# QuID: Quantum-Resistant Identity Protocol

## A Foundation for Extensible Decentralized Systems


### Abstract

The QuID (Quantum-resistant Identity Protocol) introduces a novel approach to creating persistent digital identities designed specifically for the post-quantum era. The protocol employs quantum-resistant cryptographic primitives to ensure long-term security against both classical and quantum attacks while maintaining an extensible architecture that allows for future capability expansion. By providing a robust identity layer with built-in quantum resistance, QuID serves as a foundation for developing secure decentralized social networks, cryptocurrency systems, and other distributed applications that need to remain secure in a post-quantum world.


### 1. Introduction

As quantum computing capabilities advance, the cryptographic foundations of current digital identity systems face increasing vulnerability. Modern decentralized systems require a strong foundation of digital identity that can withstand both present-day classical attacks and future quantum threats. Current solutions often lack comprehensive quantum resistance or are too rigidly designed for specific applications. QuID addresses these limitations by providing a protocol that is both quantum-resistant from the ground up and extensible to accommodate future use cases.

The key innovation of QuID lies in its consistent use of quantum-resistant primitives throughout the entire protocol stack, from basic operations like hashing to advanced features like digital signatures. This approach ensures that no component of the system becomes a weak link in the post-quantum security chain.

### 2. Design Goals

The protocol aims to achieve the following objectives:

2.1. Quantum Resistance: Ensure all cryptographic operations remain secure against attacks from both classical and quantum computers by exclusively using post-quantum cryptographic primitives.

2.2. Extensibility: Support the addition of new capabilities without requiring protocol modification, allowing the identity system to evolve with emerging needs.

2.3. Self-Sovereignty: Enable users to maintain complete control over their identities without relying on central authorities or trusted third parties.

2.4. Verifiability: Allow third parties to verify claims and extensions associated with identities using quantum-resistant verification mechanisms.

2.5. Privacy: Provide mechanisms for selective disclosure of identity attributes while maintaining quantum resistance.

## 3. Core Protocol Components

**3.1 Identity Generation** Each QuID identity consists of the following components:

```
Identity {
   id: SHAKE256(public_key || creation_timestamp)
   public_key: Bytes            // From ML-DSA
   private_key: Bytes          // From ML-DSA
   creation_timestamp: Uint64
   version: String
   metadata: Map<String, Bytes>
   extensions: Map<String, Extension>
}
```

The identity generation process uses ML-DSA (formerly CRYSTALS-Dilithium) for key generation, providing NIST-standardized post-quantum security. The unique identifier is derived using SHAKE256, which offers quantum resistance as part of the SHA-3 family.

**3.2 Extension Framework** Extensions are self-contained modules that add functionality to the base identity:

```
Extension {
   type: String
   data: Bytes
   signature: Bytes            // ML-DSA signature
   timestamp: Uint64
   version: String
   metadata: Map<String, Bytes>
}
```

All extensions must be signed using the identity's ML-DSA private key, creating a verifiable chain of ownership that remains secure against quantum attacks.

## 4. Protocol Operations

### 4.1 Identity Creation

1. Generate quantum-resistant keypair using ML-DSA
2. Record creation timestamp
3. Generate unique identifier using SHAKE256
4. Initialize empty extension and metadata stores
5. Perform quantum-resistant proof of possession

**4.2 Extension Management** Extensions are added through a defined process:

1. Serialize extension data using quantum-resistant encoding
2. Sign serialized data with identity's ML-DSA private key
3. Create extension structure
4. Add to extension store with quantum-resistant integrity protection

**4.3 Verification** The protocol provides quantum-resistant mechanisms for:

1. Identity ownership verification using ML-DSA
2. Extension authenticity verification
3. Claim verification with zero-knowledge proofs when needed

## 5. Security Analysis

**5.1 Quantum Security Considerations** All cryptographic primitives in QuID are selected to provide security against quantum attacks. The protocol uses:

1. ML-DSA for signatures (NIST standardized PQC)
2. SHAKE256 for hashing (quantum resistant)
3. ML-KEM for any required key encapsulation
4. Quantum-resistant authenticated encryption for data protection

**5.2 Security Levels** QuID supports multiple security levels corresponding to NIST security categories:

1. Level 1: 128 bits of quantum security
2. Level 3: 192 bits of quantum security
3. Level 5: 256 bits of quantum security

## 6. Implementation Guidelines

6.1 The protocol should be implemented using constant-time operations to prevent timing attacks

6.2 All random number generation must use quantum-resistant entropy sources

6.3 Implementation should use the liboqs library for standardized quantum-resistant primitives

6.4 Memory handling must be secure to prevent key material leakage

## 7. Future Considerations

7.1 Protocol upgrade paths for quantum-resistant algorithm transitions

7.2 Integration with quantum key distribution systems

7.3 Extension to support quantum-resistant distributed identity verification

## 8. Conclusion

QuID provides a robust foundation for building quantum-resistant decentralized systems that require strong identity guarantees. Its extensible architecture and comprehensive use of post-quantum cryptography make it suitable for long-term use in critical applications.

## 9. Network Protocol Specification

The QuID network layer ensures secure communication between identities while maintaining quantum resistance throughout the networking stack.

**9.1 Peer Discovery** The peer discovery mechanism uses a quantum-resistant Distributed Hash Table (DHT) based on the following principles:

**9.1.1 Bootstrap Process** Nodes enter the network through quantum-resistant bootstrap nodes. The bootstrap sequence: 1. New node generates temporary ML-KEM keypair 2. Connects to bootstrap nodes using ML-KEM for key encapsulation 3. Receives initial peer set encrypted with ML-KEM 4. Verifies peer signatures using ML-DSA

**9.1.2 DHT Structure** The DHT uses SHAKE256 for consistent hashing and network organization:

```
DHT_Node {
   node_id: SHAKE256(public_key)
   routing_table: Map<Distance, Vec<PeerInfo>>
   stored_records: Map<SHAKE256_Hash, Record>
}
PeerInfo {
   identity: QuID_Identity
   network_address: Vec<Address>
   last_seen: Uint64
   signature: ML-DSA_Signature
}
```

**9.1.3 Node Authentication** Each network interaction requires mutual authentication: 1. Nodes exchange ML-DSA signatures over session-specific challenges 2. Session keys established using ML-KEM 3. Ongoing communication encrypted using quantum-resistant authenticated encryption

**9.2 Message Routing**

**9.2.1 Overlay Network** The QuID overlay network provides: - Quantum-resistant path selection - Multi-path message routing - Traffic analysis resistance through padding and mixing

#### 9.2.2 Message Structure

```
Message {
   header: {
      version: String
      source_id: QuID_Identity
      destination_id: QuID_Identity
      timestamp: Uint64
      message_type: String
      routing_flags: Uint32
   }
   payload: EncryptedData {
   content: Bytes
   encryption_metadata: {
      algorithm: "ML-KEM-768"
      parameters: Map<String, Value>
      nonce: Bytes
   }
}
   signature: ML-DSA_Signature
}
```

**9.2.3 End-to-End Security** All messages use: - ML-KEM for initial key exchange - Quantum-resistant authenticated encryption for payload - Forward secrecy through frequent key rotation - Post-compromise security via ratcheting protocols

### 10. Key Management Procedures

#### 10.1 Key Backup and Recovery

**10.1.1 Social Recovery Protocol** QuID implements a t-of-n threshold scheme for key recovery:

```
Recovery_Share {
   shard_id: Uint32
   encrypted_key_data: {
      key_material: EncryptedData
      metadata: Map<String, Value>
      recovery_timestamp: Uint64
      verification_data: Bytes
   }
   guardian_signature: ML-DSA_Signature
}
```

The recovery process requires: 1. Collection of t valid recovery shares 2. Quantum-resistant share combination 3. Key reconstruction verification 4.

Extension key regeneration

10.1.2 Encrypted Backup Backup data is protected using: - ML-KEM for key protection - Quantum-resistant authenticated encryption - Tamper-evident sealing - Version control for recovery compatibility

10.2 Key Rotation

10.2.1 Scheduled Rotation The protocol mandates regular key rotation: 1. Generate new ML-DSA keypair 2. Create signed transition record 3. Update all dependent extensions 4. Propagate changes through network 5. Secure deletion of old keys

10.2.2 Emergency Rotation For compromised key scenarios: 1. Broadcast revocation certificate 2. Rapid new key generation and distribution 3. Extension re-signing with new keys 4. Recovery of encrypted data using backup mechanisms

## 11. System Requirements and Performance

### 11.1 Computational Requirements

**11.1.1 Processing Resources** Minimum specifications for different security levels: - Level 1 (128-bit): 2GHz processor, 1GB RAM - Level 3 (192-bit): 2.5GHz processor, 2GB RAM - Level 5 (256-bit): 3GHz processor, 4GB RAM

**11.1.2 Memory Patterns** Runtime memory requirements: - Key generation: 32-128MB temporary allocation - Signature operations: 16-64MB working set - Extension processing: Varies by type - Network operations: 64-256MB buffer space

**11.1.3 Storage Requirements** Persistent storage needs: - Core identity: 1-4KB - Extension data: Variable (typically 1-100KB per extension) - Network cache: Configurable (recommended 1-10GB) - Key backup: 16-64KB per backup set

### 11.2 Network Requirements

**11.2.1 Bandwidth Model** Expected bandwidth usage: - Node discovery: 1-5KB/s average - Extension synchronization: 0.1-1KB/s per active extension - Message routing: Dependent on usage (typically 1-10KB/s) - Key rotation: Burst traffic during rotation (approximately 10-50KB)

**11.2.2 Latency Considerations** Performance targets: - Local operations: <100ms - Network operations: <1s (95th percentile) - Key rotation: <5s complete propagation - Recovery operations: <30s for full recovery

**11.2.3 Scalability Characteristics** The system scales with: - O(log n) routing complexity - O(n) storage growth - O(m) extension cost (m = number of extensions) - O(k) key rotation cost (k = number of active connections)

## 12. Implementation Reference

### 12.1 Core Components

```
// Primary QuID implementation components
struct QuID_Implementation {
   identity_manager: IdentityManager,
   network_handler: NetworkHandler,
   extension_registry: ExtensionRegistry,
   key_manager: KeyManager,
   security_policy: SecurityPolicy
}

// Example initialization sequence
fn initialize_quid() -> Result<QuID_Implementation> {
    // Initialize with quantum-resistant entropy
    let entropy = quantum_resistant_random(32);

    // Create core components
    let implementation = QuID_Implementation {
        identity_manager: IdentityManager::new(entropy),
        network_handler: NetworkHandler::new(
            NetworkConfig::default_quantum_resistant()
        ),
        extension_registry: ExtensionRegistry::new(),
        key_manager: KeyManager::new(
            KeyConfig::high_security()
        ),
        security_policy: SecurityPolicy::strict_quantum_resistant()
    };

    // Verify quantum resistance of all components
    implementation.verify_quantum_resistance()?;

    // Initialize networking subsystem
    implementation.network_handler.start()?;

    // Set up extension handlers
    implementation.extension_registry.register_default_extensions()?;

    // Establish security monitoring
    implementation.security_policy.start_monitoring()?;
```

```
    Ok(implementation)
}
```

## 13. Threat Model Analysis

The security of the QuID protocol must be evaluated against both classical and quantum threats. This section provides a comprehensive analysis of the threat landscape and security boundaries of the system.

**13.1 Protected Attack Vectors** The protocol provides protection against the following attack vectors:

**13.1.1 Quantum Computing Attacks** QuID's security model assumes the presence of large-scale quantum computers. The protocol maintains security through: - Quantum-resistant signature schemes using ML-DSA - Post-quantum key encapsulation with ML-KEM - Quantum-resistant hash functions like SHAKE256 - Forward-secure messaging protocols resistant to retrospective quantum decryption

**13.1.2 Network-Level Attacks** The protocol defends against sophisticated network adversaries through: - Quantum-resistant authenticated encryption for all messages - Multi-path routing to prevent route analysis - Traffic padding and mixing to defeat timing analysis - Replay protection using strictly monotonic timestamps and nonces - Man-in-the-middle prevention through ML-DSA signature verification

**13.1.3 Identity and Extension Attacks** The system prevents tampering with identities and extensions via: - Unforgeable quantum-resistant signatures on all identity claims - Cryptographic binding between identities and their extensions - Version control and tamper-evident logging of all modifications - Zero-knowledge proofs for selective attribute disclosure - Strong revocation mechanisms for compromised identities

**13.1.4 Post-Compromise Security** Even after a temporary breach, the system provides: - Forward secrecy through frequent key rotation - Post-compromise security via ratcheting protocols - Secure recovery mechanisms through social backup - Ability to revoke and replace compromised credentials - Isolation of damage through extension compartmentalization

**13.2 Out of Scope Threats** The protocol explicitly does not protect against:

**13.2.1 Physical Security**

- Direct physical access to user devices

- Hardware-level tampering
- Side-channel attacks except timing attacks
- Compromised random number generators
- Physical observation of user input

#### 13.2.2 System-Level Security

- Operating system compromises
- Malware on user devices
- Memory scraping attacks
- Keyloggers and screen capture
- Privileged process interference

#### 13.2.3 Social Attacks

- Social engineering against users
- Phishing for recovery shares
- Coercion of key holders
- Insider threats in organizations
- Human operational security failures

**13.3 Security Assumptions** The security guarantees of QuID rely on the following assumptions:

#### 13.3.1 Cryptographic Assumptions

- The quantum security of ML-DSA signature scheme
- The quantum security of ML-KEM encapsulation
- The collision and preimage resistance of SHAKE256
- The security of the implemented zero-knowledge proof systems
- The quantum resistance of the authenticated encryption

#### 13.3.2 Implementation Assumptions

- Correct implementation of all cryptographic primitives
- Secure generation of random numbers
- Proper clearing of sensitive memory
- Constant-time operations where required
- Correct implementation of the protocol specification

#### 13.3.3 Operational Assumptions

- Users can securely store their private keys
- Recovery shares are distributed to trusted parties
- Network connectivity is generally available
- Users follow security best practices
- Systems running QuID maintain basic security hygiene

**13.4 Risk Mitigation Strategies** To address residual risks, the protocol recommends:

#### 13.4.1 Technical Controls

- Regular security audits of implementations
- Automated testing for cryptographic correctness
- Runtime verification of security properties
- Continuous monitoring for protocol violations
- Automated detection of potential compromises

#### 13.4.2 Operational Controls

- User education and security awareness
- Regular key rotation schedules
- Backup and recovery testing
- Incident response procedures
- Security logging and monitoring

#### 13.4.3 Governance Controls

- Clear security policies
- Regular threat model reviews
- Incident response plans
- Compliance monitoring
- Security metrics and reporting

This threat model should be regularly reviewed and updated as new attack vectors emerge and quantum computing capabilities advance.

### References

[1] Alagic G., et al., "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process", NISTIR 8413, July 2022

[2] Schwabe P., et al., "ML-DSA - A Deterministic Digital Signature Algorithm for Post-Quantum Applications", NIST FIPS 204 (Draft), 2024

[3] NIST Computer Security Division, "Post-Quantum Cryptography Standardization", 2024

[4] Bernstein D., et al., "Post-quantum cryptography—dealing with the fallout of physics success", September 2017, IACR Cryptology ePrint Archive 2017/314

[5] Bernstein D., Lange T., "Post-quantum cryptography", Nature, 549(7671):188-194, 2017

[6] Bindel N., et al., "Transitioning to a Quantum-Resistant Public Key Infrastructure", NIST NISTIR 8105, 2020

[7] Chen L., et al., "Report on Post-Quantum Cryptography", NISTIR 8309, 2023

[8] Regev O., "The Learning with Errors Problem", Proceedings of the 25th Annual IEEE Conference on Computational Complexity, 2010

[9] Open Quantum Safe Project, "liboqs: C library for quantum-safe cryptographic algorithms", Version 0.12.0, 2024

### Appendix A: Extension Interface Specification

The QuID extension interface provides a standardized way to add capabilities to identities while maintaining quantum resistance. This specification defines the requirements and interfaces for creating compatible extensions.

A.1 Extension Structure Definition

Extensions must implement the following interface:

```
ExtensionInterface {
    // Core Extension Properties
    type: String              // Unique identifier for extension type
    version: String           // Semantic version of extension
    data: EncryptedData {     // Quantum-resistant encrypted data structure
        payload: Bytes
        encryption_metadata: {
            algorithm: String  // Must be quantum-resistant
            parameters: Map<String, Value>
            nonce: Bytes
        }
    }

    // Quantum-resistant signature components
    signature: {
        algorithm: String     // Must be ML-DSA
        value: Bytes
        public_key: Bytes
        metadata: Map<String, Value>
    }

    // Extension metadata
    metadata: {
        created_at: Uint64
        updated_at: Uint64
        permissions: Map<String, Value>
        dependencies: Vec<DependencyInfo>
    }
}
```

### A.2 Extension Implementation Requirements

1. Cryptographic Requirements
   - All cryptographic operations must use quantum-resistant algorithms
   - Signatures must use ML-DSA with minimum security level matching the identity
   - Data encryption must use quantum-resistant authenticated encryption
   - Key derivation must use quantum-resistant key derivation functions
2. Data Handling
   - All sensitive data must be encrypted at rest
   - Clear memory after cryptographic operations
   - Implement secure deletion capabilities
   - Support selective disclosure mechanisms
3. Extension Lifecycle
   - Support version migration
   - Implement clean deactivation
   - Provide extension revocation mechanisms
   - Handle dependency management

### A.3 Standard Extension Types

#### A.3.1 Social Network Extension

```
SocialNetworkExtension {
    type: "social_network"
    data: {
        profile: {
            username: String
            display_name: String
            bio: String
        }
        connections: Vec<{
            identity_id: String
            relationship: String
            established_at: Uint64
            proof: ML-DSA-Signature
        }>
        posts: Vec<{
            content: EncryptedData
            timestamp: Uint64
            signature: ML-DSA-Signature
        }>
    }
}
```

#### A.3.2 Cryptocurrency Extension

```
CryptocurrencyExtension {
```

```
    type: "cryptocurrency"
    data: {
        wallet: {
            address: String
            public_parameters: Map<String, Bytes>
        }
        transactions: Vec<{
            tx_hash: SHAKE256-Hash
            timestamp: Uint64
            signature: ML-DSA-Signature
        }>
    }
}
```

**Appendix B: Security Proofs**

This appendix provides formal security proofs for the QuID protocol's core components.

B.1 Identity Uniqueness Theorem

Theorem 1: The probability of identity collision in QuID is negligible even against quantum adversaries.

Proof: Let A be a quantum adversary with access to a quantum computer with n qubits. The probability of finding a collision in the identity generation process is bounded by:

P(collision)  (q^2/2) * 2^(-min(|SHAKE256|, |ML-DSA-pk|))

where: - q is the number of quantum queries A can make - |SHAKE256| is the output length of SHAKE256 - |ML-DSA-pk| is the bit length of ML-DSA public keys

Given that both SHAKE256 and ML-DSA provide at least 128 bits of quantum security at their lowest security level, the probability remains negligible even against quantum attacks.

B.2 Extension Binding Security

Theorem 2: Extensions are unforgeable under chosen message attacks by quantum adversaries.

Proof: We reduce the unforgeability of extensions to the quantum security of ML-DSA. Assume a quantum adversary A can forge a valid extension with non-negligible probability . We construct an algorithm B that breaks ML-DSA with probability /q where q is the number of extension queries.

The reduction works as follows: 1. B receives an ML-DSA public key pk 2. B simulates the QuID protocol for A, using pk as the identity's public key 3.

B forwards A's signature queries to the ML-DSA signing oracle 4. When A outputs a forged extension, B extracts the signature

The success probability contradicts the proven security of ML-DSA against quantum adversaries, thus proving extension unforgeability.

B.3 Privacy Preservation

Theorem 3: The selective disclosure mechanism in QuID provides computational privacy against quantum adversaries.

Proof: The privacy of the selective disclosure mechanism relies on: 1. The quantum security of ML-KEM for key encapsulation 2. The quantum security of the authenticated encryption scheme 3. The zero-knowledge property of the proof system

We prove that an adversary with quantum capabilities cannot distinguish between real and simulated selective disclosures with non-negligible advantage, maintaining privacy even in a quantum setting.

These security proofs are presented for peer review and verification. They build upon the formal security definitions and proof techniques established in the quantum cryptography literature, particularly drawing from the security models presented in [1] and [3]. We encourage careful review and analysis of these proofs by the cryptographic community.

Note: Implementation of this protocol should undergo thorough security review and formal verification before deployment in critical systems.