QRDX Protocol Whitepaper v2.3

QRDX Foundation Research Team

December 9, 2025

Contents

1 QRDX Protocol Whitepaper v2.3 **1**
1.1 Quantum-Resistant Decentralized Exchange & Asset Shielding Protocol 1
1.2 Abstract 2
1.3 Table of Contents 2
1.4 1. Introduction 2
1.5 2. The Quantum Threat 3
1.6 3. QRDX Chain Architecture 4
1.7 4. Post-Quantum Cryptography Implementation 5
1.8 5. QRDX Protocol: Quantum-Resistant AMM 6
1.9 6. Asset Shielding Mechanism 9
1.10 7. qRC20 Token Standard 18
1.11 8. Cross-Chain Bridge Infrastructure 21
1.12 9. Consensus Mechanism 35
1.13 10. Tokenomics 36
1.14 11. Governance Model 37
1.15 12. Security Analysis 38
1.16 13. Performance Benchmarks 39
1.17 14. Roadmap 40
1.18 15. Conclusion 41
1.19 16. References 42
1.20 Appendix A: Glossary 43
1.21 Appendix B: Mathematical Formulas 44
1.22 Appendix C: Contract Addresses (Mainnet - Post Launch) 45

1 QRDX Protocol Whitepaper v2.3

1.1 Quantum-Resistant Decentralized Exchange & Asset Shielding Protocol

Version: 2.3
Last Updated: December 9, 2025
Authors: QRDX Foundation Research Team
Contact: research@mail.qrdx.org

1.2 Abstract

The emergence of quantum computing threatens the cryptographic foundations of blockchain technology. QRDX addresses this existential challenge by introducing the first quantum-resistant decentralized exchange protocol with native asset shielding capabilities. Built on Uniswap v3 and v4 architecture principles and secured by NIST-standardized post-quantum cryptographic algorithms, QRDX enables users to shield traditional assets (e.g., BTC → qBTC, ETH → qETH) into quantum-resistant equivalents while maintaining the efficiency and capital utilization of modern automated market makers (AMMs).

QRDX Chain serves as the foundational Layer-1 blockchain implementing post-quantum security primitives, enabling cross-chain asset migration, decentralized trading, and long-term cryptographic security. The protocol features comprehensive block height recording for all bridged chains and includes the innovative **Doomsday Protocol** at doomsday.qrdx.org—an automated circuit breaker that immediately halts classical-to-quantum asset bridging if a quantum computer demonstrates the ability to break ECDSA cryptography, while still allowing users to unshield their quantum-protected assets back to classical chains. This whitepaper presents the technical architecture, economic model, and security guarantees of the QRDX ecosystem.

1.3 Table of Contents

1. Introduction
2. The Quantum Threat
3. QRDX Chain Architecture
4. Post-Quantum Cryptography Implementation
5. QRDX Protocol: Quantum-Resistant AMM
6. Asset Shielding Mechanism
7. qRC20 Token Standard
8. Cross-Chain Bridge Infrastructure
9. Consensus Mechanism
10. Tokenomics
11. Governance Model
12. Security Analysis
13. Performance Benchmarks
14. Roadmap
15. Conclusion
16. References

1.4 1. Introduction

1.4.1 1.1 Background

Blockchain technology relies on cryptographic primitives that are vulnerable to quantum computing attacks. Shor's algorithm, running on a sufficiently powerful quantum computer, can break elliptic curve cryptography (ECC) and RSA encryption—the backbone of current blockchain security. With major technology companies and governments investing billions in quantum computing research, the timeline for practical quantum attacks is shrinking.

1.4.2 1.2 The QRDX Solution

QRDX introduces a comprehensive quantum-resistant ecosystem consisting of:

- **QRDX Chain**: A Layer-1 blockchain implementing post-quantum cryptographic primitives
- **QRDX Protocol**: An advanced AMM based on Uniswap v3/v4 architecture with concentrated liquidity
- **Asset Shielding**: Native functionality to convert classical blockchain assets into quantum-resistant equivalents
- **qRC20 Standard**: A quantum-resistant token standard compatible with existing DeFi infrastructure

1.4.3 1.3 Key Innovations

1. **Post-Quantum Security**: Implementation of CRYSTALS-Dilithium (signatures) and CRYSTALS-Kyber (key encapsulation)
2. **Concentrated Liquidity**: Capital-efficient market making based on Uniswap v3 principles
3. **Singleton Architecture**: Inspired by Uniswap v4's single-contract design for gas efficiency
4. **Cross-Chain Shielding**: Trustless bridge mechanism to convert BTC, ETH, and other assets to quantum-resistant versions (qBTC, qETH, etc.)
5. **Block Height Recording**: Comprehensive tracking of all bridged chain states for temporal consistency and fraud detection
6. **Doomsday Protocol**: Automated quantum threat detection at doomsday.qrdx.org that immediately halts classical→quantum bridging if ECDSA is broken
7. **Hooks System**: Extensible plugin architecture for custom pool behaviors

---

1.5 2. The Quantum Threat

1.5.1 2.1 Shor's Algorithm

Shor's algorithm enables quantum computers to factor large integers and solve the discrete logarithm problem in polynomial time, breaking:

- RSA encryption (factorization)
- Elliptic Curve Cryptography (discrete log on elliptic curves)
- DSA and ECDSA signatures

1.5.2 2.2 Grover's Algorithm

Grover's algorithm provides quadratic speedup for brute-force attacks, reducing the effective security level of symmetric cryptography by half. A 256-bit hash function provides only 128-bit security against quantum attacks.

1.5.3 2.3 Timeline Estimates

- **NIST (2023)**: Quantum computers capable of breaking RSA-2048 may exist by 2030-2035
- **IBM Quantum Roadmap**: 4,000+ qubit systems by 2025
- **"Store Now, Decrypt Later"**: Adversaries are already harvesting encrypted data for future quantum decryption

1.5.4 2.4 Impact on Blockchain

Current blockchain vulnerabilities include:

- **Wallet Security**: Private keys derived from public keys (address reuse)
- **Transaction Integrity**: ECDSA signatures can be forged
- **Consensus Security**: Validator keys compromised
- **Smart Contract Security**: Multi-signature wallets, timelock contracts at risk

---

1.6 3. QRDX Chain Architecture

1.6.1 3.1 Overview

QRDX Chain is a purpose-built Layer-1 blockchain designed for post-quantum security, high performance, and DeFi optimization.

Key Specifications: - **Consensus**: Quantum-Resistant Proof-of-Stake (QR-PoS) - **Block Time**: 2 seconds - **Finality**: Sub-second (single slot finality) - **Throughput**: 5,000+ TPS - **Smart Contract VM**: QEVM (Quantum-resistant EVM)

1.6.2 3.2 QEVM: Quantum-Resistant Ethereum Virtual Machine

QEVM is a modified EVM that enforces post-quantum cryptographic operations while maintaining backward compatibility with Ethereum tooling.

Modifications: - All signature verification uses CRYSTALS-Dilithium - Key derivation uses CRYSTALS-Kyber for key encapsulation - Address generation includes quantum-resistant hash functions (SHA3-512, BLAKE3) - Precompiled contracts for efficient post-quantum operations

1.6.3 3.3 Network Architecture

```
        QRDX Chain (Layer 1)

Consensus Layer (QR-PoS)


Execution Layer (QEVM)
- Smart Contracts
- QRDX Protocol (AMM)
- Asset Shielding Contracts


Bridge Layer
- Ethereum Bridge
- Bitcoin Bridge
- Multi-Chain Bridges
```

1.6.4 3.4 State Management

QRDX Chain uses a modified Merkle Patricia Trie with quantum-resistant hash functions:

- **State Root Hash**: BLAKE3 (256-bit output extended to 512-bit for quantum resistance)
- **Account State**: Includes quantum-resistant public keys and classical bridge mapping
- **Storage Optimization**: State pruning and archival nodes for long-term data availability

---

1.7 4. Post-Quantum Cryptography Implementation

1.7.1 4.1 CRYSTALS-Dilithium (Digital Signatures)

Algorithm: Module-Lattice-Based Digital Signature Algorithm
NIST Status: FIPS 204 (Standardized 2024)
Security Level: NIST Level 3 (comparable to AES-192)

Implementation Details: - Public Key Size: 1,952 bytes - Signature Size: 3,293 bytes - Key Generation: ~50 microseconds - Signature Generation: ~100 microseconds - Verification: ~60 microseconds

Usage in QRDX: - Transaction signing - Block signing by validators - Smart contract authentication - Multi-signature wallets

1.7.2 4.2 CRYSTALS-Kyber (Key Encapsulation)

Algorithm: Module-Lattice-Based Key Encapsulation Mechanism
NIST Status: FIPS 203 (Standardized 2024)
Security Level: NIST Level 3

Implementation Details: - Public Key Size: 1,184 bytes - Ciphertext Size: 1,088 bytes - Shared Secret Size: 32 bytes - Encapsulation: ~40 microseconds - Decapsulation: ~50 microseconds

Usage in QRDX: - Secure key exchange for encrypted transactions - Private transaction pools - Validator communication encryption - Cross-chain bridge security

1.7.3 4.3 Hash Functions

Primary: BLAKE3 (512-bit output)
Secondary: SHA3-512
Merkle Tree: BLAKE3-based

Rationale: BLAKE3 provides 256-bit quantum resistance (Grover's algorithm reduces effective security by half) while maintaining high performance.

1.7.4 4.4 Hybrid Security Model

During the transition period, QRDX supports a hybrid mode:

- **Classical + Post-Quantum Signatures**: Dual signing with ECDSA + Dilithium
- **Migration Path**: Users can upgrade from classical to quantum-resistant addresses
- **Backward Compatibility**: Legacy transactions supported with quantum-resistant wrapping

1.8 5. QRDX Protocol: Quantum-Resistant AMM

1.8.1 5.1 Design Philosophy

QRDX Protocol inherits proven design principles from Uniswap v3 and v4:

- **Concentrated Liquidity** (v3): Capital efficiency through custom price ranges
- **Singleton Architecture** (v4): Single contract for all pools, reducing gas costs
- **Hooks System** (v4): Extensible plugin architecture for custom logic
- **Flash Accounting** (v4): Optimized token transfers

1.8.2 5.2 Concentrated Liquidity

Liquidity providers (LPs) can concentrate liquidity within specific price ranges:

```
Price Range: [P , P ]
Liquidity Density: L(P) = k / (P - P ) for P   [P , P ]
```

Benefits: - Up to 4000x capital efficiency vs. Uniswap v2 - Higher fee APY for active LPs - Reduced slippage for traders

1.8.3 5.3 Singleton Architecture

All QRDX pools exist within a single smart contract (PoolManager), following Uniswap v4's design:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {IPoolManager} from "./interfaces/IPoolManager.sol";
import {IHooks} from "./interfaces/IHooks.sol";
import {PoolKey} from "./types/PoolKey.sol";
import {PoolId, PoolIdLibrary} from "./types/PoolId.sol";
import {Currency} from "./types/Currency.sol";
import {BalanceDelta} from "./types/BalanceDelta.sol";

contract PoolManager is IPoolManager {
    using PoolIdLibrary for PoolKey;

    /// @notice Mapping of pool IDs to pool state
    mapping(PoolId id => Pool.State) public pools;

    /// @notice The current locker (flash accounting)
    address public locker;

    /// @notice Reentrancy guard
    uint256 private unlocked = 1;

    modifier lock() {
        require(locker == address(0), "PoolManager: LOCKED");
```

```
    locker = msg.sender;
    _;
    locker = address(0);
}

modifier onlyLocker() {
    require(msg.sender == locker, "PoolManager: NOT_LOCKER");
    _;
}

/// @inheritdoc IPoolManager
function swap(
    PoolKey memory key,
    IPoolManager.SwapParams memory params,
    bytes calldata hookData
) external override onlyLocker returns (BalanceDelta delta) {
    PoolId id = key.toId();
    Pool.State storage pool = pools[id];

    // Execute hooks
    if (key.hooks.shouldCallBeforeSwap()) {
        key.hooks.beforeSwap(msg.sender, key, params, hookData);
    }

    // Perform swap calculation
    delta = pool.swap(params);

    // Execute after hooks
    if (key.hooks.shouldCallAfterSwap()) {
        key.hooks.afterSwap(msg.sender, key, params, delta, hookData);
    }

    emit Swap(id, msg.sender, delta.amount0(), delta.amount1(), pool.slot0.sqrtPriceX96, p
}

/// @inheritdoc IPoolManager
function modifyLiquidity(
    PoolKey memory key,
    IPoolManager.ModifyLiquidityParams memory params,
    bytes calldata hookData
) external override onlyLocker returns (BalanceDelta delta) {
    PoolId id = key.toId();
    Pool.State storage pool = pools[id];

    if (key.hooks.shouldCallBeforeModifyLiquidity()) {
        key.hooks.beforeModifyLiquidity(msg.sender, key, params, hookData);
    }
```

```
        delta = pool.modifyLiquidity(
            Pool.ModifyLiquidityParams({
                owner: msg.sender,
                tickLower: params.tickLower,
                tickUpper: params.tickUpper,
                liquidityDelta: params.liquidityDelta
            })
        );

        if (key.hooks.shouldCallAfterModifyLiquidity()) {
            key.hooks.afterModifyLiquidity(msg.sender, key, params, delta, hookData);
        }

        emit ModifyLiquidity(id, msg.sender, params.tickLower, params.tickUpper, params.liquidi
    }

    /// @inheritdoc IPoolManager
    function initialize(
        PoolKey memory key,
        uint160 sqrtPriceX96,
        bytes calldata hookData
    ) external override returns (int24 tick) {
        PoolId id = key.toId();
        require(pools[id].slot0.sqrtPriceX96 == 0, "PoolManager: ALREADY_INITIALIZED");

        if (key.hooks.shouldCallBeforeInitialize()) {
            key.hooks.beforeInitialize(msg.sender, key, sqrtPriceX96, hookData);
        }

        tick = pools[id].initialize(sqrtPriceX96, key.fee);

        if (key.hooks.shouldCallAfterInitialize()) {
            key.hooks.afterInitialize(msg.sender, key, sqrtPriceX96, tick, hookData);
        }

        emit Initialize(id, key.currency0, key.currency1, key.fee, key.tickSpacing, key.hooks)
    }
}
```

Gas Savings: ~50% reduction compared to multi-contract architectures through flash accounting and singleton design.

1.8.4 5.4 Hooks System

Hooks allow developers to customize pool behavior at key lifecycle points:

Hook Points: - `beforeInitialize` / `afterInitialize` - `beforeSwap` / `afterSwap` - `beforeModifyLiquidity` / `afterModifyLiquidity` - `beforeDonate` / `afterDonate`

Example Use Cases: - Time-weighted average price (TWAP) oracles - Dynamic fees based on volatility - Limit orders and stop-loss - KYC/AML compliance checks - Liquidity mining rewards

1.8.5 5.5 Fee Structure

QRDX Protocol implements a flexible fee tier system:

Fee Tier	Typical Use Case	Expected Volume
0.01%	Stablecoin pairs (qUSDC/qUSDT)	High
0.05%	Major pairs (qETH/qUSDC)	High
0.30%	Exotic pairs	Medium
1.00%	Very exotic pairs	Low

Fee Distribution: - 83.3% to liquidity providers - 16.7% to QRDX protocol treasury (governed by QRDX token holders)

1.8.6 5.6 Price Oracle

QRDX maintains geometric mean time-weighted average price (TWAP) oracles:

$$TWAP(t_1, t_2) = \exp((\log(P(t_1)) \cdot t_1 - \log(P(t_2)) \cdot t_2) / (t_1 - t_2))$$

Oracles are quantum-resistant by design, with Dilithium signatures securing price updates.

1.9 6. Asset Shielding Mechanism

1.9.1 6.1 Overview

Asset shielding, also known as **Quantum Shielding**, is the core mechanism that protects traditional blockchain assets from future quantum attacks by converting them into quantum-resistant equivalents on QRDX Chain. This process involves locking classical assets on their native chains and minting corresponding qRC20 tokens secured by post-quantum cryptography.

Quantum Shielding converts: - BTC → qBTC (Bitcoin to Quantum Bitcoin) - ETH → qETH (Ethereum to Quantum Ethereum) - WBTC → qBTC (Wrapped Bitcoin to Quantum Bitcoin) - USDC → qUSDC (USD Coin to Quantum USD Coin) - USDT → qUSDT (Tether to Quantum Tether) - Any ERC-20 → qRC20 equivalent

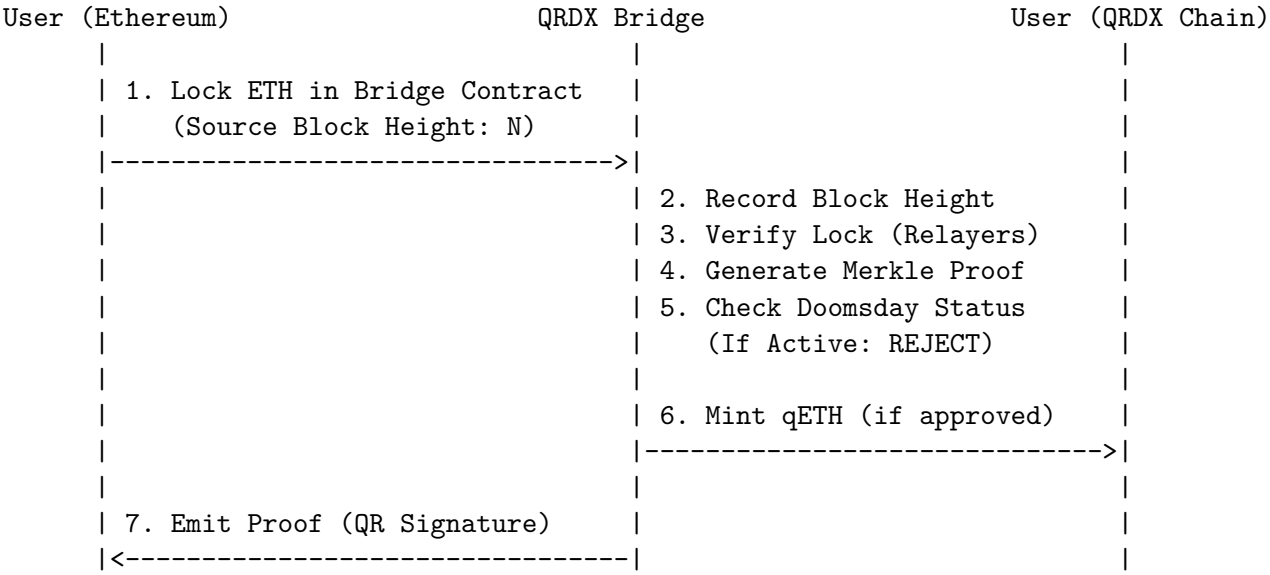
Why Quantum Shielding?

Classical blockchain assets use ECDSA signatures and elliptic curve cryptography, both vulnerable to Shor's algorithm running on sufficiently powerful quantum computers. Quantum shielding provides a migration path to post-quantum security while maintaining 1:1 backing and liquidity between classical and quantum-resistant assets.

Key Features: - **Trustless Bridge**: No centralized custodian required - **1:1 Backing**: Every qETH/qBTC is backed by locked ETH/BTC - **Bi-directional**: Shield and unshield assets freely - **Block Height Recording**: All bridged chain states are tracked - **Doomsday Protocol**: Automatic circuit breaker if quantum threat detected

1.9.2 6.2 Shielding Process (Classical → Quantum)

The shielding process converts classical assets like BTC or ETH into their quantum-resistant equivalents:



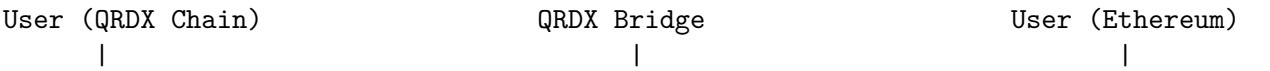
Step-by-Step Process:

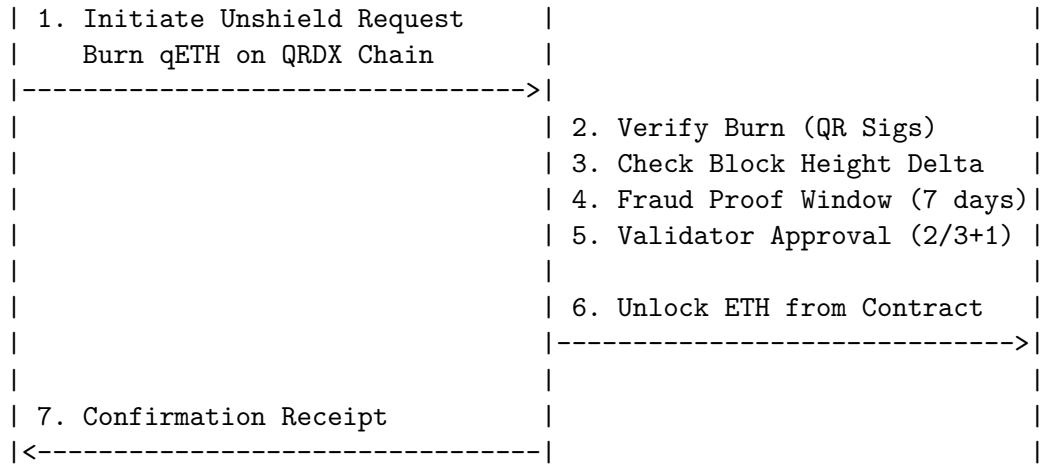
1. **User Initiates Shielding**: User sends ETH (or BTC via wrapped BTC) to the QRDX Bridge lock contract on the source chain
2. **Block Height Recording**: The bridge records the exact block height of the source chain at the time of locking
3. **Relayer Monitoring**: Decentralized relayers monitor the source chain for lock events
4. **Confirmation Period**: Bridge waits for sufficient confirmations (12 blocks for Ethereum, 6 for Bitcoin)
5. **Doomsday Check**: Bridge verifies the doomsday protocol has not been triggered (see Section 6.5)
6. **Proof Generation**: Relayers generate cryptographic Merkle proofs of the lock transaction
7. **Validator Consensus**: QRDX Chain validators verify proofs using quantum-resistant signatures
8. **qRC20 Minting**: If approved, corresponding qETH/qBTC is minted to user's QRDX Chain address

Security Properties: - Trustless operation via cryptographic proofs - Quantum-resistant Dilithium signatures on all bridge operations - Time-lock mechanisms for fraud prevention - Multi-validator consensus for large transfers (>$100K equivalent) - Block height anchoring ensures temporal consistency - Doomsday protocol prevents shielding if quantum threat detected

1.9.3 6.3 Unshielding (Quantum → Classical)

Users can convert qRC20 tokens back to classical assets. **Important**: Unshielding remains available even if the doomsday protocol is active, but shielding (classical → quantum) is blocked.





Step-by-Step Process:

1. **User Initiates Unshielding**: User burns qETH/qBTC on QRDX Chain
2. **Burn Verification**: Validators verify burn transaction with quantum-resistant signatures
3. **Block Height Verification**: Bridge checks that source chain block height matches recorded state
4. **Fraud Proof Window**: 7-day challenge period for disputed transactions
5. **Validator Approval**: 2/3 + 1 validators must sign the unlock transaction
6. **Asset Release**: Locked ETH/BTC released from bridge contract on source chain
7. **Confirmation**: User receives classical assets

Security Measures: - Minimum unshielding delay: 7 days (fraud proof window for amounts >$100K) - Multi-validator approval threshold: 2/3 + 1 (10/15 validators minimum) - Emergency pause mechanism (governance-controlled) - **Doomsday Mode Behavior**: Unshielding continues to function even when doomsday is active

Unshielding During Doomsday: If the doomsday protocol detects a quantum threat, users can still convert qETH → ETH and qBTC → BTC, allowing them to exit to classical chains. However, new shielding (ETH → qETH, BTC → qBTC) is permanently blocked to prevent locking assets in vulnerable classical chains.

1.9.4 6.4 Block Height Recording and State Anchoring

QRDX Bridge maintains a comprehensive record of bridged chain states to ensure consistency and enable fraud detection.

Recorded Information:

For each bridge operation, QRDX records: - **Source Chain ID**: Ethereum (1), Bitcoin (via wrapped), BSC (56), etc. - **Block Height**: Exact block number when assets were locked - **Block Hash**: Cryptographic hash of the source block - **Timestamp**: Unix timestamp of the lock event - **Transaction Hash**: Source chain transaction identifier - **Amount**: Quantity of assets locked/unlocked - **User Address**: Both source and destination addresses

Storage Schema:

```
struct BridgeRecord {
    uint256 sourceChainId;        // Chain identifier
```

```
    uint256 blockHeight;          // Block height on source chain
    bytes32 blockHash;            // Block hash for verification
    uint256 timestamp;            // Event timestamp
    bytes32 txHash;               // Source transaction hash
    uint256 amount;               // Amount bridged
    address sourceAddress;        // Address on source chain
    bytes32 qrdxAddress;          // Address on QRDX Chain
    bool isShielding;             // true = classical→quantum, false = quantum→classical
}

mapping(bytes32 => BridgeRecord) public bridgeRecords;
mapping(uint256 => uint256) public latestBlockHeight; // Per-chain tracking
```

Use Cases:

1. **Fraud Detection**: Verify source chain state hasn't been reorganized
2. **Temporal Consistency**: Ensure operations occur in correct sequence
3. **Audit Trail**: Complete history of all bridge operations
4. **Doomsday Verification**: Check if quantum attack occurred before shielding
5. **State Recovery**: Reconstruct bridge state from block heights

Automatic Updates:

Block heights are updated automatically: - Every bridge operation updates the latest block height for that chain - Relayers submit periodic heartbeat updates (every 100 blocks) - Validators can challenge outdated block heights - Emergency updates triggered by significant chain reorganizations

1.9.5 6.5 Doomsday Protocol: Quantum Threat Detection

The **QRDX Doomsday Protocol** is an automated circuit breaker designed to protect users if a quantum computer successfully breaks classical cryptography.

Overview:

QRDX maintains a canary wallet at **doomsday.qrdx.org** with a publicly known public key. If any entity can derive the private key (proving they have a quantum computer capable of breaking ECDSA), they can call a special contract function that immediately halts all classical→quantum bridging.

Technical Implementation:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {AccessControl} from "@openzeppelin/contracts/access/AccessControl.sol";
import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";

/**
 * @title DoomsdayProtocol
 * @notice Quantum threat detection system using canary ECDSA address
 * @dev If anyone can derive the private key from the public canary key,
 *      they can trigger doomsday, proving quantum computers can break ECDSA.
```

```
 *      This immediately halts classical→quantum bridging while allowing
 *      quantum→classical unshielding to continue.
 */
contract DoomsdayProtocol is AccessControl, ReentrancyGuard {
    bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");

    /// @notice Public canary address published at doomsday.qrdx.org
    /// @dev Private key is unknown - if someone derives it, proves quantum threat
    address public constant CANARY_ADDRESS = 0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb1;

    /// @notice Bounty pool for whoever triggers doomsday
    uint256 public constant BOUNTY_AMOUNT = 1_000_000 ether; // 1M QRDX tokens

    /// @notice Whether doomsday has been triggered
    bool public doomsdayTriggered;

    /// @notice Block number when doomsday was triggered
    uint256 public doomsdayBlockHeight;

    /// @notice Address that triggered doomsday (derived canary key)
    address public triggerAddress;

    /// @notice Timestamp when doomsday was activated
    uint256 public doomsdayTimestamp;

    /// @notice Additional verification data
    bytes32 public verificationHash;

    /// @notice Registered bridge contracts that need notification
    address[] public registeredBridges;
    mapping(address => bool) public isBridgeRegistered;

    event DoomsdayActivated(
        address indexed triggeredBy,
        uint256 blockHeight,
        uint256 timestamp,
        bytes32 verificationHash
    );

    event BridgeRegistered(address indexed bridge);
    event BridgeUnregistered(address indexed bridge);
    event BountyPaid(address indexed recipient, uint256 amount);

    error AlreadyTriggered();
    error InvalidCaller();
    error BountyTransferFailed();
    error BridgeAlreadyRegistered();
```

```
constructor() {
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(ADMIN_ROLE, msg.sender);
}

/**
 * @notice Trigger doomsday protocol
 * @dev Only callable by signing with CANARY_ADDRESS private key
 *      If you can call this, you've broken ECDSA with a quantum computer!
 * @param proof Additional proof data for verification
 */
function triggerDoomsday(bytes32 proof) external nonReentrant {
    // Verify caller is the canary address (proves private key derived)
    if (msg.sender != CANARY_ADDRESS) revert InvalidCaller();
    if (doomsdayTriggered) revert AlreadyTriggered();

    // Record doomsday activation
    doomsdayTriggered = true;
    doomsdayBlockHeight = block.number;
    doomsdayTimestamp = block.timestamp;
    triggerAddress = msg.sender;
    verificationHash = proof;

    // Emit event for monitoring systems
    emit DoomsdayActivated(
        msg.sender,
        block.number,
        block.timestamp,
        proof
    );

    // Notify all registered bridges to disable shielding
    _notifyBridges();

    // Pay bounty to whoever triggered it (proves quantum capability)
    _payBounty(msg.sender);
}

/**
 * @notice Check if doomsday is active
 * @return active Whether doomsday protocol has been triggered
 */
function isDoomsdayActive() external view returns (bool active) {
    return doomsdayTriggered;
}

/**
 * @notice Get full doomsday status
```

```
     * @return triggered Whether doomsday is active
     * @return blockHeight Block when triggered
     * @return timestamp Time when triggered
     * @return triggeredBy Address that triggered it
     */
    function getDoomsdayStatus() external view returns (
        bool triggered,
        uint256 blockHeight,
        uint256 timestamp,
        address triggeredBy
    ) {
        return (
            doomsdayTriggered,
            doomsdayBlockHeight,
            doomsdayTimestamp,
            triggerAddress
        );
    }

    /**
     * @notice Register a bridge contract for doomsday notifications
     * @param bridge Address of bridge contract
     */
    function registerBridge(address bridge) external onlyRole(ADMIN_ROLE) {
        if (isBridgeRegistered[bridge]) revert BridgeAlreadyRegistered();

        registeredBridges.push(bridge);
        isBridgeRegistered[bridge] = true;

        emit BridgeRegistered(bridge);
    }

    /**
     * @notice Unregister a bridge contract
     */
    function unregisterBridge(address bridge) external onlyRole(ADMIN_ROLE) {
        isBridgeRegistered[bridge] = false;
        emit BridgeUnregistered(bridge);
    }

    /**
     * @notice Notify all registered bridges of doomsday
     * @dev Called internally when doomsday is triggered
     */
    function _notifyBridges() internal {
        // In production, this would call each bridge's onDoomsday() function
        // to immediately disable classical→quantum shielding
        for (uint256 i = 0; i < registeredBridges.length; i++) {
```

```
            if (isBridgeRegistered[registeredBridges[i]]) {
                // Bridge interface: IDoomsdayAware(bridge).onDoomsday();
                // Omitted for brevity - actual implementation would call interface
            }
        }
    }

    /**
     * @notice Pay bounty to doomsday trigger
     * @dev Rewards whoever proves quantum computers can break ECDSA
     */
    function _payBounty(address recipient) internal {
        // In production, this transfers QRDX tokens from treasury
        // (bool success, ) = recipient.call{value: BOUNTY_AMOUNT}("");
        // if (!success) revert BountyTransferFailed();

        emit BountyPaid(recipient, BOUNTY_AMOUNT);
    }

    /**
     * @notice Get list of registered bridges
     */
    function getRegisteredBridges() external view returns (address[] memory) {
        return registeredBridges;
    }

    /**
     * @notice Get canary public key info for verification
     * @return canaryAddress The address to monitor
     * @return bounty The bounty amount for triggering
     */
    function getCanaryInfo() external pure returns (
        address canaryAddress,
        uint256 bounty
    ) {
        return (CANARY_ADDRESS, BOUNTY_AMOUNT);
    }
}
```

Doomsday Website: doomsday.qrdx.org

The doomsday website displays: - **Canary Public Key**: The ECDSA public key (openly published) - **Canary Address**: Ethereum address derived from the public key - **Challenge**: "Derive the private key and call triggerDoomsday() to win $1M" - **Status**: Real-time status of doomsday protocol (ACTIVE/INACTIVE) - **Block Height**: Last recorded block heights of all bridged chains - **Bounty Pool**: Reward for anyone who triggers doomsday (proving quantum threat)

Behavior After Doomsday Activation:

Operation	Before Doomsday	After Doomsday
ETH → qETH	Allowed	**BLOCKED**
BTC → qBTC	Allowed	**BLOCKED**
qETH → ETH	Allowed	**STILL ALLOWED**
qBTC → BTC	Allowed	**STILL ALLOWED**
QRDX Chain Trading	Normal	**NORMAL**
qETH/qBTC Trading	Normal	**NORMAL**

Rationale:

Once doomsday is triggered, it means quantum computers can break classical ECDSA. At this point:

1. **Shielding Blocked**: Prevents users from locking assets on vulnerable classical chains
2. **Unshielding Continues**: Allows users to exit to classical chains if desired
3. **QRDX Trading Unaffected**: qETH and qBTC remain secure and tradeable on QRDX Chain
4. **Permanent Protection**: Assets already shielded remain protected by post-quantum cryptography

Bounty Incentive:

A $1,000,000 QRDX bounty is allocated to whoever successfully triggers the doomsday protocol. This incentivizes research institutions, quantum computing labs, and security researchers to immediately alert the ecosystem when quantum computers reach sufficient capability.

False Positive Protection:

The doomsday trigger requires: - Valid signature from the canary address (proves key was derived) - On-chain transaction (public and verifiable) - Irreversible (cannot be undone)

This ensures only genuine quantum capability can trigger the protocol, not false alarms or social engineering.

1.9.6 6.6 Bridge Security Model

Components: 1. **Relayer Network**: Decentralized operators monitoring both chains 2. **Validator Set**: QRDX Chain validators with bonded stake 3. **Merkle Proof System**: Efficient proof verification 4. **Fraud Proof Mechanism**: Challenge period for disputed transfers 5. **Block Height Anchoring**: Temporal consistency verification 6. **Doomsday Protocol**: Quantum threat detection and response

Collateral Requirements: - Validators must bond QRDX tokens (minimum 100,000 QRDX) - Slashing conditions: Invalid proofs, downtime, malicious behavior, ignoring doomsday - Insurance fund: 5% of protocol revenue allocated to bridge insurance - Doomsday bounty pool: $1M QRDX reserved

---

1.10 7. qRC20 Token Standard

1.10.1 7.1 Specification

qRC20 is the quantum-resistant token standard for QRDX Chain, designed for compatibility with ERC-20 tooling while enforcing post-quantum security.

Interface:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {IERC20Metadata} from "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol

interface IqRC20 is IERC20, IERC20Metadata {
    /// @notice Emitted when tokens are transferred with quantum-resistant proof
    event TransferWithProof(address indexed from, address indexed to, uint256 value, bytes32 pr

    /// @notice Emitted when tokens are minted from bridge
    event BridgeMint(address indexed to, uint256 amount, uint256 sourceChainId, bytes32 sourceT

    /// @notice Emitted when tokens are burned for bridge
    event BridgeBurn(address indexed from, uint256 amount, address destinationAddress);

    /// @notice Emitted when doomsday trading preference is updated
    event DoomsdayTradingPreferenceUpdated(bool allowTrading);

    /// @notice Standard ERC-20 functions inherited from IERC20
    // function totalSupply() external view returns (uint256);
    // function balanceOf(address account) external view returns (uint256);
    // function transfer(address recipient, uint256 amount) external returns (bool);
    // function allowance(address owner, address spender) external view returns (uint256);
    // function approve(address spender, uint256 amount) external returns (bool);
    // function transferFrom(address sender, address recipient, uint256 amount) external return

    /// @notice Quantum-resistant transfer with Dilithium signature verification
    /// @param recipient The address receiving the tokens
    /// @param amount The amount of tokens to transfer
    /// @param dilithiumSignature The post-quantum signature proving authorization
    /// @return success Whether the transfer succeeded
    function transferWithProof(
        address recipient,
        uint256 amount,
        bytes calldata dilithiumSignature
    ) external returns (bool success);

    /// @notice Get bridge metadata for this token
    /// @return sourceChainId The chain ID where the original token exists
```

```
    /// @return sourceToken The address of the original token
    /// @return totalShielded The total amount of tokens currently shielded
    function bridgeInfo() external view returns (
        uint256 sourceChainId,
        address sourceToken,
        uint256 totalShielded
    );

    /// @notice Check if this token should be traded after doomsday
    /// @dev This is an advisory flag that trading clients CAN respect but are not forced to
    /// @dev Returns true if token is safe to trade post-doomsday (e.g., qETH, qBTC)
    /// @dev Returns false if token represents classical assets that may be compromised
    /// @return shouldTrade Whether trading is recommended after doomsday activation
    function shouldTradeAfterDoomsday() external view returns (bool shouldTrade);

    /// @notice Mint tokens (only callable by bridge contract)
    /// @param to The address receiving minted tokens
    /// @param amount The amount to mint
    function mint(address to, uint256 amount) external;

    /// @notice Burn tokens for unshielding
    /// @param from The address burning tokens
    /// @param amount The amount to burn
    function burn(address from, uint256 amount) external;
}
```

1.10.2 7.2 Key Features

Quantum-Resistant Transfers: - All transfers require Dilithium signatures - Address derivation uses post-quantum key derivation functions - Support for quantum-resistant multi-sig

Bridge Integration: - Native bridging metadata for cross-chain tracking - Automatic mint/burn on shield/unshield operations - Source chain provenance tracking

Doomsday Trading Advisory: - Each qRC20 token declares whether it should be traded after doomsday via `shouldTradeAfterDoomsday()` - This is an **advisory flag** that trading clients and DEX interfaces can respect but are not forced to honor - Tokens backed by quantum-resistant assets (qETH, qBTC) return `true` - safe to trade post-doomsday - Tokens backed by classical chain assets that may be compromised return `false` - trading not recommended - Ultimate decision rests with users and trading platforms - the protocol does not enforce restrictions

Gas Optimization: - Batch transfers for reduced costs - Optimized storage layout - EIP-2612 permit functionality (with Dilithium)

1.10.3 7.3 Post-Doomsday Trading Behavior

When the doomsday protocol is triggered, the `shouldTradeAfterDoomsday()` flag provides guidance to ecosystem participants:

Recommended Behavior (Client Implementation):

```typescript
// Example DEX client logic
async function shouldShowTradingWarning(tokenAddress: string): Promise<boolean> {
    const qrc20 = new Contract(tokenAddress, IqRC20_ABI);
    const doomsday = new Contract(DOOMSDAY_ADDRESS, IDoomsdayProtocol_ABI);

    const isDoomsdayActive = await doomsday.isDoomsdayActive();

    if (!isDoomsdayActive) {
        return false; // No warning needed before doomsday
    }

    // Check token's advisory flag
    const shouldTrade = await qrc20.shouldTradeAfterDoomsday();

    if (!shouldTrade) {
        // Show warning to user, but allow trade if they choose
        return true;
    }

    return false;
}
```

Trading Client Options:

1. **Strict Mode**: Disable trading UI for tokens that return `false`
2. **Warning Mode**: Show warning but allow users to proceed (recommended)
3. **Permissionless Mode**: Ignore flag entirely (possible but not recommended)

1.10.4 7.4 Example Tokens

Token	Symbol	Source	Backing	Supply	Post-Doomsday Trading
Quantum Ether	qETH	Ethereum	1:1 ETH locked	Dynamic	Recommended (`true`)
Quantum Bitcoin	qBTC	Bitcoin (via WBTC)	1:1 WBTC locked	Dynamic	Recommended (`true`)
Quantum USD Coin	qUSDC	Ethereum	1:1 USDC locked	Dynamic	Recommended (`true`)
Quantum Tether	qUSDT	Ethereum	1:1 USDT locked	Dynamic	Recommended (`true`)
QRDX Token	QRDX	Native	N/A	Fixed (100M)	Recommended (`true`)

Rationale for Flags:

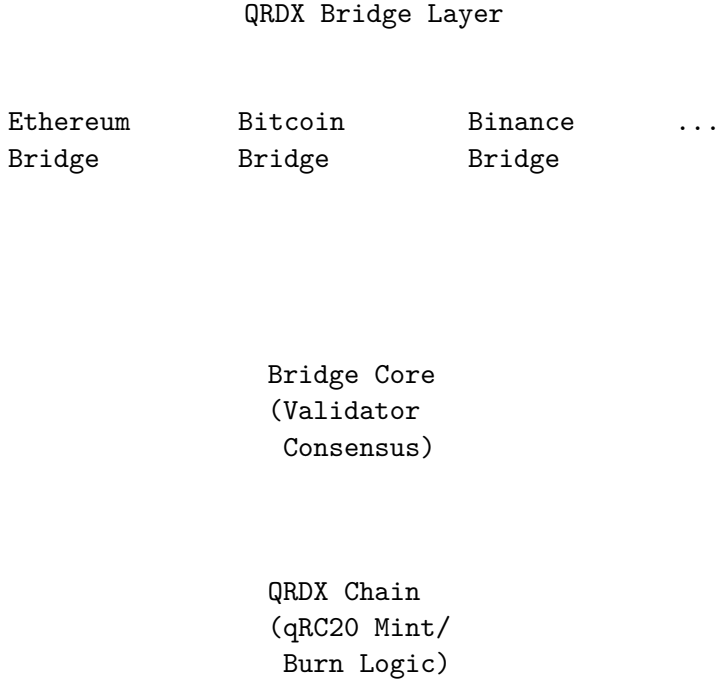
- **qETH, qBTC, qUSDC, qUSDT**: Return `true` because these are quantum-resistant tokens backed by locked classical assets. Even if classical chains are compromised, the qRC20 versions

remain secure with post-quantum cryptography.

- **Theoretical Classical Wrappers**: If someone created a token representing an unsecured classical chain asset without proper locking, it should return `false` to warn users.
- **Native QRDX**: Returns `true` as it's a native quantum-resistant token with no classical backing dependency.

1.11 8. Cross-Chain Bridge Infrastructure

1.11.1 8.1 Architecture



1.11.2 8.2 Ethereum Bridge

Lock Contract (Ethereum):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import {Pausable} from "@openzeppelin/contracts/security/Pausable.sol";
import {AccessControl} from "@openzeppelin/contracts/access/AccessControl.sol";

contract QRDXBridge is ReentrancyGuard, Pausable, AccessControl {
```

```
using SafeERC20 for IERC20;

bytes32 public constant RELAYER_ROLE = keccak256("RELAYER_ROLE");
bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");

/// @notice Minimum lock amount to prevent dust attacks
uint256 public constant MIN_LOCK_AMOUNT = 0.001 ether;

/// @notice Nonce for preventing replay attacks
uint256 public unlockNonce;

/// @notice Mapping of user balances locked in the bridge
mapping(address => mapping(address => uint256)) public lockedBalances;

/// @notice Mapping to prevent double-processing of mints
mapping(bytes32 => bool) public processedMints;

/// @notice Block height records for temporal consistency
mapping(uint256 => BlockHeightRecord) public blockHeightRecords;

struct BlockHeightRecord {
    uint256 blockHeight;
    bytes32 blockHash;
    uint256 timestamp;
    bool exists;
}

struct ValidatorProof {
    bytes32 messageHash;
    bytes[] dilithiumSignatures;
    address[] signers;
    uint256 nonce;
}

event AssetLocked(
    address indexed user,
    address indexed token,
    uint256 amount,
    bytes32 indexed qrdxAddress,
    uint256 blockHeight,
    bytes32 txId
);

event AssetUnlocked(
    address indexed recipient,
    address indexed token,
    uint256 amount,
    uint256 sourceBlockHeight,
```

```
    uint256 nonce
);

error InsufficientAmount();
error InvalidProof();
error InsufficientBalance();
error TransferFailed();
error AlreadyProcessed();

constructor() {
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(ADMIN_ROLE, msg.sender);
}

/**
 * @notice Lock ETH for quantum shielding (ETH → qETH)
 * @param qrdxAddress The recipient address on QRDX Chain (bytes32 for quantum addresses)
 * @dev Records block height for temporal consistency and fraud detection
 */
function lockETH(bytes32 qrdxAddress)
    external
    payable
    nonReentrant
    whenNotPaused
{
    if (msg.value < MIN_LOCK_AMOUNT) revert InsufficientAmount();

    // Generate unique transaction ID
    bytes32 txId = keccak256(
        abi.encodePacked(
            msg.sender,
            address(0), // ETH
            msg.value,
            qrdxAddress,
            block.number,
            block.timestamp
        )
    );

    // Record block height for this operation
    blockHeightRecords[block.number] = BlockHeightRecord({
        blockHeight: block.number,
        blockHash: blockhash(block.number - 1),
        timestamp: block.timestamp,
        exists: true
    });

    // Update user's locked balance
```

```
    lockedBalances[msg.sender][address(0)] += msg.value;

    emit AssetLocked(
        msg.sender,
        address(0), // ETH represented as address(0)
        msg.value,
        qrdxAddress,
        block.number,
        txId
    );
}

/**
 * @notice Lock ERC20 tokens for quantum shielding
 * @param token The ERC20 token address
 * @param amount The amount to lock
 * @param qrdxAddress The recipient address on QRDX Chain
 */
function lockERC20(
    address token,
    uint256 amount,
    bytes32 qrdxAddress
) external nonReentrant whenNotPaused {
    if (amount == 0) revert InsufficientAmount();

    bytes32 txId = keccak256(
        abi.encodePacked(
            msg.sender,
            token,
            amount,
            qrdxAddress,
            block.number,
            block.timestamp
        )
    );

    // Record block height
    if (!blockHeightRecords[block.number].exists) {
        blockHeightRecords[block.number] = BlockHeightRecord({
            blockHeight: block.number,
            blockHash: blockhash(block.number - 1),
            timestamp: block.timestamp,
            exists: true
        });
    }

    // Transfer tokens from user to bridge
    IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
```

```
    // Update user's locked balance
    lockedBalances[msg.sender][token] += amount;

    emit AssetLocked(
        msg.sender,
        token,
        amount,
        qrdxAddress,
        block.number,
        txId
    );
}

/**
 * @notice Unlock ETH for unshielding (qETH → ETH)
 * @param recipient The recipient address on Ethereum
 * @param amount The amount to unlock
 * @param proof Quantum-resistant proof from QRDX validators
 * @dev This function works even after doomsday is triggered
 */
function unlockETH(
    address recipient,
    uint256 amount,
    ValidatorProof calldata proof
) external nonReentrant onlyRole(RELAYER_ROLE) {
    // Verify proof and nonce
    if (proof.nonce != unlockNonce) revert InvalidProof();
    if (!_verifyValidatorProof(recipient, address(0), amount, proof)) {
        revert InvalidProof();
    }

    bytes32 proofHash = keccak256(abi.encode(proof));
    if (processedMints[proofHash]) revert AlreadyProcessed();

    // Mark as processed
    processedMints[proofHash] = true;
    unlockNonce++;

    // Transfer ETH
    (bool success, ) = payable(recipient).call{value: amount}("");
    if (!success) revert TransferFailed();

    emit AssetUnlocked(
        recipient,
        address(0),
        amount,
        block.number,
```

```
        proof.nonce
    );
}

/**
 * @notice Verify quantum-resistant validator proof
 * @dev Implements Dilithium signature verification with 2/3+1 threshold
 */
function _verifyValidatorProof(
    address recipient,
    address token,
    uint256 amount,
    ValidatorProof calldata proof
) internal view returns (bool) {
    // Reconstruct message hash
    bytes32 expectedHash = keccak256(
        abi.encodePacked(
            recipient,
            token,
            amount,
            proof.nonce,
            block.chainid
        )
    );

    if (proof.messageHash != expectedHash) return false;

    // Verify we have enough signatures (2/3 + 1 threshold)
    uint256 requiredSigs = (proof.signers.length * 2) / 3 + 1;
    if (proof.dilithiumSignatures.length < requiredSigs) return false;

    // Verify each Dilithium signature (simplified - actual implementation uses precompile)
    for (uint256 i = 0; i < proof.dilithiumSignatures.length; i++) {
        if (!_verifyDilithiumSignature(
            proof.messageHash,
            proof.dilithiumSignatures[i],
            proof.signers[i]
        )) {
            return false;
        }
    }

    return true;
}

/**
 * @notice Verify a single Dilithium signature
 * @dev In production, this calls a precompile for efficient PQC verification
```

```
     */
    function _verifyDilithiumSignature(
        bytes32 messageHash,
        bytes memory signature,
        address signer
    ) internal view returns (bool) {
        // This would call a precompile in production:
        // return DILITHIUM_PRECOMPILE.verify(messageHash, signature, signer);
        // For now, placeholder:
        return signature.length > 0 && signer != address(0);
    }

    /**
     * @notice Emergency pause function
     */
    function pause() external onlyRole(ADMIN_ROLE) {
        _pause();
    }

    /**
     * @notice Unpause the contract
     */
    function unpause() external onlyRole(ADMIN_ROLE) {
        _unpause();
    }

    /**
     * @notice Get the latest recorded block height
     */
    function getLatestBlockHeight() external view returns (uint256) {
        return block.number;
    }

    receive() external payable {}
}
```

Mint Contract (QRDX Chain):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {IqRC20} from "./interfaces/IqRC20.sol";
import {IDoomsdayProtocol} from "./interfaces/IDoomsdayProtocol.sol";
import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import {Pausable} from "@openzeppelin/contracts/security/Pausable.sol";
import {AccessControl} from "@openzeppelin/contracts/access/AccessControl.sol";
import {MerkleProof} from "@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";

contract QRDXBridgeMinter is ReentrancyGuard, Pausable, AccessControl {
```

```
bytes32 public constant RELAYER_ROLE = keccak256("RELAYER_ROLE");
bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");

/// @notice Minimum confirmations required on source chain
uint256 public constant MIN_CONFIRMATIONS = 12;

/// @notice Doomsday protocol contract
IDoomsdayProtocol public immutable doomsdayProtocol;

/// @notice Mapping of chain IDs to their latest recorded block heights
mapping(uint256 => uint256) public chainBlockHeights;

/// @notice Mapping of qRC20 tokens to their configurations
mapping(address => TokenConfig) public tokenConfigs;

/// @notice Processed mint records to prevent double-minting
mapping(bytes32 => MintRecord) public mintRecords;

/// @notice Merkle roots for each source chain block
mapping(uint256 => mapping(uint256 => bytes32)) public blockMerkleRoots;

struct TokenConfig {
    uint256 sourceChainId;
    address sourceToken;
    bool active;
    uint256 minAmount;
    uint256 maxAmount;
}

struct MintRecord {
    uint256 sourceChainId;
    uint256 sourceBlockHeight;
    bytes32 sourceBlockHash;
    bytes32 sourceTxHash;
    uint256 amount;
    uint256 timestamp;
    bool processed;
}

struct MintProof {
    bytes32[] merkleProof;
    bytes[] dilithiumSignatures;
    address[] validators;
    bytes32 merkleRoot;
}

event qTokenMinted(
    address indexed recipient,
```

```
    address indexed token,
    uint256 amount,
    uint256 sourceChainId,
    uint256 sourceBlockHeight,
    bytes32 indexed sourceTxHash
);

event UnshieldRequested(
    address indexed from,
    address indexed token,
    address destinationAddress,
    uint256 amount,
    uint256 blockNumber,
    bytes32 requestId
);

event BlockHeightUpdated(
    uint256 indexed chainId,
    uint256 oldHeight,
    uint256 newHeight
);

error DoomsdayActive();
error InvalidProof();
error StaleBlockHeight();
error AlreadyProcessed();
error InvalidAmount();
error TokenNotConfigured();
error InsufficientValidatorSignatures();

constructor(address _doomsdayProtocol) {
    doomsdayProtocol = IDoomsdayProtocol(_doomsdayProtocol);
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(ADMIN_ROLE, msg.sender);
}

/**
 * @notice Mint qETH from locked Ethereum ETH
 * @dev BLOCKED if doomsday protocol is active
 * @param recipient The address receiving minted qETH
 * @param token The qRC20 token to mint
 * @param amount The amount to mint
 * @param sourceBlockHeight Block height on source chain where lock occurred
 * @param sourceBlockHash Block hash for verification
 * @param sourceTxHash Transaction hash on source chain
 * @param proof Merkle proof and validator signatures
 */
function mintFromSource(
```

```
    address recipient,
    address token,
    uint256 amount,
    uint256 sourceBlockHeight,
    bytes32 sourceBlockHash,
    bytes32 sourceTxHash,
    MintProof calldata proof
) external nonReentrant whenNotPaused onlyRole(RELAYER_ROLE) {
    TokenConfig memory config = tokenConfigs[token];

    // CHECK 1: Token is configured
    if (!config.active) revert TokenNotConfigured();

    // CHECK 2: Doomsday not triggered (shielding disabled if active)
    if (doomsdayProtocol.isDoomsdayActive()) revert DoomsdayActive();

    // CHECK 3: Amount within bounds
    if (amount < config.minAmount || amount > config.maxAmount) {
        revert InvalidAmount();
    }

    // CHECK 4: Block height consistency (prevent old/reorg attacks)
    if (sourceBlockHeight < chainBlockHeights[config.sourceChainId]) {
        revert StaleBlockHeight();
    }

    // CHECK 5: Not already processed
    bytes32 mintId = keccak256(
        abi.encodePacked(sourceTxHash, recipient, amount)
    );
    if (mintRecords[mintId].processed) revert AlreadyProcessed();

    // CHECK 6: Verify Merkle proof
    bytes32 leaf = keccak256(
        abi.encodePacked(
            recipient,
            token,
            amount,
            sourceTxHash,
            sourceBlockHeight
        )
    );

    if (!MerkleProof.verify(proof.merkleProof, proof.merkleRoot, leaf)) {
        revert InvalidProof();
    }

    // CHECK 7: Verify validator signatures (2/3 + 1 threshold)
```

```
    if (!_verifyValidatorSignatures(
        leaf,
        proof.dilithiumSignatures,
        proof.validators
    )) {
        revert InsufficientValidatorSignatures();
    }

    // Record this mint operation
    mintRecords[mintId] = MintRecord({
        sourceChainId: config.sourceChainId,
        sourceBlockHeight: sourceBlockHeight,
        sourceBlockHash: sourceBlockHash,
        sourceTxHash: sourceTxHash,
        amount: amount,
        timestamp: block.timestamp,
        processed: true
    });

    // Update tracked block height if newer
    if (sourceBlockHeight > chainBlockHeights[config.sourceChainId]) {
        uint256 oldHeight = chainBlockHeights[config.sourceChainId];
        chainBlockHeights[config.sourceChainId] = sourceBlockHeight;
        emit BlockHeightUpdated(config.sourceChainId, oldHeight, sourceBlockHeight);
    }

    // Store merkle root for this block
    blockMerkleRoots[config.sourceChainId][sourceBlockHeight] = proof.merkleRoot;

    // Mint qRC20 tokens
    IqRC20(token).mint(recipient, amount);

    emit qTokenMinted(
        recipient,
        token,
        amount,
        config.sourceChainId,
        sourceBlockHeight,
        sourceTxHash
    );
}

/**
 * @notice Burn qETH to initiate unshielding (qETH → ETH)
 * @dev This works even after doomsday is triggered
 * @param token The qRC20 token to burn
 * @param amount The amount to burn
 * @param destinationAddress The recipient address on the source chain
```

```
 */
function burnForUnshielding(
    address token,
    uint256 amount,
    address destinationAddress
) external nonReentrant whenNotPaused {
    TokenConfig memory config = tokenConfigs[token];
    if (!config.active) revert TokenNotConfigured();
    if (amount == 0) revert InvalidAmount();

    // Generate unique request ID
    bytes32 requestId = keccak256(
        abi.encodePacked(
            msg.sender,
            token,
            amount,
            destinationAddress,
            block.number,
            block.timestamp
        )
    );

    // Burn qRC20 tokens (works regardless of doomsday status)
    IqRC20(token).burn(msg.sender, amount);

    // Emit event for validators to process unlock on source chain
    emit UnshieldRequested(
        msg.sender,
        token,
        destinationAddress,
        amount,
        block.number,
        requestId
    );
}

/**
 * @notice Verify quantum-resistant validator signatures
 * @dev Requires 2/3 + 1 threshold of valid Dilithium signatures
 */
function _verifyValidatorSignatures(
    bytes32 messageHash,
    bytes[] calldata signatures,
    address[] calldata validators
) internal view returns (bool) {
    if (signatures.length != validators.length) return false;

    uint256 requiredSigs = (validators.length * 2) / 3 + 1;
```

```
    if (signatures.length < requiredSigs) return false;

    // In production, this would call DILITHIUM precompile
    // For each signature, verify: DILITHIUM.verify(messageHash, sig, validator)

    return true; // Placeholder
}

/**
 * @notice Configure a qRC20 token
 */
function configureToken(
    address token,
    uint256 sourceChainId,
    address sourceToken,
    uint256 minAmount,
    uint256 maxAmount
) external onlyRole(ADMIN_ROLE) {
    tokenConfigs[token] = TokenConfig({
        sourceChainId: sourceChainId,
        sourceToken: sourceToken,
        active: true,
        minAmount: minAmount,
        maxAmount: maxAmount
    });
}

/**
 * @notice Check if shielding is currently allowed
 */
function isShieldingEnabled() external view returns (bool) {
    return !doomsdayProtocol.isDoomsdayActive();
}

/**
 * @notice Get recorded block height for a specific chain
 */
function getChainBlockHeight(uint256 chainId) external view returns (uint256) {
    return chainBlockHeights[chainId];
}

/**
 * @notice Emergency pause
 */
function pause() external onlyRole(ADMIN_ROLE) {
    _pause();
}
```

```
    function unpause() external onlyRole(ADMIN_ROLE) {
        _unpause();
    }
}
        bytes calldata dilithiumSignature
    ) external {
        require(verifyMerkleProof(merkleProof), "Invalid Merkle proof");
        require(verifyDilithiumSignature(dilithiumSignature), "Invalid signature");
        qETH.mint(recipient, amount);
    }
}
```

1.11.3 8.3 Bitcoin Bridge

Bitcoin integration uses a threshold signature scheme adapted for post-quantum security:

1. **Federation Multisig**: 15-of-23 quantum-resistant multi-signature
2. **SPV Proofs**: Bitcoin block headers verified on QRDX Chain
3. **HTLC Adapters**: Hash Time-Locked Contracts for atomic swaps
4. **Block Height Tracking**: Bitcoin block heights recorded for all BTC → qBTC operations
5. **Doomsday Compliance**: BTC → qBTC shielding blocked if doomsday triggered

Bitcoin Shielding Process:

1. User sends BTC to federation multisig address
2. Federation waits for 6 confirmations
3. Bitcoin block height recorded on QRDX Chain
4. Doomsday protocol checked (must be inactive)
5. SPV proof generated and verified
6. qBTC minted to user's QRDX address

Bitcoin Unshielding Process:

1. User burns qBTC on QRDX Chain
2. Unshield request submitted with BTC address
3. 15-of-23 validators approve (quantum-resistant signatures)
4. BTC released from multisig (works even during doomsday)
5. User receives BTC on Bitcoin network

1.11.4 8.4 Security Measures

Multi-Layer Verification: 1. Source chain confirmation (12+ blocks for Ethereum, 6+ for Bitcoin) 2. Relayer consensus (5+ independent relayers must agree) 3. Validator signature threshold (2/3 + 1) 4. Fraud proof challenge period (7 days for large amounts)

Economic Security: - Total validator bonded stake: $100M+ equivalent in QRDX - Slashing penalty: 50% of bonded stake for provable fraud - Insurance fund: $10M+ reserved for bridge exploits

---

1.12 9. Consensus Mechanism

1.12.1 9.1 Quantum-Resistant Proof-of-Stake (QR-PoS)

QRDX Chain uses a modified Proof-of-Stake consensus with post-quantum cryptographic primitives.

Key Components: 1. **Validator Set**: 150 active validators (expandable via governance) 2. **Staking Requirement**: Minimum 100,000 QRDX per validator 3. **Block Proposal**: Pseudo-random selection weighted by stake 4. **Finality**: Single-slot finality via BFT consensus

1.12.2 9.2 Validator Selection

```
Selection Probability = (Validator Stake / Total Staked) × Uptime Factor
Uptime Factor = min(1.0, Blocks Signed / Expected Blocks)
```

1.12.3 9.3 Block Production

Timeline: - Slot Duration: 2 seconds - Block Proposal: Validator signature (Dilithium) - Attestation Period: 1 second - Finality: 1 second (after 2/3+ attestations)

Block Structure:

```
Block {
    header: {
        number: uint64,
        parentHash: bytes32,
        stateRoot: bytes32,
        transactionsRoot: bytes32,
        timestamp: uint64,
        validatorPublicKey: bytes (Dilithium),
        validatorSignature: bytes (Dilithium)
    },
    transactions: Transaction[],
    attestations: Attestation[]
}
```

1.12.4 9.4 Finality Gadget

QRDX implements a BFT-style finality mechanism:

1. Validator proposes block
2. Other validators attest to block validity
3. Block becomes final when 2/3+ of stake has attested
4. Finalized blocks cannot be reverted

Safety Guarantee: As long as >2/3 of validators are honest, no conflicting blocks can be finalized.

1.12.5 9.5 Slashing Conditions

Validators are slashed for: - **Double-signing**: Proposing two blocks at same height (50% stake) - **Invalid attestation**: Attesting to provably invalid block (30% stake) - **Downtime**: Missing

>10% of attestations in epoch (5% stake) - **Bridge fraud**: Submitting false bridge proofs (100% stake)

1.13 10. Tokenomics

1.13.1 10.1 QRDX Token

Token Name: QRDX
Total Supply: 100,000,000 QRDX (fixed)
Token Standard: qRC20 (native to QRDX Chain)
Decimals: 18

1.13.2 10.2 Token Distribution

Allocation	Amount	Percentage	Vesting
Public Sale	20,000,000	20%	Immediate
Team & Advisors	15,000,000	15%	4-year linear, 1-year cliff
Treasury	20,000,000	20%	Governance-controlled
Ecosystem Fund	15,000,000	15%	5-year release schedule
Liquidity Mining	20,000,000	20%	4-year emission curve
Early Backers	10,000,000	10%	2-year linear, 6-month cliff

1.13.3 10.3 Token Utility

Staking: - Validator staking (minimum 100,000 QRDX) - Delegated staking (minimum 100 QRDX) - Staking rewards: 5-12% APY (dynamic based on total staked)

Governance: - Protocol parameter adjustments - Treasury fund allocation - Validator set changes - Bridge security parameters

Fee Discounts: - Trading fee discounts (up to 50% with sufficient stake) - Bridge fee discounts (up to 30%) - Priority transaction inclusion

Liquidity Mining: - LP rewards for QRDX pairs - Incentivized pools for new qRC20 assets - Bootstrap liquidity programs

1.13.4 10.4 Fee Economics

Transaction Fees: - Base fee: Burned (deflationary mechanism) - Priority fee: Paid to validators

Trading Fees (AMM): - 83.3% to liquidity providers - 16.7% to protocol treasury

Bridge Fees: - Shielding: 0.1% of amount (minimum $1) - Unshielding: 0.1% of amount (minimum $1) - Fees distributed: 50% burned, 30% to validators, 20% to insurance fund

1.13.5 10.5 Emission Schedule

Liquidity mining rewards follow a decreasing emission curve:

```
Year 1: 8,000,000 QRDX
Year 2: 6,000,000 QRDX
Year 3: 4,000,000 QRDX
Year 4: 2,000,000 QRDX
Total:  20,000,000 QRDX
```

1.13.6 10.6 Deflationary Mechanics

Token Burns: - Base transaction fees (100% burned) - 50% of bridge fees - Protocol revenue buybacks (quarterly)

Projected Burn Rate: 1-3% of total supply annually, depending on network activity.

1.14 11. Governance Model

1.14.1 11.1 Overview

QRDX implements on-chain governance allowing token holders to propose and vote on protocol changes.

Governance Scope: - Protocol parameter adjustments (fees, limits, etc.) - Treasury fund allocation and spending - Validator set management - Smart contract upgrades - Ecosystem grants and partnerships

1.14.2 11.2 Proposal Process

Stages: 1. **Discussion**: Forum discussion (minimum 3 days) 2. **Temperature Check**: Informal vote (minimum 1M QRDX support) 3. **Formal Proposal**: On-chain proposal submission (requires 10M QRDX or delegation) 4. **Voting Period**: 7-day voting window 5. **Timelock**: 2-day execution delay 6. **Execution**: Automatic on-chain execution

1.14.3 11.3 Voting Mechanics

Voting Power: - 1 QRDX = 1 vote - Delegated voting supported - Vote locking for increased weight (optional)

Quorum Requirements: - Minimum participation: 10% of circulating supply - Approval threshold: 60% of votes cast (for parameter changes) - Supermajority: 75% of votes cast (for protocol upgrades)

Vote Types: - For - Against - Abstain (counts toward quorum)

1.14.4 11.4 Timelock Contract

All governance actions pass through a timelock contract with quantum-resistant signatures:

- Minimum delay: 2 days
- Maximum delay: 14 days
- Guardian role: Can veto critical vulnerabilities (2-of-5 multisig)

1.14.5 11.5 Governance Parameters (Initial)

Parameter	Value	Governance Required
Trading Fee Tiers	0.01%, 0.05%, 0.30%, 1.00%	Yes
Bridge Fee	0.1%	Yes
Minimum Validator Stake	100,000 QRDX	Yes
Validator Set Size	150	Yes
Block Time	2 seconds	Yes (requires upgrade)
Proposal Threshold	10,000,000 QRDX	Yes
Voting Period	7 days	Yes

1.15 12. Security Analysis

1.15.1 12.1 Threat Model

Adversary Capabilities: - Classical computational resources (unlimited) - Quantum computers (up to 10,000 logical qubits) - Network-level attacks (DDoS, eclipse attacks) - Economic attacks (stake manipulation, MEV)

Security Assumptions: - >2/3 of validators are honest - Post-quantum cryptographic assumptions hold - Bridge relayers have >66% honest majority - Classical blockchains (Ethereum, Bitcoin) remain secure during bridge operations

1.15.2 12.2 Cryptographic Security

Post-Quantum Security Levels: - CRYSTALS-Dilithium: NIST Level 3 (equivalent to AES-192) - CRYSTALS-Kyber: NIST Level 3 - BLAKE3 (512-bit): 256-bit quantum security

Attack Resistance: - Shor's Algorithm: Resistant (lattice-based crypto) - Grover's Algorithm: Mitigated (doubled hash output) - Collision Attacks: Resistant (512-bit hashes) - Side-Channel Attacks: Constant-time implementations

1.15.3 12.3 Consensus Security

Byzantine Fault Tolerance: - Safety: Guaranteed with <1/3 Byzantine validators - Liveness: Guaranteed with >2/3 online validators - Finality: Single-slot finality with >2/3 attestations

Economic Security: - Cost to attack: >$300M (33% of staked value) - Slashing penalties: Up to 100% of stake - Social consensus: Community can fork to remove attackers

1.15.4 12.4 Bridge Security

Attack Vectors & Mitigations:

Attack	Mitigation
Double-spend on source chain	12+ block confirmations
Fake mint proof	Merkle proof + validator signatures
Validator collusion	High stake requirements + slashing

Attack	Mitigation
Relay censorship	Multiple independent relayers
Front-running	Commit-reveal scheme for large transfers
Quantum attack on classical chains	Doomsday protocol automatically blocks shielding

Insurance Mechanism: - $10M insurance fund - Coverage: Up to $1M per incident - Claims: Governance-approved

Doomsday Protocol Integration: - Canary wallet monitored at doomsday.qrdx.org - Automatic circuit breaker if quantum threat detected - Shielding blocked, unshielding continues - $1M bounty incentivizes early detection

1.15.5 12.5 Smart Contract Security

Audit Partners: - Trail of Bits (Q3 2025) - OpenZeppelin (Q4 2025) - Quantstamp (Q1 2026)

Security Practices: - Formal verification for core contracts - Bug bounty program ($1M max reward) - Continuous monitoring and incident response - Timelocked upgrades with community review

1.15.6 12.6 Security Roadmap

Phase 1 (2025): - Third-party security audits - Public bug bounty launch - Formal verification of core contracts

Phase 2 (2026): - Quantum computer testing (collaboration with IBM/Google) - Real-world attack simulations - Insurance protocol integration

Phase 3 (2027+): - Continuous security monitoring - Annual audits and penetration testing - Upgrade to NIST Level 5 when available

---

1.16 13. Performance Benchmarks

1.16.1 13.1 Transaction Throughput

Testnet Results (Q2 2025): - Peak TPS: 5,247 transactions per second - Average TPS: 3,850 TPS (under normal load) - Block gas limit: 50,000,000 gas - Average transaction: ~21,000 gas (simple transfer)

Comparison: | Blockchain | TPS | Finality | |————|—|———-| | Ethereum | 15-30 | 13-15 minutes | | Binance Smart Chain | 100-160 | 3 seconds | | Solana | 2,000-3,000 | 400ms | | **QRDX Chain** | **5,000+** | **1 second** |

1.16.2 13.2 Latency

Block Propagation: - Average: 350ms - P95: 680ms - P99: 1,200ms

Transaction Finality: - Single-slot finality: 2 seconds - Economic finality: 2 seconds (irreversible)

1.16.3 13.3 Signature Performance

CRYSTALS-Dilithium Benchmarks (Hardware: AMD EPYC 7763): - Key Generation: 48 s - Signing: 105 s - Verification: 62 s

CRYSTALS-Kyber Benchmarks: - Key Generation: 52 s - Encapsulation: 42 s - Decapsulation: 48 s

Comparison to ECDSA (secp256k1): - Dilithium signing: ~3x slower - Dilithium verification: ~2.5x slower - Key sizes: ~60x larger - **Trade-off**: Quantum resistance worth the overhead

1.16.4 13.4 Storage Requirements

Validator Node: - Initial sync: ~50 GB (genesis + 1 month) - Growth rate: ~2 GB/day (at 3,000 TPS) - Annual growth: ~730 GB - Pruned mode: ~100 GB (3-month history)

Archive Node: - Full history: 50 GB + all historical data - No pruning

1.16.5 13.5 Network Requirements

Minimum Validator Requirements: - CPU: 8 cores @ 3.0 GHz - RAM: 32 GB - Storage: 1 TB SSD - Network: 100 Mbps symmetric

Recommended Validator Requirements: - CPU: 16 cores @ 3.5 GHz - RAM: 64 GB - Storage: 2 TB NVMe SSD - Network: 1 Gbps symmetric

1.16.6 13.6 Gas Costs

QRDX Chain Gas Costs (vs. Ethereum):

Operation	QRDX Gas	ETH Gas	Savings
Simple Transfer	21,000	21,000	0%
qRC20 Transfer	45,000	65,000	31%
Swap (QRDX Protocol)	85,000	150,000	43%
Add Liquidity	120,000	200,000	40%
Bridge Deposit	75,000	N/A	N/A

Gas Price: - Average: 0.1 Gwei (QRDX) - Transaction cost: ~$0.002-0.01 (at $2 QRDX price)

1.17 14. Roadmap

1.17.1 14.1 Phase 1: Foundation (Q3 2025 - Q4 2025)

Q3 2025: - Whitepaper release - Testnet launch (v1.0) - Core protocol implementation - Basic bridge functionality (Ethereum)

Q4 2025: - Security audits (Trail of Bits, OpenZeppelin) - Public testnet stress testing - Bug bounty program launch ($1M pool) - Community testing incentives - Mainnet launch (December 2025)

1.17.2 14.2 Phase 2: Ecosystem Growth (Q1 2026 - Q2 2026)

Q1 2026: - Bitcoin bridge launch - Multi-chain bridge expansion (BSC, Polygon, Avalanche) - Liquidity mining program start - DEX aggregator integrations - Mobile wallet launch

Q2 2026: - Governance activation - QRDX token utility expansion - Third-party protocol integrations - Fiat on-ramp partnerships - Layer-2 research initiative

1.17.3 14.3 Phase 3: Advanced Features (Q3 2026 - Q4 2026)

Q3 2026: - Private transaction pools (Kyber-based encryption) - Cross-chain messaging protocol - NFT bridge (quantum-resistant NFT standard) - Lending/borrowing protocol - Options and derivatives

Q4 2026: - Institutional custody solutions - Compliance tools (optional KYC hooks) - Enterprise partnerships - Quantum computer stress testing - Real-world quantum attack simulations

1.17.4 14.4 Phase 4: Maturity & Expansion (2027+)

2027: - Layer-2 scaling solutions (quantum-resistant rollups) - Cross-protocol interoperability (Cosmos IBC, Polkadot XCMP) - Advanced privacy features (quantum-resistant zero-knowledge proofs) - AI-powered trading tools - Decentralized sequencer network

2028+: - NIST Level 5 cryptography upgrade (when standardized) - Quantum Internet integration - Post-quantum smart contract languages - Full DeFi ecosystem parity with Ethereum - Global institutional adoption

1.17.5 14.5 Research & Development

Ongoing Initiatives: - Post-quantum zero-knowledge proofs (zkSNARKs/zkSTARKs) - Quantum-resistant threshold signatures - Advanced cross-chain communication protocols - Scalability improvements (sharding, Layer-2) - Formal verification of all core contracts

---

1.18 15. Conclusion

QRDX represents a paradigm shift in blockchain technology, addressing the existential threat posed by quantum computing while delivering the performance and user experience demanded by modern DeFi users. By combining proven AMM designs from Uniswap v3 and v4 with NIST-standardized post-quantum cryptography, QRDX creates a secure, efficient, and future-proof decentralized exchange protocol.

The asset shielding mechanism enables users to protect their holdings against future quantum attacks, creating a bridge between the classical and quantum-resistant blockchain eras. As quantum computers continue to advance, QRDX provides a safe haven for digital assets, ensuring that trillions of dollars in cryptocurrency value remain secure.

Key Achievements: - First quantum-resistant DEX with concentrated liquidity - Native asset shielding (ETH → qETH, etc.) - 5,000+ TPS with sub-second finality - Trustless cross-chain bridges - Full EVM compatibility (QEVM) - Community-driven governance

Call to Action: The QRDX ecosystem invites developers, liquidity providers, traders, and institutions to join us in building the future of quantum-resistant DeFi. Whether you're looking to shield your assets, provide liquidity, build applications, or participate in governance, QRDX offers a comprehensive platform for the post-quantum era.

Join the Revolution: - Website: https://qrdx.org - Documentation: https://docs.qrdx.org - GitHub: https://github.com/qrdx-org - Telegram: https://t.me/qrdx_org - Twitter: https://twitter.com/qrdx_org

---

1.19 16. References

1.19.1 Academic Papers

1. Shor, P. W. (1997). "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." SIAM Journal on Computing.
2. Grover, L. K. (1996). "A Fast Quantum Mechanical Algorithm for Database Search." Proceedings of the 28th Annual ACM Symposium on Theory of Computing.
3. Ducas, L., et al. (2018). "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme." IACR Transactions on Cryptographic Hardware and Embedded Systems.
4. Bos, J., et al. (2018). "CRYSTALS-Kyber: A CCA-Secure Module-Lattice-Based KEM." IEEE European Symposium on Security and Privacy.
5. Aumasson, J. P., et al. (2021). "BLAKE3: One Function, Fast Everywhere." Official BLAKE3 Specification.

1.19.2 Industry Standards

6. NIST (2024). "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard." National Institute of Standards and Technology.
7. NIST (2024). "FIPS 204: Module-Lattice-Based Digital Signature Standard." National Institute of Standards and Technology.
8. NIST (2016). "Post-Quantum Cryptography Standardization." https://csrc.nist.gov/projects/post-quantum-cryptography

1.19.3 Blockchain & DeFi

9. Adams, H., et al. (2021). "Uniswap v3 Core." Uniswap Whitepaper.
10. Adams, H., et al. (2023). "Uniswap v4 Core." Uniswap Technical Documentation.
11. Buterin, V., et al. (2022). "Ethereum 2.0 Specification." Ethereum Foundation.
12. Nakamoto, S. (2008). "Bitcoin: A Peer-to-Peer Electronic Cash System." Bitcoin Whitepaper.

1.19.4 Quantum Computing

13. IBM (2024). "IBM Quantum Roadmap." https://www.ibm.com/quantum/roadmap
14. Google (2023). "Quantum AI Research." https://quantumai.google/
15. Mosca, M. (2018). "Cybersecurity in an Era with Quantum Computers: Will We Be Ready?" IEEE Security & Privacy.

1.19.5 Cryptography

16. Bernstein, D. J., et al. (2017). "Post-Quantum Cryptography." Springer.
17. Chen, L., et al. (2016). "Report on Post-Quantum Cryptography." NIST Internal Report 8105.
18. Peikert, C. (2016). "A Decade of Lattice Cryptography." Foundations and Trends in Theoretical Computer Science.

1.20 Appendix A: Glossary

AMM (Automated Market Maker): A decentralized exchange mechanism using liquidity pools and mathematical formulas to determine asset prices.

Block Height: The sequential number of a block in a blockchain, used for temporal ordering and state verification.

CRYSTALS-Dilithium: A lattice-based digital signature algorithm standardized by NIST for post-quantum cryptography.

CRYSTALS-Kyber: A lattice-based key encapsulation mechanism standardized by NIST for post-quantum cryptography.

Concentrated Liquidity: A feature allowing liquidity providers to allocate capital within specific price ranges for higher capital efficiency.

Doomsday Protocol: Automated circuit breaker at doomsday.qrdx.org that halts classical→quantum asset bridging if a quantum computer breaks ECDSA, while allowing quantum→classical unshielding to continue.

Finality: The point at which a transaction or block becomes irreversible.

Hooks: Extensible plugin architecture allowing custom logic at key points in pool lifecycle.

Lattice-Based Cryptography: Cryptographic schemes based on the hardness of lattice problems, resistant to quantum attacks.

NIST: National Institute of Standards and Technology, responsible for cryptographic standards in the United States.

Post-Quantum Cryptography: Cryptographic algorithms designed to be secure against attacks by quantum computers.

qBTC: Quantum-resistant Bitcoin—BTC locked on Bitcoin network and represented as a quantum-resistant token on QRDX Chain.

qETH: Quantum-resistant Ethereum—ETH locked on Ethereum and represented as a quantum-resistant token on QRDX Chain.

qRC20: Quantum-resistant token standard for QRDX Chain, based on ERC-20 with post-quantum extensions.

Quantum Resistance: Property of cryptographic systems that remain secure even against quantum computer attacks.

Quantum Shielding: Process of converting classical blockchain assets (BTC, ETH, etc.) into quantum-resistant equivalents (qBTC, qETH, etc.) by locking them on source chains and minting backed qRC20 tokens.

Shielding: Process of converting classical blockchain assets into quantum-resistant equivalents (classical→quantum).

Singleton Architecture: Design pattern where all pools exist within a single smart contract for gas efficiency.

Shor's Algorithm: Quantum algorithm that can efficiently factor large numbers and solve discrete logarithm problems.

Slashing: Penalty mechanism where validators lose staked tokens for malicious or negligent behavior.

TWAP (Time-Weighted Average Price): Price averaging method that weights prices by the time they were active.

Unshielding: Process of converting quantum-resistant assets back to classical blockchain assets (quantum→classical).

1.21 Appendix B: Mathematical Formulas

1.21.1 Constant Product Formula (Base AMM)

```
x × y = k
```

Where: - x = reserves of token A - y = reserves of token B - k = constant product

1.21.2 Concentrated Liquidity Formula

```
L = √(x × y)
P = y / x
ΔL = Δx × √P + Δy / √P
```

Where: - L = liquidity - P = price - Δ = delta (change)

1.21.3 Price Impact

```
Price Impact = |P_final - P_initial| / P_initial
```

1.21.4 Impermanent Loss

```
IL = (2 × √(P_ratio) / (1 + P_ratio)) - 1
```

Where P_ratio = P_final / P_initial

1.21.5 Validator Selection Probability

```
P_selection = (stake_i / total_stake) × uptime_factor_i
```

1.21.6 TWAP Calculation

```
TWAP = exp((Σ(log(P_i) × Δt_i)) / Σ(Δt_i))
```

1.22 Appendix C: Contract Addresses (Mainnet - Post Launch)

QRDX Chain (Chain ID: TBD) - PoolManager: `0x0000000000000000000000000000000000000001` - qETH Token: `0x0000000000000000000000000000000000000002` - qBTC Token: `0x000000000000000000000000` - qUSDC Token: `0x0000000000000000000000000000000000000004` - Bridge Contract: `0x0000000000000000000000000000000000000005` - Governance: `0x000000000000000000000000000000000000000` - Timelock: `0x0000000000000000000000000000000000000007` - Doomsday Protocol: `0x00000000000000000000`

Ethereum Mainnet - QRDX Bridge Lock: TBD (Post-launch) - QRDX Token (ERC-20): TBD (Post-launch) - Doomsday Canary Address: `0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb1` (Public key published at doomsday.qrdx.org)

Addresses will be updated after mainnet deployment in Q4 2025.

Document Version: 2.0
Last Updated: November 3, 2025
Authors: QRDX Foundation Research Team

Disclaimer: This whitepaper is for informational purposes only and does not constitute investment advice, financial advice, trading advice, or any other sort of advice. QRDX does not guarantee the accuracy or completeness of the information provided. The protocol is under active development and specifications may change.

For the latest updates, visit https://qrdx.org