credit return

Once two parties have established a conversation state, either side may transfer credits back to the other by including a transfer-bundle in any subsequent message.

This allows:

- reciprocation of initial handshake costs,
- reimbursement of credits used during contact initiation, and
- optional cost-sharing models between trusted peers.

Returning credits is purely optional and does not affect the protocol’s state transition logic.

7. Handshake as a Special Transmission Type

The Unlink handshake is a specialized message flow that establishes mutual cryptographic context between two parties. It enables encrypted communication without any prior coordination or identity registration, preserving pseudonymity and unlinkability by default.

7.1 Overview

The handshake is a two-part process:

- **Invitation**: The initiator sends a handshake request containing their Kyber public key.
- **Confirmation**: The recipient replies with a Kyber ciphertext generated using the initiator's Kyber public key, enabling both sides to compute a shared post-quantum secret.

This exchange creates a mutual cryptographic state: a Curve25519 keypair per side (view keys), and a derived Kyber shared secret, which is used to enhance message encryption for post-quantum security.

Although logically a pair, the handshake spans multiple messages due to Kyber's large key sizes. However, on-chain these are indistinguishable from ordinary messages. Clients can obscure handshake structure using decoy chunks, shuffled message types, or randomized timing to preserve indistinguishability.

7.2 Initiation

To initiate a handshake, the sender transmits a standard message containing a chunk with type 0 (Handshake Request). The payload of this chunk encodes:

- **Version (1 byte)**: Indicates the version of the handshake payload structure.
- **Message Length Prefix (1 byte)**: Specifies the length of the optional introductory message, in bytes.
- **Kyber Public Key (1568 bytes)**: The sender's Kyber key, used by the receiver to perform encapsulation and derive a shared secret.
- **Introductory Message (up to 255 bytes)**: An optional, UTF-8 encoded message that may contain human-readable context or intent. Its length is constrained to a single byte (maximum 255) for efficiency and to discourage meaningful communication prior to quantum-safe encryption.

The receiver, upon accepting, will send a confirm message with the Kyber ciphertext, enabling derivation of the shared secret on both sides.

This structure ensures that handshake requests:

- Are indistinguishable from regular messages in on-chain structure and metadata.
- Can be relayed, filtered, and decrypted using the standard protocol pipeline.
- Embed no unencrypted identifiers or addresses, preserving unlinkability.

7.3 Confirmation

The handshake confirmation completes the key agreement by delivering the sender's encapsulated Kyber ciphertext to the recipient. This ciphertext is the output of a Kyber encapsulation operation using the recipient's Kyber public key, and allows both parties to derive a shared post-quantum secret.

The confirmation is embedded in a standard message using messageType = 1 (handshake confirm) and a dedicated payload structure:

- **Version (1 byte):** Indicates the format version (currently 1).
- **Ciphertext Length (2 bytes):** Specifies the length of the following Kyber ciphertext.
- **Kyber Ciphertext (1568 bytes):** The ciphertext from the sender's encapsulation operation.
- **Receiver Kyber Public Key Hash (32 bytes):** SHA-256 hash of the receiver's Kyber public key.

Upon receiving this confirmation, the recipient uses their Kyber private key to decapsulate and recover the shared Kyber secret. Both parties now hold a matching 32-byte post-quantum shared secret. This is combined with the existing 32-byte Curve25519 ECDH shared secret to derive a 64-byte encryption key: the Kyber secret forms the first half, and the ECDH output forms the second. This combined key is used for the inner layer of encryption, ensuring both forward secrecy and post-quantum resistance.

Like all messages, this confirmation is indistinguishable from any other on-chain message due to full encryption and standardized padding. The client is responsible for parsing the payload type and processing it accordingly.

7.4 Rotation and Rekeying

To maintain forward secrecy, every message exchanged after the handshake must rotate both the view public key and the associated filter tag. This rotation ensures that compromise of a single message key does not expose previous or future messages.

Additionally, clients may choose to periodically rotate the Kyber shared secret to further strengthen post-quantum security. This can be initiated by either party by embedding a new Kyber public key in a message (e.g., via a new handshake request with a different messageType), prompting the recipient to respond with a new confirmation. Since the protocol treats all messages uniformly at the transport level, this rekeying process remains indistinguishable from ordinary message exchange.

Key rotation strategies can be client-specific, depending on use case, threat model, and device constraints. However, the protocol enforces forward-secret view key rotation with every message to ensure baseline unlinkability and security.

7.5 Multiple Handshake Attempts

To account for unreliable network conditions or dropped messages, the protocol permits multiple handshake requests to be sent to the same recipient. These redundant attempts may include the same or a new Kyber public key.

Each handshake request is treated as an ordinary message at the protocol level, indistinguishable from others in terms of encryption, size, and layout. This ensures that failed or repeated handshakes do not leak metadata or intent.

Clients are expected to handle deduplication or confirmation tracking locally. A receiver may choose to respond to the first valid handshake request, ignore duplicates, or allow multiple confirmations depending on application logic.

This approach favors resilience over efficiency, accepting minimal redundancy to guarantee connection establishment without central coordination or session state.

7.6 Privacy and Deniability

The handshake process is designed to be indistinguishable from ordinary message transmission. Both the invitation and confirmation are encoded as standard messages, without any on-chain indication of their special purpose.

Key privacy properties include:

- **No On-Chain Linkage**: Handshake messages are encrypted and padded like any other, with no metadata revealing that a connection is being initiated.
- **Unobservable Handshake Size**: Handshakes span multiple messages due to the size of Kyber keys. To prevent adversaries from recognizing this pattern, clients should obscure it through padding, batching, or decoy traffic. For strong anonymity guarantees, the recommended approach is to route each request (both sending and receiving) through a fresh Tor circuit. This provides unlinkability at the network layer; other measures are partial mitigations.

- **View Key Unlinkability**: Each message includes a new public view key and therefore a new filter tag. This ensures that even a passive observer cannot correlate sequential messages between two parties.
- **No Mutual Acknowledgment Required**: A participant can initiate a handshake without the other party ever responding. This asymmetry avoids exposing intent or identity unless the receiver explicitly replies.
- **Offline Handshake Option:** If the handshake remains offline (e.g., via QR code scanning), the connection never touches the chain. A participant can passively observe their own message flow without producing any on-chain trace, eliminating any residual leakage.

In effect, the handshake blends into the broader flow of protocol messages, supporting strong deniability and resistance to analysis, even under extensive passive surveillance.

8. Filter Tag System

Unlink uses compact, prefix-based filter tags to support scalable message discovery without exposing recipient identities or social-graph structure. Each client maintains a fixed set of discovery roots that deterministically produce a stable set of tag prefixes for each epoch. Senders addressing a specific recipient do not choose prefixes; they use the single discovery root that the recipient explicitly provides for that relationship. Relayers observe only pseudorandom prefixes and cannot determine their meaning, origin, or relationship structure. Because tags appear at the top level of the message envelope, they are visible to all observers; however, the prefixes themselves are unlinkable without knowledge of the underlying discovery root. Only the intended recipient can derive and follow their tag set for a given epoch and identify which messages may belong to them.

8.1 Discovery Surface

Each user maintains a fixed discovery surface composed of:

- **16 personal unicast roots**
  Deterministically derived from the user's mnemonic seed. These roots are used for 1:1 and small-contact relationships.
- **Up to 16 multicast roots**
  Each multicast root is derived from a publisher's multicast discovery keypair. When a user subscribes to a multicast channel, they receive the publisher's multicast keypair, derive that channel's discovery roots, and select exactly one root to follow.

Together, this yields **32 logical roots per epoch** per user:

- 16 personal roots from the user's own seed.
- Up to 16 multicast roots from followed publishers.

Clients always maintain 32 slots; if fewer multicast channels are followed, unused slots are filled with decoy roots.

8.2 Prefix Derivation and Epoch Rotation

Discovery is anchored to Solana's block progression.
epoch = block_height mod 1500

For each root R, the client derives a per-epoch tag value:
tag_value_e = BLAKE3(R || e)

This is serialized as a fixed-size filter tag:

- **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

Two-epoch window
To avoid race conditions at epoch boundaries, clients always follow:

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

In total, each client tracks 64 active tags. This set is stable in size across all users and independent of how many contacts or channels they maintain.

8.3 Sender Routing

Unicast

When a recipient accepts a new contact, they reveal exactly one personal unicast root to the peer. The sender must:

- Derive the per-epoch tag from this root.
- Place all messages for that relationship under that tag.

Thus:

- The sender does not choose arbitrary prefixes.
- Each unicast relationship is bound to a single root at any given time.

Multicast

For multicast channels:

1. The publisher exposes a **multicast decryption keypair** and the associated multicast discovery roots (conceptually indexed 0–15 for that keypair).
2. Each subscriber deterministically derives the same 16 multicast roots from the provided keypair.
3. Each subscriber selects exactly **one** of those roots as their subscription root for that channel and adds the corresponding tags for e and e − 1 to their follow set (in a multicast slot).
4. The publisher encrypts each message once under the multicast decryption keypair and addresses it under a small, fixed set of multicast roots for that channel (e.g. one per index or per active audience cluster).

Subscribers on different multicast roots decrypt the same ciphertext but observe it under different prefixes. Relayers see only pseudorandom tags and cannot infer audience size or membership.

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 discovery-surface and activity-level inference:

- Clients always maintain **32 logical roots** per epoch (16 personal + 16 multicast slots).
- If fewer than 16 multicast channels are followed, unused slots are filled with decoy roots and synthetic tags.
- Clients may inject decoy retrievals and vary query patterns (e.g., fetching entire blocks or additional messages).

This keeps the observable number of active tags and the approximate traffic envelope uniform across users.

8.6 Privacy Properties

The filter tag system provides:

- **Fixed discovery surface:** Observers see 64 active tags per client (32 roots × 2 epochs), regardless of the number of contacts or channels.
- **Per-root isolation:** Compromise or deliberate disclosure of one root affects only the relationships or channels bound to that root.

- **Unlinkability across epochs:** Tags are derived as BLAKE3 outputs keyed by R || e. Without knowledge of R, prefixes from different epochs cannot be linked.
- **Multicast anonymity:** Subscribers to the same channel typically listen on different roots. Relayers cannot reconstruct the audience or its size from observed prefixes.

Overall, the filter tag design gives receivers scalable, private message discovery while preserving a uniform, protocol-wide footprint and leaving routing infrastructure fully untrusted.

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:

- A fixed domain separation prefix (to prevent cross-protocol misuse)
- the conversation ID,
- the receiver's view key,
- the next view key for forward secrecy,
- and 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., Solana RPC providers, indexers, or network monitors):
  Cannot correlate messages to identities, as the sender is never on-chain and the HMAC is unlinkable to any wallet or account.

- **Active matchers or malicious clients**:
  Cannot forge messages without the private view key. Even if acting as relayers or trying to inject spoofed traffic, all unauthorized messages will fail HMAC verification.
- **Key compromise and IP-linking**:
  A compromised private view key allows future impersonation, but not message decryption or past forgery. Clients must protect IP anonymity layers (e.g., Tor, relays), as metadata leakage could correlate a view key to a network identity.

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

- **Encryption Layers**: 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.
- **On-Chain Opaqueness**: 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 their filter tags 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 discovery keys and a fixed index range, used for efficient message matching without revealing recipient identity.
- 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

- Blockchain indexers, 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.

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 RPC providers, relayers, 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. Scalability & Performance

Unlink is engineered for extreme throughput and low-latency operation across a decentralized network, with no reliance on centralized relays or infrastructure. The protocol targets **100 million+ messages per day**, achieved through careful design of its message structure, filtering model, and execution cost on Solana.

12.1 On-Chain Efficiency

Each Unlink message is a single, fixed-size Solana instruction consuming approximately 25,000–40,000 compute units, depending on decryption paths and optional validation logic. This allows thousands of messages per block, with low variance and no program state bottlenecks. The protocol is entirely stateless: all necessary data (encryption, filtering, routing) is embedded within the message envelope. There is no need for account lookups, memory growth, or CPI calls.
Solana's high-throughput architecture enables parallel validation of independent messages across validators, making Unlink compatible with large-scale, asynchronous message flows without contention or centralized verification.

12.2 Matcher-Side Performance

Clients may scan on-chain messages themselves or offload the task to third-party matchers, untrusted nodes that filter messages using public filter tags and shared secret hints.
This offloading is efficient:

- Matchers only perform simple hash and prefix comparisons.
- They forward candidate message hints without knowing whether the client will actually accept them.
- With a modest setup, a matcher can handle millions of filter tag comparisons per second.

Because matching is stateless and read-only, relayers can scale horizontally and remain completely blind to message contents or identity mappings. This enables pluggable indexing infrastructure, including local clients, community-run matchers, or sponsor-funded relayers.

12.3 Filter Tag Set Size and Tradeoffs

Unlink uses a fixed discovery footprint: each client follows 32 discovery roots per epoch (16 personal and up to 16 multicast), doubled across the current and previous epoch for a stable set of 64 active tags. This uniform surface prevents observers from inferring contact volume or channel membership and keeps resource usage predictable.

The tunable parameter is the prefix length L (1–40 bits). Shorter prefixes enlarge the anonymity set but increase collision traffic; longer prefixes reduce collisions and lower bandwidth and CPU cost. Prefix length affects only collision domain size, not the number of tags.

After prefix filtering, the shared-secret hint provides a second, low-cost check. Only messages that match both the prefix and hint reach decryption. This two-stage process eliminates most irrelevant traffic early, keeping discovery efficient even at high global throughput and allowing Unlink to scale without overloading mobile or constrained devices.

12.4 Batching and Throughput

Messages are fixed-size, allowing efficient batching, indexing, and transport:

- Matchers and indexers can process entire blocks without schema variation.
- All messages are uniform in format and size, allowing hardware acceleration or WASM-based fast filters.

Unlink's architectural constraints make it well-suited for integration into bandwidth- and resource-constrained devices, including phones and embedded systems. Combined with client-managed encryption and filtering, the system avoids any performance bottleneck at the protocol level, supporting global-scale, real-time private communication.

13. Comparison to Existing Systems

In section 13.1 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.

13.1 Overview

Messenger	**Privacy**	**Metadata resilience**	**Linkability resilience**	**Post Quantum Secure**	**Scalability**	**Regulatory Resilience**	**Trust required**
Signal	Medium	Medium	Low	No	High	Low	Medium
Telegram	Low	Very low	None	No	High	Very low	Very high
WhatsApp	Medium	Low	Low	No	High	Very low	Very high
Matrix	Low	Low	Low	Partial	Medium	Low	High
Keet	Medium	Low	Low	No	Poor	Strong	Medium
Nostr DM	Low	Very low	None	No	Medium	Low	Medium
Session	Medium	Medium	Medium	No	Medium	Medium	Medium
Waku	Low	Low	None	No	Moderate	Low	High
Briar	High	High	High	No	Low	High	Low
Tox	Medium	Medium	Medium	No	Limited	Medium	Medium
Wire	Medium	Low	Low	No	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**

13.2 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.

13.3 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.

14. 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**
  Used only for user-to-user transfer. Ignored by operator during redemption.

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

Message Envelope

A complete Unlink message consists of the following fields in strict order:

- 1 byte: Discriminator — Identifies message type (e.g., message, fee update).
- 700 bytes: Ciphertext — Encrypted DataFrame, padded to fixed length.
- 32 bytes: Ephemeral public key — Used for ECDH shared secret derivation.
- 4 bytes: Shared secret hint — Truncated hash prefix of the ECDH-derived secret.
- 6 bytes: Filter tag — For recipient-side filtering.

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:
  - Chunk Header (fixed size; see Section 6.2.1)
  - Chunk Body (slice of the logical message body; may be complete or partial)
- Padding: Random bytes to reach 700 bytes

Payload Layouts

The logical message body is defined over the concatenation of all Chunk 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: 700 bytes
- Filter tag size: 6 bytes (1-byte length prefix + 5 bytes tag data)
- Hint size: 4 bytes
- 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.