QUBIC MESSENGER

A Decentralized End-to-End Encrypted Messaging Protocol

Built on the Qubic Blockchain

Version	0.1.0 — Testnet Preview
Authors	Nubi305
Repository	github.com/Nubi305/qubic-messenger
Network	Qubic Testnet
Date	February 2026
License	MIT Open Source

This document describes the architecture, cryptographic design, and implementation of Qubic Messenger — a feeless, pseudonymous, end-to-end encrypted messaging protocol built natively on the Qubic blockchain.

1. Abstract

Qubic Messenger is an open-source, decentralized messaging protocol that enables end-to-end encrypted (E2EE) communication using Qubic wallet identities as pseudonymous user accounts. Unlike traditional messaging applications that require phone numbers, email addresses, or centralized servers, Qubic Messenger uses the Qubic blockchain as a decentralized user registry and message metadata store.

Messages are encrypted client-side using X25519 Elliptic Curve Diffie-Hellman key exchange combined with XSalsa20-Poly1305 authenticated encryption, the same cryptographic primitives used by Signal and WhatsApp. Message content is never stored on-chain — only a BLAKE2b hash fingerprint is posted as cryptographic delivery proof. Message bodies are delivered peer-to-peer via libp2p GossipSub with IPFS as a fallback inbox for offline recipients.

The result is a messaging system where no company, server operator, or network observer can read messages, identify users, or censor communication — while maintaining cryptographic proof of delivery on the immutable Qubic ledger.

Traditional Messengers	Qubic Messenger
✓ Requires phone number or email	✓ Qubic wallet address only
✓ Messages stored on central servers	✓ Messages delivered peer-to-peer
✓ Company can read metadata	✓ Only hashes on-chain — no content
✓ Account can be banned or deleted	✓ Censorship-resistant by design
✓ Subject to government requests	✓ No personally identifiable data
✓ Fees for premium features	✓ 100% feeless on Qubic

2. Background & Motivation

2.1 The Privacy Problem

Modern messaging applications have a fundamental privacy problem: they require personally identifiable information to create an account. WhatsApp and Telegram require a phone number. Email-based services tie identity to an address. This creates a permanent link between a user's real identity and their communications.

Even applications that offer E2EE for message content — such as Signal — still collect metadata: who you talk to, when, and how often. This metadata is highly sensitive and can reveal social graphs, relationships, and behavioral patterns.

2.2 Why Qubic

Qubic is uniquely suited for a private messaging layer for several reasons:

- Feeless transactions — no gas costs for posting message metadata
- High throughput — Qubic processes millions of transactions per second
- Smart contract support — user registry and delivery proof on-chain
- Wallet-based identity — pseudonymous by design, no KYC required
- Open source — fully auditable, community-governed

2.3 Prior Art

Several projects have attempted decentralized messaging, including Status (Ethereum), Bitmessage (P2P broadcast), and Session (Session Network). Qubic Messenger differentiates itself by leveraging Qubic's feeless, high-speed infrastructure and combining it with modern E2EE primitives and a practical user experience.

3. Cryptographic Design

3.1 Key Generation

Each user generates a 32-byte X25519 keypair derived from their Qubic wallet seed using BLAKE2b-256. The public key is registered on-chain via the QubicMessenger smart contract. The private key never leaves the user's device.

```
encKeyPair = X25519.fromSeed(BLAKE2b(walletSeed + 'messenger-v1'))
```

3.2 Key Exchange

When Alice wants to message Bob, she performs an X25519 Diffie-Hellman key exchange using her private key and Bob's registered public key. This produces a shared secret that only Alice and Bob can derive. No third party — including relay nodes — can compute this shared secret without access to a private key.

```
sharedSecret = X25519.dh(alice.privateKey, bob.publicKey)
```

3.3 Message Encryption

Messages are encrypted using XSalsa20-Poly1305 (via TweetNaCl) with a random 24-byte nonce. The nonce is prepended to the ciphertext and transmitted with the message. The authentication tag provides integrity and authenticity — any tampering is detected during decryption.

```
ciphertext = XSalsa20Poly1305.encrypt(plaintext, nonce, sharedSecret)
```

3.4 On-Chain Delivery Proof

Only a BLAKE2b-256 hash of the ciphertext is posted to the Qubic smart contract as delivery proof. This allows recipients to verify that a message was sent at a specific tick without revealing any content or metadata about the message.

```
contentHash = BLAKE2b(ciphertext)[0:32] // First 32 bytes only
```

3.5 Key Rotation

Users can rotate their encryption keypair at any time. The new public key is posted to the smart contract with a version number. Senders check the latest key version before encrypting. Old messages remain readable with the old key, which is archived locally in the user's encrypted key store.

3.6 Group Chat Keys

Group chats use a shared 32-byte symmetric key encrypted individually for each member using their X25519 public key. When a member is removed, a new group key is generated and re-encrypted for remaining members (forward secrecy).

4. Smart Contract Architecture

4.1 Contract Overview

The QubicMessenger smart contract (index 25, asset name QMSG) is implemented in C++ using Qubic's QPI (Qubic Programming Interface). It provides three core functions: user registration, public key lookup, and message metadata storage.

4.2 User Registry

Users register by calling RegisterUser with their 32-byte X25519 public key and an optional 32-character nickname. The contract stores a mapping from Qubic address to public key, enabling any sender to look up a recipient's encryption key.

Function	RegisterUser(pubKey: id, nickname: char[32])
Storage	HashMap — up to 1M users
Fee	0 QUBIC (feeless)
On-chain	pubKey, nickname, registrationTick

4.3 Message Metadata

Senders post message metadata on-chain via PostMessageMeta. Only the content hash and recipient address are stored — never the message content itself. This provides delivery proof while preserving privacy.

Function	PostMessageMeta(recipient: id, contentHash: id, timestamp: uint64)
Storage	Ring buffer — last 10,000 messages per user
Fee	0 QUBIC (feeless)
On-chain	contentHash, sender, recipient, tick

4.4 Contract Limits

The contract is designed to operate within Qubic's 1GB state limit per contract:

- User registry: 1,000,000 users × 128 bytes = 128 MB
- Message metadata: 10,000 entries × 96 bytes = ~1 MB ring buffer
- Total estimated state: < 200 MB — well within limits

5. Message Delivery Architecture

5.1 Primary: P2P via libp2p GossipSub

When both sender and recipient are online, messages are delivered directly peer-to-peer using libp2p GossipSub. Each user subscribes to their own topic: qm/inbox/{address}. Messages are routed through the gossip network without passing through any central server.

5.2 Fallback: IPFS Offline Inbox

When a recipient is offline, the sender uploads the encrypted message blob to IPFS via Helia (in-browser IPFS node) and receives a Content Identifier (CID). The CID is posted to the Qubic contract as the contentHash. When the recipient comes online, they retrieve pending CIDs from the contract and fetch the corresponding blobs from IPFS.

5.3 Relay Infrastructure

A lightweight libp2p circuit relay server enables NAT traversal for peers behind firewalls. The relay server can be deployed on any $5/month VPS and handles up to 512 simultaneous relay slots. The relay never sees message content — it only routes encrypted packets.

5.4 Delivery Guarantee

The combination of P2P delivery, IPFS fallback, and on-chain hash verification provides a three-layer delivery guarantee: messages are either delivered in real-time, retrievable from IPFS within 24 hours, or verifiable as sent via the on-chain hash even if the IPFS node goes offline.

6. Token-Gated Features

Qubic Messenger uses a four-tier access model based on QUBIC balance, verified in real-time via the Qubic RPC. This creates a sustainable incentive for QUBIC holders while keeping basic messaging free for all users.

Free	0 QUBIC	Basic E2EE messaging, up to 10 conversations
Silver ■	1,000 QUBIC	Unlimited conversations, message history & search, custom avatar
Gold ■	10,000 QUBIC	Group chats (50 members), priority P2P routing, scheduling
Diamond ■	100,000 QUBIC	Create gated channels, verified badge, early access

Balance checks are cached for 60 seconds and verified against the Qubic testnet RPC. Gates are enforced both client-side (UX) and by the smart contract for on-chain operations such as creating gated channels.

7. Roadmap

Q1 2026 — Testnet	Smart contract deployment · P2P relay live · Web beta launch
Q2 2026 — Mobile	React Native iOS & Android apps · Push notifications · QR codes
Q3 2026 — Groups	Group chats · Token gating · Message reactions · Read receipts
Q4 2026 — Mainnet	IPO · Mainnet deployment · Audit · Key rotation v2
2027 — Ecosystem	SDK for third-party integrations · Oracle-verified delivery · DAO governance

8. Conclusion

Qubic Messenger demonstrates that private, feeless, censorship-resistant messaging is achievable on the Qubic blockchain today. By combining battle-tested cryptographic primitives (X25519, XSalsa20-Poly1305) with Qubic's unique feeless smart contract infrastructure and libp2p peer-to-peer networking, we have built a messaging protocol that is genuinely private by design — not just by policy.

The smart contract is deployed to testnet and the full open-source codebase is available at github.com/Nubi305/qubic-messenger. We welcome contributions, security reviews, and community feedback as we work toward mainnet deployment.

Private. Feeless. Yours.

github.com/Nubi305/qubic-messenger