Zixt: A Quantum-Resistant, Blockchain-Based Secure Messaging Platform with Zero-Knowledge Metadata Privacy

Whitepaper Author: Ryan Huff (@NetworkNerd1337) Version 1.0.2 April 19, 2025



Abstract

Zixt is a decentralized, secure messaging and authentication platform that leverages a custom blockchain, quantum-resistant cryptographic primitives, and advanced privacy mechanisms. This updated whitepaper introduces significant enhancements, including Lyra, a lattice-based zero-knowledge proof (ZKP) scheme for metadata privacy, DHT encryption with DTLS and SPHINCS+-based node authentication for secure peer discovery, and the use of CRYSTALS-Kyber, a quantum-resistant key encapsulation mechanism (KEM), for secure key exchange and message escrow. By combining SPHINCS+ for authentication, Kyber for key encapsulation, a multi-node consensus mechanism, and end-to-end encrypted messaging, Zixt offers a robust, privacy-focused platform resilient to classical and quantum threats.

1. Introduction

The rise of quantum computing and growing privacy concerns in digital communication demand solutions that ensure security, anonymity, and future-proofing. Zixt addresses these challenges by integrating a lightweight blockchain with post-quantum cryptography, privacy-preserving techniques, and secure key exchange mechanisms.

1.1 Problem Statement

- **Quantum Threats**: Quantum algorithms like Shor's threaten traditional cryptographic systems (e.g., RSA, ECC), while Grover's algorithm impacts symmetric-key systems, necessitating quantum-resistant alternatives.
- Metadata Privacy: Centralized messaging platforms expose metadata (e.g., sender, receiver, timestamps), enabling surveillance and communication pattern analysis.
- Centralization Risks: Centralized systems are prone to data breaches, surveillance, and single points of failure.
- **State Persistence**: Traditional server-based applications lose in-memory state during restarts, disrupting user experience.
- Network Security: Peer-to-peer communication in distributed systems, such as Distributed Hash Tables (DHTs), often lacks encryption and authentication.
- Key Exchange Vulnerability: Classical key exchange protocols (e.g., Diffie-Hellman) are vulnerable to quantum attacks, risking the security of end-to-end encrypted messages.

1.2 Zixt's Solution

Zixt provides:

- **Quantum-Resistant Security**: SPHINCS+ signatures for authentication, Lyra ZKPs for metadata privacy, and CRYSTALS-Kyber for secure key encapsulation.
- Metadata Anonymity: Zero-knowledge proofs hide message metadata while proving its validity.
- Decentralized Architecture: A custom blockchain for immutable user identities and message metadata.
- **State Persistence**: Multi-node synchronization with leader-based consensus for reliable state management.
- Secure Peer Discovery: DHT encryption with DTLS and SPHINCS+-based node authentication.
- Secure Key Exchange and Escrow: Kyber ensures quantum-safe key exchange for endto-end encryption and supports time-locked message escrow via smart contracts.
- Privacy and Control: Cryptographic pseudonyms and end-to-end encrypted messaging.

2. System Architecture

Zixt's architecture comprises a Flask-based user interface, a custom blockchain for data storage, a peer-to-peer network for state synchronization, and advanced cryptographic mechanisms for privacy, security, and key exchange.

2.1 Core Components

1. Blockchain:

• Stores user identities (public keys tied to pseudonyms), message metadata as Lyra ZKPs, and smart contracts for message escrow.

- Each block contains a header (previous block hash, timestamp, nonce) and a payload (user data, ZKP proofs, or escrow contracts).
- Blocks are cryptographically linked using SHA-256, with SPHINCS+ signatures for authenticity.

2. SPHINCS+ Integration:

- A stateless, hash-based signature scheme (NIST-standardized in 2022) used for user authentication, block signing, and node authentication in DHT.
- Key properties: quantum-resistant, deterministic, and stateless, avoiding state reuse vulnerabilities.

3. Lyra Zero-Knowledge Proofs:

- A lattice-based ZKP scheme for proving metadata validity without revealing sensitive details (e.g., sender, receiver, timestamp).
- Based on Learning With Errors (LWE), Lyra ensures quantum resistance and succinct proofs, suitable for blockchain storage.
- Non-interactive via the Fiat-Shamir transform, enabling efficient verification by nodes.

4. CRYSTALS-Kyber for Key Encapsulation:

- A lattice-based key encapsulation mechanism (KEM) standardized by NIST in 2022, used for quantum-safe key exchange and message escrow.
- **Zixt Quantum Key Exchange (ZQKE)**: Kyber enables cross-platform key exchange for end-to-end encryption, ensuring security against quantum attacks.
- **Message Escrow**: Kyber encrypts messages in time-locked smart contracts, allowing conditional delivery (e.g., "deliver on a specific date").
- Key properties: quantum-resistant, efficient, and standardized for post-quantum security.

5. Peer-to-Peer Network with DHT:

- Multiple nodes synchronize blockchain state using a Kademlia-based Distributed Hash Table (DHT) for peer discovery and block propagation.
- **DHT Encryption**: Communication is encrypted with Datagram Transport Layer Security (DTLS) to prevent eavesdropping.
- **Node Authentication**: Nodes authenticate each other using SPHINCS+ signatures, ensuring only trusted nodes participate.
- A leader-based consensus mechanism ensures agreement on the canonical chain, with nodes validating blocks using SPHINCS+ signatures and Lyra ZKP verification.

6. Flask UI:

- Provides a web interface for user registration, login, messaging, and escrow setup.
- Templates are modularized into separate HTML files for maintainability.
- SMTP integration enables email notifications for user actions.

7. Persistence Layer:

- A file-system cache stores blockchain data to optimize performance.
- Database-backed storage ensures uploaded files and user data persist across restarts.

2.2 Workflow

1. User Registration:

- Users generate a SPHINCS+ key pair and register a pseudonym.
- The public key and pseudonym are recorded on the blockchain.

2. Authentication:

- Users log in by signing a challenge with their SPHINCS+ private key.
- The server verifies the signature against the stored public key.

3. Key Exchange with Kyber (ZQKE):

- Users establish a shared symmetric key using CRYSTALS-Kyber for end-to-end encryption.
- The key exchange metadata (public keys, encapsulated secrets) is logged on the blockchain for auditability.

4. Messaging with ZKP:

- Messages are encrypted end-to-end using symmetric keys derived via Kyber.
- A Lyra ZKP is generated to prove metadata validity (e.g., sender/receiver exist, timestamp is valid) without revealing the metadata.
- The ZKP is stored on the blockchain, while the encrypted message content is stored off-chain.

5. Message Escrow with Kyber:

- Users can create time-locked smart contracts to escrow messages (e.g., "deliver on a future date").
- The message is encrypted with Kyber, and the contract stores the ciphertext, releasing the decryption key upon meeting the condition.

6. State Synchronization with DHT:

- Nodes use a Kademlia DHT to discover peers and propagate blocks.
- DHT communication is encrypted with DTLS, and nodes authenticate using SPHINCS+ signatures.
- The leader node resolves conflicts by selecting the longest valid chain.

3. Technical Implementation

3.1 Blockchain Design

The Zixt blockchain is a lightweight, permissioned ledger:

```
Block Structure:
•
•
   {
     "index": ,
•
     "previous hash": ,
•
•
     "timestamp": ,
     "nonce": ,
     "data": {
       "type": "user|message|escrow",
•
•
       "payload": {
          "pseudonym": ,
•
          "public key": ,
```

```
• "proof": ,
```

```
 "encrypted_message": ,
    "contract":
    }
  },
  "signature":
  }
```

- **Consensus**: Leader-based, with the leader elected based on node uptime and chain length.
- Validation: Nodes verify SPHINCS+ signatures, Lyra ZKPs, and Kyber-encrypted contracts.

3.2 SPHINCS+ Implementation

- Library: Sourced from liboqs-python.
- Key Generation:
- from sphincs import SPHINCS
- sphincs = SPHINCS()
- private_key, public_key = sphincs.generate_keypair()
- Signing and Verification:
- signature = sphincs.sign(message, private_key)
- is_valid = sphincs.verify(message, signature, public_key)

3.3 Lyra ZKP Implementation

- Library: Simulated in zkp_lyra.py using liboqs-python; real implementation requires C++ adaptation.
- Proof Generation:
- proof = lyra_zkp.generate_proof(
- statement="valid_user_to_valid_user_at_valid_time",
- witness={"sender": "alice", "recipient": "bob", "timestamp": 1625097600},
- public_inputs={"sender_exists": True, "recipient_exists": True,
- "timestamp_valid": True}
-)
- Verification:
- is_valid = lyra_zkp.verify_proof(proof, public_inputs)

3.4 CRYSTALS-Kyber Implementation

- Library: Sourced from liboqs-python, which includes Kyber as a NIST-standardized KEM.
- Zixt Quantum Key Exchange (ZQKE):
 - Users establish a shared symmetric key for end-to-end encryption.
- from oqs import Kem
- kyber = Kem("Kyber512") # Use Kyber512 variant
- pk_a, sk_a = kyber.keypair() # User A generates keypair
- shared_secret, ciphertext = kyber.encapsulate(pk_a) # User B encapsulates

- # User A decapsulates to get the same shared secret
- shared secret a = kyber.decapsulate(ciphertext, sk a)
- # Use shared_secret for AES encryption
- Message Escrow:
 - Messages are encrypted with Kyber and stored in a smart contract for conditional delivery.
- contract = {
- "recipient": "bob",
- "release date": "2026-04-17",
- "ciphertext": kyber.encapsulate(pk_bob)[1],
- "signature": sphincs.sign(ciphertext, private_key)
- •
- blockchain.add_contract(contract)

3.5 DHT with Encryption and Authentication

- Kademlia DHT: Used for peer discovery and block propagation.
- **DTLS Encryption**: Ensures confidentiality and integrity of DHT communication.
- SPHINCS+-Based Node Authentication:
- message = {"node id": "node1", "action": "lookup"}
- signature = sphincs.sign(json.dumps(message).encode(), private_key)
- if sphincs.verify(json.dumps(message).encode(), signature, public_key):
- process_message(message)

3.6 Flask Application

• Sample Route for Escrow:

- @app.route("/create_escrow", methods=["POST"])
- def create escrow():
- recipient = request.form["recipient"]
- message = request.form["message"]
- release date = request.form["release date"]
- pk_recipient = blockchain.get_user(recipient)["public_key"]
- ciphertext, shared secret =

```
kyber.encapsulate(bytes.fromhex(pk_recipient))
```

- contract = {
- "recipient": recipient,

```
• "release date": release date,
```

```
    "ciphertext": ciphertext.hex(),
```

• "signature": sphincs.sign(ciphertext, private key).hex()

```
• }
```

- blockchain.add_contract(contract)
- return jsonify({"status": "Escrow created"})

4. Security Features

• Quantum Resistance: SPHINCS+, Lyra, and Kyber protect against quantum attacks.

- Metadata Privacy: Lyra ZKPs hide sensitive metadata.
- Secure Key Exchange: Kyber ensures quantum-safe key exchange for end-to-end encryption.
- Immutability: Blockchain ensures tamper-proof records.
- End-to-End Encryption: Messages are encrypted with AES-256 using Kyber-derived keys.
- Secure Peer Discovery: DTLS encryption and SPHINCS+-based authentication protect DHT communication.
- Decentralization: Multi-node setup eliminates single points of failure.
- Auditability: Blockchain allows verification of ZKPs and escrow contracts without revealing sensitive data.

5. Use Cases

- Anonymous Messaging: Privacy-focused communication with hidden metadata.
- Identity Management: Decentralized, pseudonymous authentication.
- Data Integrity: Immutable logging for audit trails.
- Quantum-Safe Systems: Early adoption of post-quantum cryptography.
- Secure Peer Networks: Encrypted and authenticated peer discovery.
- **Conditional Messaging**: Time-locked message escrow for scheduled or conditional delivery.