# Lux Security Token Standard: ERC-1400 on Post-Quantum EVM with MPC Custody

Zach Kelling

*Lux Industries*

2025

**Abstract**

We present the Lux Security Token Standard, an implementation of ERC-1400 (Security Token Standard) on Lux EVM with post-quantum cryptographic precompiles and MPC threshold custody for controller operations. This standard evolves from Arca's `st-contracts` (2018), which implemented ERC-1400/ERC-1404 security token protocols for ArCoin and the Arca Funds. The Lux implementation extends the original standard with: (i) transfer restriction enforcement via on-chain compliance oracles linked to I-Chain DIDs, (ii) post-quantum signature verification through ETHFALCON and Pulsar precompiles that protect long-lived securities tokens against quantum adversaries, (iii) MPC threshold signing for controller operations so that no single entity—not the issuer, not the platform, not any regulator—can unilaterally force-transfer securities, and (iv) partition management for complex capital structures including preferred/common partitions, lockup tranches, and vesting schedules. We provide formal specifications for all token operations, prove that the transfer restriction mechanism is sound and complete with respect to a given compliance policy, and compare the standard against ERC-3643 (T-REX), ERC-1404, and proprietary alternatives.

# 1 Introduction

Security tokens—blockchain-based representations of regulated securities—require capabilities far beyond those of standard fungible tokens. An ERC-20 token can be transferred freely between any two addresses. A security token must enforce transfer restrictions based on investor accreditation, jurisdictional rules, lockup periods, and regulatory holds. It must support forced transfers for court orders and regulatory actions. It must maintain auditable records of all ownership changes. It must manage complex capital structures through partitions.

## 1.1 Heritage: Arca st-contracts (2018)

The first production implementation of these requirements was Arca's `st-contracts`, developed in 2018 for the ArCoin digital security. ArCoin was an interest-bearing token backed by US Treasury bills, requiring:

- Transfer restrictions: Only accredited investors in permitted jurisdictions could hold ArCoin.
- Interest distribution: Automated calculation and distribution of T-bill interest to token holders.
- NAV tracking: On-chain net asset value updates reflecting the underlying T-bill portfolio.
- Regulatory compliance: Compliance with SEC Regulation D, Rule 506(c) for accredited investor verification.
- Forced transfers: Ability to execute transfers mandated by court orders or regulatory actions.

The `st-contracts` implementation used ERC-1400 for the core security token interface and ERC-1404 for transfer restriction logic. This implementation was deployed on Ethereum mainnet and operated without incident for the lifetime of the ArCoin program.

## 1.2 Limitations of the Original Design

The Arca implementation, while functional, had several architectural limitations that motivated the Lux standard:

1. **Centralized controller**: A single Ethereum address controlled forced transfers and document management. Compromise of this key would grant full control over all securities.
2. **Static restrictions**: Transfer restriction rules were compiled into the contract and required redeployment to update.
3. **No quantum resistance**: ECDSA signatures protecting controller operations and investor wallets are vulnerable to Shor's algorithm on future quantum computers.

4. **Limited partitioning**: The original ERC-1400 partition implementation did not support the complex capital structures required for private equity, real estate, and structured products.
5. **Gas costs**: On Ethereum mainnet, the gas costs for compliance-checked transfers made small-denomination trading economically impractical.

### 1.3 Contributions

This paper presents the Lux Security Token Standard, which addresses each limitation:

1. MPC threshold controller: Controller operations require $t$-of-$n$ threshold signatures, eliminating single points of compromise [1, 2].
2. Dynamic compliance oracles: Transfer restrictions are evaluated by on-chain compliance oracles that reference I-Chain DID claims, enabling rule updates without contract redeployment.
3. Post-quantum precompiles: ETHFALCON and Pulsar signature verification is available as EVM precompiles, enabling quantum-resistant token operations [5].
4. Extended partitions: Support for hierarchical partitions with independent restriction rules, enabling complex capital structures.
5. Lux EVM execution: Low gas costs (sub-cent per transfer) make small-denomination trading viable.

## 2 Token Interface

### 2.1 Core ERC-1400 Interface

The Lux Security Token Standard implements the full ERC-1400 interface:

Listing 1: Core security token interface.

```
interface ISecurityToken is IERC20 {
    // Document management
    function getDocument(bytes32 name)
        external view returns (string memory uri, bytes32 hash, uint256 timestamp);
    function setDocument(bytes32 name, string calldata uri, bytes32 hash) external;
    function getAllDocuments() external view returns (bytes32[] memory);

    // Controller operations (MPC threshold required)
    function isControllable() external view returns (bool);
    function controllerTransfer(
        address from, address to, uint256 value,
        bytes calldata data, bytes calldata operatorData
    ) external;
    function controllerRedeem(
        address holder, uint256 value,
        bytes calldata data, bytes calldata operatorData
    ) external;

    // Partition management
    function balanceOfByPartition(bytes32 partition, address holder)
        external view returns (uint256);
    function partitionsOf(address holder)
        external view returns (bytes32[] memory);
    function transferByPartition(
        bytes32 partition, address from, address to,
        uint256 value, bytes calldata data
    ) external returns (bytes32);

    // Transfer restriction
    function canTransfer(address to, uint256 value, bytes calldata data)
        external view returns (bytes1 statusCode, bytes32 appCode);
    function canTransferByPartition(
        bytes32 partition, address to, uint256 value, bytes calldata data
    ) external view returns (bytes1 statusCode, bytes32 appCode, bytes32 partition);
```

```
35
36      // Issuance
37      function isIssuable() external view returns (bool);
38      function issue(address holder, uint256 value, bytes calldata data) external;
39      function issueByPartition(
40          bytes32 partition, address holder, uint256 value, bytes calldata data
41      ) external;
42
43      // Redemption
44      function redeem(uint256 value, bytes calldata data) external;
45      function redeemByPartition(
46          bytes32 partition, uint256 value, bytes calldata data
47      ) external;
48  }
```

### 2.2 ERC-1404 Transfer Restrictions

Transfer restriction checking extends ERC-1404 with a richer status code system:

Listing 2: Transfer restriction interface.

```
interface ITransferRestriction {
    function detectTransferRestriction(
        address from, address to, uint256 value
    ) external view returns (uint8 restrictionCode);

    function messageForTransferRestriction(uint8 restrictionCode)
        external pure returns (string memory message);
}
```

Table 1: Transfer restriction codes.

| Code | Name | Description |
|---|---|---|
| 0x00 | SUCCESS | Transfer permitted |
| 0x01 | SENDER_NOT_ELIGIBLE | Sender fails compliance check |
| 0x02 | RECEIVER_NOT_ELIGIBLE | Receiver fails compliance check |
| 0x03 | SENDER_LOCKUP | Sender's tokens are in lockup period |
| 0x04 | RECEIVER_LIMIT | Receiver would exceed maximum holding |
| 0x05 | JURISDICTION_BLOCKED | Transfer blocked by jurisdictional rule |
| 0x06 | ACCREDITATION_EXPIRED | Receiver's accreditation has expired |
| 0x07 | REGULATORY_HOLD | Tokens subject to regulatory hold/freeze |
| 0x08 | MAX_HOLDERS | Transfer would exceed maximum holder count |
| 0x09 | PARTITION_RESTRICTED | Partition-specific restriction applies |
| 0x0A | KYC_INCOMPLETE | Receiver has not completed KYC |
| 0x0B | AML_FLAG | AML screening flag on sender or receiver |
| 0xFF | UNKNOWN | Unspecified restriction |

## 3 Compliance Oracle

### 3.1 Architecture

Transfer restrictions are evaluated by on-chain compliance oracles rather than hardcoded contract logic. Each security token references a compliance oracle contract that implements the

restriction checking interface:

Listing 3: Compliance oracle.

```
contract ComplianceOracle is ITransferRestriction {
    IIdentityRegistry public identityRegistry; // I-Chain DID registry
    mapping(bytes32 => Rule[]) public partitionRules;
    Rule[] public globalRules;

    struct Rule {
        RuleType ruleType;
        bytes params;
        bool active;
    }

    enum RuleType {
        WHITELIST,          // only whitelisted addresses
        ACCREDITATION,      // require valid accreditation credential
        JURISDICTION,       // block/allow specific jurisdictions
        LOCKUP,             // time-based transfer restriction
        MAX_HOLDERS,        // cap on total holder count
        MAX_BALANCE,        // cap on individual holding
        MIN_TRANSFER,       // minimum transfer amount
        VELOCITY            // maximum transfer frequency
    }

    function detectTransferRestriction(
        address from, address to, uint256 value
    ) external view override returns (uint8) {
        // Check global rules
        for (uint i = 0; i < globalRules.length; i++) {
            uint8 code = evaluateRule(globalRules[i], from, to, value);
            if (code != 0x00) return code;
        }
        return 0x00; // SUCCESS
    }

    function evaluateRule(
        Rule memory rule, address from, address to, uint256 value
    ) internal view returns (uint8) {
        if (!rule.active) return 0x00;

        if (rule.ruleType == RuleType.ACCREDITATION) {
            bytes32 did = identityRegistry.didOf(to);
            if (!identityRegistry.hasValidCredential(did, "accreditation")) {
                return 0x06; // ACCREDITATION_EXPIRED
            }
        }
        // ... additional rule evaluation
        return 0x00;
    }
}
```

## 3.2 I-Chain DID Integration

The compliance oracle queries the I-Chain identity registry to verify investor credentials. Each investor's DID contains verifiable credentials issued by KYC/AML providers (KYC providers integrated via I-Chain):

- **KYC credential**: Issued upon successful identity verification, contains verification level and expiry.
- **Accreditation credential**: Issued upon accredited investor verification per SEC Rule 501(a), contains basis (income, net worth, professional) and expiry date.
- **Jurisdiction credential**: Contains the investor's country and state/province of residence for jurisdictional rule evaluation.
- **AML credential**: Contains ongoing AML monitoring status, updated by continuous screening.

The oracle verifies credentials without accessing the underlying personal data, preserving investor privacy while enforcing compliance requirements.

### 3.3 Formal Properties

**Definition 1** (Compliance Policy)**.** *A compliance policy* $\Pi$ *is a finite set of rules* $\{r_1, r_2, \ldots, r_k\}$ *where each rule* $r_i : (\mathsf{Address} \times \mathsf{Address} \times \mathbb{N}) \to \{0, 1\}$ *maps a (sender, receiver, amount) triple to a boolean decision.*

**Definition 2** (Sound Restriction)**.** *A transfer restriction mechanism is* sound *with respect to policy* $\Pi$ *if every transfer that the mechanism permits also satisfies* $\Pi$*:* $\forall(s, r, v)$*, if* $\mathsf{detectRestriction}(s, r, v) = 0$ *then* $\forall r_i \in \Pi$*,* $r_i(s, r, v) = 1$*.*

**Definition 3** (Complete Restriction)**.** *A transfer restriction mechanism is* complete *with respect to policy* $\Pi$ *if every transfer that satisfies* $\Pi$ *is also permitted by the mechanism:* $\forall(s, r, v)$*, if* $\forall r_i \in \Pi$*,* $r_i(s, r, v) = 1$ *then* $\mathsf{detectRestriction}(s, r, v) = 0$*.*

**Theorem 1** (Soundness and Completeness)**.** *The compliance oracle transfer restriction mechanism is sound and complete with respect to the configured compliance policy* $\Pi$*, provided that the I-Chain identity registry returns correct credential status.*

*Proof.* Soundness: `detectTransferRestriction` iterates over all rules in $\Pi$ (both global and partition-specific). If any rule $r_i$ evaluates to a non-zero restriction code, the function returns that code and the transfer is blocked. A transfer is permitted (code 0x00) only if every rule returns 0x00. By construction, code 0x00 is returned by `evaluateRule` only when the rule's condition is satisfied. Therefore, a permitted transfer satisfies all rules in $\Pi$.

Completeness: If all rules in $\Pi$ are satisfied for a given transfer, then each call to `evaluateRule` returns 0x00. The loop produces no non-zero code. The function returns 0x00, permitting the transfer.

The correctness assumption on the I-Chain registry is necessary because the oracle delegates credential verification to the registry. If the registry returns incorrect status (e.g., reports an expired accreditation as valid), soundness may be violated. The I-Chain registry's correctness is maintained by the DID credential verification protocol. □

## 4 Partition Management

### 4.1 Partition Types

The Lux standard extends ERC-1400 partitions to support complex capital structures:

Table 2: Partition types and use cases.

| Partition | Use Case |
|---|---|
| `COMMON` | Common shares with standard voting and dividend rights |
| `PREFERRED_A` | Series A preferred with liquidation preference and anti-dilution |
| `PREFERRED_B` | Series B preferred with distinct terms |
| `LOCKUP_6M` | Shares subject to 6-month lockup from issuance |
| `LOCKUP_12M` | Shares subject to 12-month lockup (Rule 144) |
| `VESTING` | Shares subject to vesting schedule (employee grants) |
| `RESTRICTED` | Shares subject to regulatory hold |
| `CONVERTIBLE` | Convertible instruments (notes, SAFEs) pending conversion |
| `TREASURY` | Issuer treasury shares (not outstanding, no voting rights) |
| `ESOP` | Employee stock option pool allocation |

## 4.2 Partition Operations

Listing 4: Partition management operations.

```
contract SecurityToken is ISecurityToken {
    mapping(bytes32 => mapping(address => uint256)) internal partitionBalances;
    mapping(bytes32 => PartitionConfig) internal partitionConfigs;

    struct PartitionConfig {
        bool transferable;
        uint256 lockupExpiry;       // 0 = no lockup
        address complianceOracle;   // partition-specific oracle
        uint256 maxHolders;         // 0 = unlimited
        VestingSchedule vesting;    // empty = no vesting
        bool convertible;
        bytes32 convertTarget;      // target partition for conversion
        uint256 convertRatio;       // conversion ratio (basis points)
    }

    struct VestingSchedule {
        uint256 cliffDuration;
        uint256 totalDuration;
        uint256 startTime;
    }

    function transferByPartition(
        bytes32 partition, address from, address to,
        uint256 value, bytes calldata data
    ) external override returns (bytes32) {
        PartitionConfig memory config = partitionConfigs[partition];

        // Lockup check
        require(
            config.lockupExpiry == 0 || block.timestamp >= config.lockupExpiry,
            "partition locked"
        );

        // Vesting check
        if (config.vesting.totalDuration > 0) {
            uint256 vested = vestedAmount(partition, from);
            require(value <= vested - partitionTransferred[partition][from],
                "exceeds vested amount");
        }
```

```
40
41          // Compliance check via partition-specific or global oracle
42          address oracle = config.complianceOracle != address(0)
43              ? config.complianceOracle
44              : defaultComplianceOracle;
45
46          (bytes1 status,,) = ITransferRestriction(oracle)
47              .detectTransferRestriction(from, to, value);
48          require(status == 0x00, "transfer restricted");
49
50          // Execute transfer
51          partitionBalances[partition][from] -= value;
52          partitionBalances[partition][to] += value;
53
54          emit TransferByPartition(partition, from, to, value, data);
55          return partition;
56      }
57  }
```

# 5 Controller Operations with MPC

## 5.1 Threshold Controller

In the original Arca `st-contracts`, the controller was a single Ethereum address. If that key was compromised, the attacker could force-transfer any holder's securities. The Lux standard replaces the single controller with an MPC threshold controller.

Controller operations (forced transfers, redemptions, document updates) require a $t$-of-$n$ threshold signature from a controller quorum. The default configuration uses 2-of-3:

1. **Issuer share**: Held by the securities issuer.
2. **Platform share**: Held by the ATS operator [2].
3. **Regulatory share**: Held by the regulatory custodian or compliance officer.

Listing 5: MPC-controlled forced transfer.

```
1   contract SecurityToken is ISecurityToken {
2       address public mpcController; // MPC threshold address
3
4       modifier onlyController() {
5           require(msg.sender == mpcController, "not controller");
6           _;
7       }
8
9       function controllerTransfer(
10          address from, address to, uint256 value,
11          bytes calldata data, bytes calldata operatorData
12      ) external override onlyController {
13          // Controller can transfer regardless of restrictions
14          // Used for: court orders, regulatory seizures, error correction
15          _transfer(from, to, value);
16          emit ControllerTransfer(msg.sender, from, to, value, data, operatorData);
17      }
18
19      function controllerRedeem(
20          address holder, uint256 value,
21          bytes calldata data, bytes calldata operatorData
22      ) external override onlyController {
23          _burn(holder, value);
24          emit ControllerRedemption(msg.sender, holder, value, data, operatorData);
25      }
26  }
```

The `mpcController` address is the public key derived from the distributed key generation ceremony. No single party ever possesses the full private key.

### 5.2 Controller Action Types

Table 3: Controller actions and their legal bases.

| Action | Legal Basis |
|---|---|
| Forced transfer | Court order, regulatory directive |
| Forced redemption | Issuer buyback, regulatory delisting |
| Account freeze | SAR investigation, OFAC designation |
| Document update | Prospectus amendment, offering circular update |
| Partition migration | Corporate action (merger, conversion) |
| Emergency pause | Material adverse event, security incident |

## 6 Post-Quantum Security

### 6.1 Threat Model

Securities tokens are long-lived assets. A bond may have a 30-year maturity. A common stock may exist indefinitely. The ECDSA signatures used in standard EVM are vulnerable to Shor's algorithm, which can derive private keys from public keys in polynomial time on a cryptographically relevant quantum computer.

For securities tokens, the threat is not abstract. An attacker who records current ECDSA public keys can, once a quantum computer is available, derive the corresponding private keys and forge transfers. This "harvest now, decrypt later" attack is particularly dangerous for long-lived securities.

### 6.2 Lux PQ Precompiles

The Lux EVM includes precompiled contracts for post-quantum signature verification:

- **ETHFALCON** (precompile 0x0301): Falcon-512 signature verification, providing 128-bit post-quantum security [5].
- **Pulsar** (precompile 0x0302): Hybrid classical/post-quantum signature verification combining ECDSA with ML-DSA [1].

Listing 6: Post-quantum transfer verification.

```
contract PQSecurityToken is SecurityToken {
    // Precompile addresses
    address constant ETHFALCON = address(0x0301);
    address constant CORONA = address(0x0302);

    bool public pqRequired; // enforce PQ signatures

    function transferByPartitionPQ(
        bytes32 partition, address from, address to,
        uint256 value, bytes calldata data,
        bytes calldata pqSignature
    ) external returns (bytes32) {
        if (pqRequired) {
            bytes32 msgHash = keccak256(abi.encodePacked(
                partition, from, to, value, block.chainid, nonces[from]++
            ));
            (bool ok,) = CORONA.staticcall(
                abi.encodePacked(msgHash, pqSignature, pqPublicKeys[from])
            );
            require(ok, "PQ signature invalid");
        }

```

```
        return transferByPartition(partition, from, to, value, data);
    }
}
```

### 6.3 Migration Path

Existing securities tokens can migrate to post-quantum security incrementally:

1. **Phase 1**: Deploy PQ-aware token contracts alongside classical contracts.
2. **Phase 2**: Investors register PQ public keys via I-Chain DID updates.
3. **Phase 3**: Enable optional PQ signature verification for transfers.
4. **Phase 4**: Require PQ signatures for all transfers (set `pqRequired = true`).
5. **Phase 5**: Rotate controller MPC shares to PQ-safe threshold scheme.

## 7 Comparison with Existing Standards

Table 4: Security token standard comparison.

| Feature | Lux | ERC-3643 | ERC-1400 | ERC-1404 |
|---|---|---|---|---|
| Partitioned holdings | Yes | No | Yes | No |
| Dynamic compliance oracle | Yes | Yes (ONCHAINID) | No | No |
| MPC controller | Yes | No | No | No |
| Post-quantum signatures | Yes | No | No | No |
| Forced transfers | Yes | Yes | Yes | No |
| Document management | Yes | No | Yes | No |
| DID-based identity | Yes | Yes (ONCHAINID) | No | No |
| Hierarchical partitions | Yes | No | Partial | No |
| Vesting schedules | Yes | No | No | No |
| Convertible instruments | Yes | No | No | No |

### 7.1 ERC-3643 (T-REX)

ERC-3643, the Token for Regulated Exchanges (T-REX) standard, is the closest competitor to the Lux standard. Both use on-chain identity registries for compliance checking. Key differences:

- T-REX uses ONCHAINID, a centralized identity provider. Lux uses I-Chain DIDs with verifiable credentials from multiple providers.
- T-REX has a single controller address. Lux uses MPC threshold controllers.
- T-REX does not support partitioned holdings or post-quantum signatures.
- T-REX is deployed on Ethereum mainnet with high gas costs. Lux EVM provides sub-cent transfer costs.

## 8 Document Management

Securities require extensive documentation: prospectuses, offering circulars, subscription agreements, financial statements, and regulatory filings. The Lux standard stores document references on-chain with content hashes for integrity verification:

Listing 7: Document management.

```
contract SecurityToken is ISecurityToken {
    struct Document {
        string uri;       // IPFS or HTTPS URI
        bytes32 hash;     // SHA-256 of document content
        uint256 timestamp;
```

```
        address updatedBy; // must be MPC controller
    }

    mapping(bytes32 => Document) internal documents;
    bytes32[] internal documentNames;

    function setDocument(
        bytes32 name, string calldata uri, bytes32 hash
    ) external override onlyController {
        documents[name] = Document(uri, hash, block.timestamp, msg.sender);
        if (!documentExists[name]) {
            documentNames.push(name);
            documentExists[name] = true;
        }
        emit DocumentUpdated(name, uri, hash);
    }
}
```

Standard document names include:

- `PROSPECTUS`: The offering prospectus or offering circular.
- `SUBSCRIPTION`: The subscription agreement template.
- `PPM`: Private placement memorandum.
- `FINANCIALS`: Most recent audited financial statements.
- `LEGAL_OPINION`: Legal opinion on token characterization.
- `TAX_OPINION`: Tax opinion on token treatment.
- `TRANSFER_RESTRICTIONS`: Summary of applicable transfer restrictions.

## 9 Gas Cost Analysis

Table 5: Gas costs for security token operations on Lux EVM vs Ethereum.

| **Operation** | **Gas (Lux)** | **Gas (ETH)** | **Cost (Lux)*** |
|---|---|---|---|
| Simple transfer | 65,000 | 65,000 | \$0.003 |
| Partition transfer | 85,000 | 85,000 | \$0.004 |
| Transfer with compliance check | 120,000 | 120,000 | \$0.006 |
| Partition transfer + PQ verify | 180,000 | N/A | \$0.009 |
| Issue tokens | 95,000 | 95,000 | \$0.005 |
| Controller forced transfer | 75,000 | 75,000 | \$0.004 |
| Set document | 55,000 | 55,000 | \$0.003 |

*At Lux base gas price of 25 nLUX per gas, LUX price \$2.00.

## 10 Conclusion

The Lux Security Token Standard represents the natural evolution of the Arca `st-contracts` architecture from 2018. By combining ERC-1400 partitioned security tokens with dynamic compliance oracles, MPC threshold controllers, and post-quantum cryptographic precompiles, the standard eliminates the security and architectural limitations of first-generation security token implementations while preserving full regulatory compliance.

The standard is deployed on Lux EVM and serves as the token layer for the non-custodial ATS described in [2]. Transfer restrictions are enforced in coordination with the transfer agent registry [4], and securities are routable through the smart order router [3].

## References


[1] Z. Kelling, V. Seesahai, "M-Chain: Decentralized Multi-Party Computation Custody with Quantum-Safe Threshold Signatures," Lux Partners, 2023.

[2] Z. Kelling, "Non-Custodial Blockchain ATS: Architecture of a Regulatory-Compliant Securities Exchange," Lux Partners, 2025.

[3] Z. Kelling, "Smart Order Routing for Digital Securities: Multi-Provider Execution on Lux Network," Lux Partners, 2025.

[4] Z. Kelling, "Blockchain Transfer Agent: On-Chain Cap Table with MPC-Custodied Registry," Lux Partners, 2025.

[5] Z. Kelling, "ETHFALCON: Post-Quantum Signature Verification as EVM Precompile," Lux Partners, 2024.

[6] Z. Kelling, "Lux Credit Protocol Specification," Lux Partners, 2023.

[7] A. Dossa, P. Ruiz, F. Vogelsteller, S. Gosselin, "ERC-1400: Security Token Standard," Ethereum Improvement Proposals, 2018.

[8] R. Bose, J. Cawley, "ERC-1404: Simple Restricted Token Standard," Ethereum Improvement Proposals, 2018.

[9] J. Music, R. Music, "ERC-3643: T-REX — Token for Regulated Exchanges," Ethereum Improvement Proposals, 2021.

[10] R. Gennaro, S. Goldfeder, "One Round Threshold ECDSA with Identifiable Abort," Cryptology ePrint Archive, 2020.

[11] P. W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," FOCS, 1994.

[12] National Institute of Standards and Technology, "Post-Quantum Cryptography Standardization," NIST, 2024.