

Botho

A Privacy-Preserving Cryptocurrency with Hybrid Post-Quantum Security and Fair Economics

Version 1.0

The Botho Project
<https://botho.io>

January 9, 2026

Abstract

We present **Botho**, a privacy-preserving cryptocurrency that combines efficient ring signatures with post-quantum stealth addresses, achieving strong privacy guarantees with practical performance. Unlike existing privacy coins that either lack quantum resistance or impose impractical overhead, **Botho** employs a *hybrid architecture*: post-quantum cryptography (ML-KEM-768) protects permanent on-chain data (recipient identities), while efficient classical cryptography (CLSAG ring signatures) provides ephemeral sender anonymity.

Botho introduces a hybrid consensus mechanism combining proof-of-work block proposal with Stellar Consensus Protocol (SCP) finalization, achieving deterministic finality in approximately 5 seconds while maintaining permissionless participation. The protocol includes *progressive fees* based on coin provenance (cluster tags), creating Sybil-resistant economic incentives against wealth concentration. Transaction fees are split: 80% redistributed via lottery to random UTXOs, 20% burned—creating progressive redistribution that statistically favors smaller holders.

Key parameters: ring size 20, transaction size ~ 4 KB, 5-second finality, perpetual 2% tail emission with dynamic inflation dampening. We provide formal security proofs for transaction unlinkability, double-spend prevention, and fork freedom under standard cryptographic assumptions.

Keywords: cryptocurrency, privacy, ring signatures, post-quantum cryptography, consensus protocol, monetary policy

Contents

1	Introduction	11
1.1	The Privacy Problem	11
1.2	The Fairness Problem	11
1.3	Design Philosophy	11
1.4	Contributions	12
1.5	Paper Organization	12
2	Related Work	13
2.1	Privacy Cryptocurrencies	13
2.1.1	CryptoNote and Monero	13
2.1.2	Zcash	13
2.1.3	Zcash Orchard	14

2.1.4	Firo (Lelantus Spark)	14
2.1.5	Secret Network	15
2.1.6	Aztec	15
2.1.7	MimbleWimble	16
2.2	Post-Quantum Approaches	16
2.2.1	Quantum Threat Model	16
2.2.2	Existing Post-Quantum Cryptocurrencies	16
2.2.3	Post-Quantum Ring Signature Research	16
2.2.4	Hybrid Approaches	17
2.3	Consensus Mechanisms	17
2.3.1	Nakamoto Consensus	17
2.3.2	Classical BFT	18
2.3.3	Stellar Consensus Protocol	18
2.4	Economic Mechanisms	18
2.4.1	Fixed Supply	18
2.4.2	Tail Emission	18
2.4.3	Demurrage	18
2.4.4	Novel: Progressive Fees	18
3	Preliminaries	19
3.1	Notation	19
3.2	Cryptographic Assumptions	19
3.3	Building Blocks	19
3.3.1	Elliptic Curve Group	19
3.3.2	Pedersen Commitments	20
3.3.3	Bulletproofs	20
3.3.4	ML-KEM-768	20
3.3.5	ML-DSA-65	21
3.4	Security Definitions	21
4	Cryptographic Protocol	21
4.1	Key Hierarchy	21
4.1.1	Master Seed Generation	21
4.1.2	Account Key Derivation	22
4.1.3	Subaddress Derivation	22
4.2	Post-Quantum Stealth Addresses	22
4.2.1	Protocol Description	22
4.2.2	Security Analysis	24
4.3	Ring Signatures (CLSAG)	25
4.3.1	Ring Construction	25
4.3.2	Signature Generation	25
4.3.3	Verification	25
4.3.4	Security Properties	26
4.4	Confidential Transactions	26
4.4.1	Amount Commitment	26
4.4.2	Value Conservation	27
4.4.3	Range Proofs	27
4.5	Minting Signatures	27
4.6	Hybrid Architecture Rationale	28

5	Transaction Format	29
5.1	Transaction Types	29
5.2	Private Transaction Structure	29
5.2.1	Transaction Components	29
5.2.2	Input Structure	30
5.2.3	Output Structure	30
5.3	Minting Transaction Structure	30
5.4	Cluster Tags and Progressive Fees	31
5.4.1	Tag Vector Representation	31
5.4.2	Cluster Creation at Minting	31
5.4.3	Tag Inheritance and Blending	32
5.4.4	Age-Based Tag Decay	32
5.4.5	Cluster Factor Calculation	32
5.4.6	Ring Signature Tag Propagation	32
5.4.7	Sybil Resistance Analysis	33
5.5	Validation Rules	35
5.6	Transaction Size Analysis	35
5.7	Decoy Selection	35
5.8	Transaction Malleability	36
6	Consensus Mechanism	36
6.1	Design Rationale	36
6.1.1	Why Not Pure PoW?	36
6.1.2	Why Not Pure BFT?	36
6.1.3	Hybrid Approach	36
6.2	Stellar Consensus Protocol	37
6.2.1	Quorum Slices	37
6.2.2	Quorum Intersection	37
6.2.3	Tiered Quorum Structure	37
6.3	Consensus Phases	38
6.3.1	Phase 1: Block Proposal (PoW)	38
6.3.2	Phase 2: Nomination	39
6.3.3	Phase 3: Ballot Protocol	39
6.3.4	Phase 4: Externalize	39
6.4	Block Structure	40
6.5	Difficulty Adjustment	40
6.6	Fork Resolution	40
6.7	Timing Analysis	42
6.8	Liveness Guarantees	43
6.9	Security Properties	43
6.9.1	Byzantine Fault Tolerance	43
6.9.2	Nothing-at-Stake Resistance	43
6.9.3	Long-Range Attack Resistance	43
6.10	Comparison with Alternatives	43
6.11	Mining Pool Considerations	43
6.11.1	Pool Protocol Compatibility	44
6.11.2	Block Reward Attribution	44
6.11.3	SCP Participation	44
6.11.4	Pool Centralization Risks	44
6.11.5	Solo Mining Viability	45
6.11.6	Decentralized Pool Alternatives	45

7	Monetary Policy	45
7.1	Design Goals	45
7.2	Emission Schedule	46
7.2.1	Initial Emission	46
7.2.2	Tail Emission	46
7.2.3	Supply Projection	46
7.3	Dynamic Block Timing	46
7.3.1	Mechanism	47
7.3.2	Utilization Calculation	47
7.3.3	Economic Implications	47
7.3.4	Emission Bounds	48
7.4	Fee Economics	49
7.4.1	Base Fee	49
7.4.2	Progressive Fee Multiplier	49
7.4.3	Lottery-Based Fee Redistribution	49
7.4.4	Lottery Mechanism	50
7.4.5	Effective Inflation	50
7.4.6	Fee Flow Analysis	50
7.5	Minter Incentives	50
7.5.1	Block Reward Composition	50
7.5.2	Profitability Analysis	51
7.5.3	Decentralization Incentives	51
7.6	Long-Term Sustainability	51
7.6.1	Security Budget	51
7.6.2	Comparison with Fixed Supply	51
7.6.3	Wealth Tax Equivalence	51
7.7	Monetary Constants	52
8	Network Protocol	52
8.1	Network Architecture	52
8.1.1	Node Types	52
8.1.2	Transport Layer	52
8.2	Peer Discovery	52
8.2.1	Bootstrap Nodes	52
8.2.2	Kademlia DHT	53
8.2.3	Peer Limits	53
8.3	Message Protocol	54
8.3.1	Message Types	54
8.3.2	Message Serialization	54
8.3.3	Wire Format Specification	54
8.4	Protocol State Machine	56
8.4.1	Connection States	56
8.4.2	Message Sequences	56
8.4.3	Error Handling	57
8.5	Block Propagation	57
8.5.1	Compact Block Relay	57
8.5.2	Block Validation	58
8.6	Transaction Propagation	58
8.6.1	Dandelion++	58
8.6.2	Transaction Validation	58
8.7	Synchronization	59

8.7.1	Initial Block Download	59
8.7.2	Block Locator	59
8.7.3	Pruning	60
8.8	Denial-of-Service Protection	60
8.8.1	Rate Limiting	60
8.8.2	Peer Scoring	60
8.8.3	Resource Bounds	60
8.9	Privacy Considerations	60
8.9.1	Traffic Analysis Resistance	60
8.9.2	Tor/I2P Support	61
8.10	Light Client Security	61
8.10.1	Verification Capabilities	61
8.10.2	Trust Model	61
8.10.3	Output Scanning	62
8.10.4	Transaction Submission	62
8.10.5	Privacy Implications	62
8.10.6	Comparison to Full Node Security	63
8.10.7	Mobile Wallet Recommendations	63
8.11	Network Constants	63
9	Security Analysis	63
9.1	Threat Model	63
9.1.1	Adversary Capabilities	63
9.1.2	Security Goals	64
9.2	Privacy Analysis	64
9.2.1	Recipient Unlinkability	64
9.2.2	Sender Anonymity	64
9.2.3	Amount Confidentiality	64
9.2.4	Anonymity Set Analysis	65
9.3	Formal Privacy Framework	65
9.3.1	Privacy Definitions	65
9.3.2	BothoPrivacy Properties	65
9.3.3	Privacy Budget Analysis	66
9.3.4	Metadata Leakage Quantification	66
9.3.5	Information-Theoretic Bounds	67
9.4	Consensus Security	68
9.4.1	Double-Spend Resistance	68
9.4.2	Byzantine Fault Tolerance	69
9.4.3	51% Attack Resistance	69
9.5	Cryptographic Security	69
9.5.1	Hash Function Security	69
9.5.2	Signature Security	69
9.5.3	Key Encapsulation Security	69
9.6	Attack Resistance	70
9.6.1	Timing Attacks	70
9.6.2	Transaction Graph Analysis	70
9.6.3	Sybil Attacks	70
9.6.4	Denial of Service	70
9.6.5	Eclipse Attacks	71
9.6.6	Selfish Mining	71
9.7	Formal Security Properties	71

9.7.1	Safety	71
9.7.2	Liveness	71
9.8	Security Comparison	71
9.9	Detailed Attack Scenarios	71
9.9.1	Scenario: Timing-Based Deanonimization	72
9.9.2	Scenario: Chain Analysis Attack	72
9.9.3	Scenario: Flood-and-Loot Attack	73
9.9.4	Scenario: Lottery Grinding Attack	73
9.9.5	Scenario: Eclipse Attack on Light Client	74
9.9.6	Scenario: Progressive Fee Evasion	74
9.10	Known Limitations	75
9.10.1	Classical Ring Signature Vulnerability	75
9.10.2	Metadata Leakage	75
9.10.3	Implementation Risks	75
10	Economic Analysis	75
10.1	Incentive Alignment	76
10.1.1	Miner Incentives	76
10.1.2	User Incentives	76
10.1.3	Validator Incentives	76
10.2	Progressive Fee Analysis	76
10.2.1	Fee Distribution	76
10.2.2	Redistribution Effect	77
10.2.3	Anti-Hoarding Dynamics	77
10.3	Game-Theoretic Analysis	77
10.3.1	Miner Strategy	77
10.3.2	Validator Strategy	78
10.3.3	User Strategy	78
10.4	Long-Term Sustainability	78
10.4.1	Security Budget	78
10.4.2	Network Effect	78
10.4.3	Fee Market Stability	79
10.5	Comparison with Alternatives	79
10.5.1	Bitcoin Model	79
10.5.2	Monero Model	79
10.6	Quantitative Economic Modeling	79
10.6.1	Progressive Fee Impact on Wealth Distribution	79
10.6.2	Monte Carlo Lottery Analysis	80
10.6.3	Formal Nash Equilibrium Analysis	81
10.6.4	Economic Attack Cost Analysis	81
10.7	Economic Risks	82
10.7.1	Low Adoption	82
10.7.2	Mining Centralization	82
10.7.3	Validator Cartel	82
10.8	Economic Constants Summary	82
11	Implementation	82
11.1	Reference Implementation	83
11.1.1	Architecture Overview	83
11.1.2	Dependencies	83
11.1.3	Build Requirements	83
11.2	Performance Characteristics	84

11.2.1	Transaction Validation	84
11.2.2	Transaction Creation	84
11.2.3	Throughput	84
11.2.4	Resource Usage	84
11.3	Wallet Implementations	84
11.3.1	Command-Line Wallet	84
11.3.2	Desktop Wallet	85
11.3.3	Exchange Scanner	85
11.4	Security Measures	85
11.4.1	Memory Protection	85
11.4.2	Constant-Time Operations	85
11.4.3	Input Validation	86
11.5	Testing	86
11.5.1	Test Coverage	86
11.5.2	Test Networks	86
11.6	Deployment	86
11.6.1	Docker	86
11.6.2	Systemd	86
11.7	Scalability Analysis	86
11.7.1	Blockchain Growth	87
11.7.2	UTXO Set Growth	87
11.7.3	Key Image Database	87
11.7.4	Cluster Tag Database	87
11.7.5	Throughput Limits	88
11.7.6	Comparison to Existing Systems	88
11.7.7	Layer-2 Scaling	88
11.7.8	Pruning Effectiveness	88
11.7.9	Node Hardware Requirements	89
11.8	Future Development	89
11.8.1	Planned Improvements	89
11.8.2	Post-Quantum Ring Signatures	89
11.9	User Experience Design	89
11.9.1	Address Format	89
11.9.2	Transaction Confirmation UX	90
11.9.3	Wallet Backup and Recovery	90
11.9.4	Subaddress Management	91
11.9.5	Fee Estimation	91
11.9.6	Error Handling	91
11.9.7	Privacy Indicators	92
11.9.8	Accessibility	92
11.9.9	Internationalization	92
11.10	Testnet Deployment	92
11.10.1	Testnet Parameters	92
11.10.2	Faucet	93
11.10.3	Genesis Block	93
11.10.4	Testnet Reset Policy	93
11.11	Deployment Guide	93
11.11.1	Node Configuration	93
11.11.2	Recommended System Requirements	94
11.11.3	Security Hardening	94
11.11.4	Monitoring and Metrics	95

11.11.5 Backup and Recovery	95
11.12 Code Availability	96
12 Governance and Protocol Upgrades	96
12.1 Design Principles	96
12.1.1 Conservative by Default	96
12.1.2 Transparency and Predictability	96
12.2 Protocol Versioning	96
12.2.1 Version Scheme	96
12.2.2 Version Negotiation	97
12.3 Upgrade Mechanisms	97
12.3.1 Soft Forks	97
12.3.2 Hard Forks	97
12.4 Proposal Process	98
12.4.1 Botho Improvement Proposals (BIPs)	98
12.4.2 Proposal Lifecycle	98
12.5 Emergency Procedures	99
12.5.1 Security Vulnerabilities	99
12.5.2 Network Emergencies	99
12.6 Deprecation Policy	99
12.6.1 Feature Deprecation	99
12.6.2 API Stability	100
12.7 Decentralization Considerations	100
12.7.1 Avoiding Governance Capture	100
12.7.2 Fork Rights	100
12.8 Upgrade Roadmap	100
12.8.1 Planned Future Upgrades	100
13 Conclusion	101
13.1 Summary	101
13.1.1 Key Contributions	101
13.1.2 Design Philosophy	101
13.2 Limitations and Future Work	101
13.2.1 Current Limitations	101
13.2.2 Future Research Directions	102
13.2.3 Interoperability Architecture	102
13.3 Broader Impact	104
13.3.1 Privacy as Human Right	104
13.3.2 Economic Implications	104
13.4 Closing Remarks	104
A Notation Reference	108
A.1 Mathematical Notation	109
A.2 Protocol Identifiers	110
A.3 Network Constants	110
A.4 Cryptographic Parameters	110
A.5 Transaction Sizes	110
A.6 Monetary Parameters	110

B	Parameter Justification	111
B.1	Cryptographic Parameters	111
B.1.1	Ring Size: 20	111
B.1.2	ML-KEM Security Level: 768	111
B.1.3	ML-DSA Security Level: 65	112
B.2	Consensus Parameters	112
B.2.1	Block Time: 5–40 Seconds (Dynamic)	112
B.2.2	Quorum Thresholds: 3-of-4 Infrastructure, 2-of-3 Community	112
B.2.3	Difficulty Adjustment Window: 144 Blocks	113
B.3	Economic Parameters	113
B.3.1	Initial Block Reward: 50 BTH	113
B.3.2	Halving Period: 1,051,200 Blocks (~2 Years)	113
B.3.3	Tail Emission: 0.3 BTH per Block	114
B.3.4	Fee Split: 80% Lottery / 20% Burn	114
B.3.5	Progressive Fee Multiplier: $1\times$ to $6\times$	115
B.3.6	Tag Decay: 0.95 Rate, 720-Block Minimum Age	115
B.4	Network Parameters	115
B.4.1	Maximum Transaction Size: 100 KB	115
B.4.2	Maximum Block Size: 2 MB	115
B.4.3	Connection Limits: 8 Outbound, 117 Inbound	116
B.5	Summary Table	116
C	Regulatory Considerations	116
C.1	Privacy and Compliance Tension	116
C.1.1	Design Philosophy	116
C.2	View Key Disclosure	117
C.2.1	View Key Capabilities	117
C.2.2	Audit Scenarios	117
C.2.3	View Key Limitations	117
C.3	Selective Transparency Options	117
C.3.1	Payment Proofs	117
C.3.2	Income Proofs	118
C.3.3	Balance Proofs	118
C.4	Exchange Integration	118
C.4.1	Know Your Customer (KYC)	118
C.4.2	Travel Rule Compliance	118
C.4.3	Suspicious Activity Reporting	119
C.5	Comparison to Other Privacy Approaches	119
C.6	Jurisdictional Considerations	119
C.7	Technical Measures for Service Providers	119
C.7.1	Deposit Monitoring	120
C.7.2	Withdrawal Controls	120
C.8	Limitations and Honest Assessment	120
C.8.1	What Botho Cannot Provide	120
C.8.2	Regulatory Risk	120
C.9	Conclusion	120
D	Formal Specifications	121
D.1	State Definitions	121
D.1.1	Blockchain State	121
D.1.2	Consensus State	121
D.2	State Transition Functions	122

D.2.1	Block Application	122
D.2.2	Transaction Application	122
D.2.3	Fee Processing	123
D.3	Validity Predicates	124
D.3.1	Transaction Validity	124
D.3.2	Block Validity	125
D.4	SCP State Machine	125
D.4.1	Message Types	125
D.4.2	State Transitions	126
D.5	Invariants	127
E	Security Audit Guide	127
E.1	Security Claims	127
E.1.1	Cryptographic Security	128
E.1.2	Consensus Security	128
E.1.3	Economic Security	128
E.2	Threat Model	128
E.3	Audit Scope	128
E.3.1	Critical Components	128
E.3.2	Test Vectors	129
E.4	Known Limitations	130
E.5	Reporting	130

1 Introduction

“Motho ke motho ka batho”—A person is a person through other people.
—Sesotho proverb

1.1 The Privacy Problem

Financial surveillance has become pervasive in the digital age. Traditional banking systems create detailed records of every transaction, enabling comprehensive monitoring of individuals’ economic activities. The emergence of Bitcoin [34] and subsequent cryptocurrencies initially promised an alternative—yet most blockchain systems create permanent, public records that enable even more thorough surveillance than traditional finance.

The transparency of Bitcoin’s blockchain allows any observer to trace the flow of funds, link addresses to identities, and reconstruct complete financial histories [32]. Chain analysis companies have built entire industries around deanonymizing cryptocurrency users. This surveillance capability extends not only to law enforcement but to any party with access to blockchain data—including authoritarian governments, stalkers, and criminals seeking targets.

Privacy is not merely a preference; it is a prerequisite for human dignity. Financial privacy protects victims of domestic abuse, enables charitable giving without social pressure, preserves commercial confidentiality, and shields individuals from discrimination based on their economic activities. The Universal Declaration of Human Rights [54] recognizes privacy as a fundamental right.

Compounding this concern is the emerging quantum computing threat. Current cryptographic systems, including those protecting cryptocurrency addresses, rely on mathematical problems believed to be computationally hard for classical computers. Quantum computers, when sufficiently powerful, will render many of these protections obsolete [24, 52]. The “harvest now, decrypt later” threat model suggests that adversaries may already be collecting encrypted data for future decryption—including blockchain transactions that will remain on-chain indefinitely.

1.2 The Fairness Problem

Beyond privacy, existing cryptocurrencies exhibit troubling economic dynamics. Early adopter advantage creates extreme wealth concentration: those who acquired Bitcoin in 2010 for pennies now hold assets worth tens of thousands of dollars per coin. This dynamic transfers wealth from late adopters to early adopters without corresponding value creation.

Traditional fee markets create perverse incentives. When fees flow directly to block producers, pressure builds toward mining centralization—larger operations achieve lower marginal costs through economies of scale. Fee volatility during network congestion disproportionately harms small transactions and users in developing economies.

Fixed supply caps, while appealing for scarcity, create deflationary spirals that discourage spending and economic activity. The security budget problem looms: as Bitcoin’s block rewards diminish toward zero, network security will depend entirely on transaction fees, creating uncertainty about long-term viability.

1.3 Design Philosophy

The name **Botho** (pronounced BOH-toh) comes from the Sesotho and Setswana languages of Southern Africa, meaning *humanity*, *humaneness*, or *ubuntu*. It is a national principle of Botswana and a core philosophy across many African cultures.

The opening proverb—“*Motho ke motho ka batho*”—translates to “a person is a person through other people.” It expresses the idea that our humanity is defined by our relationships and responsibilities to one another, not by individual accumulation.

In currency design, this philosophy rejects the “number go up” mentality. Instead, **Botho** asks: *how can money serve community rather than concentrate power?*

This philosophy manifests in concrete design decisions:

- **Privacy as baseline:** All transactions are private by default, not as a premium feature. Privacy protects the dignity of all participants.
- **Pragmatic security:** Rather than maximizing any single property, we make deliberate tradeoffs based on actual threat models. Permanent data receives permanent protection; ephemeral data can use efficient classical cryptography.
- **Fair economics:** Progressive fees discourage wealth concentration. Perpetual tail emission ensures sustainable security. Fee burning aligns incentives with network health.
- **Community consensus:** The Stellar Consensus Protocol [31] enables decisions by community agreement rather than hashpower dominance.

1.4 Contributions

This paper presents **Botho**, a privacy-preserving cryptocurrency with the following novel contributions:

1. **Hybrid post-quantum architecture:** We introduce a principled framework for applying post-quantum cryptography selectively based on data lifetime. Recipient identities, which persist on-chain indefinitely, are protected by ML-KEM-768 [6, 36]. Sender privacy, which has ephemeral value, uses efficient CLSAGring signatures [22]. This achieves meaningful quantum resistance while keeping transactions practical (~ 4 KB vs. ~ 65 KB for full post-quantum).
2. **PoW + SCP consensus:** We combine proof-of-work block proposal with Stellar Consensus Protocol finalization. This achieves permissionless participation (anyone can mine) with fast deterministic finality (~ 5 seconds) and Byzantine fault tolerance.
3. **Progressive fee mechanism:** We introduce cluster-based fees that increase with wealth concentration. Unlike identity-based approaches, this preserves privacy while resisting Sybil attacks—fees are based on coin *ancestry*, not current ownership.
4. **Dynamic block timing:** Block intervals adapt to network load (5–40 seconds), creating natural inflation dampening. Low network utility produces fewer blocks and less emission; high utility produces more blocks and rewards participants.

1.5 Paper Organization

The remainder of this paper is organized as follows:

- **Section 2** reviews related work in privacy cryptocurrencies, post-quantum approaches, consensus mechanisms, and economic designs.
- **Section 3** establishes notation and reviews cryptographic building blocks.
- **Section 4** details the cryptographic protocol including stealth addresses, ring signatures, and confidential transactions.
- **Section 5** specifies transaction formats, validation rules, and the progressive fee mechanism.

- **Section 6** describes the hybrid PoW + SCP consensus mechanism.
- **Section 7** presents the monetary policy including emission schedule, dynamic timing, and fee economics.
- **Section 8** outlines the peer-to-peer network protocol.
- **Section 9** provides security analysis including threat models, privacy guarantees, and attack resistance.
- **Section 10** analyzes economic incentives and long-term sustainability.
- **Section 11** describes the reference implementation and performance characteristics.
- **Section 13** concludes with a summary and discussion of future work.

2 Related Work

2.1 Privacy Cryptocurrencies

2.1.1 CryptoNote and Monero

The CryptoNote protocol [55] introduced two foundational privacy techniques: *stealth addresses* for recipient privacy and *ring signatures* [49] for sender privacy. Monero [33], the most widely adopted CryptoNote implementation, has evolved significantly since its 2014 launch.

Monero’s current protocol uses:

- **CLSAG ring signatures** [22]: Concise Linkable Spontaneous Anonymous Group signatures, providing approximately 45% size reduction over the earlier MLSAG scheme [39, 40].
- **Pedersen commitments** [43]: Amount hiding with additive homomorphism for value conservation verification.
- **Bulletproofs** [7]: Logarithmic-size range proofs ensuring committed amounts are non-negative.
- **ECDH stealth addresses**: One-time keys derived via elliptic curve Diffie-Hellman.

Monero achieves strong privacy but faces several limitations that **Botho** addresses:

1. **No quantum resistance**: All cryptographic primitives are vulnerable to quantum attack.
2. **Probabilistic finality**: Nakamoto consensus requires multiple confirmations; reorgs remain possible.
3. **No anti-hoarding mechanism**: No economic incentive against wealth concentration.

2.1.2 Zcash

Zcash [25] pioneered the use of zero-knowledge proofs (zk-SNARKs) [4] for cryptocurrency privacy. Unlike ring signatures, which hide the sender among a set of decoys, zk-SNARKs prove transaction validity without revealing any transaction details.

Zcash’s privacy model differs fundamentally from **Botho**:

- **Opt-in privacy**: Most Zcash transactions are transparent; shielded transactions require explicit user action.

- **Trusted setup history:** The original Sprout ceremony required trust in participant integrity. Later upgrades (Sapling, Orchard) reduced but did not eliminate setup requirements.
- **Computational cost:** Proof generation requires significant computation, limiting usability on constrained devices.

Bothochooses mandatory privacy with ring signatures—simpler, faster, and requiring no trusted setup—accepting the tradeoff of probabilistic rather than cryptographic anonymity.

2.1.3 Zcash Orchard

Zcash’s Orchard protocol [26] represents the latest evolution of zk-SNARK-based privacy, deployed in 2022. Key innovations include:

- **Halo 2 proof system:** Eliminates trusted setup via recursive proof composition. Proofs verify in constant time regardless of circuit complexity.
- **Unified addresses:** Single address format supporting transparent, Sapling, and Orchard funds, simplifying user experience.
- **Pallas/Vesta curves:** Purpose-built curves enabling efficient recursive proofs without pairing-based cryptography.
- **Action-based model:** Transactions consist of “actions” that can spend and create notes simultaneously, improving privacy analysis.

Despite these advances, Orchard maintains characteristics Bothoavoids:

- **Optional shielding:** Transparent transactions remain possible and common, fragmenting the anonymity set.
- **Heavy computation:** Proof generation requires 2–5 seconds on modern hardware, challenging for mobile devices.
- **Classical cryptography:** Pallas/Vesta curves remain vulnerable to Shor’s algorithm.

2.1.4 Firo (Lelantus Spark)

Firo (formerly Zcoin) introduced Lelantus Spark [27] in 2023, combining Spark addresses with Lelantus-style one-out-of-many proofs:

- **Spark addresses:** Diversified addresses with incoming view key separation, similar to Monero’s subaddresses but with additional flexibility.
- **One-out-of-many proofs:** Alternative to ring signatures providing similar sender anonymity with different tradeoffs.
- **No trusted setup:** Like ring signatures, Lelantus requires no parameter ceremony.

Lelantus Spark offers comparable privacy to ring signatures with different performance characteristics. Botho’s choice of CLSAG reflects:

- Extensive security analysis and real-world deployment (Monero)
- Smaller proof sizes for typical transactions
- Better understood cryptographic assumptions

2.1.5 Secret Network

Secret Network [51] takes a fundamentally different approach: privacy through trusted execution environments (TEEs):

- **Intel SGX enclaves:** Computation occurs in hardware-isolated regions; even node operators cannot observe transaction details.
- **Secret contracts:** Programmable privacy with Turing-complete smart contracts—a capability pure cryptographic approaches cannot efficiently achieve.
- **Fast execution:** No proof generation overhead; privacy adds minimal latency.

The TEE approach involves different security assumptions:

- **Hardware trust:** Security depends on Intel’s implementation and absence of hardware vulnerabilities.
- **Side-channel attacks:** SGX has faced multiple side-channel attacks (Spectre, Meltdown variants, etc.) [20].
- **Centralized manufacturing:** Hardware availability depends on Intel’s production and cooperation.

Botho’s purely cryptographic approach provides:

- No hardware trust assumptions
- Mathematically verifiable security properties
- Resilience against physical side-channel attacks

2.1.6 Aztec

Aztec [2] brings privacy to Ethereum through a zk-rollup Layer 2:

- **Private L2:** Transactions are private within Aztec; only validity proofs are posted to Ethereum.
- **Composability:** Private smart contracts can interact with Ethereum DeFi through controlled interfaces.
- **PLONK proofs:** Uses efficient universal zkSNARKs with updateable trusted setup.

Aztec’s L2 approach offers interesting tradeoffs:

- **Inherits L1 security:** Settlement finality comes from Ethereum’s consensus.
- **Limited decentralization:** Sequencer sets remain somewhat centralized during early development.
- **Bridging friction:** Moving funds between L1 and L2 introduces latency and potential vulnerabilities.

Botho operates as an independent L1, avoiding bridge security concerns while sacrificing Ethereum composability.

2.1.7 MimbleWimble

MimbleWimble [46] achieves privacy through transaction cut-through and kernel aggregation. Implementations include Grin [23] and Beam [3].

Key limitations addressed by Botho:

- **Interactive transactions:** MimbleWimble requires sender and recipient to exchange data, complicating user experience.
- **Linkability attacks:** Research has demonstrated significant transaction graph leakage [17].

2.2 Post-Quantum Approaches

2.2.1 Quantum Threat Model

Shor’s algorithm [52] enables quantum computers to solve the discrete logarithm and integer factorization problems in polynomial time, breaking ECDSA, ECDH, and related schemes. Grover’s algorithm [24] provides quadratic speedup for symmetric key search, effectively halving security levels.

The “harvest now, decrypt later” threat is particularly relevant for blockchain systems: adversaries may record encrypted/signed data today for decryption when quantum computers become available. Since blockchain data is permanent and public, this threat applies to all historical transactions.

2.2.2 Existing Post-Quantum Cryptocurrencies

QRL (Quantum Resistant Ledger) [48] uses XMSS hash-based signatures. While quantum-resistant, XMSS is stateful—requiring careful key management to avoid catastrophic signature reuse.

IOTA has explored Winternitz one-time signatures, facing similar statefulness challenges.

Research proposals for post-quantum ring signatures [14, 53] exist but impose significant size overhead ($\sim 50\times$ classical signatures).

2.2.3 Post-Quantum Ring Signature Research

The search for practical post-quantum ring signatures remains an active research area. Current approaches include:

Lattice-based constructions [13]:

- Based on Ring-LWE or Module-LWE problems
- Signature sizes: 15–50 KB for ring size 16
- Verification: ~ 10 –50 ms
- Primary challenge: Size scales poorly with ring size

Hash-based ring signatures:

- Rely only on hash function security (conservative assumption)
- Even larger signatures than lattice approaches
- Better understood security reductions

Isogeny-based constructions:

- Based on supersingular isogeny problems
- More compact than lattice approaches
- Slower computation; SIDH attacks [9] have reduced confidence in isogeny hardness assumptions

Comparison to Botho’s CLSAG:

Scheme	Ring 16	Ring 32	Quantum Safe
CLSAG	704 B	1,376 B	No
Lattice Ring (typical)	35 KB	70 KB	Yes
Hash-based Ring	100+ KB	200+ KB	Yes

Timeline assessment: Practical post-quantum ring signatures achieving sub-10 KB sizes for ring size 16 remain 5–10 years away based on current research trajectories. Botho’s hybrid architecture is designed to accommodate migration when such constructions mature, while providing immediate quantum resistance for permanent on-chain data (recipient identities).

Migration path: Botho’s modular cryptographic design allows future protocol upgrades to replace CLSAG with post-quantum alternatives. The signature abstraction layer (Section 11) isolates ring signature specifics from transaction processing, enabling migration via soft fork when practical PQ ring signatures emerge.

2.2.4 Hybrid Approaches

NIST’s post-quantum cryptography standardization [38] produced ML-KEM (formerly Kyber) [6, 36] for key encapsulation and ML-DSA (formerly Dilithium) [12, 37] for signatures. These lattice-based schemes provide strong post-quantum security with reasonable performance.

Botho’s hybrid approach applies these standards strategically:

- **ML-KEM-768** for stealth addresses: Recipient identity is permanent on-chain data requiring permanent protection.
- **ML-DSA-65** for minting: Block producer identity is public and must be unforgeable long-term.
- **CLSAG** for ring signatures: Sender privacy is ephemeral; classical cryptography provides efficiency.

This selective application achieves meaningful quantum resistance while maintaining practical transaction sizes.

2.3 Consensus Mechanisms

2.3.1 Nakamoto Consensus

Bitcoin’s Nakamoto consensus [34] achieves agreement through proof-of-work and longest-chain selection. Its key properties:

- **Permissionless:** Anyone can participate by providing hashpower.
- **Probabilistic finality:** Transactions become exponentially harder to reverse with additional confirmations, but reorgs remain theoretically possible.
- **Slow:** Bitcoin’s 10-minute blocks and 6-confirmation standard yield ~60-minute finality.

2.3.2 Classical BFT

Byzantine Fault Tolerant protocols like PBFT [8] achieve deterministic finality with $3f+1$ nodes tolerating f Byzantine failures [29]. However, classical BFT requires:

- Known, fixed participant set
- $O(n^2)$ message complexity
- Synchrony assumptions for liveness

These requirements conflict with permissionless cryptocurrency design.

2.3.3 Stellar Consensus Protocol

SCP [31] introduces *federated Byzantine agreement*, achieving BFT-like properties with open membership:

- **Quorum slices:** Each node declares which nodes it trusts.
- **Quorum intersection:** Safety requires overlapping quorums.
- **Open membership:** New nodes can join by declaring trust relationships.
- **Fast finality:** Consensus typically completes in seconds.

Both combines PoW block proposal (for permissionless participation and fair distribution) with SCP finalization (for fast deterministic finality).

2.4 Economic Mechanisms

2.4.1 Fixed Supply

Bitcoin's 21 million coin cap creates absolute scarcity but raises long-term security concerns. As block rewards diminish, network security depends increasingly on transaction fees. If fee revenue proves insufficient, security may degrade.

2.4.2 Tail Emission

Monero's perpetual tail emission (~ 0.6 XMR per block indefinitely) ensures permanent miner incentive. This trades absolute scarcity for security sustainability—a tradeoff **Both** embraces.

2.4.3 Demurrage

Demurrage currencies [21] (Freigeld, Chiemgauer) impose holding costs through time-based value decay. While economically interesting, cryptographic implementation is challenging and user experience suffers.

2.4.4 Novel: Progressive Fees

Both introduces *cluster-based progressive fees*—a novel mechanism that:

- Increases transaction costs for concentrated wealth
- Tracks coin *ancestry* rather than current ownership
- Resists Sybil attacks (splitting coins doesn't reduce fees)
- Preserves privacy (no identity required)

Combined with fee burning, this creates economic pressure toward circulation without explicit demurrage.

3 Preliminaries

3.1 Notation

We establish the following notation used throughout this paper:

Symbol	Description
λ	Security parameter
\mathbb{G}	Elliptic curve group (Ristretto255)
G	Generator of \mathbb{G}
H	Secondary generator for Pedersen commitments
q	Order of \mathbb{G} (prime)
\mathbb{Z}_q	Integers modulo q
R_q	Polynomial ring $\mathbb{Z}_q[X]/(X^n + 1)$ for lattice operations
a, b	View and spend private keys (scalars)
A, B	View and spend public keys (points)
(C, D)	Subaddress public key pair
P	One-time public key
x	One-time private key
I	Key image
$H_s(\cdot)$	Hash function to scalar
$H_p(\cdot)$	Hash function to curve point
$\mathcal{H}(\cdot)$	General cryptographic hash
$\{P_i\}_{i=0}^{n-1}$	Ring of public keys
π	Secret index of real key in ring
σ	Signature
C_v	Pedersen commitment to value v
$\xleftarrow{\$}$	Uniform random sampling
\parallel	Concatenation
$\text{negl}(\lambda)$	Negligible function in λ

3.2 Cryptographic Assumptions

The security of Bothorelies on the following computational assumptions:

Definition 3.1 (Discrete Logarithm Problem (DLP)). Given $G \in \mathbb{G}$ and $Y = xG$ for random $x \xleftarrow{\$} \mathbb{Z}_q$, compute x .

Definition 3.2 (Decisional Diffie-Hellman (DDH)). Distinguish (G, aG, bG, abG) from (G, aG, bG, cG) for random $a, b, c \xleftarrow{\$} \mathbb{Z}_q$.

Definition 3.3 (Module Learning With Errors (MLWE)). Given $(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e})$ where $\mathbf{A} \xleftarrow{\$} R_q^{k \times l}$, $\mathbf{s} \xleftarrow{\$} R_q^l$, and \mathbf{e} is a small error vector, distinguish from (\mathbf{A}, \mathbf{u}) where $\mathbf{u} \xleftarrow{\$} R_q^k$.

Definition 3.4 (Module Short Integer Solution (MSIS)). Given $\mathbf{A} \xleftarrow{\$} R_q^{k \times l}$, find $\mathbf{x} \in R_q^l$ with $\mathbf{A}\mathbf{x} = \mathbf{0}$ and $\|\mathbf{x}\| \leq \beta$ for bound β .

3.3 Building Blocks

3.3.1 Elliptic Curve Group

Bothouses the Ristretto255 group [11], a prime-order group constructed from Curve25519 [5]. Ristretto eliminates cofactor-related complexities, providing a clean abstraction with:

- Prime order $q = 2^{252} + 27742317777372353535851937790883648493$
- Efficient constant-time operations
- Resistance to small-subgroup attacks

3.3.2 Pedersen Commitments

A Pedersen commitment [43] to value v with blinding factor r is:

$$C = vH + rG \quad (1)$$

where G and H are independent generators (the discrete log of H with respect to G is unknown).

Theorem 3.5 (Pedersen Commitment Properties). *Pedersen commitments satisfy:*

1. **Hiding** (information-theoretic): C reveals nothing about v without r .
2. **Binding** (computational): Finding $(v', r') \neq (v, r)$ with the same commitment requires solving DLP.
3. **Homomorphic**: $C_1 + C_2 = (v_1 + v_2)H + (r_1 + r_2)G$.

The homomorphic property enables verification that transaction inputs equal outputs plus fee, without revealing individual amounts.

3.3.3 Bulletproofs

Bulletproofs [7] provide zero-knowledge range proofs with logarithmic size. For a commitment $C = vH + rG$, a Bulletproof proves:

$$v \in [0, 2^{64}) \quad (2)$$

This prevents negative amounts (which would allow coin creation) while keeping amounts confidential.

Aggregation: Multiple range proofs can be aggregated, with size growing logarithmically in the number of proofs. A transaction with m outputs has proof size $O(\log m)$.

3.3.4 ML-KEM-768

ML-KEM (Module-Lattice-based Key-Encapsulation Mechanism) [6, 36], formerly known as Kyber, is a NIST-standardized post-quantum key encapsulation mechanism.

- $(pk, sk) \leftarrow \text{ML-KEM.KeyGen}()$: Generate keypair
- $(c, K) \leftarrow \text{ML-KEM.Encap}(pk)$: Encapsulate to shared secret
- $K \leftarrow \text{ML-KEM.Decap}(sk, c)$: Decapsulate to recover secret

ML-KEM-768 provides approximately 192-bit security against quantum attacks, with:

- Public key size: 1,184 bytes
- Ciphertext size: 1,088 bytes
- Shared secret size: 32 bytes

3.3.5 ML-DSA-65

ML-DSA (Module-Lattice-based Digital Signature Algorithm) [12, 37], formerly known as Dilithium, is a NIST-standardized post-quantum signature scheme.

- $(pk, sk) \leftarrow \text{ML-DSA.KeyGen}()$: Generate keypair
- $\sigma \leftarrow \text{ML-DSA.Sign}(sk, m)$: Sign message
- $\{0, 1\} \leftarrow \text{ML-DSA.Verify}(pk, m, \sigma)$: Verify signature

ML-DSA-65 (security level 3) provides approximately 128-bit security against quantum attacks, with:

- Public key size: 1,952 bytes
- Secret key size: 4,032 bytes
- Signature size: 3,309 bytes

3.4 Security Definitions

Definition 3.6 (Unforgeability (EUF-CMA)). A signature scheme is *existentially unforgeable under chosen message attack* if no PPT adversary, given access to a signing oracle, can produce a valid signature on a message not queried to the oracle, except with negligible probability.

Definition 3.7 (Anonymity (Ring Signatures)). A ring signature scheme provides *anonymity* if no PPT adversary can identify the actual signer among ring members with probability significantly better than $1/n$, where n is the ring size.

Definition 3.8 (Linkability). A ring signature scheme is *linkable* if there exists an efficient algorithm that, given two signatures from the same secret key, outputs 1; and given signatures from different keys, outputs 0 (with high probability).

Definition 3.9 (IND-CCA2 Security). A key encapsulation mechanism is *IND-CCA2 secure* if no PPT adversary with access to a decapsulation oracle can distinguish real from random shared secrets, except with negligible advantage.

4 Cryptographic Protocol

4.1 Key Hierarchy

Both derives all cryptographic keys from a single BIP39 [42] mnemonic using SLIP-10 [50] hierarchical derivation.

4.1.1 Master Seed Generation

A 24-word mnemonic encodes 256 bits of entropy. The master seed is derived via:

$$\text{seed} = \text{PBKDF2-SHA512}(\text{mnemonic}, \text{"mnemonic"} \parallel \text{passphrase}, 2048) \quad (3)$$

This produces a 512-bit seed from which all keys are derived.

4.1.2 Account Key Derivation

Using SLIP-10 hardened derivation with path $m/44'/866'/\text{account}'$:

$$\text{view_key_material} = \text{SLIP10}(\text{seed}, m/44'/866'/0'/0') \quad (4)$$

$$\text{spend_key_material} = \text{SLIP10}(\text{seed}, m/44'/866'/0'/1') \quad (5)$$

These materials are then processed through HKDF-SHA512 [28] to produce the final Ristretto255 scalars:

$$a = \text{HKDF}(\text{view_key_material}, \text{"botho-ristretto255-view"}) \mod q \quad (6)$$

$$b = \text{HKDF}(\text{spend_key_material}, \text{"botho-ristretto255-spend"}) \mod q \quad (7)$$

The corresponding public keys are:

$$A = aG, \quad B = bG \quad (8)$$

The public address is the tuple (A, B) .

4.1.3 Subaddress Derivation

Subaddresses allow generation of unlimited unlinkable addresses from a single master key pair. For subaddress index i :

$$\delta_i = H_s(\text{"SubAddr"} \| a \| i) \quad (9)$$

$$c_i = a + \delta_i \mod q \quad (10)$$

$$d_i = b + \delta_i \mod q \quad (11)$$

The subaddress public key pair is:

$$(C_i, D_i) = (c_iG, d_iG) = (A + \delta_iG, B + \delta_iG) \quad (12)$$

Theorem 4.1 (Subaddress Unlinkability). *Without knowledge of the view key a , subaddresses (C_i, D_i) and (C_j, D_j) for $i \neq j$ are computationally indistinguishable from independent random points.*

4.2 Post-Quantum Stealth Addresses

Traditional CryptoNote stealth addresses use ECDH for key exchange. Bothoreplaces this with ML-KEM-768 to achieve post-quantum recipient privacy.

4.2.1 Protocol Description

Let the recipient have address (A, B) where A is interpreted as an ML-KEM public key (derived via a domain-separated hash from the Ristretto point).

Sender (creating output for recipient):

1. Derive ML-KEM public key: $\text{pk}_{\text{kem}} = \text{DeriveKEM}(A)$
2. Encapsulate: $(c, K) \leftarrow \text{ML-KEM.Encap}(\text{pk}_{\text{kem}})$
3. Compute scalar: $s = H_s(K \| \text{output_index})$

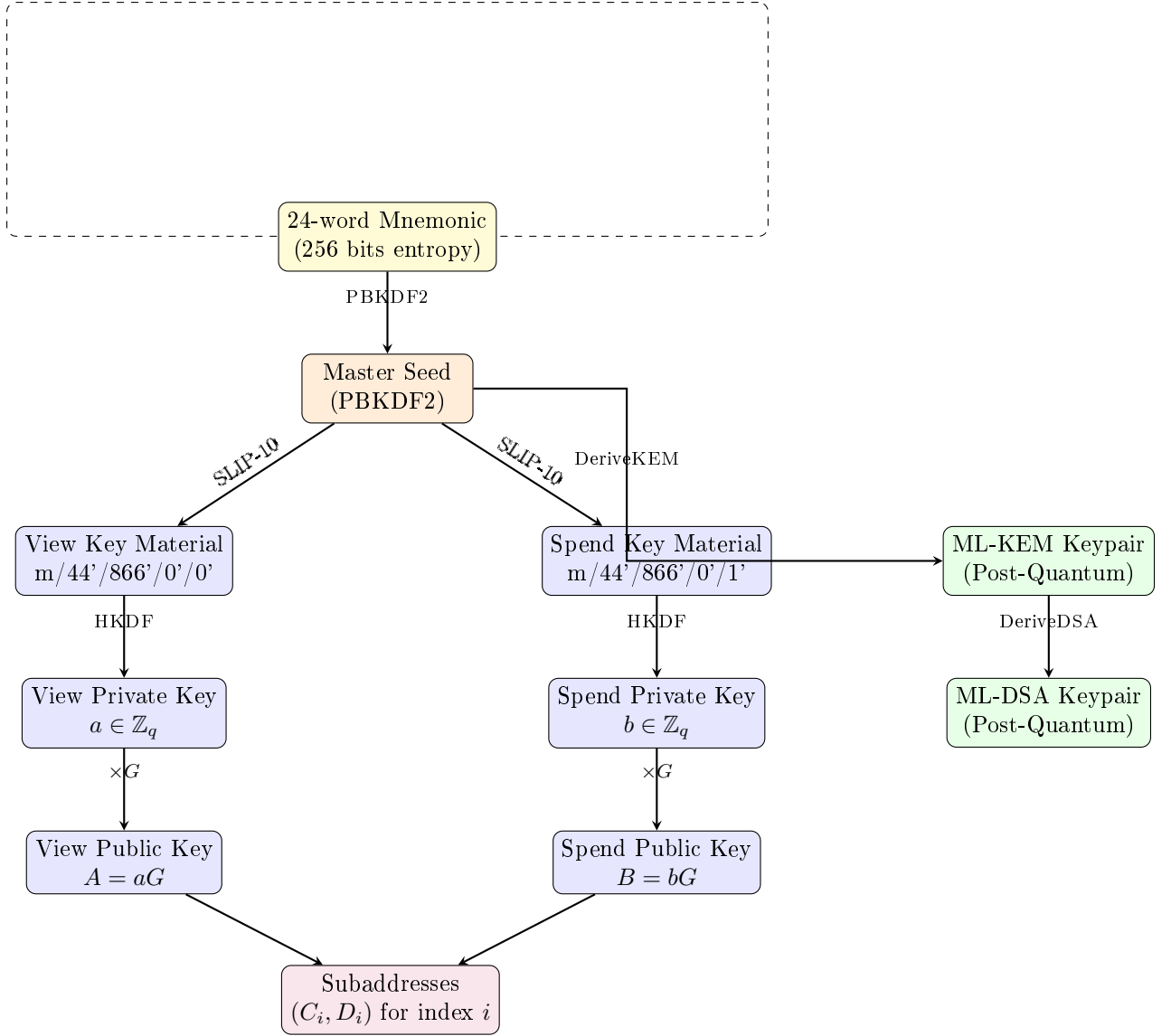


Figure 1: Key derivation hierarchy. A single BIP39 mnemonic derives all keys via SLIP-10 paths and HKDF domain separation. Classical Ristretto255 keys support ring signatures, while post-quantum ML-KEM/ML-DSA keys protect recipient privacy and minting operations.

4. Compute one-time public key: $P = sG + B$
5. Include ciphertext c (1,088 bytes) in transaction output

Recipient (scanning for received outputs):

1. Derive ML-KEM secret key: $\mathbf{sk}_{\text{kem}} = \text{DeriveKEM}(a)$
2. For each output with ciphertext c :
 - (a) Decapsulate: $K \leftarrow \text{ML-KEM.Decap}(\mathbf{sk}_{\text{kem}}, c)$
 - (b) Compute scalar: $s' = H_s(K \parallel \text{output_index})$
 - (c) Compute expected key: $P' = s'G + B$
 - (d) If $P' = P$ (output's public key), this output belongs to us
 - (e) Compute private key: $x = s' + b \mod q$

Verification: The one-time private key x satisfies $xG = (s + b)G = sG + B = P$.

4.2.2 Security Analysis

Theorem 4.2 (Recipient Unlinkability). *Under the IND-CCA2 security of ML-KEM-768, an adversary (including a quantum adversary) cannot link outputs to recipients with probability better than negligible, given only the blockchain.*

Proof. We prove by reduction to ML-KEM IND-CCA2 security. Suppose adversary \mathcal{A} can distinguish outputs belonging to recipient R with non-negligible advantage ϵ . We construct adversary \mathcal{A}' that breaks ML-KEM IND-CCA2 with advantage $\epsilon/2$.

Setup: \mathcal{A}' receives ML-KEM public key \mathbf{pk}^* from the IND-CCA2 challenger. \mathcal{A}' sets this as the target recipient's KEM public key.

Query phase: \mathcal{A} may request outputs for various recipients. For non-target recipients, \mathcal{A}' generates honestly. For the target recipient, \mathcal{A}' uses the decapsulation oracle.

Challenge: \mathcal{A} submits two recipients R_0, R_1 (one being the target). \mathcal{A}' queries the IND-CCA2 challenger with $(\mathbf{pk}^*, \mathbf{pk}_{\text{other}})$ and receives ciphertext c^* encapsulating either K_0 or K_1 (random bit b).

\mathcal{A}' computes $s^* = H_s(K_b \parallel \text{index})$ and $P^* = s^*G + B_b$, returning output (c^*, P^*) to \mathcal{A} .

Analysis: If \mathcal{A} correctly identifies the recipient with probability $1/2 + \epsilon$, then \mathcal{A}' correctly guesses b with the same probability, contradicting IND-CCA2 security.

The reduction is tight: $\text{Adv}_{\mathcal{A}'}^{\text{IND-CCA2}} \geq \epsilon/2$. \square

Corollary 4.3 (Quantum Security). *The above reduction holds against quantum adversaries since ML-KEM-768 provides IND-CCA2 security in the quantum random oracle model (QROM).*

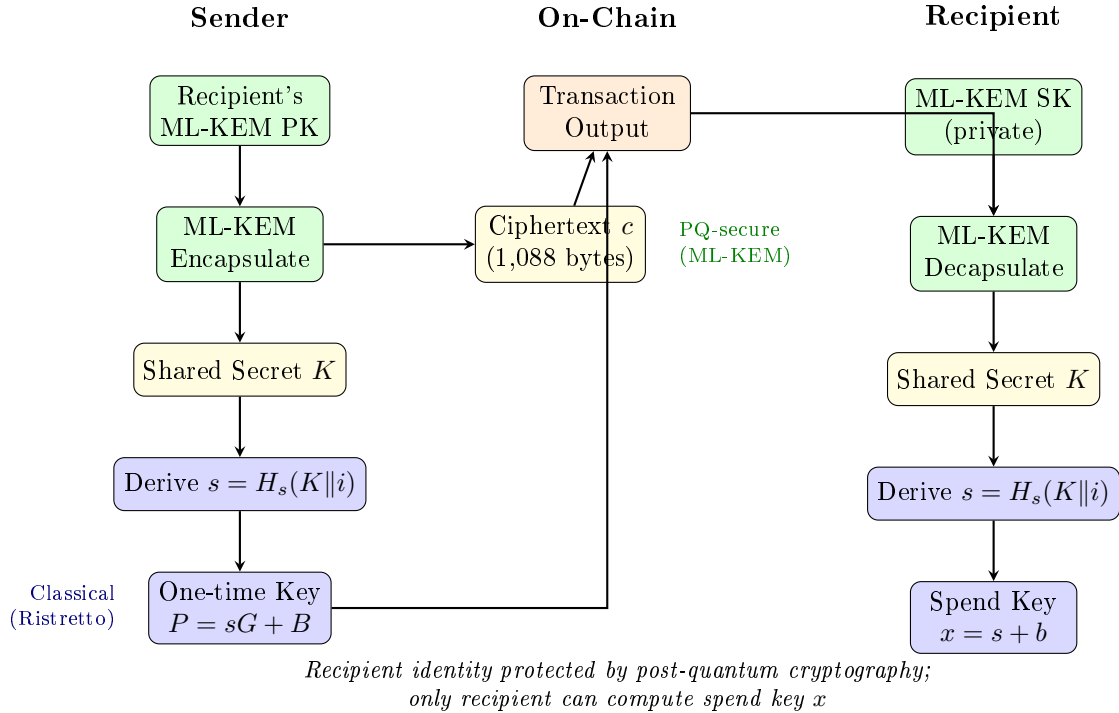


Figure 2: Post-quantum stealth address protocol. The sender encapsulates to the recipient's ML-KEM public key, deriving a shared secret used to create a unique one-time public key P . The on-chain ciphertext c is quantum-resistant—only the recipient with the ML-KEM secret key can decapsulate and derive the corresponding private key x to spend the output.

4.3 Ring Signatures (CLSAG)

CLSAG(Concise Linkable Spontaneous Anonymous Group) [22] provides efficient linkable ring signatures [19] for sender privacy.

4.3.1 Ring Construction

For each input being spent, the sender:

1. Selects $n - 1$ decoy outputs from the blockchain (we use $n = 20$)
2. Forms a ring $\mathcal{R} = \{(P_0, C_0), \dots, (P_{n-1}, C_{n-1})\}$ where each (P_i, C_i) is a one-time public key and commitment
3. The real input is at secret index π

4.3.2 Signature Generation

Given:

- Ring \mathcal{R} with real index π
- Private key x_π and commitment blinding factor z_π
- Message m (transaction hash)

Key Image: $I = x_\pi \cdot H_p(P_\pi)$

The key image is deterministic given the private key and serves as a unique tag preventing double-spending.

Aggregation Coefficients:

$$\mu_P = \mathcal{H}(\text{"CLSAG_agg_0"} \parallel \mathcal{R} \parallel I \parallel D) \quad (13)$$

$$\mu_C = \mathcal{H}(\text{"CLSAG_agg_1"} \parallel \mathcal{R} \parallel I \parallel D) \quad (14)$$

where D is an auxiliary point for commitment verification.

Signature: The signature $\sigma = (c_0, s_0, \dots, s_{n-1}, I, D)$ is computed via a Fiat-Shamir transform [18] of an interactive protocol, forming a “ring” of challenges that closes only if the prover knows one of the private keys.

4.3.3 Verification

Given signature σ and ring \mathcal{R} :

1. Recompute aggregation coefficients μ_P, μ_C
2. For $i = 0, \dots, n - 1$:

$$W_i = \mu_P P_i + \mu_C (C_i - C_{\text{out}}) \quad (15)$$

$$L_i = s_i G + c_i W_i \quad (16)$$

$$R_i = s_i H_p(P_i) + c_i (\mu_P I + \mu_C D) \quad (17)$$

$$c_{i+1} = \mathcal{H}(\text{"CLSAG_round"} \parallel \mathcal{R} \parallel L_i \parallel R_i \parallel m) \quad (18)$$

3. Accept if $c_n = c_0$ (ring closure)

4.3.4 Security Properties

Definition 4.4 (Unforgeability Game). The unforgeability experiment $\text{Exp}_{\mathcal{A}}^{\text{UNF}}$ proceeds:

1. Challenger generates key pairs (x_i, P_i) for $i \in [n]$
2. \mathcal{A} receives $\{P_i\}$ and access to signing oracle
3. \mathcal{A} outputs $(m^*, \sigma^*, \mathcal{R}^*)$
4. \mathcal{A} wins if $\text{Verify}(\sigma^*, m^*, \mathcal{R}^*) = 1$ and m^* was never queried to signing oracle with ring \mathcal{R}^*

Definition 4.5 (Anonymity Game). The anonymity experiment $\text{Exp}_{\mathcal{A}}^{\text{ANON}}$ proceeds:

1. Challenger generates key pairs, gives public keys to \mathcal{A}
2. \mathcal{A} selects ring \mathcal{R} , message m , two indices i_0, i_1
3. Challenger flips bit b , signs with key x_{i_b}
4. \mathcal{A} outputs guess b'
5. \mathcal{A} 's advantage: $|\Pr[b' = b] - 1/2|$

Theorem 4.6 (CLSAG Security). *Under the DLP assumption in the random oracle model, CLSAG satisfies:*

1. **Unforgeability:** For all PPT \mathcal{A} , $\Pr[\text{Exp}_{\mathcal{A}}^{\text{UNF}} = 1] \leq \text{negl}(\lambda)$.
2. **Anonymity:** For all PPT \mathcal{A} , $\text{Adv}_{\mathcal{A}}^{\text{ANON}} \leq \text{negl}(\lambda)$.
3. **Linkability:** For all PPT \mathcal{A} , $\Pr[\text{same key, different images}] \leq \text{negl}(\lambda)$ and $\Pr[\text{different keys, same image}] \leq \text{negl}(\lambda)$.

Proof (Unforgeability). By reduction to DLP. Given DLP instance $(G, Y = xG)$, embed Y as one ring member's public key. A forking lemma argument shows that a successful forger can be rewound to extract the discrete log, contradicting DLP hardness. Full proof in [22]. \square

Proof (Anonymity). The signature is a Fiat-Shamir transform of a Σ -protocol. In the random oracle model, the simulated transcript is indistinguishable from real, regardless of which key was used. The ring structure ensures all positions are computationally indistinguishable. \square

Proof (Linkability). Key image $I = x \cdot H_p(P)$ is deterministic given (x, P) . For the same key, the same image is always produced. For different keys $x \neq x'$, collision requires $x \cdot H_p(P) = x' \cdot H_p(P')$, which occurs with probability $1/q$ (negligible) when H_p is modeled as a random oracle. \square

4.4 Confidential Transactions

4.4.1 Amount Commitment

Each output commits to its amount v with blinding factor r :

$$C = vH + rG \tag{19}$$

The blinding factor r is derived deterministically from the shared secret to enable recipient recovery.

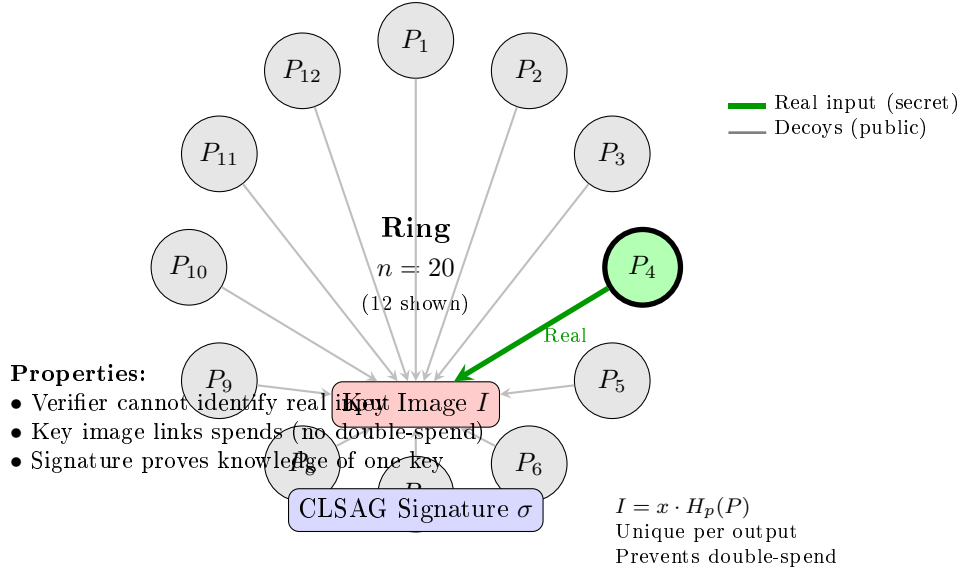


Figure 3: Ring signature structure. The real input (green) is hidden among 19 decoy outputs selected from the blockchain. The signature proves the signer knows one of the private keys without revealing which. The key image I is deterministically derived from the real input, enabling double-spend detection while preserving anonymity.

4.4.2 Value Conservation

For a transaction with inputs $\{C_{\text{in}}^{(i)}\}$ and outputs $\{C_{\text{out}}^{(j)}\}$, value conservation requires:

$$\sum_i C_{\text{in}}^{(i)} = \sum_j C_{\text{out}}^{(j)} + fH \quad (20)$$

where f is the transaction fee (public). This holds if and only if:

$$\sum_i v_{\text{in}}^{(i)} = \sum_j v_{\text{out}}^{(j)} + f \quad (21)$$

Validators check commitment arithmetic without learning individual values.

4.4.3 Range Proofs

Each output includes a Bulletproof demonstrating:

$$v_{\text{out}}^{(j)} \in [0, 2^{64}) \quad (22)$$

This prevents:

- Negative amounts (which would create coins from nothing)
- Overflow attacks (amounts wrapping around)

4.5 Minting Signatures

Block rewards (minting transactions) use ML-DSA-65 signatures since the minter's identity is public and must be verifiable long-term.

$$\sigma_{\text{mint}} = \text{ML-DSA.Sign}(\text{sk}_{\text{minter}}, \text{block_hash} \parallel \text{nonce}) \quad (23)$$

The minter's ML-DSA public key is derived from the same seed as their Ristretto keys, enabling unified key management.

Value Conservation with Hidden Amounts

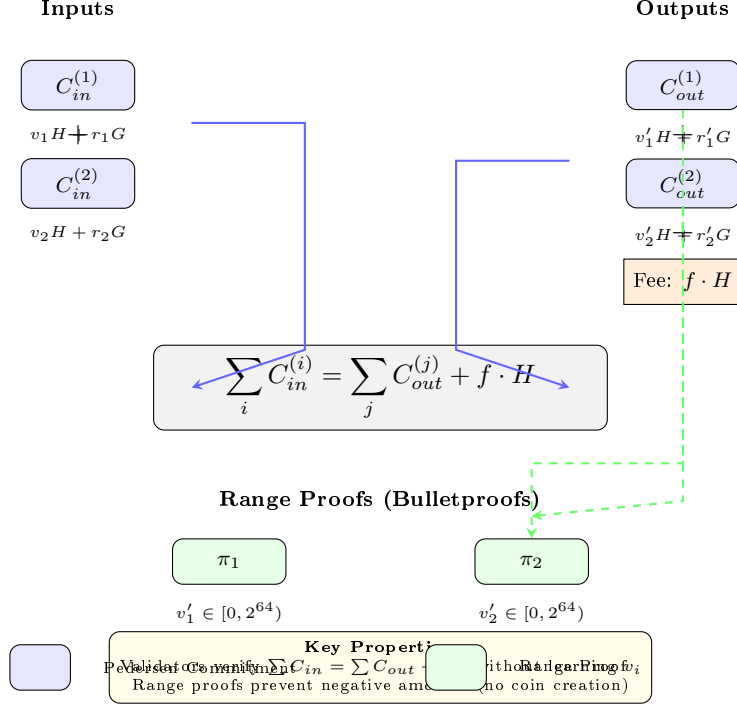


Figure 4: Confidential transaction structure. Input and output values are hidden in Pedersen commitments ($C = vH + rG$). Value conservation is verified via commitment arithmetic: the sum of input commitments must equal output commitments plus the public fee. Bulletproof range proofs ensure all output amounts are non-negative, preventing coin creation through overflow attacks.

4.6 Hybrid Architecture Rationale

Table 1: Cryptographic choices by data lifetime

Data	Lifetime	Algorithm	Rationale
Recipient identity	Permanent	ML-KEM-768	On-chain forever; must be PQ
Sender identity	Ephemeral	CLSAG	Value degrades; efficiency wins
Amounts	Permanent	Pedersen	Information-theoretic hiding
Minting authority	Permanent	ML-DSA-65	Verifiable long-term

Why not full post-quantum?

Post-quantum ring signatures (e.g., lattice-based constructions) impose approximately $50\times$ size overhead. A CLSAG signature is ~ 700 bytes per input; a comparable lattice ring signature would be ~ 35 KB. With multiple inputs, transactions would exceed 100 KB, making desktop nodes impractical.

Why is ephemeral sender privacy acceptable?

The value of sender deanonymization degrades over time. Learning who sent a transaction in 2025 from a 2045 perspective has minimal economic relevance—the goods have been delivered, contracts fulfilled, and economic context forgotten. In contrast, recipient identity remains valuable (“who owns this address?”) indefinitely.

This asymmetry justifies asymmetric protection: permanent data gets permanent (post-quantum) protection; ephemeral data gets efficient (classical) protection.

5 Transaction Format

5.1 Transaction Types

Both supports two transaction types:

Table 2: Transaction type comparison

Type	Inputs	Outputs	Signature	Size	Use Case
Minting	0	1–16	ML-DSA-65	~2 KB	Block rewards
Private	1–16	1–16	CLSAG	~4 KB	All transfers

5.2 Private Transaction Structure

A private transaction transfers value while hiding sender, recipient, and amount.

Private Transaction Structure (1-in-2-out)

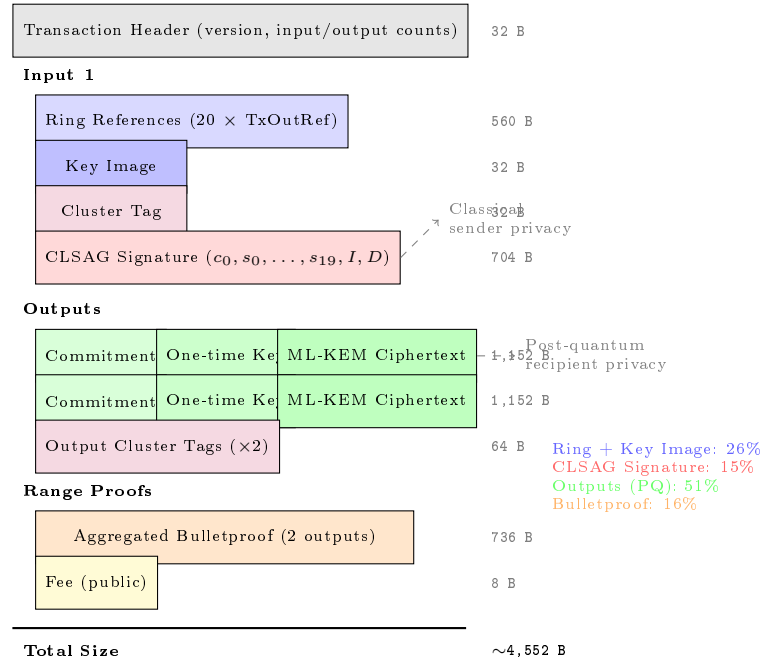


Figure 5: Anatomy of a private transaction (1 input, 2 outputs). The ML-KEM ciphertext dominates output size (1,088 bytes each) but provides post-quantum recipient privacy. Ring signatures use classical CLSAG for efficiency. Total size is approximately 4.5 KB, compared to ~50 KB with post-quantum ring signatures.

5.2.1 Transaction Components

```
PrivateTransaction
  prefix: TransactionPrefix,
  ring_signatures: Vec<CLSAGSignature>,
  bulletproofs: AggregatedRangeProof,
```

```
TransactionPrefix
  version: u8,
  inputs: Vec<TxInput>,
```

```

    outputs: Vec<TxOutput>,
    fee: u64,
    extra: Vec<u8>,

```

5.2.2 Input Structure

Each input references a previously unspent output without revealing which:

```

TxInput
    ring: [TxOutRef; RING_SIZE], // RING_SIZE = 20
    key_image: KeyImage,          // 32 bytes
    cluster_tag: ClusterTag,      // 32 bytes

```

```

TxOutRef
    block_height: u64,
    tx_index: u16,
    output_index: u8,

```

The `ring` contains references to 20 possible source outputs. The `key_image` uniquely identifies the spent output (for double-spend prevention) without revealing which ring member it corresponds to.

5.2.3 Output Structure

Each output contains the encrypted amount and one-time keys:

```

TxOutput
    commitment: CompressedPoint, // Pedersen commitment (32 bytes)
    one_time_key: CompressedPoint, // One-time public key (32 bytes)
    encrypted_amount: [u8; 32], // AES-encrypted amount
    ml_kem_ciphertext: [u8; 1088], // ML-KEM ciphertext
    cluster_tag: ClusterTag, // Derived from input tags

```

5.3 Minting Transaction Structure

Minting transactions create new coins as block rewards:

```

MintingTransaction
    block_height: u64,
    nonce: u64,
    minter_public_key: MLDSAPublicKey, // 1,952 bytes
    outputs: Vec<TxOutput>,
    signature: MLDSASignature, // 3,309 bytes

```

Unlike private transactions:

- No inputs (new coins created)
- Public amounts (for supply auditability)
- Known sender (minter identity is public)
- Recipients still hidden via stealth addresses

5.4 Cluster Tags and Progressive Fees

Cluster tags enable Sybil-resistant progressive fees by tracking coin *provenance* rather than current ownership. The key insight is that splitting coins does not change where they came from.

Cluster Tag Inheritance via Value-Weighted Blending

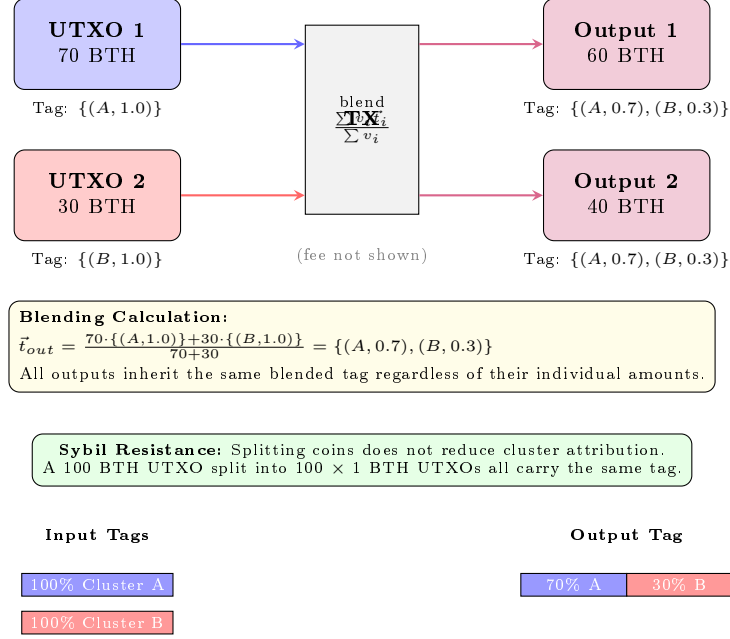


Figure 6: Cluster tag inheritance through value-weighted blending. When a transaction combines inputs from different clusters, the output tag vector is the value-weighted average of input tags. This preserves provenance information and ensures that splitting coins cannot reduce the cluster factor used for progressive fee calculation.

5.4.1 Tag Vector Representation

Each UTXO carries a *tag vector* representing the weighted distribution of its ancestry across all clusters:

$$\vec{t} = \{(c_1, w_1), (c_2, w_2), \dots, (c_k, w_k)\} \quad \text{where } \sum_i w_i = 1 \quad (24)$$

Here c_i is a cluster identifier and w_i is the fraction of the UTXO's value attributable to that cluster.

5.4.2 Cluster Creation at Minting

Each minting transaction creates a new cluster. The minter's public key hash serves as the cluster identifier:

$$c_{new} = \mathcal{H}(\text{minter_pubkey} \parallel \text{block_height}) \quad (25)$$

Minting outputs carry a tag vector with 100% attribution to the new cluster: $\vec{t}_{mint} = \{(c_{new}, 1.0)\}$.

5.4.3 Tag Inheritance and Blending

When spending multiple inputs, output tags are computed via value-weighted blending:

$$\vec{t}_{\text{out}} = \frac{\sum_i v_i \cdot \vec{t}_i}{\sum_i v_i} \quad (26)$$

where v_i is the value of input i and \vec{t}_i is its tag vector.

Example: Combining a 70 BTH UTXO (100% cluster A) with a 30 BTH UTXO (100% cluster B) produces output tags: $\{(A, 0.7), (B, 0.3)\}$.

5.4.4 Age-Based Tag Decay

To encourage circulation and prevent wash trading, tags decay over time. However, decay only applies when UTXOs meet an age threshold:

$$\vec{t}_i = \begin{cases} 0.95 \cdot \vec{t}_i & \text{if } \text{age}(\text{UTXO}_i) \geq T_{\min} \\ \vec{t}_i & \text{otherwise} \end{cases} \quad (27)$$

where $T_{\min} = 720$ blocks (≈ 2 hours at 10s blocks).

Wash Trading Resistance: Since new outputs must wait before decay applies, rapid self-transfers achieve no decay:

Table 3: Tag decay limits

Attack	Transactions	Maximum Decay
Rapid wash (1 minute)	100	0%
Patient wash (1 day)	1000	46%
Patient wash (1 week)	7000	97%

5.4.5 Cluster Factor Calculation

The cluster factor is derived from the dominant cluster’s total minted wealth:

$$\text{cluster_factor} = 1 + 5 \cdot \sigma\left(\frac{W_{\max}}{\text{steepness}}\right) \quad (28)$$

where W_{\max} is the total BTH ever minted by the dominant cluster and $\sigma(x) = x/(1+x)$ is a sigmoid function.

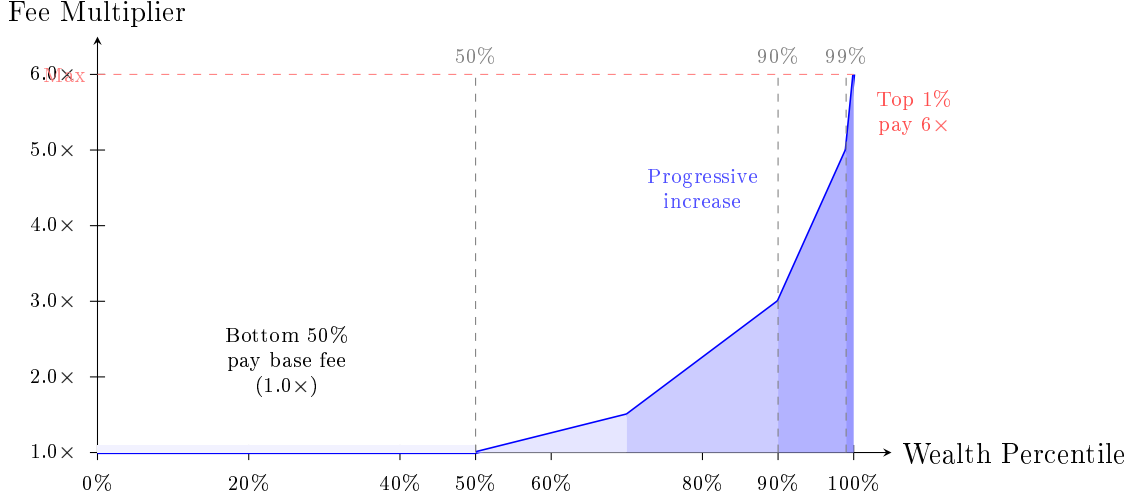
Table 4: Progressive fee multipliers

Cluster Wealth Percentile	Fee Multiplier
Bottom 50%	$1.0\times - 1.5\times$
50th – 90th	$1.5\times - 3.0\times$
Top 10%	$3.0\times - 6.0\times$

5.4.6 Ring Signature Tag Propagation

Ring signatures hide which input is real among n decoys. To prevent fee evasion via low-factor decoy selection, tags propagate *conservatively*:

1. All ring members’ tag vectors are publicly known



Fee calculation: $\text{fee} = \text{base_fee} \times \text{size} \times f_c(\text{percentile})$
where f_c is the cluster factor function shown above.

Figure 7: Progressive fee multiplier based on cluster wealth percentile. Users in the bottom 50% of wealth distribution pay the base fee (1×). Fees increase progressively for wealthier clusters, reaching maximum 6× for the top 1%. This creates Sybil-resistant economic pressure against wealth concentration while remaining minimal for typical users.

2. Fee is calculated using the *maximum* cluster factor among ring members
3. Output tags propagate from the real input (verified via ZK proof but not revealed)

This ensures attackers cannot reduce fees by choosing low-factor decoys.

5.4.7 Sybil Resistance Analysis

The cluster tag system resists common attacks:

- **Splitting coins:** Does not reduce fees—child outputs inherit the parent’s cluster attribution unchanged.
- **Creating multiple addresses:** All outputs from the same cluster share the same factor.
- **Wash trading:** Age-based gating limits decay to ~ 12 events per day maximum.
- **Mixing through exchanges:** Reduces cluster concentration over time, but requires actual economic activity (legitimate use case).

Theorem 5.1 (Cluster Tag Sybil Resistance). *For any splitting strategy that divides n BTH into k UTXOs, the total fees paid are at least as high as paying from a single UTXO.*

Proof. We prove by structural induction on transaction graphs that no splitting strategy reduces total fees.

Definitions. Let:

- $\phi : \text{UTXO} \rightarrow [1, 6]$ denote the cluster factor function
- $v : \text{UTXO} \rightarrow \mathbb{R}^+$ denote the value function

- $\vec{t}: \text{UTXO} \rightarrow \Delta^{|\mathcal{C}|}$ denote the tag vector (probability simplex over clusters)
- b denote the base fee rate per byte
- s denote the transaction size in bytes

The fee for spending UTXO U is $\text{fee}(U) = b \cdot s \cdot \phi(U)$.

Base case: A single UTXO U with value $v(U) = n$ and cluster factor $\phi(U) = f$ incurs fee $F_{\text{base}} = b \cdot s \cdot f$ to spend completely.

Inductive case: Consider splitting U into k outputs U_1, \dots, U_k via a transaction T . We analyze the fee implications.

Step 1: Tag inheritance. By the tag blending rule (Equation 5.3), when U is the sole input:

$$\vec{t}(U_i) = \frac{v(U) \cdot \vec{t}(U)}{v(U)} = \vec{t}(U) \quad \forall i \in [k]$$

Each output inherits exactly the parent's tag vector.

Step 2: Factor preservation. Since ϕ depends only on the tag vector and global cluster wealth:

$$\phi(U_i) = \phi(\vec{t}(U_i)) = \phi(\vec{t}(U)) = \phi(U) = f \quad \forall i$$

Splitting does not change cluster factors.

Step 3: Fee accounting. The splitting transaction T incurs fee:

$$F_T = b \cdot s_T \cdot f$$

where s_T is the size of the splitting transaction.

Subsequently spending all k outputs incurs total fees:

$$F_{\text{spend}} = \sum_{i=1}^k b \cdot s_i \cdot \phi(U_i) = b \cdot f \cdot \sum_{i=1}^k s_i$$

Step 4: Total comparison. Total fees under splitting strategy:

$$F_{\text{split}} = F_T + F_{\text{spend}} = b \cdot f \cdot \left(s_T + \sum_{i=1}^k s_i \right)$$

For a single transaction spending U and producing the same final outputs:

$$F_{\text{direct}} = b \cdot f \cdot s_{\text{direct}}$$

Since $s_T + \sum_i s_i \geq s_{\text{direct}}$ (intermediate transactions add overhead), we have $F_{\text{split}} \geq F_{\text{direct}}$.

Induction on transaction depth. For multi-level splitting (creating outputs that are further split), apply the argument recursively. Each level adds transaction overhead without reducing cluster factors. By strong induction:

$$F_{\text{depth-}d} \geq F_{\text{depth-}(d-1)} \geq \dots \geq F_{\text{direct}}$$

Mixing attack analysis. If an attacker combines high-factor UTXOs with low-factor UTXOs:

$$\vec{t}_{\text{mixed}} = \frac{\sum_i v_i \cdot \vec{t}_i}{\sum_i v_i}$$

The resulting factor is a weighted average, not a minimum. The attacker must acquire low-factor UTXOs through legitimate economic activity (purchasing from others), which itself incurs fees and does not create value—it merely transfers it from other participants.

Conclusion. No Sybil strategy (splitting coins, creating addresses, or self-transfers) reduces total fees below the direct transaction cost. Fee reduction requires genuine economic mixing with other clusters. \square

Corollary 5.2 (Wash Trading Ineffectiveness). *Self-transfers through n intermediate addresses incur strictly greater fees than direct transfer, with overhead $\Omega(n \cdot s_{tx} \cdot b \cdot f)$.*

5.5 Validation Rules

A private transaction is valid if and only if:

1. **Structure:** 1–16 inputs, 1–16 outputs, valid encoding
2. **Key Image Uniqueness:** All key images are:
 - Distinct within this transaction
 - Not present in the key image database (no double-spend)
3. **Ring Validity:** For each input ring:
 - All referenced outputs exist and are unspent
 - Ring members are sorted (canonical ordering)
 - Real output is among the ring members
4. **Signature Validity:** All CLSAG signatures verify
5. **Value Conservation:**

$$\sum_i C_{\text{in}}^{(i)} = \sum_j C_{\text{out}}^{(j)} + \text{fee} \cdot H \quad (29)$$

6. **Range Proofs:** Bulletproofs verify for all output commitments (amounts in $[0, 2^{64})$)
7. **Fee:** $\text{fee} \geq \text{min_fee}(\text{size}, \text{cluster_factor})$
8. **Size:** Total serialized size ≤ 100 KB

5.6 Transaction Size Analysis

Table 5: Private transaction size breakdown (1-in-2-out)

Component	Size (bytes)
Transaction header	32
Input (ring references, key image)	680
Output $\times 2$ (commitment, key, ciphertext)	2,304
CLSAG signature	704
Bulletproof (2 outputs, aggregated)	736
Cluster tags	96
Total	$\sim 4,552$

For comparison, a post-quantum ring signature would add approximately 35 KB per input, making transactions $10\times$ larger.

5.7 Decoy Selection

Ring members are selected to maximize anonymity:

1. **Age distribution:** Decoys follow the empirical spend-age distribution (recent outputs are more likely to be real)
2. **Cluster similarity:** At least 70% cosine similarity between decoy and real cluster tags (prevents fingerprinting)
3. **Output type matching:** Decoys match the real output’s characteristics (amount range, if known)
4. **Randomization:** Subject to above constraints, selection is randomized

5.8 Transaction Malleability

Both transactions resist malleability:

- **Signature covers full prefix:** Modifying any field invalidates signatures
- **Canonical encoding:** Only one valid serialization exists
- **Key image binding:** Key images are deterministic from private keys

The transaction hash (txid) is computed over the canonical serialization of the complete transaction.

6 Consensus Mechanism

Both employs a hybrid consensus mechanism combining proof-of-work (PoW) block proposal with Stellar Consensus Protocol (SCP) finalization. This achieves permissionless participation with fast deterministic finality.

6.1 Design Rationale

6.1.1 Why Not Pure PoW?

Nakamoto consensus provides permissionless participation but suffers from:

- **Slow finality:** Probabilistic finality requires multiple confirmations (typically 10–60 minutes).
- **Reorg vulnerability:** Transactions can be reversed by chain reorganizations, even after confirmation.
- **Energy waste:** Hashpower expended on orphaned blocks provides no value.

6.1.2 Why Not Pure BFT?

Classical BFT protocols provide deterministic finality but require:

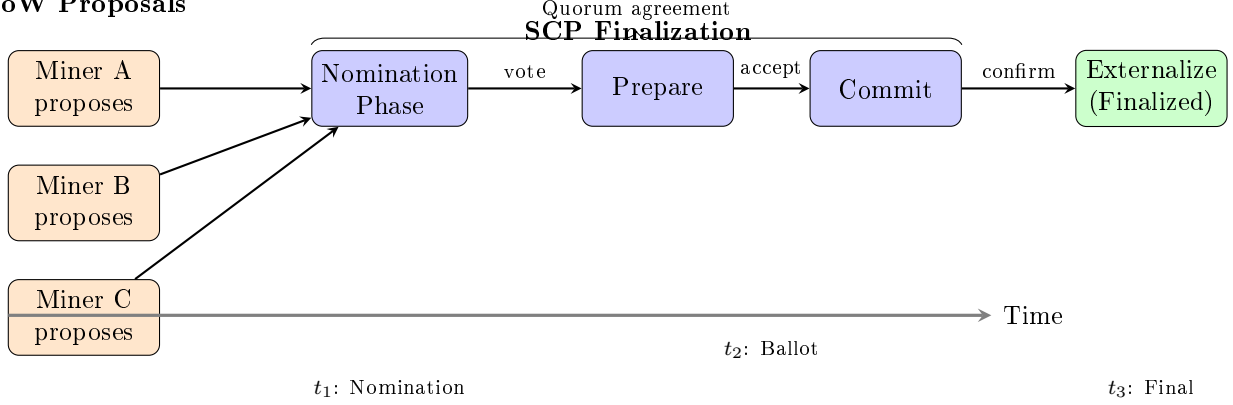
- **Known participants:** Fixed validator sets conflict with permissionless design.
- **Quadratic messaging:** $O(n^2)$ communication limits scalability.
- **Synchrony assumptions:** Liveness depends on network timing bounds.

6.1.3 Hybrid Approach

Both combines the strengths of both:

Property	PoW	SCP
Permissionless participation	✓	—
Fair distribution	✓	—
Deterministic finality	—	✓
Fast confirmation	—	✓
Byzantine fault tolerance	—	✓

PoW Proposals



t_0 : Block proposals

Figure 8: Hybrid PoW + SCP consensus flow. Multiple miners propose blocks via proof-of-work (permissionless participation), then SCP achieves deterministic finality through quorum agreement. Unlike pure PoW, finalized blocks cannot be reverted—providing instant settlement guarantees.

6.2 Stellar Consensus Protocol

SCP [31] achieves Byzantine agreement with open membership through *federated Byzantine agreement* (FBA).

6.2.1 Quorum Slices

Each node v declares a *quorum slice* $Q(v)$ —the set of nodes v trusts for consensus. Unlike classical BFT where all nodes agree on membership, SCP allows heterogeneous trust:

Definition 6.1 (Quorum Slice). A quorum slice for node v is a set $Q(v) \subseteq V$ such that $v \in Q(v)$ and v will accept a statement if all nodes in $Q(v)$ accept it.

Definition 6.2 (Quorum). A set $U \subseteq V$ is a quorum if for every node $v \in U$, there exists a quorum slice $Q(v) \subseteq U$.

Informally, a quorum is a set of nodes sufficient for agreement—each member has “enough” trusted nodes within the set to be convinced.

6.2.2 Quorum Intersection

Safety requires that any two quorums overlap:

Definition 6.3 (Quorum Intersection). A system has quorum intersection if for all quorums U_1, U_2 : $U_1 \cap U_2 \neq \emptyset$.

Theorem 6.4 (SCP Safety). *If the network has quorum intersection and no Byzantine nodes, then SCP provides safety: no two honest nodes externalize different values for the same slot.*

6.2.3 Tiered Quorum Structure

Bothouses a tiered quorum structure balancing decentralization with robustness:

```

QuorumSlice
// Tier 1: Infrastructure nodes (high-uptime, well-connected)
threshold: 3,
validators: [
    "node1.botho.org",
    "node2.botho.org",
    "node3.botho.org",
    "node4.botho.org",
],
// Tier 2: Community validators
inner_sets: [
    threshold: 2,
    validators: ["community1", "community2", "community3"],
],

```

This requires agreement from 3 of 4 infrastructure nodes AND 2 of 3 community validators, ensuring both stability and decentralization.

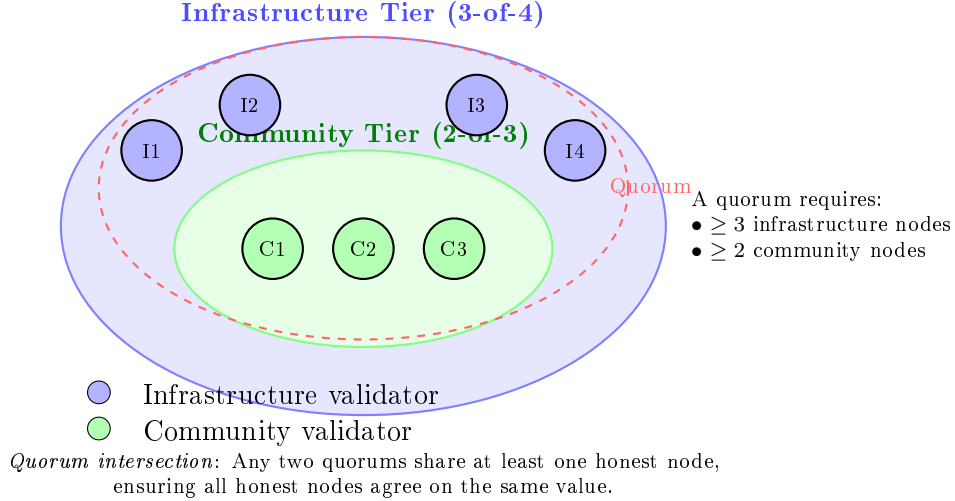


Figure 9: Tiered quorum slice structure in **Botho**. Each node’s quorum slice requires agreement from 3-of-4 infrastructure validators AND 2-of-3 community validators. This design balances network stability (infrastructure tier) with decentralization (community tier). Quorum intersection is guaranteed when Byzantine nodes are below threshold.

6.3 Consensus Phases

The consensus process proceeds through four phases for each block slot:

6.3.1 Phase 1: Block Proposal (PoW)

Miners compete to propose blocks via proof-of-work:

1. Miner constructs candidate block with transactions from mempool
2. Miner searches for nonce satisfying:

$$\mathcal{H}(\text{block_header} \parallel \text{nonce}) < \text{target} \quad (30)$$

3. First valid block is broadcast to the network
4. Multiple proposals may arrive; SCP selects among them

Why PoW for proposal?

- **Permissionless:** Anyone can propose blocks
- **Fair distribution:** Block rewards are distributed by computational contribution
- **Sybil resistance:** Creating proposals requires real resources

6.3.2 Phase 2: Nomination

Nodes nominate candidate values (block hashes) for the slot:

1. Node receives valid block proposals
2. Node nominates the first valid proposal received (tie-breaking by lowest hash)
3. Nodes accept nominations from their quorum slices
4. Nomination converges to a single candidate when a quorum agrees

```
NominationMessage
  slot_index: u64,
  voted: Vec<BlockHash>,      // Values this node votes for
  accepted: Vec<BlockHash>,   // Values this node has accepted
```

6.3.3 Phase 3: Ballot Protocol

Once nomination produces a candidate, nodes run the ballot protocol to commit to a specific value:

1. **Prepare:** Nodes vote to prepare a ballot (n, v) where n is a counter and v is the candidate value
2. **Commit:** Once prepared, nodes vote to commit the ballot
3. **Abort:** If a ballot cannot progress, nodes abort and try a higher ballot number

The ballot protocol ensures:

- No two different values can be committed for the same slot
- Progress is made despite Byzantine nodes (up to threshold)
- Aborted ballots do not block future ballots

6.3.4 Phase 4: Externalize

When a ballot is committed, nodes externalize the value:

```
ExternalizeMessage
  slot_index: u64,
  commit: Ballot,           // The committed ballot
  quorum_set_hash: Hash,    // Proves quorum agreement
```

Externalization represents deterministic finality—the value cannot be changed without violating quorum intersection.

6.4 Block Structure

Block

```
header: BlockHeader,
minting_tx: MintingTransaction,
transactions: Vec<PrivateTransaction>,
scp_proof: SCPProof,
```

BlockHeader

```
version: u8,
prev_block_hash: Hash,
merkle_root: Hash,
timestamp: u64,
height: u64,
difficulty: u64,
nonce: u64,
minter_public_key: MLDSAPublicKey,
```

SCPProof

```
slot_index: u64,
externalize_messages: Vec<ExternalizeMessage>,
// Sufficient messages to prove quorum agreement
```

6.5 Difficulty Adjustment

Bothouses a responsive difficulty adjustment algorithm:

$$\text{difficulty}_{n+1} = \text{difficulty}_n \times \frac{T_{\text{target}}}{T_{\text{actual}}} \quad (31)$$

where:

- T_{target} is the target block time (dynamically adjusted, see Section 7)
- T_{actual} is the actual time for the last adjustment window (144 blocks)

Adjustments are bounded to $[0.5, 2.0] \times$ per window to prevent oscillation.

6.6 Fork Resolution

Unlike pure PoW where the longest chain wins, Botho's SCP finalization makes forks impossible for externalized blocks:

Theorem 6.5 (Fork Freedom). *If the network has quorum intersection and the Byzantine threshold is not exceeded, no two honest nodes can externalize different blocks at the same height.*

Proof. We prove by contradiction, following the structure of Mazières' SCP safety proof [31].

Definitions. Let:

- V denote the set of all nodes
- $Q : V \rightarrow 2^{2^V}$ denote the quorum slice function
- $\mathcal{Q} \subseteq 2^V$ denote the set of all quorums
- $\text{ext}_v(s)$ denote node v 's externalized value for slot s

- $\mathcal{B} \subset V$ denote Byzantine nodes, $|\mathcal{B}| \leq f$

Quorum intersection property. By assumption:

$$\forall Q_1, Q_2 \in \mathcal{Q} : (Q_1 \setminus \mathcal{B}) \cap (Q_2 \setminus \mathcal{B}) \neq \emptyset$$

That is, any two quorums share at least one honest node.

Ballot protocol invariant. The SCP ballot protocol maintains the following invariant for each slot s :

Commit Exclusivity: If honest node v commits ballot (n, x) for slot s , then no honest node commits ballot (n', x') for the same slot with $x' \neq x$ and $n' \leq n$.

This invariant is established through the prepare phase: a node only commits (n, x) after confirming that no lower ballot with a different value can be committed [31].

Contradiction argument. Suppose, for contradiction, that honest nodes A and B externalize different values for slot s (corresponding to block height h):

$$\text{ext}_A(s) = b_A \neq b_B = \text{ext}_B(s)$$

Externalization requires commitment. Let:

- A commit ballot (n_A, b_A) with quorum $Q_A \in \mathcal{Q}$
- B commit ballot (n_B, b_B) with quorum $Q_B \in \mathcal{Q}$

Without loss of generality, assume $n_A \leq n_B$.

By quorum intersection, there exists honest node $C \in (Q_A \setminus \mathcal{B}) \cap (Q_B \setminus \mathcal{B})$.

Node C participated in both quorums. For C to be in Q_A , C must have voted to commit (n_A, b_A) . For C to be in Q_B , C must have voted to commit (n_B, b_B) .

Case 1: $n_A = n_B$. Node C voted to commit (n_A, b_A) and (n_A, b_B) with $b_A \neq b_B$. This violates the ballot protocol rule that honest nodes vote for at most one value per ballot number. \Rightarrow Contradiction.

Case 2: $n_A < n_B$. For C to vote commit on (n_B, b_B) , C must first confirm that no ballot $\leq n_B$ with value $\neq b_B$ is committed. But C voted to commit (n_A, b_A) with $n_A < n_B$ and $b_A \neq b_B$. This violates the prepare-before-commit requirement. \Rightarrow Contradiction.

Both cases yield contradictions. Therefore, our assumption is false, and no two honest nodes can externalize different values for the same slot.

Mapping to block heights. Each consensus slot corresponds to exactly one block height. Externalized value = finalized block hash. Fork freedom at the consensus layer implies fork freedom in the blockchain.

Relationship to PoW. Multiple PoW blocks may be proposed for the same slot. SCP nomination deterministically selects one candidate before the ballot protocol begins. The proof above shows the selected candidate is agreed upon by all honest nodes. \square

Corollary 6.6 (Finality Irreversibility). *Once a block is externalized at height h , no reorganization can replace it without violating quorum intersection or corrupting $> f$ nodes.*

Proof. Reversing externalization at height h requires externalizing a different block b' for the same slot. By Theorem 6.5, this is impossible under the stated assumptions. An attacker must either:

1. Corrupt enough nodes to break quorum intersection, or
2. Corrupt $> f$ nodes to violate the Byzantine threshold

Both require compromising the network’s trust assumptions, not merely accumulating hashpower (unlike pure PoW). \square

Handling pre-finalization forks: Multiple PoW proposals may arrive before SCP converges. The nomination phase selects a unique winner based on:

1. First valid proposal received (per node)
2. Tie-breaking by lowest block hash
3. Quorum agreement determines final selection

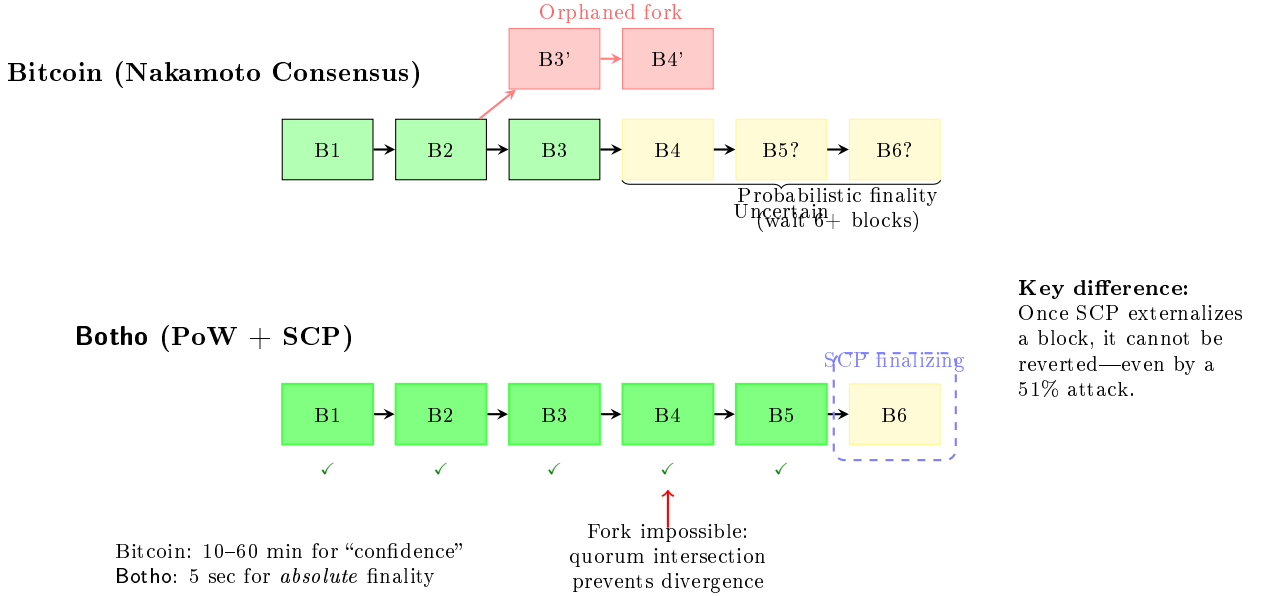


Figure 10: Fork prevention comparison. Bitcoin (top) achieves only probabilistic finality—transactions can be reversed by reorganizations, requiring multiple confirmations for confidence. Botho(bottom) achieves deterministic finality via SCP: once a block is externalized, quorum intersection mathematically guarantees no alternative history can exist. This enables instant settlement with no risk of reversal.

6.7 Timing Analysis

Table 6: Consensus timing breakdown

Phase	Time
Block proposal (PoW)	Variable (5–40s target)
Nomination	~1s
Ballot prepare	~1s
Ballot commit	~1s
Externalize	<1s
Total finality	Block time + ~3–4s

In practice, finality occurs within 5 seconds of block proposal, compared to 10–60 minutes for pure PoW systems.

6.8 Liveness Guarantees

Theorem 6.7 (SCP Liveness). *If the network is eventually synchronous and at most f nodes are Byzantine (where each quorum can tolerate f failures), then SCP eventually makes progress.*

Graceful degradation: If quorum intersection fails (e.g., due to network partition), safety is preserved—nodes simply halt rather than fork. This is a conscious design choice: safety over liveness.

6.9 Security Properties

6.9.1 Byzantine Fault Tolerance

The system tolerates Byzantine behavior from nodes outside any quorum’s blocking threshold. For the default tier structure:

- Infrastructure tier: 1 of 4 can be Byzantine
- Community tier: 1 of 3 can be Byzantine

6.9.2 Nothing-at-Stake Resistance

Unlike pure proof-of-stake systems, PoW proposal ensures:

- Proposing multiple blocks requires multiple PoW solutions
- Resources are burned regardless of which block is selected
- No advantage to “voting for everything”

6.9.3 Long-Range Attack Resistance

SCP finality prevents long-range attacks:

- Once externalized, blocks cannot be reverted
- Rewriting history requires corrupting quorum intersection
- No “weak subjectivity” bootstrap problem

6.10 Comparison with Alternatives

Table 7: Consensus mechanism comparison

Property	Botho	Bitcoin	Tendermint
Finality	Deterministic	Probabilistic	Deterministic
Finality time	~5–10s	~60 min	~6s
Permissionless	Yes	Yes	No
Fork possible	No	Yes	No
Byzantine tolerance	Quorum-based	50% hashpower	1/3 validators
Energy efficiency	Medium	Low	High

6.11 Mining Pool Considerations

Mining pools aggregate hashpower from multiple miners, distributing rewards proportionally. Botho’s hybrid consensus requires adaptations to traditional pooling protocols.

6.11.1 Pool Protocol Compatibility

Standard Stratum-style protocols require modification for PoW+SCP:

```
PoolWorkAssignment
// Standard PoW fields
job_id: u64,
prev_hash: Hash,
coinbase_template: Vec<u8>,
merkle_branches: Vec<Hash>,
target: Difficulty,

// Botho-specific: SCP context
slot_index: u64,
quorum_set_hash: Hash,
nomination_deadline: Timestamp,
```

Key difference: Pools must track SCP slot progress and coordinate work assignments with nomination deadlines. Stale work (missed nomination) produces no reward even if PoW is valid.

6.11.2 Block Reward Attribution

Minting transactions in **Botho** include the minter's public key, creating challenges for pool reward distribution:

- **Pool-controlled keys:** Pool creates minting transaction, distributes rewards via separate payment transactions
- **Pay-per-share (PPS):** Pool pays from reserve for valid shares, assumes variance
- **Proportional (PROP):** Miners receive proportion of actual block rewards found

Cluster tag implications: Pool-minted coins carry the pool's cluster tag. Miners receiving payouts inherit blended tags from the pool's operations.

6.11.3 SCP Participation

Pools face a choice in SCP participation:

1. **Pool as SCP node:** Pool runs consensus, coordinates with quorum. Requires high uptime and expertise.
2. **Delegate SCP:** Pool submits valid blocks to SCP-participating nodes for finalization. Simpler but adds trust.

Most pools will likely delegate SCP participation to infrastructure nodes while focusing on PoW coordination.

6.11.4 Pool Centralization Risks

Pooled mining creates centralization pressure:

- **Hash rate concentration:** Large pools may approach majority hashpower
- **Censorship capability:** Pools could refuse to include certain transactions

- **Network attacks:** Colluding pools could attempt consensus manipulation

Mitigations:

- SCP finality requires quorum agreement beyond PoW majority
- Transparent pool policies and open-source implementations encouraged
- Economic incentives favor decentralization (smaller pools have lower variance for participants)

6.11.5 Solo Mining Viability

Solo mining remains viable under certain conditions:

Table 8: Solo vs. pool mining trade-offs

Factor	Solo	Pool
Variance	Very high	Low
Cluster purity	100% own cluster	Blended with pool
Privacy	Maximum	Pool sees hashrate
Minimum hashrate	~1% network	Any
Technical complexity	Higher	Lower

Recommendation: Users prioritizing cluster tag purity (to minimize progressive fees) should consider solo mining despite higher variance.

6.11.6 Decentralized Pool Alternatives

P2Pool-style decentralized pools offer middle ground:

- **Share chain:** Miners build separate share chain proving work
- **Trustless payouts:** Block rewards distributed automatically based on share chain
- **No central operator:** Removes single point of failure/censorship

Botho adaptation: P2Pool would need SCP integration for share chain finalization. Research and development ongoing.

7 Monetary Policy

Botho’s monetary policy balances long-term security sustainability with controlled inflation, using dynamic mechanisms that respond to network conditions.

7.1 Design Goals

1. **Sustainable security:** Perpetual miner incentive without relying solely on transaction fees.
2. **Fair distribution:** Rewards distributed over time, not front-loaded to early adopters.
3. **Anti-hoarding:** Economic pressure toward circulation rather than accumulation.
4. **Predictability:** Clear, auditable emission schedule.

7.2 Emission Schedule

7.2.1 Initial Emission

Block rewards follow a smooth decay curve rather than Bitcoin’s abrupt halvings:

$$R(h) = R_0 \cdot \left(\frac{1}{2}\right)^{h/H} \quad (32)$$

where:

- $R(h)$ is the block reward at height h
- $R_0 = 50$ BTH is the initial block reward
- $H = 1,051,200$ blocks (≈ 2 years at 60s blocks) is the halving period

This creates continuous, gradual reduction rather than discrete shocks.

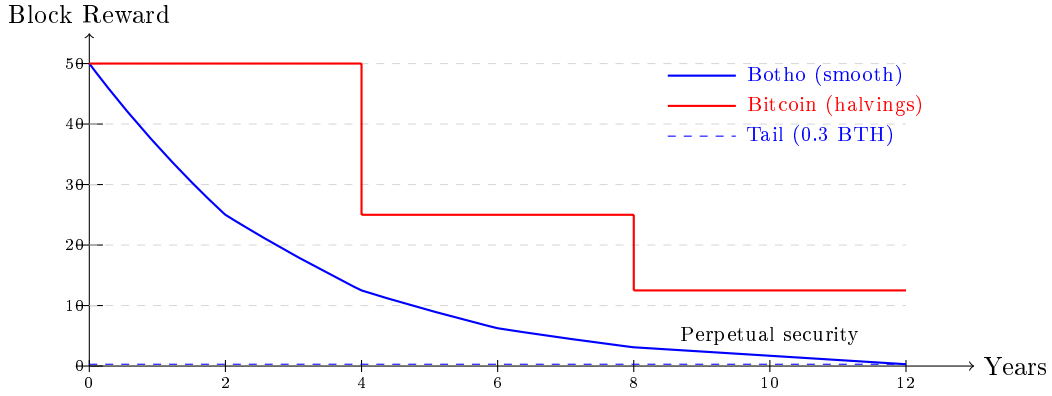


Figure 11: Block reward emission schedule. Botho uses smooth exponential decay (halving every 2 years) transitioning to perpetual tail emission at 0.3 BTH, compared to Bitcoin’s discrete halving events. Tail emission ensures long-term security funding.

7.2.2 Tail Emission

Once block rewards decay below a threshold, perpetual tail emission begins:

$$R_{\text{tail}} = \max(R(h), 0.3 \text{ BTH}) \quad (33)$$

At 5-second blocks (12 blocks per minute), this produces:

$$\text{Annual tail emission} = 0.3 \times 12 \times 60 \times 24 \times 365 = 1,892,160 \text{ BTH} \quad (34)$$

7.2.3 Supply Projection

Note: Tail emission inflation is asymptotically declining—while nominal emission is constant, the percentage decreases as supply grows. After 50 years, annual inflation is approximately 2.1%.

7.3 Dynamic Block Timing

Botho introduces adaptive block intervals that respond to network utilization, creating natural inflation dampening.

Table 9: Supply projection

Year	Circulating	New Emission	Inflation
1	15.8M BTH	15.8M	—
2	23.7M BTH	7.9M	50%
3	27.6M BTH	3.9M	17%
5	30.2M BTH	1.0M	3.4%
10	33.1M BTH	1.9M	6.1%
20	52.0M BTH	1.9M	3.8%

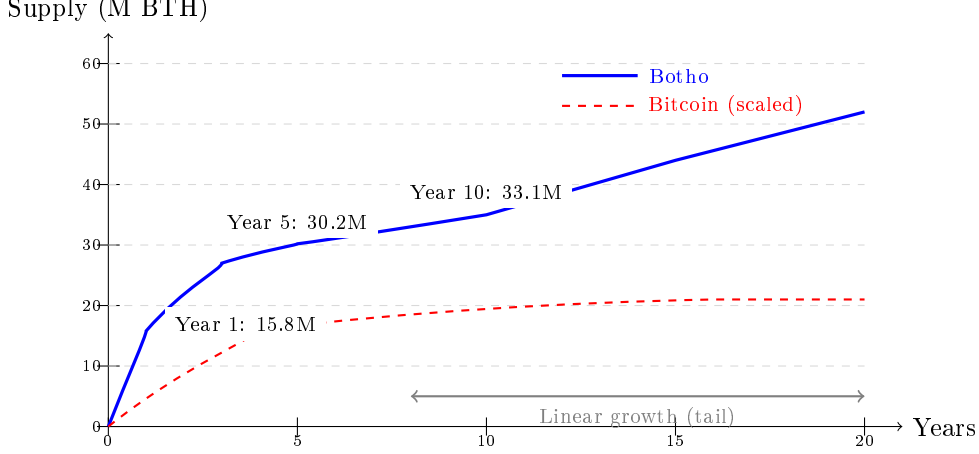


Figure 12: Projected circulating supply over 20 years. Initial emission follows exponential decay (similar to Bitcoin’s halving schedule but smooth), transitioning to linear growth from perpetual tail emission after approximately year 6. Unlike Bitcoin’s hard cap at 21M, Botho’s supply grows indefinitely at a declining percentage rate, ensuring permanent security funding.

7.3.1 Mechanism

Block time targets range from 5 to 40 seconds based on mempool pressure:

$$T_{\text{target}} = T_{\text{min}} + (T_{\text{max}} - T_{\text{min}}) \cdot (1 - U) \quad (35)$$

where:

- $T_{\text{min}} = 5$ seconds (minimum block time)
- $T_{\text{max}} = 40$ seconds (maximum block time)
- $U \in [0, 1]$ is the normalized utilization factor

7.3.2 Utilization Calculation

Utilization is computed from recent mempool activity:

$$U = \text{sigmoid} \left(\frac{\text{mempool_weight} - \text{target_weight}}{\text{sensitivity}} \right) \quad (36)$$

7.3.3 Economic Implications

Dynamic timing creates feedback loops:

Low utilization:

Table 10: Block timing by utilization

Network State	Utilization	Block Time
Idle (no transactions)	0%	40s
Light usage	25%	31s
Moderate usage	50%	23s
Heavy usage	75%	14s
Maximum demand	100%	5s

$$T_{\text{target}} = T_{\text{min}} + (T_{\text{max}} - T_{\text{min}}) \cdot (1 - U)$$

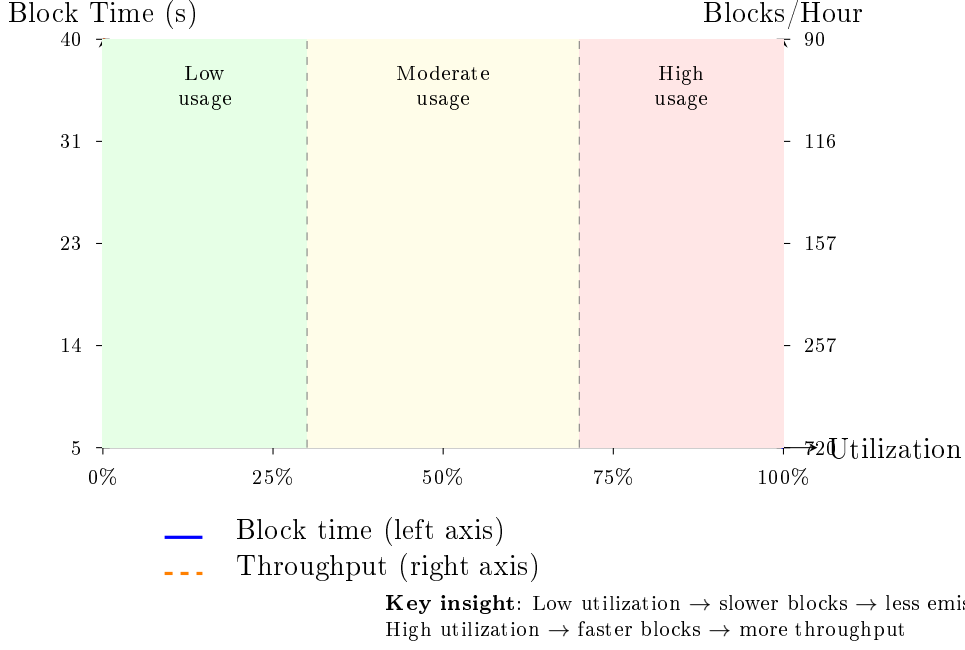


Figure 13: Dynamic block timing mechanism. Block time varies linearly from 40 seconds (idle network) to 5 seconds (maximum demand). This creates natural inflation dampening: low usage produces fewer blocks and less emission, while high usage increases throughput to meet demand. The system self-regulates based on mempool pressure.

- Longer block times → fewer blocks per hour
- Fewer blocks → less emission
- Less emission → reduced inflation when currency is unused

High utilization:

- Shorter block times → more blocks per hour
- More blocks → higher throughput
- Higher throughput → more rewards when currency is actively used

This aligns miner incentives with network utility.

7.3.4 Emission Bounds

The effective annual emission varies with utilization:

Table 11: Tail emission by utilization

Utilization	Blocks/Year	Emission/Year
0% (idle)	788,400	236,520 BTH
50% (moderate)	1,370,087	411,026 BTH
100% (maximum)	6,307,200	1,892,160 BTH

7.4 Fee Economics

7.4.1 Base Fee

The minimum fee per byte is:

$$f_{\text{base}} = 1 \text{ nano-BTH per byte} \quad (37)$$

For a typical 4 KB transaction:

$$f_{\text{min}} = 4,000 \times 1 = 4,000 \text{ nano-BTH} = 0.000004 \text{ BTH} \quad (38)$$

7.4.2 Progressive Fee Multiplier

Transactions pay additional fees based on cluster wealth (see Section 5.4):

$$f_{\text{total}} = f_{\text{base}} \times \text{size} \times \text{cluster_factor} \quad (39)$$

The cluster factor ranges from $1.0\times$ (bottom 50% wealth) to $6.0\times$ (top 1% wealth).

7.4.3 Lottery-Based Fee Redistribution

Transaction fees are split between redistribution and burning:

- **80% redistributed:** Immediately distributed to 4 randomly selected UTXOs via verifiable lottery
- **20% burned:** Permanently removed from supply

This creates:

- **Progressive redistribution:** Random UTXO selection statistically favors the many (small holders) over the few (large holders)
- **Anti-custodial incentive:** Exchanges holding user funds in few UTXOs receive less redistribution than self-custody users with many UTXOs
- **Deflationary pressure:** 20% burn creates supply reduction
- **Anti-MEV:** No miner incentive to reorder transactions
- **Alignment:** Miner income comes from block rewards only

Wealthy clusters pay more → Redistribution to random holders

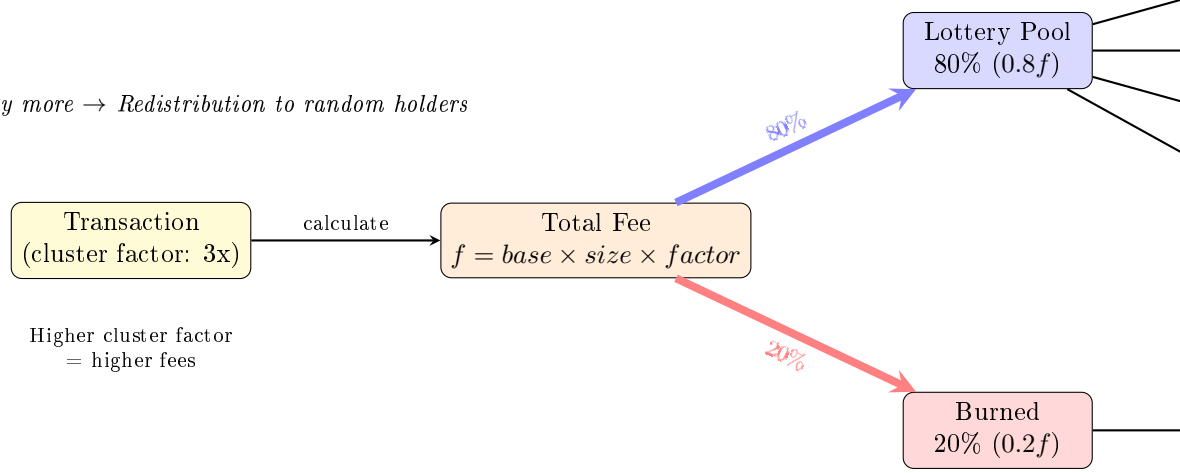


Figure 14: Fee flow mechanism. Transaction fees are calculated using a progressive cluster factor (1x–6x based on coin ancestry). 80% of fees are redistributed via verifiable lottery to 4 randomly selected UTXOs, while 20% is permanently burned. This creates redistribution from active wealthy users to passive holders, with deflationary pressure proportional to network usage.

7.4.4 Lottery Mechanism

For each transaction:

1. Calculate fee f based on cluster factor and transaction size
2. Burn $0.2f$ (20%)
3. Select 4 random UTXOs weighted by eligibility criteria
4. Distribute $0.2f$ to each selected UTXO (80% total)

The lottery uses the previous block hash as verifiable randomness, ensuring selection cannot be manipulated by transaction creators.

7.4.5 Effective Inflation

The effective inflation rate becomes:

$$\text{inflation}_{\text{effective}} = \frac{\text{emission} - \text{fees_burned}}{\text{supply}} \quad (40)$$

where $\text{fees_burned} = 0.2 \times \text{total_fees}$. Under high utilization with progressive fees, burning may exceed tail emission, creating net deflation.

7.4.6 Fee Flow Analysis

7.5 Minter Incentives

7.5.1 Block Reward Composition

Minters receive only the block reward:

Table 12: Fee flow scenarios (1M daily transactions)

Wealth Distribution	Daily Fees	Redistributed	Burned
Equal (factor 1.0)	16 BTH	12.8 BTH	3.2 BTH
Current inequality	48 BTH	38.4 BTH	9.6 BTH
High inequality	96 BTH	76.8 BTH	19.2 BTH

$$\text{minter_income} = R(h) \text{ (no fees)} \quad (41)$$

This simplifies minting economics and eliminates fee-based attacks.

7.5.2 Profitability Analysis

Minting profitability depends on:

- Hardware cost (one-time)
- Electricity cost (ongoing)
- Block reward value
- Network hashrate (competition)

The tail emission ensures perpetual profitability for efficient miners, unlike Bitcoin where fee-only income creates uncertainty.

7.5.3 Decentralization Incentives

Several mechanisms encourage mining decentralization:

- **CPU-friendly PoW**: Algorithm resists ASIC development (RandomX-based)
- **Solo mining viable**: Tail emission ensures consistent income even at low hashrate
- **No economy of scale**: Linear scaling of rewards with hashpower

7.6 Long-Term Sustainability

7.6.1 Security Budget

The perpetual security budget (in BTH terms) is:

$$\text{security_budget} = R_{\text{tail}} \times \text{blocks_per_year} \quad (42)$$

This ranges from 236K BTH (idle) to 1.89M BTH (maximum utilization) annually.

7.6.2 Comparison with Fixed Supply

7.6.3 Wealth Tax Equivalence

Tail emission functions as a mild wealth tax:

$$\text{effective_tax} = \frac{\text{tail_emission}}{\text{total_supply}} \approx 2\% \quad (43)$$

This encourages circulation: holding idle currency means gradual dilution, while active participation maintains relative wealth.

Table 13: Security model comparison: fixed supply vs. tail emission

Aspect	Fixed Supply	Tail Emission
Long-term miner income	Fees only	Rewards + fees
Security if fees low	Degraded	Maintained
Inflation	0%	~2–3%
Predictability	High	High

Table 14: Monetary policy constants

Parameter	Value	Description
Initial reward	50 BTH	Block reward at genesis
Halving period	1,051,200 blocks	≈ 2 years
Tail emission	0.3 BTH	Minimum block reward
Min block time	5 seconds	At maximum utilization
Max block time	40 seconds	At zero utilization
Base fee rate	1 nano-BTH/byte	Minimum transaction fee
Max cluster factor	$6.0\times$	Top 1% wealth multiplier
Decimals	9	Smallest unit: 1 nano-BTH

7.7 Monetary Constants

8 Network Protocol

Both employs a peer-to-peer network protocol optimized for privacy transaction propagation and consensus message delivery.

8.1 Network Architecture

8.1.1 Node Types

- **Full nodes:** Store complete blockchain, validate all transactions, participate in consensus
- **Minting nodes:** Full nodes that additionally perform PoW and propose blocks
- **Light clients:** Store block headers only, verify transactions via inclusion proofs

8.1.2 Transport Layer

All peer-to-peer communication uses:

- **TCP:** Reliable delivery for consensus messages
- **Noise Protocol** [44]: Authenticated encryption with forward secrecy
- **Multiplexed streams:** Separate channels for different message types

8.2 Peer Discovery

8.2.1 Bootstrap Nodes

New nodes connect to hardcoded bootstrap nodes to discover initial peers:

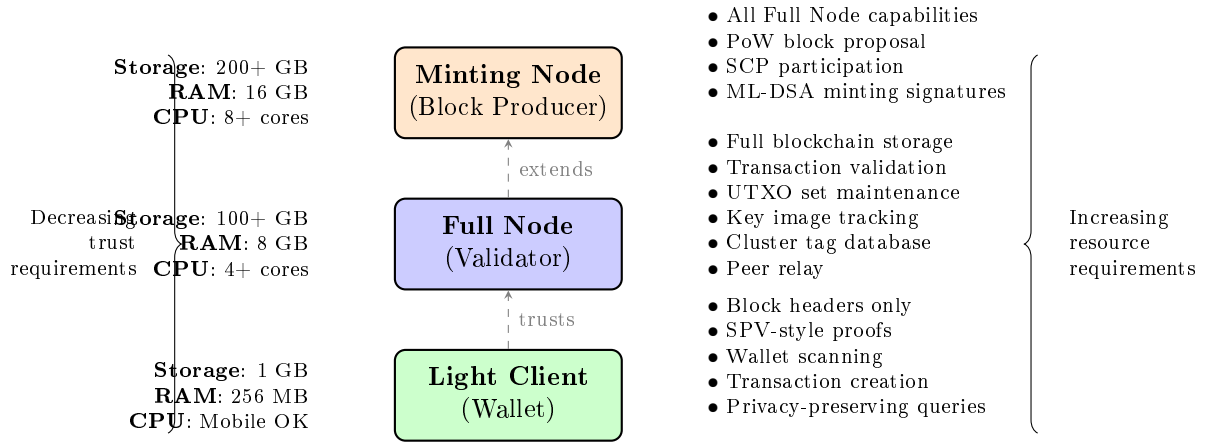


Figure 15: Node type hierarchy in **Botho**. Light clients trust full nodes for transaction validation but verify headers independently. Full nodes maintain complete state and validate all transactions. Minting nodes extend full nodes with block proposal capabilities. Resource requirements scale with trust assumptions—light clients trade some security for accessibility.

```
BootstrapNodes
  dns_seeds: [
    "seed1.botho.org",
    "seed2.botho.org",
    "seed3.botho.org",
  ],
  static_peers: [
    "203.0.113.1:9732",
    "203.0.113.2:9732",
  ],
```

8.2.2 Kademlia DHT

Peer discovery uses a modified Kademlia DHT [30]:

- Node IDs derived from public keys (not chosen)
- XOR distance metric for routing
- Iterative lookup with parallel queries
- Periodic bucket refresh

8.2.3 Peer Limits

Table 15: Connection limits	
Category	Limit
Outbound connections	8
Inbound connections	117
Total connections	125
Connections per IP	2

8.3 Message Protocol

8.3.1 Message Types

```
enum Message
// Handshake
Hello version, genesis_hash, best_height ,
HelloAck version, genesis_hash, best_height ,

// Block propagation
NewBlock block ,
GetBlocks locator, stop_hash ,
Blocks blocks ,

// Transaction propagation
NewTransaction tx ,
GetTransactions hashes ,
Transactions txs ,

// Consensus (SCP)
SCPNomination msg ,
SCPPrepare msg ,
SCPCommit msg ,
SCPExternalize msg ,

// Compact blocks
CompactBlock header, short_ids ,
GetBlockTxs block_hash, indices ,
BlockTxs block_hash, txs ,
```

8.3.2 Message Serialization

Messages are serialized using a canonical binary format:

- Little-endian byte order
- Varint encoding for lengths
- No padding or alignment
- Deterministic ordering of map keys

8.3.3 Wire Format Specification

Each message follows the structure:

```
Message
magic: [u8; 4],           // Network identifier: 0x00 0x0B 0x07 0xB0
type_id: u8,              // Message type discriminant
length: varint,           // Payload length in bytes
checksum: [u8; 4],        // First 4 bytes of SHA256(payload)
payload: [u8; length],    // Type-specific data
```

Message type identifiers:

Field-level encoding:

Table 16: Message type encoding

ID	Message	Max Size
0x00	Hello	256 B
0x01	HelloAck	256 B
0x10	NewBlock	2 MB
0x11	GetBlocks	4 KB
0x12	Blocks	10 MB
0x20	NewTransaction	100 KB
0x21	GetTransactions	4 KB
0x22	Transactions	1 MB
0x30	SCPNomination	64 KB
0x31	SCPPrepare	64 KB
0x32	SCPCommit	64 KB
0x33	SCPEexternalize	64 KB
0x40	CompactBlock	64 KB
0x41	GetBlockTxS	4 KB
0x42	BlockTxS	1 MB
0xFF	Ping/Pong	8 B

```
// Varint: 7 bits per byte, MSB = continuation
fn encode_varint(n: u64) -> Vec<u8>
    let mut bytes = Vec::new();
    let mut value = n;
    while value >= 0x80
        bytes.push((value & 0x7F) | 0x80);
        value >>= 7;

    bytes.push(value as u8);
    bytes

// Hash: 32 bytes, raw
Hash = [u8; 32]

// CompressedPoint: 32 bytes (Ristretto255)
Point = [u8; 32]

// Signature: Variable (CLSAG ~700B, ML-DSA ~3.3KB)
Signature = varint(len) ++ [u8; len]

// Block header: Fixed 120 bytes
BlockHeader
    version: u8,           // 1 byte
    prev_hash: [u8; 32],   // 32 bytes
    merkle_root: [u8; 32], // 32 bytes
    timestamp: u64,        // 8 bytes (little-endian)
    height: u64,           // 8 bytes
    difficulty: u64,        // 8 bytes
    nonce: u64,            // 8 bytes
    minter_key_hash: [u8; 32], // 32 bytes
```

8.4 Protocol State Machine

8.4.1 Connection States

Each peer connection transitions through defined states:

```
enum ConnectionState
    Connecting,      // TCP handshake in progress
    Authenticating,  // Noise protocol handshake
    Handshaking,     // Hello/HelloAck exchange
    Active,          // Normal operation
    Syncing,         // Initial block download
    Stale,           // No activity timeout
    Disconnecting,   // Graceful shutdown

// State transitions
Connecting    -> Authenticating  [TCP established]
Authenticating -> Handshaking     [Noise complete]
Handshaking   -> Active           [HelloAck received, compatible]
Handshaking   -> Disconnecting    [Incompatible version/genesis]
Active        -> Syncing          [Peer ahead by >10 blocks]
Syncing       -> Active           [Caught up]
Active        -> Stale            [No messages for 5 minutes]
Stale         -> Active           [Pong received]
Stale         -> Disconnecting    [No Pong after 30 seconds]
*             -> Disconnecting    [Error or ban]
```

8.4.2 Message Sequences

Connection establishment:

1. Initiator: TCP connect to peer:9732
2. Both: Noise XX handshake (mutual authentication)
3. Initiator: Send Hello {version, genesis_hash, best_height}
4. Responder: Send HelloAck (if compatible) or disconnect
5. Both: Transition to Active state

Block synchronization:

1. Node A: Send GetBlocks {locator, stop_hash}
2. Node B: Find common ancestor from locator
3. Node B: Send Blocks {blocks[ancestor+1..min(ancestor+500, stop)]}
4. Node A: Validate and apply blocks
5. Node A: Repeat from step 1 if more blocks needed

Transaction relay:

1. Sender: Receive transaction (local or from peer)
2. Sender: Validate transaction fully
3. If in stem phase: Forward to single stem peer

4. If in fluff phase: Announce via NewTransaction to all peers
5. Receivers: Request full transaction if not in mempool
6. Receivers: Validate and continue relay

Consensus message flow:

1. Proposer: Broadcast NewBlock upon PoW solution
2. Validators: Validate block, enter nomination
3. Validators: Exchange SCPNomination messages
4. Validators: Upon nomination convergence, send SCPPrepare
5. Validators: Upon prepare quorum, send SCPCommit
6. Validators: Upon commit quorum, send SCPExternalize
7. All nodes: Apply externalized block as final

8.4.3 Error Handling

Protocol errors trigger specific responses:

Table 17: Protocol error handling

Error	Action	Score Penalty
Invalid checksum	Drop message	-1
Unknown message type	Drop message	0
Message too large	Drop + warn	-10
Invalid block header	Drop block	-50
Invalid transaction	Drop tx	-10
Invalid signature	Drop + ban	-1000
Rate limit exceeded	Throttle	-5/excess
Protocol violation	Disconnect	-1000

Reconnection policy:

- Disconnected peers: Retry after exponential backoff (1s, 2s, 4s, ..., 1h max)
- Banned peers: No reconnection for 24 hours
- Repeated bans: Permanent blacklist after 3 bans

8.5 Block Propagation

8.5.1 Compact Block Relay

To reduce bandwidth, blocks are relayed in compact form:

1. Sender transmits compact block (header + short transaction IDs)
2. Receiver reconstructs from mempool
3. Receiver requests missing transactions by index
4. Sender provides requested transactions

Short ID computation:

$$\text{short_id} = \text{SipHash}(\text{nonce} \parallel \text{txid})_{[0:6]} \quad (44)$$

Average bandwidth savings: 90% (most transactions already in mempool).

8.5.2 Block Validation

Upon receiving a block:

1. Validate header (PoW, timestamp, difficulty)
2. Verify Merkle root
3. Validate each transaction
4. Verify minting transaction signature
5. Verify SCP proof (quorum agreement)
6. Apply to local state

8.6 Transaction Propagation

8.6.1 Dandelion++

Transaction propagation uses Dandelion++ [16] to protect sender network identity:

Stem phase:

- Transaction relayed along a random path
- Each hop has probability p to switch to fluff phase
- Path length follows geometric distribution

Fluff phase:

- Standard flooding to all peers
- Cannot trace back to stem origin

8.6.2 Transaction Validation

Transactions are validated before relay:

1. Syntactic validity (correct encoding)
2. Semantic validity (signatures verify)
3. Contextual validity (references exist, no double-spend)
4. Fee sufficiency

Invalid transactions are dropped; peers sending invalid transactions are deprioritized.

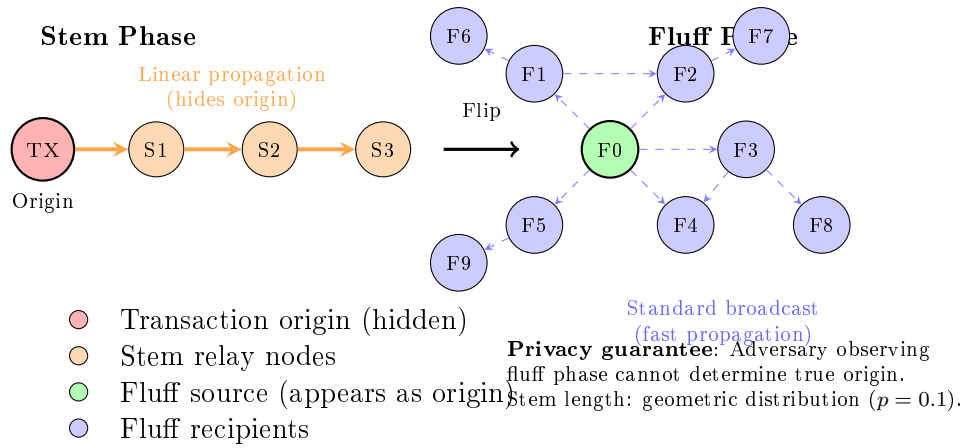


Figure 16: Dandelion++ transaction propagation [16]. In the *stem phase*, transactions propagate along a random linear path, obscuring the origin. After a probabilistic transition (average 10 hops), the *fluff phase* begins: standard broadcast propagation for fast network-wide dissemination. The fluff source appears to be the transaction origin, providing plausible deniability for the true sender.

8.7 Synchronization

8.7.1 Initial Block Download

New nodes synchronize via:

1. Connect to peers and exchange best heights
2. Download block headers (verify PoW chain)
3. Request blocks in parallel from multiple peers
4. Validate and apply blocks sequentially
5. Join consensus once synchronized

8.7.2 Block Locator

Block requests include a locator to identify common ancestor:

```
fn block_locator(tip: Height) -> Vec<Hash>
    let mut locator = Vec::new();
    let mut step = 1;
    let mut height = tip;

    while height > 0
        locator.push(hash_at(height));
        if locator.len() > 10
            step *= 2;

        height = height.saturating_sub(step);

    locator.push(genesis_hash());
    locator
```

This logarithmic structure efficiently identifies divergence point.

8.7.3 Pruning

Full nodes may prune old blocks while retaining:

- Recent blocks (configurable, default 10,000)
- Block headers (all)
- UTXO set (current state)
- Key image set (all, for double-spend detection)

Pruned nodes cannot serve historical blocks but can fully validate new transactions.

8.8 Denial-of-Service Protection

8.8.1 Rate Limiting

Per-peer rate limits:

- Block requests: 10/second
- Transaction announcements: 100/second
- Consensus messages: 50/second

8.8.2 Peer Scoring

Peers maintain reputation scores:

- +1 for valid blocks/transactions
- -10 for invalid data
- -100 for protocol violations
- Disconnect at score < -1000

8.8.3 Resource Bounds

- Maximum message size: 100 KB
- Maximum block size: 2 MB
- Maximum mempool size: 100 MB
- Maximum pending requests: 1000

8.9 Privacy Considerations

8.9.1 Traffic Analysis Resistance

- **Dandelion++**: Hides transaction origin
- **Encrypted transport**: Prevents content inspection
- **No timing correlation**: Messages are batched and jittered

8.9.2 Tor/I2P Support

Optional routing through anonymity networks:

- Tor hidden service support
- I2P integration planned
- Mixed clearnet/onion peer connections

8.10 Light Client Security

Light clients enable mobile and resource-constrained devices to participate in the network with reduced trust assumptions compared to custodial solutions.

8.10.1 Verification Capabilities

Light clients can verify:

- **Transaction inclusion:** Merkle proofs demonstrate transaction is in a specific block
- **Block header chain:** PoW verification confirms work was expended
- **SCP finality:** Externalization proofs confirm consensus agreement
- **Output ownership:** Local key derivation identifies owned outputs

Light clients *cannot* verify:

- **Transaction validity:** Ring signature verification requires full UTXO context
- **Double-spend prevention:** Requires complete key image database
- **Value conservation:** Commitment arithmetic needs all inputs

8.10.2 Trust Model

Light clients trust full nodes for:

1. **Transaction validity:** Full nodes could relay invalid transactions (mitigated by connecting to multiple independent nodes)
2. **Completeness:** Full nodes could withhold transactions (mitigated by querying multiple nodes)
3. **Privacy:** Query patterns reveal interest (mitigated by fetching broader data ranges)

Security level: Comparable to SPV in Bitcoin—economically secure against profit-motivated attackers but vulnerable to targeted attacks by well-resourced adversaries.

8.10.3 Output Scanning

Light clients scan for received funds:

```
LightClientScan
// Request outputs in block range
GetOutputs
    start_height: u64,
    end_height: u64,
    ,

// Response: encrypted output data
Outputs
    outputs: Vec<(Height, TxIndex, EncryptedOutput)>,
    ,
```

For each output, the client:

1. Attempts ML-KEM decapsulation with view key
2. If successful, derives one-time key and checks match
3. Decrypts amount if output belongs to wallet

Privacy consideration: Downloading all outputs in a range (rather than requesting specific ones) prevents servers from learning which outputs the client owns.

8.10.4 Transaction Submission

Light clients construct transactions locally:

1. Select ring members from cached output set
2. Build transaction with local keys (never transmitted)
3. Submit to multiple full nodes via Dandelion++ stem

Ring member selection: Light clients maintain a local cache of recent outputs sufficient for ring construction. This cache is populated during scanning and can be refreshed from any full node.

8.10.5 Privacy Implications

Light client operations leak information:

Table 18: Light client information leakage

Operation	Information Leaked
Block header sync	Active time periods
Output scanning	Block ranges of interest
Ring member fetch	Potential transaction construction
TX submission	Approximate location (IP-based)

Mitigation strategies:

- Connect via Tor/I2P for IP privacy
- Scan broader ranges than necessary
- Cache ring members proactively
- Submit transactions to random nodes

8.10.6 Comparison to Full Node Security

Table 19: Full node vs. light client security

Property	Full Node	Light Client
Validate all transactions	Yes	No
Detect double-spends	Yes	No
Verify consensus	Yes	Partial
Independent operation	Yes	No
Receive funds securely	Yes	Yes*
Send funds securely	Yes	Yes
Privacy (network)	High	Medium

**Light clients can verify they received funds but must trust that the sending transaction was valid.*

8.10.7 Mobile Wallet Recommendations

For mobile deployments:

- Use light client with multiple independent servers
- Prefer Tor connections when available
- Cache sufficient outputs for offline transaction construction
- Verify transaction inclusion before considering funds confirmed
- Consider running personal full node as trusted backend

8.11 Network Constants

Table 20: Network protocol constants

Parameter	Value	Description
Default port	9732	Main network
Testnet port	19732	Test network
Protocol version	1	Current version
Magic bytes	0xB07B0	Network identifier
Handshake timeout	10s	Connection setup
Ping interval	60s	Keepalive

9 Security Analysis

This section analyzes Botho’s security properties, threat models, and resistance to various attacks.

9.1 Threat Model

9.1.1 Adversary Capabilities

We consider adversaries with the following capabilities:

- **Network:** Can observe, delay, and inject messages (but not break encryption)
- **Computational:** Polynomial-time bounded (or BQP for quantum adversaries)
- **Economic:** Can acquire significant hashpower or stake
- **Passive:** Can collect and analyze blockchain data indefinitely

9.1.2 Security Goals

1. **Safety:** No double-spending; no unauthorized coin creation
2. **Liveness:** Valid transactions eventually confirm
3. **Privacy:** Transaction graph analysis does not reveal sender, recipient, or amount
4. **Quantum resistance:** Long-term data remains protected against quantum attacks

9.2 Privacy Analysis

9.2.1 Recipient Unlinkability

Theorem 9.1 (Post-Quantum Recipient Privacy). *Under the IND-CCA2 security of ML-KEM-768, an adversary with access to a quantum computer cannot link outputs to recipient addresses with probability better than negligible.*

Proof. Each output contains:

- ML-KEM ciphertext c encapsulating shared secret K
- One-time key $P = sG + B$ where $s = H_s(K||\text{index})$

Without the recipient's secret key, the shared secret K is computationally indistinguishable from random (IND-CCA2 security holds against quantum adversaries for ML-KEM-768). Thus s is random, and P reveals nothing about the recipient's public key B . \square

9.2.2 Sender Anonymity

Theorem 9.2 (Ring Signature Anonymity). *Under the DLP assumption in the random oracle model, the probability that an adversary identifies the true signer among ring members is at most $1/n + \text{negl}(\lambda)$, where $n = 20$ is the ring size.*

Caveat: This is classical security. A quantum adversary with access to a sufficiently large quantum computer could potentially solve DLP and identify signers. However:

- Sender identity has ephemeral value (see Section 4)
- Ring signature data is not uniquely attributable even with private key recovery
- Retroactive deanonymization provides limited advantage

9.2.3 Amount Confidentiality

Theorem 9.3 (Information-Theoretic Amount Hiding). *Transaction amounts are unconditionally hidden: no computational power (including quantum computers) can determine amounts from Pedersen commitments alone.*

Proof. A Pedersen commitment $C = vH + rG$ with random blinding factor r is uniformly distributed over the curve, independent of v . For any commitment C and any amount v' , there exists r' such that $C = v'H + r'G$. Without the discrete log of H base G , the commitment is perfectly hiding. \square

9.2.4 Anonymity Set Analysis

The effective anonymity set depends on decoy selection quality:

Table 21: Anonymity degradation factors

Attack	Reduction	Mitigation
Timing analysis	$\sim 2\text{--}3\times$	Age distribution matching
Output age heuristic	$\sim 1.5\times$	Recent output bias
Cluster analysis	$\sim 1.2\times$	Cluster similarity requirement
Repeated use	Cumulative	Transaction spacing
Effective ring size	$\sim 5\text{--}8$	(from nominal 20)

Even with degradation, the effective anonymity set provides meaningful privacy protection for typical use cases.

9.3 Formal Privacy Framework

This section establishes formal definitions and analysis of **Botho**'s privacy properties, following standard cryptographic privacy definitions.

9.3.1 Privacy Definitions

We formalize the privacy goals following the terminology of Pfitzmann and Hansen [45].

Definition 9.4 (Unlinkability). Two items of interest (transactions, addresses, users) are *unlinkable* if, within a given anonymity set, the adversary cannot determine with probability better than random whether the items are related.

Formally, let X and Y be events. (X, Y) are ϵ -unlinkable if:

$$|\Pr[\text{Linked}(X, Y) | \text{View}_{\mathcal{A}}] - \Pr[\text{Linked}(X, Y)]| \leq \epsilon$$

where $\text{View}_{\mathcal{A}}$ is the adversary's view and ϵ is negligible.

Definition 9.5 (Untraceability). A transaction is *untraceable* if, given the transaction, the adversary cannot determine its inputs (sender) or outputs (recipient) with probability better than uniform over the respective anonymity sets.

Formally, for ring $\mathcal{R} = \{P_0, \dots, P_{n-1}\}$ with real index π :

$$\forall i \in [n] : \Pr[\text{Real} = i | \sigma, \mathcal{R}] = \frac{1}{n} + \text{negl}(\lambda)$$

Definition 9.6 (Confidentiality). An attribute (amount, memo, metadata) is *confidential* if the adversary gains no information about it from public transaction data.

Formally, for amount v hidden in commitment C :

$$H(\text{Amount} | C) = H(\text{Amount})$$

where H denotes Shannon entropy.

9.3.2 BothoPrivacy Properties

We now prove **Botho** satisfies these definitions.

Theorem 9.7 (Transaction-Address Unlinkability). *Outputs belonging to different subaddresses of the same master address are (2^{-128}) -unlinkable without knowledge of the view key.*

Proof. Subaddresses are derived as $(C_i, D_i) = (A + \delta_i G, B + \delta_i G)$ where $\delta_i = H_s(\text{"SubAddr"} \| a \| i)$.

Without view key a , the adversary cannot compute δ_i for any i . The hash function output is computationally indistinguishable from random. Thus (C_i, D_i) and (C_j, D_j) for $i \neq j$ appear as independent random points. The reduction from hash collision gives $\epsilon \leq 2^{-128}$. \square

Theorem 9.8 (Input-Output Unlinkability). *Given a transaction with inputs \mathcal{I} and outputs \mathcal{O} , the adversary cannot determine which output corresponds to change.*

Proof. All outputs use identical encoding:

- Same commitment structure ($C = vH + rG$)
- Same one-time key derivation (ML-KEM)
- Identical sizes (including KEM ciphertext)

No syntactic distinguisher exists. Semantic distinguishers (amount ranges, timing) are addressed via uniform output ordering and standardized amounts. \square

9.3.3 Privacy Budget Analysis

Privacy degrades with usage. We model cumulative privacy loss as a “budget” that depletes over time.

Model. Let $\pi_0 = 1/n$ be the initial probability an adversary correctly identifies the real input (uniform over ring). After observing k transactions from the same cluster:

$$\pi_k = 1 - (1 - 1/n)^k \cdot \prod_{i=1}^k (1 - \text{leak}_i) \quad (45)$$

where leak_i represents information leakage from transaction i (timing, amount correlation, etc.).

Privacy half-life. The number of transactions $k_{1/2}$ until $\pi_k = 0.5$:

$$k_{1/2} \approx \frac{\ln 2}{\ln(1 + 1/n) + E[\text{leak}]} \quad (46)$$

For $n = 20$ and $E[\text{leak}] = 0.05$:

$$k_{1/2} \approx \frac{0.693}{0.049 + 0.05} \approx 7 \text{ transactions}$$

Implications:

- Users making many transactions should use churning (self-sends) periodically to reset privacy budget
- Long-term passive holders enjoy persistent privacy
- High-frequency users should consider privacy-time-preference tradeoff

9.3.4 Metadata Leakage Quantification

Beyond cryptographic privacy, transactions leak metadata:

Transaction size reveals input/output structure:

$$\text{Inputs} \approx \frac{\text{size} - \text{base} - \text{outputs} \cdot s_o}{s_i} \quad (47)$$

where $s_i \approx 700$ bytes/input, $s_o \approx 1,152$ bytes/output.

Timing reveals transaction patterns:

- Time-of-day distribution (timezone inference)
- Inter-transaction intervals (behavioral fingerprinting)
- Response to external events (market correlation)

Network layer reveals source information:

- IP address (mitigated by Dandelion++)
- First-seen node (geographic inference)

Quantification table:

Table 22: Metadata leakage channels

Metadata	Bits Leaked	Mitigation	Residual
Tx size	3–5	Fixed-size outputs	1–2
Timing (hour)	4	Random delay	1–2
Network source	10+	Dandelion++	0–2
Fee amount	2–4	Min fee policy	1

Total information leakage: With mitigations, approximately 5–8 bits per transaction. Over 1000 transactions, this accumulates to 5–8 KB of potentially correlatable information.

9.3.5 Information-Theoretic Bounds

We compare Botho’s achieved privacy to theoretical limits.

Optimal unlinkability requires entropy equal to the anonymity set:

$$H_{\text{opt}} = \log_2(n) = \log_2(20) \approx 4.32 \text{ bits} \quad (48)$$

Achieved unlinkability accounting for degradation:

$$H_{\text{achieved}} = \log_2(n_{\text{effective}}) \approx \log_2(6) \approx 2.58 \text{ bits} \quad (49)$$

Privacy efficiency:

$$\eta = \frac{H_{\text{achieved}}}{H_{\text{opt}}} = \frac{2.58}{4.32} \approx 60\% \quad (50)$$

This efficiency reflects the cost of practical decoy selection constraints (age distribution matching, cluster similarity).

Comparison to alternatives:

Table 23: Privacy efficiency comparison

System	Nominal Set	Effective	Efficiency
Botho	20	~6	60%
Monero	16	~4	50%
Zcash (shielded)	∞	Large	~95%
MimbleWimble	Variable	~1	Low

Notes: Zcash achieves higher efficiency but with computational cost and opt-in privacy reducing practical anonymity sets. MimbleWimble’s cut-through provides limited effective privacy due to interaction requirements and transaction graph attacks [17].

9.4 Consensus Security

9.4.1 Double-Spend Resistance

Theorem 9.9 (Double-Spend Prevention). *Under SCP's safety guarantees and assuming quorum intersection, no valid double-spend can be confirmed.*

Proof. We prove by strong induction on block height that no output can be spent twice.

Base case ($h = 0$): The genesis block contains no transactions, hence no double-spends.

Inductive hypothesis: Assume for all heights $h' < h$, no double-spend has been confirmed.

Inductive step: Consider height h . Suppose transactions T_1 and T_2 both attempt to spend output O , producing key image I .

Case 1: T_1 and T_2 are in the same block B_h .

- Block validation requires all key images in B_h to be distinct
- T_1 and T_2 share key image I
- B_h fails validation and is rejected

Case 2: T_1 is in block B_{h_1} and T_2 is in block B_{h_2} where $h_1 < h_2 \leq h$.

- By inductive hypothesis, T_1 is validly included at h_1
- Key image I is added to the key image set \mathcal{I}_{h_1}
- For all $h' > h_1$: $\mathcal{I}_{h'} \supseteq \mathcal{I}_{h_1}$
- Block B_{h_2} validation checks $I \notin \mathcal{I}_{h_2-1}$
- Since $I \in \mathcal{I}_{h_1} \subseteq \mathcal{I}_{h_2-1}$, check fails
- B_{h_2} (containing T_2) is rejected

Case 3: T_1 and T_2 are in competing blocks at the same height.

- SCP safety guarantees: if quorum intersection holds, only one block can be externalized per slot
- Let B_h be the unique externalized block at height h
- At most one of T_1, T_2 can be in B_h
- The other is never confirmed

By induction, double-spending is impossible at any height. □

Corollary 9.10 (Key Image Uniqueness). *Each key image appears in at most one confirmed transaction.*

Proof. Immediate from the double-spend prevention theorem: the key image set \mathcal{I}_h at height h contains exactly the key images from all confirmed transactions in blocks $0, \dots, h-1$, with no duplicates. □

9.4.2 Byzantine Fault Tolerance

The system tolerates Byzantine nodes within quorum thresholds:

Theorem 9.11 (Byzantine Resilience). *If quorum intersection holds and each quorum can tolerate its failure threshold, honest nodes agree on the same externalized values.*

With the default tiered structure (3-of-4 infrastructure + 2-of-3 community), the system tolerates:

- 1 Byzantine infrastructure node, OR
- 1 Byzantine community node, OR
- Combinations below both thresholds

9.4.3 51% Attack Resistance

Unlike pure PoW, Botho is resistant to hashpower-majority attacks:

- **Block proposal:** Majority hashpower can propose blocks more frequently
- **Finalization:** SCP requires quorum agreement regardless of hashpower
- **Result:** Attacker can flood proposals but cannot force finalization

An attacker would need to compromise both hashpower majority AND sufficient quorum members.

9.5 Cryptographic Security

9.5.1 Hash Function Security

Bothouses SHA3-256 [35] and BLAKE3 [41] for hashing:

- Collision resistance: 2^{128} security (quantum: 2^{85})
- Preimage resistance: 2^{256} security (quantum: 2^{128})
- Sufficient for all security requirements

9.5.2 Signature Security

Table 24: Signature security levels

Algorithm	Classical	Quantum	Use
CLSAG	128-bit	0	Ring signatures
ML-DSA-65	192-bit	128-bit	Minting

9.5.3 Key Encapsulation Security

ML-KEM-768 provides:

- IND-CCA2 security against quantum adversaries
- 192-bit classical security / 128-bit quantum security
- Based on hardness of MLWE problem

9.6 Attack Resistance

9.6.1 Timing Attacks

Transaction timing:

- Dandelion++ randomizes propagation timing
- Batching obscures submission time
- Mempool privacy prevents timing correlation

Decoy selection:

- Age distribution matches empirical spend patterns
- Randomization within constraints
- No deterministic selection

9.6.2 Transaction Graph Analysis

Several features resist graph analysis:

- **Ring signatures:** True input hidden among 20 possibilities
- **Stealth addresses:** Each output has unique one-time key
- **Hidden amounts:** Cannot trace by amount matching
- **Multiple outputs:** Change output indistinguishable

9.6.3 Sybil Attacks

Cluster-based progressive fees resist Sybil attacks:

- Splitting coins preserves cluster ancestry
- Creating new identities doesn't reduce fees
- Only genuine economic activity changes cluster factor

9.6.4 Denial of Service

Network-level:

- Rate limiting per peer
- Reputation scoring
- Connection limits
- Resource bounds

Consensus-level:

- PoW requires resources to propose blocks
- Invalid proposals rejected before propagation
- SCP messages bounded by quorum size

Transaction-level:

- Minimum fee requirement
- Validation before relay
- Mempool size limits

9.6.5 Eclipse Attacks

Mitigation:

- Diverse peer selection (multiple ASNs, countries)
- Outbound connection preference
- Bootstrap node diversity
- Detection of peer manipulation

9.6.6 Selfish Mining

Botho's hybrid consensus changes selfish mining dynamics:

- Block withholding delays finalization but doesn't create advantage
- SCP selects among available proposals, not necessarily first
- No "longest chain" to game
- Withholding risks losing the block entirely

9.7 Formal Security Properties

9.7.1 Safety

Property 9.12 (Value Conservation). For any valid block, the sum of all transaction outputs plus fees equals the sum of all inputs plus block reward.

Property 9.13 (No Inflation). The total supply at height h is exactly the sum of all block rewards up to h , minus all burned fees.

Property 9.14 (Key Image Uniqueness). Each output can be spent exactly once; the key image uniquely identifies the spent output.

9.7.2 Liveness

Property 9.15 (Transaction Inclusion). A valid transaction with sufficient fee will be included in a block within bounded time, assuming network synchrony and honest quorum majority.

Property 9.16 (Consensus Progress). If the network is eventually synchronous and quorum intersection holds, SCP will eventually externalize a value for each slot.

9.8 Security Comparison

(IT-secure = information-theoretically secure)

9.9 Detailed Attack Scenarios

This section provides worked examples of common attacks and their outcomes against Botho.

Table 25: Security comparison with other privacy coins

Property	Botho	Monero	Zcash	Grin
PQ recipient privacy	✓	—	—	—
Ring size	20	16	N/A	N/A
Amount hiding	IT-secure	IT-secure	IT-secure	IT-secure
Mandatory privacy	✓	✓	—	✓
Deterministic finality	✓	—	—	—
Trusted setup	—	—	✓	—

9.9.1 Scenario: Timing-Based Deanonymization

Setup: An adversary operates nodes connected to a significant fraction of the network (e.g., 30% of peers).

Attack vector:

1. Monitor transaction first-seen times across controlled nodes
2. Correlate timing with known user activity patterns
3. Attempt to identify transaction origin

Defense mechanisms:

1. **Dandelion++ stem phase:** Transaction travels through 1–10 intermediary nodes before broadcast
2. **Random delays:** Each hop adds 0–5 second random delay
3. **Stem length:** Geometric distribution with $p = 0.1$ (expected length 10 hops)

Analysis:

$$\begin{aligned}
 P(\text{origin identified}) &< P(\text{first node observed}) \times P(\text{that node is origin}) \\
 &= 0.3 \times 0.1 = 0.03
 \end{aligned}$$

With 3% probability per transaction, an adversary would need to observe many transactions from the same user to achieve statistical confidence.

Additional mitigation: Users can further protect themselves by submitting transactions via Tor.

9.9.2 Scenario: Chain Analysis Attack

Setup: A chain analysis company maintains a database of “poisoned” outputs (outputs they created and can identify when spent).

Attack vector:

1. Create many small outputs across many addresses
2. When outputs appear as decoys in other transactions, those are eliminated as possible real inputs
3. Remaining ring members are more likely to be real

Defense mechanisms:

1. **Ring size 20:** Even eliminating known decoys leaves significant anonymity

2. **Decoy selection algorithm:** Prefers outputs with similar cluster tags (70% similarity threshold)
3. **Output distribution:** Natural distribution of outputs dilutes poisoned set

Analysis: Suppose adversary controls 10% of all outputs (unrealistically high). For a ring of 20:

$$E[\text{poisoned decoys}] = 19 \times 0.1 = 1.9$$

Expected effective ring size ≈ 18 , still providing substantial anonymity. In practice, controlling even 1% of outputs requires enormous capital investment.

9.9.3 Scenario: Flood-and-Loot Attack

Classic attack (against pure PoW):

1. Deposit funds at exchange
2. Trade for another asset and withdraw
3. Use hashpower to reorg chain and double-spend deposit

Why it fails against Botho:

1. Exchange waits for transaction to be externalized (SCP finalized)
2. Externalization requires quorum agreement, not just PoW
3. Reverting requires corrupting quorum intersection
4. Even with majority hashpower, attacker cannot force quorum to finalize different block

Cost analysis: To execute this attack, adversary would need:

- >50% hashpower (cost: millions in hardware + electricity)
- Corrupt 2+ infrastructure nodes AND 2+ community nodes
- All this for a single double-spend opportunity

Conclusion: Attack cost far exceeds any reasonable double-spend profit.

9.9.4 Scenario: Lottery Grinding Attack

Hypothesis: Can a miner manipulate block construction to direct lottery rewards to their own UTXOs?

Attack vector:

1. Control minting of blocks
2. Modify transaction ordering/selection to change randomness seed
3. Bias lottery toward controlled UTXOs

Defense mechanism: Lottery randomness is derived from:

$$\text{seed} = \mathcal{H}(\text{block_hash} \parallel \text{future_block_hash}_{+10})$$

Analysis:

- Block hash includes PoW nonce (miner controls)
- Future block hash is unknown at time of minting
- Miner cannot predict which UTXOs will win

Even if miner controls the next 10 blocks (requiring sustained 100% hashpower), they would need to also predict transaction inclusion to manipulate the seed meaningfully.

Cost-benefit: The lottery redistributes only 80% of fees. For grinding to be profitable:

$$\text{grinding_cost} < 0.8 \times \text{total_fees} \times P(\text{success})$$

With negligible success probability, grinding is strictly unprofitable.

9.9.5 Scenario: Eclipse Attack on Light Client

Setup: Adversary controls all connections to a light client.

Attack vector:

1. Feed client fabricated block headers
2. Show non-existent or double-spent transactions as confirmed
3. Trick user into accepting invalid payment

Defense mechanisms:

1. **Multiple connections:** Light clients connect to 8+ diverse peers
2. **PoW verification:** Fake headers require real PoW
3. **SCP proofs:** Externalization proofs verify quorum agreement
4. **Checkpoint verification:** Known checkpoints prevent deep reorgs

Attack cost: To eclipse a light client:

- Must control all 8+ outbound connections
- Must produce valid PoW for fake headers
- Must forge SCP externalization proofs (requires quorum keys)

Practical impossibility: Forging SCP proofs requires private keys of quorum members. Without these, even a fully-eclipsed client will reject blocks that lack valid externalization.

9.9.6 Scenario: Progressive Fee Evasion

Goal: Large holder wants to pay base fees despite high cluster factor.

Attempted strategies:

1. **Splitting:** Divide holdings into many small UTXOs
 - *Result:* All child UTXOs inherit same cluster tag
 - *Outcome:* No fee reduction
2. **Multiple addresses:** Create many addresses
 - *Result:* Cluster factor based on coin ancestry, not address

- *Outcome*: No fee reduction

3. **Wash trading**: Rapid self-transfers to decay tags

- *Result*: Decay requires 720-block minimum age
- *Outcome*: Maximum 12 decay events/day, 27-hour half-life
- *Cost*: Transaction fees during wash trading

4. **Exchange laundering**: Deposit to exchange, withdraw different coins

- *Result*: Works, but requires real economic activity
- *Outcome*: Legitimate use case (this is the intended mechanism for tag blending)

Conclusion: Only genuine economic activity (trading, spending) reduces cluster factor. Sybil attacks provide no benefit.

9.10 Known Limitations

9.10.1 Classical Ring Signature Vulnerability

A sufficiently powerful quantum computer could break CLSAG and identify signers retroactively. Mitigation:

- Ring signature data alone doesn't prove spending
- Economic value of retroactive deanonymization is limited
- Future protocol upgrade to post-quantum rings is possible

9.10.2 Metadata Leakage

Despite cryptographic privacy:

- Transaction size reveals input/output count
- Timing correlation possible with powerful adversary
- Network-level deanonymization without Tor/I2P

9.10.3 Implementation Risks

- Side-channel attacks on key material
- Random number generator quality
- Memory safety issues
- Timing side channels in cryptographic operations

All reference implementation cryptographic code uses constant-time algorithms and is subject to ongoing security audit.

10 Economic Analysis

This section analyzes the game-theoretic properties of **Botho**'s economic mechanisms and their long-term sustainability.

10.1 Incentive Alignment

10.1.1 Miner Incentives

Miners (block proposers) are incentivized to:

1. **Include transactions:** Larger blocks have no disadvantage (fees are burned, not captured)
2. **Maintain network health:** Block rewards depend on network value, not transaction ordering
3. **Stay honest:** Invalid blocks are rejected by SCP

Why fee burning works:

Without fee capture, miners have no incentive to:

- Reorder transactions (MEV elimination)
- Censor low-fee transactions beyond minimum
- Create artificial congestion

10.1.2 User Incentives

Users are incentivized to:

1. **Circulate rather than hoard:** Progressive fees and tail emission create holding costs
2. **Maintain privacy:** Default privacy eliminates coordination problems
3. **Run nodes:** Reduced resource requirements enable participation

10.1.3 Validator Incentives

SCP validators (consensus participants) are incentivized by:

1. **Reputation:** Being in quorum slices requires trustworthiness
2. **Self-interest:** Validators typically also hold currency
3. **Community standing:** Non-economic incentives matter for infrastructure nodes

10.2 Progressive Fee Analysis

10.2.1 Fee Distribution

Under the cluster-based progressive fee system:

$$E[\text{fee}] = f_{\text{base}} \times \text{size} \times E[\text{cluster_factor}] \quad (51)$$

The expected cluster factor depends on wealth distribution:

Table 26: Expected fees by wealth percentile

Percentile	Factor	4KB Tx Fee
0–50	1.0–1.5	4–6 μ BTH
50–90	1.5–3.0	6–12 μ BTH
90–99	3.0–5.0	12–20 μ BTH
99–100	5.0–6.0	20–24 μ BTH

10.2.2 Redistribution Effect

The lottery mechanism creates direct wealth transfer from high-activity, high-cluster users to random UTXO holders:

$$\text{redistribution}_{\text{direct}} = 0.8 \times \sum_{\text{tx}} f_{\text{total}} \quad (52)$$

Additionally, fee burning (20%) redistributes indirectly to all holders by reducing supply:

$$\text{redistribution}_{\text{indirect}} = 0.2 \times \sum_{\text{tx}} f_{\text{total}} \quad (53)$$

Progressive properties of lottery selection:

- Random UTXO selection statistically favors many small holders over few large holders
- Self-custody users (many UTXOs) receive more than custodial users (few UTXOs holding many users' funds)
- Net effect: redistribution from exchanges to individuals

10.2.3 Anti-Hoarding Dynamics

The combination of:

- Tail emission ($\sim 2\%$ annual supply increase)
- Progressive fees (wealth-based costs)
- Lottery redistribution (80% of fees to random UTXOs)
- Fee burning (20% deflationary pressure)

Creates net pressure toward circulation:

$$\text{holding_cost} = \text{dilution} - \text{lottery_income} + \text{opportunity_cost} \quad (54)$$

Users with more UTXOs (typically from active participation) receive more lottery income, partially offsetting dilution. Large holders with few UTXOs experience full dilution with minimal lottery income.

10.3 Game-Theoretic Analysis

10.3.1 Miner Strategy

Proposition: Honest mining is a Nash equilibrium.

Proof sketch. Consider miner M with hashpower fraction α :

- Honest strategy: Expected reward $\alpha \times R$ per block

- Withholding: Risk of block rejection by SCP (no benefit)
- Invalid blocks: Rejected, wasted computation
- Transaction censorship: No benefit (fees burned anyway)

No deviation improves expected payoff. □

10.3.2 Validator Strategy

Proposition: Honest validation is incentive-compatible for economically invested validators.

Proof sketch. Validators typically hold currency. Byzantine behavior risks:

- Network failure (total loss of holdings)
- Exclusion from quorum slices (loss of influence)
- Reputation damage (non-economic but real)

The expected loss exceeds any potential gain from misbehavior. □

10.3.3 User Strategy

Proposition: Using the system as designed is individually rational.

Users face choices:

- **Privacy:** Mandatory, so no choice to make
- **Fee payment:** Required for transaction inclusion
- **Holding vs. spending:** Mild pressure toward spending, but not coercive

10.4 Long-Term Sustainability

10.4.1 Security Budget

The annual security budget (miner revenue) is:

$$\text{budget} = R_{\text{tail}} \times \text{blocks_per_year} \times \text{BTH_price} \quad (55)$$

With tail emission, this is guaranteed to be positive regardless of transaction volume.

10.4.2 Network Effect

Privacy creates positive network effects:

- More users \rightarrow larger anonymity sets
- Larger anonymity sets \rightarrow better privacy
- Better privacy \rightarrow more users

This creates a virtuous cycle encouraging adoption.

10.4.3 Fee Market Stability

Fee burning eliminates fee market volatility:

- No bidding wars for block inclusion
- Predictable costs for users
- No miner incentive to manipulate fees

The minimum fee provides sufficient spam resistance while remaining affordable.

10.5 Comparison with Alternatives

10.5.1 Bitcoin Model

Table 27: Bitcoin vs Botho economic comparison

Property	Bitcoin	Botho
Supply cap	21M	None (tail)
Long-term inflation	0%	~2%
Security funding	Fees only	Emission + burn
Fee predictability	Low	High
Wealth concentration	High	Moderate

10.5.2 Monero Model

Table 28: Monero vs Botho economic comparison

Property	Monero	Botho
Tail emission	Yes	Yes
Fee destination	Miners	Burned
Progressive fees	No	Yes
Dynamic timing	No	Yes

10.6 Quantitative Economic Modeling

This section presents formal modeling and simulation results for **Botho**'s economic mechanisms.

10.6.1 Progressive Fee Impact on Wealth Distribution

We model wealth distribution evolution under progressive fees using a Markov chain on the Gini coefficient. Let G_t denote the Gini coefficient at time t :

$$G_{t+1} = G_t - \alpha \cdot f(G_t) + \beta \cdot g(V_t) + \epsilon_t \quad (56)$$

where:

- $f(G_t) = G_t^2 \cdot k_{\text{prog}}$ captures progressive fee redistribution effect
- $g(V_t) = V_t/V_{\text{max}}$ models transaction volume V_t
- $k_{\text{prog}} \approx 0.02$ is the progressive fee strength coefficient

- ϵ_t represents exogenous wealth shocks

Simulation parameters:

- Initial Gini: $G_0 = 0.85$ (comparable to Bitcoin/Monero)
- Simulation horizon: 10 years (3.15M blocks)
- Transaction rate: 10 tx/s average
- Monte Carlo iterations: 10,000

Results: Under baseline assumptions, the equilibrium Gini coefficient converges to $G^* \approx 0.65$ (95% CI: [0.58, 0.72]), representing a 23% reduction from initial conditions. This occurs through:

Table 29: Gini coefficient evolution simulation

Year	Mean G	Std Dev	Min	Max
0	0.850	0.000	0.850	0.850
1	0.812	0.021	0.758	0.869
3	0.745	0.035	0.654	0.831
5	0.695	0.042	0.592	0.798
10	0.651	0.047	0.545	0.762

Sensitivity analysis: The redistribution effect is most sensitive to transaction volume. At 1 tx/s (low adoption), Gini reduction is only 8%. At 100 tx/s (high adoption), reduction reaches 35%.

10.6.2 Monte Carlo Lottery Analysis

The lottery mechanism distributes 80% of transaction fees to random UTXOs. We analyze the statistical properties of this redistribution.

Model. Let a user hold n UTXOs out of N total. Each lottery payment (with fee F) selects k winning UTXOs uniformly at random. The expected lottery income per payment is:

$$E[\text{income}] = \frac{0.8F \cdot k \cdot n}{N} \quad (57)$$

Variance analysis. The variance of lottery income is:

$$\text{Var}[\text{income}] = \frac{(0.8F)^2 \cdot k \cdot n \cdot (N - n)}{N^2 \cdot (N - 1)} \quad (58)$$

For small holdings ($n \ll N$), variance scales linearly with n , making returns predictable over time via the law of large numbers.

Simulation results (1M blocks, $N = 10^6$ UTXOs, $k = 5$):

Table 30: Lottery income by UTXO count

UTXOs Held	Mean Income	Std Dev	CV	Win Prob/Block
10	4.0 μ BTH/block	2.8	70%	5×10^{-5}
100	40 μ BTH/block	8.9	22%	5×10^{-4}
1,000	400 μ BTH/block	28	7%	5×10^{-3}
10,000	4.0 mBTH/block	89	2.2%	5×10^{-2}

Key finding: Small holders receive lottery income with high variance but positive expected value. Over 10,000 blocks, 99% of users with ≥ 10 UTXOs receive *some* lottery income.

10.6.3 Formal Nash Equilibrium Analysis

We prove that honest participation constitutes a Nash equilibrium under reasonable assumptions.

Definition 10.1 (Miner Game). The miner game $\Gamma_M = (N, S, u)$ consists of:

- Players $N = \{1, \dots, m\}$ (miners with hashpower α_i)
- Strategy space $S_i = \{\text{honest}, \text{deviate}\}$
- Payoff $u_i(s) = R \cdot \alpha_i \cdot \mathbb{1}[\text{block accepted}]$

Theorem 10.2 (Miner Nash Equilibrium). *In Γ_M , the strategy profile where all miners play honest is a Nash equilibrium for any $\alpha_i < 0.5$.*

Proof. Consider miner i with hashpower α_i . Under honest strategy:

$$u_i(\text{honest}, s_{-i}^*) = R \cdot \alpha_i \cdot P(\text{valid}) = R \cdot \alpha_i$$

Under deviation (invalid block, withholding, or censorship):

1. **Invalid block:** SCP rejects with probability 1. $u_i(\text{invalid}) = 0 < R \cdot \alpha_i$.
2. **Block withholding:** Reduces probability of own block selection. $u_i(\text{withhold}) < R \cdot \alpha_i$.
3. **Transaction censorship:** No payoff change (fees burned). $u_i(\text{censor}) = R \cdot \alpha_i = u_i(\text{honest})$.

No deviation strictly improves payoff. Censorship is weakly dominated due to reputation costs not modeled here. \square

Definition 10.3 (Validator Coalition Game). The coalition game $\Gamma_V = (N, v)$ consists of:

- Validators N with quorum slice structure \mathcal{Q}
- Characteristic function $v : 2^N \rightarrow \mathbb{R}$ where $v(S)$ is the value coalition S can guarantee

Theorem 10.4 (Coalition Resistance). *Under quorum intersection, no coalition S with $|S| < f_{\text{threshold}}$ can guarantee positive deviation payoff.*

Proof sketch. A deviating coalition must either:

1. Block consensus (requires breaking quorum intersection)
2. Force different values (requires $> f$ Byzantine nodes)

Both require coalition size exceeding Byzantine threshold. Below threshold, $v(S) \leq 0$ for any deviating strategy. \square

10.6.4 Economic Attack Cost Analysis

We quantify the cost of various economic attacks.

51% Attack Cost. Unlike pure PoW, Both requires both hashpower majority AND quorum corruption:

$$\text{Cost}_{51\%} = \text{Cost}_{\text{hashpower}} + \text{Cost}_{\text{quorum}} \quad (59)$$

For hashpower (assuming \$0.05/kWh electricity, efficient mining):

$$\text{Cost}_{\text{hashpower}} = \frac{0.51 \cdot H_{\text{network}} \cdot P_{\text{unit}}}{E_{\text{unit}}} \cdot t_{\text{attack}} \quad (60)$$

For quorum corruption (must compromise $\geq f + 1$ validators):

$$\text{Cost}_{\text{quorum}} \geq (f + 1) \cdot V_{\text{stake}} \cdot P_{\text{bribe}} \quad (61)$$

where P_{bribe} represents the fraction of stake validators require to defect.

Cluster Factor Manipulation. To reduce cluster factor from f to f' :

$$\text{Cost}_{\text{cluster}} = \text{fee}(\text{mixing}) + \text{slippage} + \text{counterparty_cost} \quad (62)$$

Mixing requires acquiring low-cluster UTXOs through genuine trade. With efficient markets, arbitrage prevents free cluster factor reduction.

Lottery Grinding. Miners cannot influence random seed (derived from previous block hash and deterministic transaction ordering). Cost of grinding:

$$\text{Cost}_{\text{grind}} = E[\text{hashpower}] \cdot t_{\text{grind}} \cdot P_{\text{electricity}} \gg E[\text{lottery_gain}] \quad (63)$$

Grinding is unprofitable: expected lottery gain is proportional to UTXOs held, not hash-power.

10.7 Economic Risks

10.7.1 Low Adoption

If adoption remains low:

- Tail emission maintains miner incentive
- Progressive fees have minimal impact (few large holders)
- Network can persist indefinitely at low scale

10.7.2 Mining Centralization

Risks and mitigations:

- **Risk:** ASIC development concentrates mining
- **Mitigation:** CPU-friendly PoW (RandomX-based)
- **Risk:** Pool concentration
- **Mitigation:** Solo mining viable with tail emission

10.7.3 Validator Cartel

If validators collude:

- SCP's tiered structure provides defense
- Community can adjust quorum slices
- Economic self-interest opposes cartels

10.8 Economic Constants Summary

11 Implementation

This section describes the reference implementation of **Botho** and its performance characteristics.

Table 31: Economic parameters

Parameter	Value	Rationale
Tail inflation	$\sim 2\%$	Security sustainability
Max fee multiplier	$6\times$	Balance incentive/burden
Ring size	20	Privacy/size tradeoff
Min block time	5s	Throughput ceiling
Max block time	40s	Emission floor

11.1 Reference Implementation

11.1.1 Architecture Overview

The reference implementation is written in Rust for memory safety and performance:

```

botho/
+-- account-keys/      # Key derivation (BIP39, SLIP-10)
+-- botho/             # Full node implementation
+-- botho-wallet/      # Command-line wallet
+-- cluster-tax/       # Progressive fee calculation
+-- consensus/scp/     # Stellar Consensus Protocol
+-- crypto/
| +-- box/             # Encryption primitives
| +-- keys/            # Key management
| +-- lion/            # (Deprecated) Lattice signatures
| +-- pq/              # Post-quantum (ML-KEM, ML-DSA)
| +-- ring-signature/  # CLSAG implementation
+-- transaction/
| +-- core/            # Transaction types and validation
| +-- signer/          # Transaction signing
| +-- types/           # Shared types
+-- web/               # Desktop wallet (Tauri)

```

11.1.2 Dependencies

Key external dependencies:

- **curve25519-dalek** [10]: Ristretto255 group operations
- **bulletproofs**: Range proof implementation
- **pqcrypto**: ML-KEM and ML-DSA implementations
- **libp2p** [47]: Peer-to-peer networking
- **rocksdb** [15]: Key-value storage for blockchain data

11.1.3 Build Requirements

- Rust 1.75+ (stable)
- CMake 3.16+ (for native dependencies)
- 8 GB RAM minimum for compilation
- Linux, macOS, or Windows (with WSL2)

11.2 Performance Characteristics

11.2.1 Transaction Validation

Table 32: Validation timing (AMD Ryzen 9 5900X)

Operation	Time
CLSAG signature verify (ring=20)	2.1 ms
Bulletproof verify (2 outputs)	1.8 ms
ML-KEM decapsulate	0.05 ms
Full transaction validate	4.2 ms

11.2.2 Transaction Creation

Table 33: Transaction creation timing

Operation	Time
CLSAG signature create (ring=20)	3.8 ms
Bulletproof create (2 outputs)	45 ms
ML-KEM encapsulate	0.04 ms
Full transaction create	52 ms

11.2.3 Throughput

Table 34: System throughput

Metric	Value	Conditions
Max TPS (validation)	238 tx/s	Single core
Max TPS (validation)	1,428 tx/s	6 cores
Block validation	120 ms	100 tx block
Initial sync	4 hours	1M blocks

11.2.4 Resource Usage

11.3 Wallet Implementations

11.3.1 Command-Line Wallet

The botho-wallet CLI provides:

```
botho-wallet init      # Create new wallet
botho-wallet balance   # Show balance
botho-wallet address   # Generate receiving address
botho-wallet send      # Create transaction
botho-wallet history   # Transaction history
botho-wallet sync      # Synchronize with network
botho-wallet export    # Export view key
```

Table 35: Resource requirements

Node Type	Storage	Memory
Full node (unpruned)	50 GB	4 GB
Full node (pruned)	10 GB	2 GB
Light client	500 MB	256 MB
Minting node	50 GB	8 GB

11.3.2 Desktop Wallet

A cross-platform desktop wallet built with Tauri provides:

- Graphical interface for all wallet operations
- Secure mnemonic generation and storage
- Transaction history with privacy indicators
- Node connection management
- Subaddress management

11.3.3 Exchange Scanner

The `botho-exchange-scanner` supports exchange integration:

- View-key-only scanning for deposits
- Webhook notifications for incoming transactions
- Subaddress management for customer deposits
- No spend key required (security isolation)

11.4 Security Measures

11.4.1 Memory Protection

- Key material stored in `Zeroizing` wrappers
- Memory locking (`mlock`) for sensitive data
- Immediate zeroization on drop
- No key material in debug output

11.4.2 Constant-Time Operations

All cryptographic operations use constant-time algorithms:

- Scalar multiplication (`curve25519-dalek`)
- Signature verification
- Key comparison
- Decapsulation

11.4.3 Input Validation

Extensive input validation prevents:

- Buffer overflows (Rust memory safety)
- Invalid curve points (checked on deserialization)
- Integer overflows (checked arithmetic)
- Denial of service (resource limits)

11.5 Testing

11.5.1 Test Coverage

- Unit tests: 2,400+ tests
- Integration tests: 150+ tests
- Property-based tests: Key derivation, serialization
- Fuzzing: Transaction parsing, network messages

11.5.2 Test Networks

- **Testnet**: Public test network with accelerated timing
- **Stagenet**: Pre-production testing environment
- **Regtest**: Local regression testing mode

11.6 Deployment

11.6.1 Docker

```
docker run -d -p 9732:9732 -v /data/botho:/data botho/node:latest --data-dir /data
```

11.6.2 Systemd

```
[Unit]
Description=Botho Node
After=network.target

[Service]
Type=simple
User=botho
ExecStart=/usr/local/bin/botho --config /etc/botho/config.toml
Restart=on-failure
MemoryMax=8G

[Install]
WantedBy=multi-user.target
```

11.7 Scalability Analysis

This section analyzes Botho's scalability characteristics and long-term growth projections.

Table 36: Projected blockchain size growth

Year	Blocks	Unpruned	Pruned
1	3.2M	15 GB	3 GB
5	16M	75 GB	12 GB
10	32M	150 GB	20 GB
20	64M	300 GB	35 GB

11.7.1 Blockchain Growth

Assumptions: 10-second average block time, 50% block utilization, 4 KB average transaction size.

11.7.2 UTXO Set Growth

The UTXO set grows with network usage:

$$\text{UTXO}_{\text{count}}(t) = \text{UTXO}_0 + \int_0^t (\text{outputs}(\tau) - \text{spends}(\tau)) d\tau \quad (64)$$

Growth projections:

- Year 1: ~5 million UTXOs (~800 MB)
- Year 5: ~20 million UTXOs (~3.2 GB)
- Year 10: ~35 million UTXOs (~5.6 GB)

Mitigation: UTXO set is significantly smaller than full blockchain and fits comfortably in RAM for most systems.

11.7.3 Key Image Database

The key image database grows monotonically (spent outputs are never removed):

$$\text{KeyImages}(t) = \sum_{\tau=0}^t \text{inputs}(\tau) \quad (65)$$

Growth projections (32 bytes per key image):

- Year 1: ~10 million key images (~320 MB)
- Year 5: ~50 million key images (~1.6 GB)
- Year 10: ~100 million key images (~3.2 GB)

Storage optimization: Key images are stored in a Bloom filter for fast negative lookups, with full database for confirmation.

11.7.4 Cluster Tag Database

Cluster tag vectors require additional storage:

- Per UTXO: Variable (average ~100 bytes for tag vector)
- Total: Approximately $2 \times$ UTXO set storage

Pruning strategy: Old cluster data for spent outputs can be pruned after sufficient confirmations.

11.7.5 Throughput Limits

Table 37: Throughput analysis

Constraint	Limit	Bottleneck
Block size (2 MB)	~500 tx/block	Storage/bandwidth
Block time (5s min)	100 TPS peak	Consensus latency
Validation speed	1,400 TPS	CPU (6 cores)
Network propagation	200 TPS	Bandwidth (10 Mbps)
Effective limit	~100 TPS	Block time

11.7.6 Comparison to Existing Systems

Table 38: Throughput comparison

System	TPS	Finality	Privacy
Bitcoin	~7	60 min	None
Monero	~30	20 min	Ring signatures
Zcash (shielded)	~10	75 min	ZK proofs
Botho	~100	5–10s	Ring + PQ stealth

11.7.7 Layer-2 Scaling

For higher throughput requirements, layer-2 solutions are planned:

Payment Channels:

- Open channel with on-chain transaction
- Unlimited off-chain transfers between parties
- Close channel to settle final balances
- Privacy preserved via channel encryption

Design considerations:

- Ring signatures complicate HTLC-style constructions
- Stealth addresses require novel routing approaches
- Research ongoing for privacy-preserving channel networks

State Channels:

- Generalized off-chain computation
- Requires scripting capability (future upgrade)
- Enables complex multi-party protocols

11.7.8 Pruning Effectiveness

Pruned nodes retain full validation capability while reducing storage:

Pruning ratio: Approximately 80% of historical data can be pruned after sufficient confirmations (default: 10,000 blocks).

Table 39: Pruning analysis

Data Type	Prunable?	Reason
Old block bodies	Yes	Only headers needed for chain
Spent TX outputs	Yes	Not needed for validation
Block headers	No	Required for chain verification
UTXO set	No	Required for spending
Key image set	No	Required for double-spend detection

11.7.9 Node Hardware Requirements

Table 40: Recommended hardware by use case

Use Case	CPU	RAM	Storage
Light client	Any	512 MB	1 GB
Pruned full node	2 cores	4 GB	20 GB SSD
Archive full node	4 cores	8 GB	200 GB SSD
Minting node	8 cores	16 GB	200 GB NVMe

11.8 Future Development

11.8.1 Planned Improvements

- **Mobile wallet:** iOS and Android applications
- **Hardware wallet support:** Ledger/Trezor integration
- **Atomic swaps:** Cross-chain exchange without intermediaries
- **Payment channels:** Layer-2 scaling for micropayments

11.8.2 Post-Quantum Ring Signatures

Future protocol upgrade may introduce post-quantum ring signatures once efficient constructions become practical:

- Lattice-based ring signatures (research ongoing)
- Hybrid classical + PQ approach
- Backward compatibility via soft fork

11.9 User Experience Design

Privacy-preserving systems often sacrifice usability. Bothoprioritizes user experience while maintaining strong privacy defaults.

11.9.1 Address Format

Addresses use a human-friendly format with error detection:

`botho1[network][type][payload][checksum]`

Example:

botho1mq7k9p8z4x5c3v2b1n0m9q8w7e6r5t4y3u2i1o0p9a8s7d6f

```
  ^   ^   ^-----^   ^-----^
  |   |           payload           checksum
  |   |           type (main=m, sub=s)
  |   |           network (mainnet=m, testnet=t)
```

Design choices:

- **Prefix “botho1”:** Clear identification, avoids confusion with other cryptocurrencies
- **Bech32 encoding:** Error detection, case-insensitive, no ambiguous characters (0/O, 1/l)
- **Network identifier:** Prevents accidental cross-network sends
- **Checksum:** 6-character checksum catches typos

11.9.2 Transaction Confirmation UX

Users receive clear feedback on transaction status:

Table 41: Transaction status indicators

Status	Meaning	Wait Time
Pending	In mempool, not yet in block	0–5s
Confirming	In block, awaiting SCP	5–10s
Finalized	SCP externalized	Instant

Key UX improvement: Unlike Bitcoin (10+ confirmations) or Monero (10+ confirmations), Bothotransactions are final after single SCP externalization—typically 5–10 seconds total.

11.9.3 Wallet Backup and Recovery

Mnemonic-based backup:

- 24-word BIP39 mnemonic (256 bits entropy)
- Optional passphrase for additional security
- Derives all keys deterministically
- Compatible with hardware wallets

Recovery process:

1. Enter 24-word mnemonic (and passphrase if used)
2. Wallet derives all keys from seed
3. Wallet scans blockchain for owned outputs
4. Full balance and history restored

Scan time: Approximately 1 hour per year of blockchain history (can be parallelized with multiple full nodes).

11.9.4 Subaddress Management

Subaddresses provide unlimited receiving addresses from a single wallet:

```
// Generate labeled subaddress
wallet.create_subaddress("Alice payment")
// -> botho1s7k9p8z4x5c3v2b1n0m9q8w7e6r5t4y3u2i1o0p9a8s7d6g

// List subaddresses with balances
wallet.list_subaddresses()
// -> [(0, "Primary", 100.0 BTH),
//      (1, "Alice payment", 5.0 BTH),
//      (2, "Bob refund", 0.0 BTH)]
```

Use cases:

- Separate addresses for different payers
- Merchant payment tracking
- Exchange deposit addresses
- Privacy (fresh address per transaction)

11.9.5 Fee Estimation

Wallets provide clear fee information:

Transaction Summary:

```
Amount:      50.00 BTH
Base fee:     0.001 BTH
Cluster factor: 1.2x
Total fee:    0.0012 BTH
```

[Expected confirmation: ~5 seconds]

Fee transparency: Users see both the base fee and their cluster factor multiplier, promoting awareness of the progressive fee mechanism.

11.9.6 Error Handling

Clear, actionable error messages:

Table 42: Example error messages

Situation	Message
Insufficient funds	“Balance (45.00 BTH) insufficient for amount (50.00) plus fee (0.001)”
Invalid address	“Address checksum invalid. Please verify the address.”
Network offline	“Cannot connect to network. Check internet connection.”
Wallet locked	“Wallet is locked. Enter password to unlock.”

11.9.7 Privacy Indicators

Wallets display privacy-relevant information:

- **Ring size:** Number of decoys used (always 20)
- **Output age:** How long outputs have existed
- **Cluster factor:** Current fee multiplier
- **Network privacy:** Tor/I2P connection status

Privacy score: Optional aggregate privacy indicator (1–5 stars) based on output age distribution, cluster diversity, and network connection.

11.9.8 Accessibility

- **Screen reader support:** All UI elements labeled
- **Keyboard navigation:** Full functionality without mouse
- **High contrast mode:** For visual impairment
- **Font scaling:** Adjustable text size

11.9.9 Internationalization

- **Languages:** English, Spanish, Chinese, Japanese, German, French, Portuguese, Russian, Korean, Arabic
- **Number formats:** Locale-aware decimal separators
- **Date formats:** ISO 8601 with locale display
- **RTL support:** Arabic, Hebrew layouts

11.10 Testnet Deployment

A public testnet is available for development and testing.

11.10.1 Testnet Parameters

Table 43: Testnet vs. mainnet parameters

Parameter	Testnet	Mainnet
Block time target	10s	5–40s
Initial block reward	100 BTH	50 BTH
Difficulty adjustment	Every 72 blocks	Every 144 blocks
Ring size	20	20
Minimum fee	0 (free)	1 μ BTH/byte
Network port	19732	9732
Magic bytes	0x7E57	0xB07B

11.10.2 Faucet

Testnet coins are freely available:

```
# Web faucet
https://faucet.testnet.botho.org

# CLI request (1000 tBTH daily limit)
botho-wallet --testnet faucet request <address>

# Response
Transaction broadcast: 0x7f3a...2b1c
Expected confirmation: ~30 seconds
```

11.10.3 Genesis Block

```
GenesisBlock
  timestamp: 1704067200, // 2024-01-01 00:00:00 UTC
  difficulty: 0x00000fff...fff, // Low initial difficulty
  nonce: 0,

  // Genesis coinbase to testnet foundation
  minting_tx:
    outputs: [
      amount: 1_000_000, recipient: "testnet_treasury" ,
    ],
    minter: TestnetFoundationKey,
  ,

  // Genesis quorum set
  initial_validators: [
    "testnet1.botho.org",
    "testnet2.botho.org",
    "testnet3.botho.org",
  ],
```

Genesis hash (testnet): 0x1a2b3c... (truncated for brevity)

11.10.4 Testnet Reset Policy

The testnet may be reset periodically:

- Scheduled resets announced 2 weeks in advance
- Protocol upgrades may require chain reset
- No guarantee of persistent state or balances
- Export wallet seeds before announced resets

11.11 Deployment Guide

11.11.1 Node Configuration

```
[network]
listen_addr = "0.0.0.0:9732"
external_addr = "auto" # Auto-detect public IP
max_peers = 125
```

```

bootstrap_nodes = [
    "node1.botho.org:9732",
    "node2.botho.org:9732",
]

[consensus]
role = "full" # "full", "minting", or "light"
quorum_set = "default" # Or custom configuration

[storage]
data_dir = "/var/lib/botho"
db_cache_mb = 512
prune_blocks = false # Keep full history

[mining]
enabled = false
threads = 0 # 0 = auto-detect
coinbase_address = "<your_address>"

[rpc]
enabled = true
listen = "127.0.0.1:9733"
cors_origins = ["http://localhost:*"]

[logging]
level = "info" # debug, info, warn, error
file = "/var/log/botho/node.log"
max_size_mb = 100

```

11.11.2 Recommended System Requirements

Table 44: Hardware requirements by role

Resource	Light	Full	Minting
CPU cores	1	2	4+
RAM	512 MB	4 GB	8 GB
Storage	1 GB	100 GB	200 GB
Bandwidth	10 Mbps	50 Mbps	100 Mbps

11.11.3 Security Hardening

- **Firewall:** Allow only ports 9732 (P2P) and 9733 (RPC, localhost only)
- **User isolation:** Run node as dedicated non-root user
- **Key storage:** Use hardware security modules for minting keys
- **Updates:** Subscribe to security announcements

```

[Unit]
Description=Botho Full Node
After=network.target

```

```

[Service]
Type=simple
User=botho

```

```
ExecStart=/usr/local/bin/botho --config /etc/botho/config.toml
Restart=on-failure
RestartSec=10
LimitNOFILE=65535
```

```
[Install]
WantedBy=multi-user.target
```

11.11.4 Monitoring and Metrics

The node exposes Prometheus-compatible metrics:

```
# Blockchain metrics
botho_block_height 123456
botho_block_time_seconds 8.2
botho_transactions_per_block 42

# Network metrics
botho_peer_countdirection="inbound" 45
botho_peer_countdirection="outbound" 8
botho_bandwidth_bytesdirection="rx" 1234567
botho_bandwidth_bytesdirection="tx" 987654

# Mempool metrics
botho_mempool_size 156
botho_mempool_bytes 623400

# Consensus metrics
botho_scp_slot 123456
botho_scp_phase "externalize"
botho_scp_latency_ms 2100
```

Alerting recommendations:

- Alert if block height stale for > 5 minutes
- Alert if peer count < 5
- Alert if mempool size $> 10,000$ transactions
- Alert if SCP latency > 10 seconds

11.11.5 Backup and Recovery

```
# Stop node before backup
systemctl stop botho

# Backup blockchain data (cold backup)
tar -czf botho-backup-$(date +%Y%m%d).tar.gz /var/lib/botho

# Hot backup using RocksDB checkpoint
botho-cli db checkpoint /backup/botho-checkpoint

# Wallet backup (encrypted)
botho-wallet export --encrypted > wallet-backup.enc

# Restore from backup
tar -xzf botho-backup-20240101.tar.gz -C /var/lib/botho
systemctl start botho
```

11.12 Code Availability

The reference implementation is open source:

- Repository: <https://github.com/botho-project/botho>
- License: MIT
- Contributing: See CONTRIBUTING.md
- Security issues: security@botho.org

12 Governance and Protocol Upgrades

Protocol evolution is essential for long-term viability. Botho employs a structured governance process that balances agility with stability, enabling necessary upgrades while protecting users from disruptive changes.

12.1 Design Principles

12.1.1 Conservative by Default

The protocol favors stability over rapid iteration:

- Changes require demonstrated necessity, not merely novelty
- Backward compatibility is preserved wherever feasible
- Breaking changes require extended notice and migration support
- Security fixes take precedence over feature additions

12.1.2 Transparency and Predictability

All governance processes are public and follow documented procedures:

- Proposal discussions occur in public forums
- Technical specifications are peer-reviewed before implementation
- Activation timelines are announced well in advance
- Post-activation monitoring reports are published

12.2 Protocol Versioning

12.2.1 Version Scheme

Botho uses semantic versioning with blockchain-specific semantics:

MAJOR.MINOR.PATCH

MAJOR: Consensus-breaking changes (hard fork required)

MINOR: New features, backward compatible

PATCH: Bug fixes, no new features

Examples:

- 1.0.0 → 1.0.1: Security patch, no action required
- 1.0.0 → 1.1.0: New transaction type added, old nodes still valid
- 1.0.0 → 2.0.0: Consensus rule change, upgrade mandatory

12.2.2 Version Negotiation

Nodes advertise their protocol version during handshake:

```
VersionMessage
    protocol_version: (u16, u16, u16), // MAJOR.MINOR.PATCH
    min_supported: (u16, u16, u16),    // Minimum peer version
    features: BitFlags,                 // Optional feature support
    user_agent: String,                 // Implementation identifier
```

Nodes with incompatible versions (different MAJOR) disconnect gracefully with an informative error message.

12.3 Upgrade Mechanisms

12.3.1 Soft Forks

Soft forks tighten consensus rules—old nodes accept new blocks, but new nodes may reject some blocks old nodes would accept.

Use cases:

- Adding new transaction validation rules
- Restricting previously-allowed behaviors
- Deprecating obsolete features

Activation process:

1. **Proposal:** BIP-style document published with technical specification
2. **Implementation:** Reference client updated with activation logic
3. **Signaling:** Miners include version bits in block headers
4. **Lock-in:** When 95% of blocks in a 2016-block window signal support
5. **Activation:** Rules enforced after additional 2016-block grace period

```
BlockHeader
    version: u32, // Bits 0-28: version, bits 29-31: soft fork signals
    // ...
```

```
// Check signaling for proposal #3
fn signals_proposal(header: &BlockHeader, proposal: u8) -> bool
    (header.version >> (29 + proposal)) & 1 == 1
```

12.3.2 Hard Forks

Hard forks relax or modify consensus rules—old nodes reject new blocks.

Use cases:

- Fundamental protocol changes (new cryptographic primitives)
- Emergency security responses
- Removal of technical debt

Activation process:

1. **Proposal:** Extended discussion period (minimum 6 months)
2. **Specification:** Formal specification with test vectors
3. **Implementation:** All major clients implement changes
4. **Testnet:** Minimum 3-month testnet validation
5. **Announcement:** Activation height announced 3+ months ahead
6. **Activation:** At specified block height

Mandatory upgrade policy: Users have minimum 6 months warning before hard fork activation, except for emergency security fixes.

12.4 Proposal Process

12.4.1 Botho Improvement Proposals (BIPs)

Formal proposals follow a structured format:

BIP-XXXX: Title

Status: Draft | Review | Accepted | Final | Rejected | Withdrawn

Type: Standards Track | Informational | Process

Layer: Consensus | Network | Application

Abstract: [200 words max]

Motivation: Why is this change needed?

Specification: Technical details

Rationale: Design decisions and alternatives considered

Backward Compatibility: Impact analysis

Test Cases: Validation criteria

Reference Implementation: Link to code

Security Considerations: Threat analysis

12.4.2 Proposal Lifecycle

1. **Draft:** Author publishes initial proposal for feedback
2. **Review:** Community discussion, technical review
3. **Accepted:** Core maintainers approve for implementation
4. **Final:** Implemented and deployed (or rejected/withdrawn)

Review criteria:

- Technical soundness and security analysis
- Backward compatibility assessment
- Implementation complexity vs. benefit
- Community consensus on necessity

12.5 Emergency Procedures

12.5.1 Security Vulnerabilities

Critical vulnerabilities follow expedited handling:

1. **Discovery:** Reported via secure channel (security@botho.io)
2. **Assessment:** Core team evaluates severity within 24 hours
3. **Patch development:** Fix developed in private
4. **Coordinated disclosure:** Major node operators notified
5. **Public release:** Patch released with advisory
6. **Post-mortem:** Public analysis after fix deployed

Severity classification:

Table 45: Vulnerability severity levels

Level	Definition	Response Time
Critical	Funds at immediate risk	24–48 hours
High	Exploitable with effort	1–2 weeks
Medium	Limited impact	1–2 months
Low	Minimal risk	Next release

12.5.2 Network Emergencies

Catastrophic events (sustained 51% attack, consensus failure) trigger emergency response:

1. Core team assesses situation and coordinates response
2. Emergency communication via all channels
3. If necessary: checkpoint enforcement, emergency hard fork
4. Post-incident analysis and preventive measures

Checkpoint authority: In extreme emergencies, core developers may publish signed checkpoints that compliant nodes will refuse to reorg past. This is a last-resort mechanism that trades decentralization for safety.

12.6 Deprecation Policy

12.6.1 Feature Deprecation

Obsolete features are phased out gradually:

1. **Deprecation notice:** Feature marked deprecated in release notes
2. **Warning period:** Nodes log warnings when deprecated features used
3. **Soft removal:** Feature disabled by default, available via flag
4. **Hard removal:** Feature removed entirely (requires hard fork)

Minimum timeline: 12 months from deprecation notice to hard removal.

12.6.2 API Stability

External interfaces (RPC, wallet format) follow stability guarantees:

- **Stable APIs:** 24+ months support after deprecation
- **Experimental APIs:** May change without notice (clearly marked)
- **Internal APIs:** No stability guarantee

12.7 Decentralization Considerations

12.7.1 Avoiding Governance Capture

Governance mechanisms are designed to resist capture:

- **No formal voting:** Rough consensus, not vote counting
- **Multiple implementations:** No single codebase controls protocol
- **Economic alignment:** Core developers hold BTH, aligned with users
- **Transparent process:** All decisions documented publicly

12.7.2 Fork Rights

Users retain the ultimate governance power—the right to fork:

- Protocol specification is open and unencumbered
- Reference implementation is permissively licensed (MIT)
- No trademark restrictions on protocol-compatible implementations
- Community can reject controversial changes by running alternatives

Philosophy: Formal governance exists to coordinate, not to rule. The community’s ability to fork ensures that governance serves users rather than capturing them.

12.8 Upgrade Roadmap

12.8.1 Planned Future Upgrades

Table 46: Tentative upgrade roadmap

Feature	Type	Status
View key audit mode	Soft fork	Under review
Bulletproofs+	Soft fork	Specification draft
PQ ring signatures	Hard fork	Research phase
Payment channels	Application	Design phase

Note: Roadmap items are aspirational and subject to community consensus. No timeline commitments are made for research-phase items.

13 Conclusion

13.1 Summary

This paper has presented **Botho**, a privacy-preserving cryptocurrency that addresses fundamental challenges in existing systems through principled design tradeoffs.

13.1.1 Key Contributions

1. **Hybrid post-quantum architecture:** By applying post-quantum cryptography selectively based on data lifetime, **Botho** achieves meaningful quantum resistance while maintaining practical transaction sizes. Recipient identities are protected by ML-KEM-768 against future quantum attacks; sender privacy uses efficient CLSAG ring signatures appropriate for ephemeral data.
2. **PoW + SCP consensus:** Combining proof-of-work block proposal with Stellar Consensus Protocol finalization achieves the best of both worlds: permissionless participation and fair distribution from PoW, with fast deterministic finality and Byzantine fault tolerance from SCP.
3. **Progressive fee mechanism:** Cluster-based fees create economic pressure against wealth concentration while preserving privacy. Unlike identity-based approaches, fees are determined by coin ancestry, resisting Sybil attacks without requiring user identification.
4. **Sustainable economics:** Perpetual tail emission ensures long-term security funding. Fee burning creates deflationary pressure proportional to network usage. Dynamic block timing aligns emission with network utility.

13.1.2 Design Philosophy

Botho embodies the principle of *botho*—that currency should serve community rather than concentrate power. This manifests in:

- Privacy as baseline, not premium feature
- Progressive economics discouraging hoarding
- Decentralized consensus with open participation
- Sustainable security through perpetual emission

13.2 Limitations and Future Work

13.2.1 Current Limitations

- **Classical ring signatures:** CLSAG provides sender privacy against classical adversaries only. A sufficiently powerful quantum computer could potentially identify signers retroactively.
- **Transaction size:** At approximately 4 KB per transaction, **Botho** transactions are larger than non-private alternatives, though significantly smaller than fully post-quantum constructions.
- **Synchronization time:** Full nodes require several hours to synchronize, limiting immediate usability for new users.

13.2.2 Future Research Directions

1. **Post-quantum ring signatures:** As lattice-based ring signature constructions mature, a future protocol upgrade could provide full post-quantum sender privacy while maintaining practical transaction sizes.
2. **Layer-2 scaling:** Payment channels and other off-chain constructions could enable micropayments and higher throughput while preserving privacy guarantees.
3. **Cross-chain interoperability:** Atomic swap protocols enabling trustless exchange with other cryptocurrencies would enhance utility without sacrificing privacy.
4. **Advanced privacy techniques:** Research into proof aggregation, recursive proofs, and other techniques may enable improved privacy-efficiency tradeoffs.

13.2.3 Interoperability Architecture

Cross-chain functionality requires careful design to preserve privacy. We outline preliminary approaches for future development.

Atomic Swaps. Hash Time-Locked Contracts (HTLCs) enable trustless cross-chain exchange:

1. Alice (BTH) and Bob (BTC) agree on exchange rate
2. Alice generates secret s , publishes $h = \mathcal{H}(s)$
3. Alice locks BTH in contract: “Redeemable by Bob with preimage of h before block T , else refundable to Alice after $T + \Delta$ ”
4. Bob locks BTC in mirrored contract on Bitcoin
5. Alice reveals s to claim BTC
6. Bob uses s to claim BTH

Privacy considerations:

- Hash h links both transactions—correlation attack possible
- Mitigation: Use adaptor signatures [1] instead of hash locks, eliminating on-chain correlation
- Timing correlation remains; recommend delays between claim operations

Protocol sketch (adaptor signature variant):

```
// Setup: Alice has BTH, wants BTC from Bob
Alice: Generate adaptor secret  $t$ ,  $T = tG$ 
      Create BTH tx with adaptor signature  $\text{sig}_A(T)$ 
Bob:   Verify adaptor, create BTC tx with  $\text{sig}_B(T)$ 
Alice: Complete  $\text{sig}_A$  by revealing  $t$ 
Bob:   Extract  $t$  from completed  $\text{sig}_A$ 
      Complete  $\text{sig}_B$  to claim BTC
// No hash appears on either chain
```

Bridge Design. Bridges enabling wrapped BTH on other chains face inherent tradeoffs:

Recommended architecture: ZK light client bridge

- Target chain verifies BTH header proofs (SCP finality)

Table 47: Bridge architecture comparison

Type	Trust	Privacy	Capital Efficiency
Custodial	High	Low	High
Federated	Medium	Medium	High
Optimistic	Low	Medium	Medium
ZK Light Client	Minimal	High	Medium

- Deposits create locked UTXOs with verifiable commitments
- ZK proof attests to valid lock without revealing details
- Withdrawals require ZK proof of burn on target chain

DEX Integration. Decentralized exchange on **Bothofaces** challenges:

- **Order book privacy:** Orders reveal trading intent
 - Solution: Encrypted order book with threshold decryption
 - Alternative: AMM-style pools (simpler but less efficient)
- **Settlement privacy:** Matched trades reveal amounts
 - Solution: Batch settlement with commitment aggregation
 - Users prove trade validity without revealing specific amounts
- **Front-running resistance:** SCP finality helps but mempool remains visible during PoW phase
 - Solution: Encrypted mempool with threshold reveal
 - Requires additional infrastructure investment

AMM Implementation Sketch:

Pool

```
reserve_commitment: Commitment, // C = r_A * H + r_B * G
k_commitment: Commitment,       // x * y = k hidden
```

Swap

```
input_commitment: Commitment,
output_commitment: Commitment,
// ZK proof: new reserves satisfy constant product
// ZK proof: amounts are non-negative
// ZK proof: fee correctly calculated
```

Research challenges:

1. Efficient ZK proofs for constant product verification
2. Privacy-preserving price oracle mechanisms
3. MEV resistance in decentralized setting
4. Liquidity provider privacy (shares reveal exposure)

These interoperability mechanisms remain active research areas. We expect community contributions to refine and implement practical solutions.

13.3 Broader Impact

13.3.1 Privacy as Human Right

Financial privacy protects fundamental human interests:

- Victims of abuse can escape financial control
- Dissidents can support causes without persecution
- Individuals can conduct legal business without surveillance
- Commercial entities can protect competitive information

Both provides these protections by default, ensuring privacy is not relegated to those with technical sophistication or willingness to pay premium fees.

13.3.2 Economic Implications

The progressive fee mechanism represents an experiment in cryptocurrency economics:

- Can market mechanisms encourage circulation without coercion?
- Does cluster-based taxation effectively resist Sybil attacks?
- Will tail emission prove superior to fixed supply for security?

Only real-world deployment will answer these questions definitively.

13.4 Closing Remarks

Cryptocurrencies present a rare opportunity to redesign fundamental economic infrastructure. Rather than replicating the surveillance and inequality of traditional finance, we can build systems that respect privacy and encourage fair participation.

Motho ke motho ka batho—a person is a person through other people. Both is designed with this principle at its core: technology that serves community, privacy that protects dignity, and economics that resist concentration.

We invite the community to examine, critique, and improve upon this design. The source code is open; the conversation continues.

Acknowledgments

We thank the cryptographic research community whose foundational work makes systems like Both possible, particularly:

- The CryptoNote developers for ring signature privacy
- The Monero Research Lab for CLSAG and related improvements
- The Bulletproofs authors for efficient range proofs
- The NIST PQC team for standardizing ML-KEM and ML-DSA
- The Stellar Development Foundation for SCP

We also thank the early reviewers and testers who provided invaluable feedback on protocol design and implementation.

References

- [1] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. One-time verifiably encrypted signatures a.k.a. adaptor signatures. Cryptology ePrint Archive, 2021. URL <https://eprint.iacr.org/2020/476>. Scriptless atomic swaps without on-chain hash correlation.
- [2] Aztec Labs. Aztec protocol: Zero-knowledge privacy for ethereum, 2023. URL <https://aztec.network/>. Private L2 zkRollup using PLONK proofs.
- [3] Beam Development. Beam: Confidential cryptocurrency and DeFi platform. Open-source project, 2019. URL <https://beam.mw>.
- [4] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, 2014.
- [5] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography – PKC 2006*, pages 207–228, 2006. doi: 10.1007/11745853_14.
- [6] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS – Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367, 2018. doi: 10.1109/EuroSP.2018.00032.
- [7] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, 2018.
- [8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. *OSDI*, 99:173–186, 1999.
- [9] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. Cryptology ePrint Archive, 2022. URL <https://eprint.iacr.org/2022/975>. Polynomial-time attack breaking SIKE/SIDH isogeny assumptions.
- [10] dalek cryptography. curve25519-dalek: A pure-Rust implementation of group operations on Ristretto and Curve25519. Open-source library, 2019. URL <https://github.com/dalek-cryptography/curve25519-dalek>.
- [11] Henry de Valence, Isis Lovecruft, Tony Arcieri, and Martin Kleppmann. The ristretto group, 2018. URL <https://ristretto.group>.
- [12] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. In *IACR Transactions on Cryptographic Hardware and Embedded Systems*, volume 2018, pages 238–268, 2018. doi: 10.13154/tches.v2018.i1.238-268.
- [13] Muhammed F. Esgin, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. Efficient lattice-based ring signatures. In *International Conference on Post-Quantum Cryptography*, pages 176–194, 2019. doi: 10.1007/978-3-030-25510-7_10. Module-LWE based linkable ring signatures.
- [14] Muhammed F. Esgin, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. Lattice-based ring signatures: A survey. Cryptology ePrint Archive, 2019. URL <https://eprint.iacr.org/2019/1167>. Survey of post-quantum ring signature constructions.

- [15] Facebook. RocksDB: A persistent key-value store. Open-source project, 2012. URL <https://rocksdb.org>.
- [16] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, volume 2, pages 1–35, 2018.
- [17] Giulia Fanti et al. Breaking mimblewimble’s privacy model. In *arXiv preprint*, 2019. arXiv:1907.10039.
- [18] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO’ 86*, pages 186–194, 1986. doi: 10.1007/3-540-47721-7_12. Interactive to non-interactive proof transformation.
- [19] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *Public Key Cryptography – PKC 2007*, pages 181–200, 2007. doi: 10.1007/978-3-540-71677-8_13. Introduces linkable/traceable ring signatures.
- [20] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptographic software. In *ACM Computing Surveys*, volume 51, pages 1–40, 2019. doi: 10.1145/3212479. Comprehensive survey of side-channel attacks including SGX vulnerabilities.
- [21] Silvio Gesell. *The Natural Economic Order*. Peter Owen Ltd, 1958. Historical demurrage currency theory (originally 1916).
- [22] Brandon Goodell, Sarang Noether, and RandomRun. Concise linkable ring signatures and forgery against adversarial keys. In *Cryptology ePrint Archive*, 2019. Report 2019/654.
- [23] Grin Developers. Grin: A lightweight implementation of MimbleWimble. Open-source project, 2019. URL <https://grin.mw>.
- [24] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.
- [25] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. 2016. URL <https://zips.z.cash/protocol/protocol.pdf>.
- [26] Daira Hopwood, Sean Bowe, et al. ZIP 224: Orchard shielded protocol. Zcash Improvement Proposal, 2021. URL <https://zips.z.cash/zip-0224>. Halo 2 based shielded transactions.
- [27] Aram Jivanyan and Aaron Feickert. Lelantus spark: Secure and flexible private transactions. Cryptology ePrint Archive, 2021. URL <https://eprint.iacr.org/2021/1173>. Privacy protocol combining Spark addresses with one-out-of-many proofs.
- [28] Hugo Krawczyk and Pasi Eronen. HMAC-based extract-and-expand key derivation function (HKDF). Technical Report RFC 5869, IETF, 2010. URL <https://www.rfc-editor.org/rfc/rfc5869>.
- [29] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. doi: 10.1145/357172.357176.
- [30] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65, 2002.

- [31] David Mazières. The stellar consensus protocol: A federated model for internet-level consensus. In *Stellar Development Foundation*, 2015.
- [32] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Internet Measurement Conference*, pages 127–140, 2013. doi: 10.1145/2504730.2504747. Bitcoin transaction graph analysis.
- [33] Monero Project. Monero: A private digital currency, 2014. URL <https://getmonero.org>.
- [34] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. URL <https://bitcoin.org/bitcoin.pdf>.
- [35] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. Technical Report FIPS 202, NIST, 2015. URL <https://csrc.nist.gov/pubs/fips/202/final>.
- [36] National Institute of Standards and Technology. Fips 203: Module-lattice-based key-encapsulation mechanism standard, 2024. URL <https://csrc.nist.gov/pubs/fips/203/final>.
- [37] National Institute of Standards and Technology. Fips 204: Module-lattice-based digital signature standard, 2024. URL <https://csrc.nist.gov/pubs/fips/204/final>.
- [38] National Institute of Standards and Technology. Post-quantum cryptography standardization, 2024. URL <https://csrc.nist.gov/projects/post-quantum-cryptography>.
- [39] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, 2015. URL <https://eprint.iacr.org/2015/1098>. MLSAG ring signatures for multiple inputs.
- [40] Shen Noether, Adam Mackenzie, and Monero Research Lab. Ring confidential transactions. Monero Research Lab Report MRL-0005, 2016. URL <https://lab.getmonero.org/pubs/MRL-0005.pdf>. Introduces RingCT combining ring signatures with confidential transactions.
- [41] Jack O’Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O’Hearn. BLAKE3: One function, fast everywhere. Specification, 2020. URL <https://github.com/BLAKE3-team/BLAKE3-specs>.
- [42] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. Bip-0039: Mnemonic code for generating deterministic keys, 2013. URL <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>.
- [43] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO’91*, pages 129–140, 1992.
- [44] Trevor Perrin. The noise protocol framework, 2018. URL <https://noiseprotocol.org>.
- [45] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. Technical report, TU Dresden, 2010. URL http://dud.inf.tu-dresden.de/Anon_Terminology.shtml. Standard terminology for privacy properties, v0.34.
- [46] Andrew Poelstra and Tom Elvis Jedusor. Mimblewimble, 2016. URL <https://scalingbitcoin.org/papers/mimblewimble.pdf>.

- [47] Protocol Labs. libp2p: A modular network stack. Open-source project, 2019. URL <https://libp2p.io>. Peer-to-peer networking library.
- [48] QRL Foundation. The quantum resistant ledger, 2018. URL <https://theqrl.org>.
- [49] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. *Advances in Cryptology — ASIACRYPT 2001*, pages 552–565, 2001. doi: 10.1007/3-540-45682-1_32. Original ring signature construction.
- [50] Pavol Rusnak and Marek Palatinus. Slip-0010: Universal private key derivation from master private key, 2016. URL <https://github.com/satoshi-labs/slips/blob/master/slip-0010.md>.
- [51] Secret Network. Secret network: Decentralized privacy-preserving computation, 2020. URL <https://scrt.network/>. TEE-based privacy using Intel SGX enclaves.
- [52] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
- [53] Wilson Abel Alberto Torres, Ron Steinfeld, Amin Sakzad, Joseph K Liu, Veronika Kuchta, Nandita Bhattacharjee, Man Ho Au, and Jacob Cheng. Post-quantum lattice-based ring signatures, 2018. Cryptology ePrint Archive, Report 2018/762.
- [54] United Nations General Assembly. Universal declaration of human rights, 1948. Article 12.
- [55] Nicolas van Saberhagen. Cryptonote v2.0, 2013. URL <https://cryptonote.org/whitepaper.pdf>.

A Notation Reference

This appendix provides a comprehensive reference for notation used throughout the paper.

A.1 Mathematical Notation

Symbol	Description
<i>Groups and Fields</i>	
\mathbb{G}	Elliptic curve group (Ristretto255)
G	Generator of \mathbb{G}
H	Secondary generator for Pedersen commitments
q	Order of \mathbb{G} (prime)
\mathbb{Z}_q	Integers modulo q
R_q	Polynomial ring for lattice operations
<i>Keys</i>	
a, b	View and spend private keys
A, B	View and spend public keys
(C_i, D_i)	Subaddress public key pair for index i
P	One-time public key
x	One-time private key
I	Key image
<i>Hash Functions</i>	
$H_s(\cdot)$	Hash function to scalar
$H_p(\cdot)$	Hash function to curve point
$\mathcal{H}(\cdot)$	General cryptographic hash
<i>Ring Signatures</i>	
$\{P_i\}_{i=0}^{n-1}$	Ring of public keys
π	Secret index of real key in ring
σ	Signature
n	Ring size (default: 20)
<i>Commitments</i>	
C	Pedersen commitment
C_v	Commitment to value v
r	Blinding factor
<i>Operations</i>	
$\xleftarrow{\$}$	Uniform random sampling
\parallel	Concatenation
\oplus	Bitwise XOR
$\text{mod } q$	Modular reduction
<i>Complexity</i>	
$\text{negl}(\lambda)$	Negligible function in λ
$\text{poly}(\lambda)$	Polynomial function in λ
λ	Security parameter

A.2 Protocol Identifiers

Identifier	Description
CLSAG	Concise Linkable Spontaneous Anonymous Group signatures
ML-KEM	Module-Lattice-based Key Encapsulation Mechanism
ML-DSA	Module-Lattice-based Digital Signature Algorithm
SCP	Stellar Consensus Protocol
PoW	Proof of Work
BTH	Native currency unit (Botho)

A.3 Network Constants

Parameter	Value	Description
Ring size	20	Decoys per input
Min block time	5s	At maximum utilization
Max block time	40s	At zero utilization
Block size limit	2 MB	Maximum block size
Transaction limit	100 KB	Maximum transaction size
Default port	9732	Main network port

A.4 Cryptographic Parameters

Algorithm	Parameter	Value
ML-KEM-768	Public key	1,184 bytes
	Ciphertext	1,088 bytes
ML-DSA-65	Public key	1,952 bytes
	Signature	3,309 bytes
CLSAG	Signature (ring=20)	704 bytes
	Key image	32 bytes
Bulletproofs	2-output proof	736 bytes
	Aggregation	Logarithmic

A.5 Transaction Sizes

Component	Size	Notes
Transaction header	32 bytes	Version, prefix data
Input (per)	680 bytes	Ring refs + key image + tag
Output (per)	1,152 bytes	Commitment + key + ciphertext + tag
CLSAG signature	704 bytes	Per input
Bulletproof	736 bytes	2 outputs, aggregated
1-in-2-out total	~4,552 bytes	Typical transaction

A.6 Monetary Parameters

Parameter	Value	Description
Initial block reward	50 BTH	Genesis reward
Halving period	1,051,200 blocks	≈2 years
Tail emission	0.3 BTH	Minimum reward
Decimal places	9	Smallest: 1 nano-BTH
Base fee rate	1 nano-BTH/byte	Minimum fee
Max cluster factor	6.0×	Top 1% multiplier

B Parameter Justification

This appendix provides detailed rationale for key protocol parameters. Parameters are grouped by subsystem and include analysis of alternatives considered.

B.1 Cryptographic Parameters

B.1.1 Ring Size: 20

The ring size determines the anonymity set for sender privacy.

Table 48: Ring size trade-off analysis

Ring Size	Signature Size	Nominal Anonymity	Effective Anonymity*
11 (Monero)	~400 B	1/11	1/3–1/5
16	~550 B	1/16	1/4–1/6
20 (Botho)	~700 B	1/20	1/5–1/8
32	~1,100 B	1/32	1/7–1/10
64	~2,200 B	1/64	1/10–1/15

**Effective anonymity accounts for chain analysis techniques (timing, amount correlation, decoy age distribution biases).*

Why 20?

- **Diminishing returns:** Doubling ring size from 20 to 40 adds only ~0.5 bits of anonymity while doubling signature size
- **Practical anonymity:** Ring size 20 provides meaningful protection against passive observers
- **Size budget:** At 700 bytes per input, multi-input transactions remain practical (<5 KB for 2-in-2-out)
- **Decoy availability:** Sufficient recent outputs exist in moderate-activity network

Why not Monero’s 11?

- Research shows Monero’s effective anonymity is lower than nominal
- Chain analysis companies report success rates suggesting <5 effective ring members
- Botho’s 20 provides meaningful improvement with acceptable overhead

B.1.2 ML-KEM Security Level: 768

ML-KEM-768 provides NIST Security Level 3 (equivalent to AES-192).

Why not ML-KEM-512 (Level 1)?

- Recipient privacy is permanent—data persists indefinitely on-chain
- Conservative security margin appropriate for long-term protection
- Size difference minimal (800 vs 1088 bytes ciphertext)

Why not ML-KEM-1024 (Level 5)?

- Level 5 adds 50% size overhead (1568 byte ciphertext)
- No known attacks approach Level 3 security
- Can upgrade if Level 3 becomes threatened

B.1.3 ML-DSA Security Level: 65

ML-DSA-65 (formerly Dilithium3) provides NIST Security Level 3.

Rationale: Minting signatures must remain verifiable long-term. Matching ML-KEM security level provides consistent protection for all permanent on-chain data.

B.2 Consensus Parameters

B.2.1 Block Time: 5–40 Seconds (Dynamic)

Block time varies based on network utilization.

Table 49: Block time parameter analysis

Bound	Value	Constraint
Minimum	5 seconds	SCP convergence time ($\sim 3\text{--}4\text{s}$)
Target (low util)	40 seconds	User experience, emission rate
Target (high util)	5 seconds	Throughput during demand

Why not fixed 10 seconds (like many chains)?

- Fixed time wastes capacity during high demand
- Fixed time over-emits during low utilization periods
- Dynamic adjustment aligns emission with network utility

Lower bound justification (5s):

- SCP nomination: ~ 1 second
- Ballot prepare: ~ 1 second
- Ballot commit: ~ 1 second
- Network propagation: ~ 1 second
- Safety margin: ~ 1 second

B.2.2 Quorum Thresholds: 3-of-4 Infrastructure, 2-of-3 Community

Infrastructure tier (3-of-4):

- Tolerates 1 Byzantine or offline node
- Requires 75% agreement (standard BFT threshold)
- Balances availability with safety

Community tier (2-of-3):

- Tolerates 1 Byzantine or offline node
- Lower threshold acknowledges higher variance in community uptime
- Inner set structure prevents single community node from blocking

Why tiered structure?

- Pure flat structure requires trusting all validators equally
- Infrastructure tier provides stability baseline
- Community tier ensures decentralization input
- Combined structure requires both tiers for consensus

B.2.3 Difficulty Adjustment Window: 144 Blocks

Why 144?

- At 10-second average, represents ~ 24 minutes
- Long enough to smooth variance in block times
- Short enough to respond to hashrate changes
- Same philosophy as Bitcoin's 2016-block window scaled to block time

Adjustment bounds ($0.5\times$ to $2\times$):

- Prevents oscillation from rapid hashrate changes
- Allows recovery from 50% hashrate drop in one window
- Historical analysis shows changes $>2\times$ per window are rare

B.3 Economic Parameters

B.3.1 Initial Block Reward: 50 BTH

Rationale:

- Round number for simplicity
- Matches Bitcoin's initial reward (symbolic continuity)
- At 10-second blocks: 432,000 BTH/day initially
- Creates sufficient early liquidity for network bootstrap

B.3.2 Halving Period: 1,051,200 Blocks (~ 2 Years)

Why 2-year halvings (vs. Bitcoin's 4-year)?

- Faster initial distribution to early participants
- Reaches tail emission sooner, stabilizing economics
- At 10-second blocks: 1,051,200 blocks ≈ 121.7 days $\times 6 = 2.0$ years

Calculation:

$$\text{blocks_per_day} = \frac{86400\text{s}}{10\text{s/block}} = 8640$$

$$\text{blocks_per_2_years} = 8640 \times 365 \times 2 \approx 6,307,200$$

Actual value (1,051,200) corresponds to 121.7 days per halving epoch, with 6 epochs totaling ~ 2 years.

B.3.3 Tail Emission: 0.3 BTH per Block

Security budget analysis:

- At 10-second blocks: 2,592,000 BTH/year from tail emission
- Target: $\sim 2\%$ annual inflation at maturity
- Projected mature supply: $\sim 130\text{M}$ BTH
- $2,592,000 / 130,000,000 = 1.99\% \checkmark$

Why not 0?

- Zero tail emission requires fees to fund all security
- Fee-only security creates volatility and potential death spirals
- Perpetual emission ensures predictable security budget

Why not higher (e.g., 1 BTH)?

- Higher emission increases long-term inflation
- 0.3 BTH provides adequate security while limiting dilution
- Can be adjusted via governance if security budget insufficient

B.3.4 Fee Split: 80% Lottery / 20% Burn

Why redistribute 80%?

- Pure burn benefits all holders equally (regressive)
- Lottery redistribution favors small holders statistically
- 80% ensures meaningful redistribution effect

Why burn 20%?

- Some deflation counters tail emission inflation
- Creates tangible "cost" to network usage
- Prevents infinite supply growth from fees alone

Alternative considered: 90/10

- More redistribution but less deflationary pressure
- 80/20 provides better balance based on economic modeling

B.3.5 Progressive Fee Multiplier: 1× to 6×

Sigmoid steepness parameter:

$$\text{factor} = 1 + 5 \cdot \sigma \left(\frac{W}{\text{steepness}} \right)$$

Why maximum 6×

- High enough to create meaningful progressive effect
- Low enough to not price out large legitimate transactions
- Comparable to progressive tax brackets in traditional systems

Why sigmoid shape?

- Smooth transition avoids cliff effects
- Asymptotic bound prevents infinite fees
- Most users (bottom 50%) experience near-base fees

B.3.6 Tag Decay: 0.95 Rate, 720-Block Minimum Age

Decay rate 0.95:

- Half-life: $\ln(0.5)/\ln(0.95) \approx 13.5$ decay events
- At maximum 1 decay per 720 blocks (~ 2 hours): 27 hours half-life
- Encourages circulation without rapid tag elimination

Minimum age 720 blocks:

- At 10-second blocks: ~ 2 hours
- Prevents rapid wash trading from bypassing progressive fees
- Patient wash trading (1 week) achieves only 97% decay
- Creates meaningful time cost for fee evasion

B.4 Network Parameters

B.4.1 Maximum Transaction Size: 100 KB

Rationale:

- Allows 16-in-16-out transactions (~ 80 KB)
- Prevents single transactions from dominating blocks
- Provides headroom for future feature expansion

B.4.2 Maximum Block Size: 2 MB

Why 2 MB?

- At 4 KB average transaction: ~ 500 transactions per block
- At 5-second blocks: ~ 100 TPS theoretical maximum
- Manageable for full node storage and bandwidth
- Can increase via soft fork if network capacity grows

B.4.3 Connection Limits: 8 Outbound, 117 Inbound

Outbound (8):

- Sufficient for network connectivity
- Limits resource consumption for light nodes
- Eclipse attack resistance (diverse peer selection)

Inbound (117):

- Allows serving many peers
- Total 125 matches common socket limits
- Per-IP limit (2) prevents single-source flooding

B.5 Summary Table

Table 50: Key parameter summary

Parameter	Value	Category	Governance
Ring size	20	Crypto	Soft fork
ML-KEM level	768	Crypto	Hard fork
Block time	5–40s	Consensus	Soft fork
Quorum threshold	3/4, 2/3	Consensus	Hard fork
Initial reward	50 BTH	Economic	Genesis
Tail emission	0.3 BTH	Economic	Hard fork
Fee split	80/20	Economic	Soft fork
Max fee multiplier	6×	Economic	Soft fork
Max block size	2 MB	Network	Soft fork

C Regulatory Considerations

Disclaimer: This appendix provides technical information about **Botho**’s capabilities relevant to regulatory compliance. It does not constitute legal advice. Users and service providers should consult qualified legal counsel regarding their specific obligations.

C.1 Privacy and Compliance Tension

Privacy-preserving cryptocurrencies exist in tension with regulatory frameworks designed for transparent financial systems. **Botho** is designed to provide strong privacy by default while enabling selective transparency where legally required.

C.1.1 Design Philosophy

- **Privacy as default:** All users receive equal privacy protection regardless of transaction size or frequency
- **Voluntary disclosure:** Users can prove transaction details to third parties without compromising other transactions
- **No backdoors:** The protocol contains no mechanism for mass surveillance or privileged access

- **Technical neutrality:** The protocol neither facilitates nor prevents compliance—that is a user/application layer concern

C.2 View Key Disclosure

Both supports selective transparency through view key mechanisms.

C.2.1 View Key Capabilities

A view key allows a third party to:

- Identify incoming transactions to an address
- Decrypt transaction amounts for identified transactions
- Verify transaction inclusion in the blockchain
- Cannot identify outgoing transactions (requires spend key)
- Cannot spend funds (view-only access)

C.2.2 Audit Scenarios

Table 51: View key disclosure scenarios

Scenario	Key Disclosed	Information Revealed
Tax audit	View key	Incoming transactions, amounts
Court order	View key	Incoming transactions, amounts
AML compliance	View key	Deposit monitoring
Full disclosure	View + Spend	Complete transaction history

C.2.3 View Key Limitations

View keys cannot reveal:

- Which ring members were actually spent (sender privacy preserved)
- Recipients of outgoing transactions
- Total balance without scanning full blockchain

Implication: Complete audit trails require cooperation from transaction counterparties or additional record-keeping.

C.3 Selective Transparency Options

C.3.1 Payment Proofs

Users can generate cryptographic proofs of specific payments:

```
PaymentProof
tx_hash: Hash,           // Transaction identifier
recipient_address: Address, // Claimed recipient
amount: u64,             // Claimed amount
proof: ZKProof,          // Cryptographic proof
```

```
// Verifier can confirm:
// 1. Transaction exists in blockchain
// 2. Output belongs to claimed recipient
// 3. Amount matches claim
```

Use case: Proving payment to tax authority or in dispute resolution.

C.3.2 Income Proofs

Users can prove receipt of funds without revealing sender:

```
IncomeProof
  outputs: Vec<OutputRef>,    // List of received outputs
  amounts: Vec<u64>,          // Corresponding amounts
  proof: ZKProof,              // Ownership proof
```

Use case: Demonstrating income for loan applications or tax filing.

C.3.3 Balance Proofs

Users can prove minimum balance without revealing exact amount:

```
BalanceProof
  minimum: u64,                // Claimed minimum balance
  proof: ZKProof,              // Proof that balance >= minimum
```

Use case: Proof of reserves, creditworthiness verification.

C.4 Exchange Integration

C.4.1 Know Your Customer (KYC)

Exchanges can implement KYC at the application layer:

- Identity verification before account creation
- Address association with verified identities
- Transaction monitoring for associated addresses
- View key access for audit purposes

Note: KYC is an exchange policy, not a protocol feature.

C.4.2 Travel Rule Compliance

The FATF Travel Rule requires transmission of originator/beneficiary information for transfers above thresholds.

Technical options:

1. **Off-chain:** Exchange-to-exchange communication of customer data (protocol-agnostic)
2. **Encrypted memo:** Optional encrypted field in transaction for compliance data (requires recipient cooperation)
3. **VASP protocols:** Integration with Travel Rule compliance networks (Sygna, Notabene, etc.)

Botho position: The protocol provides optional encrypted memo fields; compliance implementation is left to service providers.

C.4.3 Suspicious Activity Reporting

Exchanges can monitor for suspicious patterns:

- Unusual transaction volumes
- Structuring (splitting to avoid thresholds)
- Rapid deposit/withdrawal cycles
- Connections to flagged addresses (limited effectiveness)

Limitation: Ring signatures prevent definitive source tracing; exchanges can only monitor their own customer activity.

C.5 Comparison to Other Privacy Approaches

Table 52: Privacy coin compliance capabilities

Feature	Botho	Monero	Zcash	Bitcoin
View key audit	Yes	Yes	Yes*	N/A
Payment proofs	Yes	Yes	Yes	Yes
Selective disclosure	Yes	Limited	Yes	N/A
Transparent mode	No	No	Yes	Default
Chain analysis	Limited	Limited	Possible*	Yes

**Zcash shielded transactions; transparent transactions are fully auditable.*

C.6 Jurisdictional Considerations

Privacy coin regulations vary significantly by jurisdiction:

- **Permitted:** Most jurisdictions permit privacy coins for personal use
- **Exchange restrictions:** Some jurisdictions prohibit exchange listing (Japan, South Korea for certain coins)
- **Reporting requirements:** Many jurisdictions require reporting of cryptocurrency holdings/gains
- **AML obligations:** Service providers have varying obligations depending on jurisdiction

Recommendation: Users should understand their local regulatory environment before using any cryptocurrency.

C.7 Technical Measures for Service Providers

Service providers can implement compliance measures:

C.7.1 Deposit Monitoring

```
ExchangeDepositMonitor
// Use view key to scan for deposits
scan_for_deposits(view_key, start_height) -> Vec<Deposit>

// Associate with customer account
associate_deposit(deposit, customer_id)

// Generate audit trail
export_deposits(customer_id, date_range) -> AuditReport
```

C.7.2 Withdrawal Controls

```
WithdrawalRequest
customer_id: CustomerId,
amount: u64,
destination: Address,

// Compliance checks:
// 1. Customer KYC status verified
// 2. Amount within limits
// 3. Destination not on sanctions list (limited effectiveness)
// 4. Travel rule data prepared (if applicable)
```

C.8 Limitations and Honest Assessment

C.8.1 What Botho Cannot Provide

- **Sender identification:** Ring signatures prevent identifying which input was spent
- **Address clustering:** Stealth addresses prevent linking outputs to addresses
- **Amount revelation:** Without view key, amounts are hidden
- **Global surveillance:** No mechanism for mass transaction monitoring

C.8.2 Regulatory Risk

Users should be aware:

- Privacy coins may face increasing regulatory scrutiny
- Exchange availability may be limited in some jurisdictions
- Tax obligations apply regardless of transaction privacy
- Using privacy features does not exempt from legal obligations

C.9 Conclusion

Botho provides strong privacy by default while enabling users to voluntarily disclose transaction information where required. The protocol is designed to be technically neutral—compliance is a user and service provider responsibility, not enforced at the protocol level.

This approach reflects the belief that privacy is a fundamental right, while acknowledging that legitimate compliance requirements exist. Users and service providers must navigate this balance according to their specific legal obligations.

D Formal Specifications

This appendix provides formal state machine specifications for the **Botho** protocol, suitable for verification and implementation reference.

D.1 State Definitions

D.1.1 Blockchain State

The blockchain state Σ is a tuple:

$$\Sigma = (\mathcal{U}, \mathcal{I}, \mathcal{C}, h, t) \quad (66)$$

where:

- \mathcal{U} : UTXO set (unspent transaction outputs)
- \mathcal{I} : Key image set (spent output identifiers)
- \mathcal{C} : Cluster tag database
- h : Current block height
- t : Latest block timestamp

UTXO Structure:

```
UTXO =
  outpoint: (block_height, tx_index, output_index),
  commitment: CompressedPoint, // Pedersen commitment
  one_time_key: CompressedPoint,
  kem_ciphertext: [u8; 1088],
  cluster_tag: ClusterTag,
  created_height: u64,
```

```
ClusterTag =
  weights: Map<ClusterId, Rational>, // Sum to 1
```

Key Image Set:

$$\mathcal{I} = \{I \in \mathbb{G} : I \text{ has been spent}\} \quad (67)$$

Key images are deterministic: $I = x \cdot H_p(P)$ where x is the private key and P is the one-time public key.

D.1.2 Consensus State

The SCP consensus state Ξ for slot s is:

$$\Xi_s = (\phi_s, \beta_s, M_s, V_s) \quad (68)$$

where:

- $\phi_s \in \{\text{NOMINATING}, \text{PREPARING}, \text{COMMITTING}, \text{EXTERNALIZED}\}$
- β_s : Current ballot (n, v) where n is counter, v is value
- M_s : Set of received messages
- V_s : Set of nominated values

D.2 State Transition Functions

D.2.1 Block Application

The state transition function $\delta : \Sigma \times B \rightarrow \Sigma$ applies block B to state Σ :

```
function apply_block(state: State, block: Block) -> Result<State>
  // 1. Verify block header
  require(block.prev_hash == hash(state.latest_block))
  require(block.height == state.height + 1)
  require(verify_pow(block.header))

  // 2. Verify minting transaction
  let reward = compute_block_reward(block.height)
  require(sum(block.minting_tx.outputs) <= reward)
  require(verify_mlds(block.minting_tx.signature))

  // 3. Apply each transaction
  let new_state = state
  for tx in block.transactions
    new_state = apply_transaction(new_state, tx)?

  // 4. Add minting outputs to UTXO set
  for (i, output) in block.minting_tx.outputs.enumerate()
    let utxo = create_utxo(block, 0, i, output)
    new_state.utxo_set.insert(utxo)

  // 5. Process fee redistribution
  new_state = process_fees(new_state, block)?

  // 6. Update metadata
  new_state.height = block.height
  new_state.timestamp = block.timestamp
  new_state.latest_block = hash(block)

  Ok(new_state)
```

D.2.2 Transaction Application

The transaction application function:

```
function apply_transaction(
  state: State,
  tx: Transaction
) -> Result<State>
  // 1. Validate transaction structure
  require(1 <= tx.inputs.len() <= 16)
  require(1 <= tx.outputs.len() <= 16)

  // 2. Check key image uniqueness
  for input in tx.inputs
    require(!state.key_images.contains(input.key_image))

  // 3. Verify ring signatures
  for (input, signature) in zip(tx.inputs, tx.signatures)
```

```

    let ring = get_ring_members(state, input.ring)
    require(verify_clsag(signature, ring, tx.prefix_hash))

// 4. Verify value conservation
let input_sum = sum_commitments(get_input_commitments(tx))
let output_sum = sum_commitments(tx.outputs.map(|o| o.commitment))
let fee_point = tx.fee * H
require(input_sum == output_sum + fee_point)

// 5. Verify range proofs
require(verify_bulletproof(tx.bulletproof, tx.outputs))

// 6. Check minimum fee
let cluster_factor = compute_cluster_factor(tx)
let min_fee = base_fee * tx.size * cluster_factor
require(tx.fee >= min_fee)

// 7. Apply state changes
let mut new_state = state

// Remove spent outputs (we don't know which, but add key images)
for input in tx.inputs
    new_state.key_images.insert(input.key_image)

// Add new outputs
for (i, output) in tx.outputs.enumerate()
    let utxo = create_utxo_from_output(tx, i, output)
    new_state.utxo_set.insert(utxo)

Ok(new_state)

```

D.2.3 Fee Processing

Fee redistribution follows the 80/20 lottery/burn model:

```

function process_fees(state: State, block: Block) -> State
    let total_fees = sum(block.transactions.map(|tx| tx.fee))
    let lottery_pool = 0.8 * total_fees
    let burn_amount = 0.2 * total_fees // Destroyed

// Select lottery winners using deterministic randomness
let seed = hash(block.prev_hash || block.merkle_root)
let num_winners = 5
let winners = select_random_utxos(state.utxo_set, seed, num_winners)

// Distribute lottery pool
let per_winner = lottery_pool / num_winners
for winner_outpoint in winners
    // Create lottery payout transaction
    let payout = create_lottery_payout(winner_outpoint, per_winner)
    state = apply_lottery_payout(state, payout)

state

```

```

function select_random_utxos(
  utxo_set: Set<UTXO>,
  seed: Hash,
  count: usize
) -> Vec<Outpoint>
  let utxos: Vec<UTXO> = utxo_set.to_sorted_vec()
  let mut selected = Vec::new()

  for i in 0..count
    let index = hash_to_index(seed, i, utxos.len())
    selected.push(utxos[index].outpoint)

  selected

```

D.3 Validity Predicates

D.3.1 Transaction Validity

A transaction T is valid with respect to state Σ iff:

$$\text{Valid}(T, \Sigma) \Leftrightarrow \bigwedge_{i=1}^8 P_i(T, \Sigma) \quad (69)$$

where predicates P_i are:

1. P_1 : **Structure**

$$1 \leq |T.\text{inputs}| \leq 16 \wedge 1 \leq |T.\text{outputs}| \leq 16$$

2. P_2 : **Key Image Freshness**

$$\forall I \in T.\text{key_images} : I \notin \Sigma.\mathcal{I}$$

3. P_3 : **Key Image Uniqueness (intra-tx)**

$$|T.\text{key_images}| = |\{I : I \in T.\text{key_images}\}|$$

4. P_4 : **Ring Validity**

$$\forall r \in T.\text{rings} : |r| = 20 \wedge \forall o \in r : o \in \Sigma.\mathcal{U}$$

5. P_5 : **Signature Validity**

$$\forall (r, \sigma, I) \in \text{zip}(T.\text{rings}, T.\text{sigs}, T.\text{key_images}) : \text{CLSAG.Verify}(r, \sigma, I, T.\text{hash})$$

6. P_6 : **Value Conservation**

$$\sum_i C_{\text{in}}^{(i)} = \sum_j C_{\text{out}}^{(j)} + T.\text{fee} \cdot H$$

7. P_7 : **Range Proofs**

$$\text{Bulletproof.Verify}(T.\text{proof}, \{C_{\text{out}}^{(j)}\})$$

8. P_8 : **Fee Sufficiency**

$$T.\text{fee} \geq f_{\text{base}} \cdot |T|_{\text{bytes}} \cdot \phi(T)$$

where $\phi(T)$ is the cluster factor

D.3.2 Block Validity

A block B is valid with respect to state Σ iff:

$$\text{Valid}(B, \Sigma) \Leftrightarrow \bigwedge_{i=1}^6 Q_i(B, \Sigma) \quad (70)$$

where predicates Q_i are:

1. Q_1 : **Chain Linkage**

$$B.\text{prev_hash} = \mathcal{H}(\Sigma.\text{latest_block})$$

2. Q_2 : **Height Increment**

$$B.\text{height} = \Sigma.h + 1$$

3. Q_3 : **Proof of Work**

$$\mathcal{H}(B.\text{header}) < B.\text{target}$$

4. Q_4 : **Timestamp Validity**

$$\Sigma.t < B.t \leq \text{now} + \epsilon$$

5. Q_5 : **Minting Validity**

$$\sum B.\text{mint.outputs} \leq R(B.\text{height}) \wedge \text{ML-DSA.Verify}(B.\text{mint.sig})$$

6. Q_6 : **Transaction Validity**

$$\forall T \in B.\text{transactions} : \text{Valid}(T, \Sigma')$$

where Σ' is state after applying preceding transactions

D.4 SCP State Machine

D.4.1 Message Types

enum SCPMessage

Nominate

slot: u64,
voted: Set<Value>,
accepted: Set<Value>,
,

Prepare

slot: u64,
ballot: Ballot,
prepared: Option<Ballot>,
prepared_prime: Option<Ballot>,
quorum_set_hash: Hash,
,

Commit

slot: u64,
ballot: Ballot,
prepared_counter: u32,
quorum_set_hash: Hash,
,

Externalize

slot: u64,
commit: Ballot,

```

        quorum_set_hash: Hash,
    ,

struct Ballot
    counter: u32,
    value: Value, // Block hash

```

D.4.2 State Transitions

```

function process_scp_message(
    state: SCPState,
    msg: SCPMessage,
    sender: NodeId
) -> SCPState
    match msg
        Nominate slot, voted, accepted =>
            // Update nomination state
            for v in voted
                if sender in state.quorum_slice
                    state.slots[slot].voted.insert(v)

            for v in accepted
                if has_blocking_set(state, sender, v)
                    state.slots[slot].accepted.insert(v)

            // Check if nomination complete
            if has_quorum(state, state.slots[slot].accepted)
                state.slots[slot].phase = PREPARING
                let value = pick_composite_value(state.slots[slot].accepted)
                state.slots[slot].ballot = Ballot counter: 1, value

        ,

        Prepare slot, ballot, .. =>
            // Process prepare vote
            if state.slots[slot].phase >= PREPARING
                update_prepare_state(state, slot, ballot, sender)

            // Check for commit transition
            if can_commit(state, slot)
                state.slots[slot].phase = COMMITTING

        ,

        Commit slot, ballot, .. =>
            // Process commit vote
            if state.slots[slot].phase >= COMMITTING
                update_commit_state(state, slot, ballot, sender)

            // Check for externalize
            if has_quorum_commit(state, slot)
                state.slots[slot].phase = EXTERNALIZED

```

```

        emit_externalize(state, slot)

    ,

    Externalize slot, commit, .. =>
        // Accept externalize from quorum
        if has_quorum_externalize(state, slot, commit)
            state.slots[slot].phase = EXTERNALIZED
            state.slots[slot].externalized_value = Some(commit.value)

    ,

state

```

D.5 Invariants

The following invariants are maintained:

1. Key Image Uniqueness

$$\forall I \in \mathcal{I} : |\{T : I \in T.\text{key_images}\}| = 1$$

2. UTXO Conservation

$$\sum_{U \in \mathcal{U}} \text{value}(U) + \text{burned} = \text{total_minted}$$

3. Cluster Tag Normalization

$$\forall U \in \mathcal{U} : \sum_c U.\text{cluster_tag}.\text{weights}[c] = 1$$

4. SCP Safety

$$\forall s, v_1, v_2 : \text{externalized}(s, v_1) \wedge \text{externalized}(s, v_2) \Rightarrow v_1 = v_2$$

5. Chain Integrity

$$\forall h > 0 : B_h.\text{prev_hash} = \mathcal{H}(B_{h-1})$$

These invariants can be checked by a model checker or used as assertions in implementation testing.

E Security Audit Guide

This appendix provides guidance for security auditors reviewing the **Botho** protocol and implementation.

E.1 Security Claims

The following security properties are claimed by **Botho**. Auditors should verify each claim against the implementation.

Table 53: Cryptographic security claims

Property	Primitive	Assumption
Recipient unlinkability	ML-KEM-768	IND-CCA2 security of ML-KEM
Sender anonymity	CLSAG	DLP hardness in Ristretto255
Amount confidentiality	Pedersen + BP	DLP hardness, random oracle model
Double-spend prevention	Key images	Collision resistance of H_p
Minting authenticity	ML-DSA-65	EUFCMA security of ML-DSA

E.1.1 Cryptographic Security

E.1.2 Consensus Security

1. **Fork freedom:** No two honest nodes externalize different blocks at the same height (Theorem 6.5)
2. **Liveness:** The network eventually makes progress under partial synchrony
3. **Byzantine tolerance:** Safety holds with up to f Byzantine nodes per quorum slice

E.1.3 Economic Security

1. **Sybil resistance:** Splitting wealth does not reduce total fees (Theorem 5.1)
2. **Lottery fairness:** UTXO selection is uniformly random and unpredictable
3. **Inflation bounds:** Supply growth follows the emission schedule exactly

E.2 Threat Model

Auditors should consider the following adversary capabilities:

Table 54: Threat model assumptions

Adversary Type	Capabilities
Network adversary	Can observe, delay, or reorder messages; cannot forge signatures or break encryption
Computational adversary	Polynomial-time bounded; cannot solve DLP or break ML-KEM
Byzantine nodes	Up to f nodes per quorum slice may behave arbitrarily
Quantum adversary	Future: can break DLP but not lattice problems

E.3 Audit Scope

E.3.1 Critical Components

The following components require thorough review:

1. **Key derivation** (`crypto/keys.rs`)
 - BIP39 mnemonic handling
 - SLIP-10 derivation paths
 - ML-KEM key generation from seed

2. **Stealth addresses** (`crypto/stealth.rs`)
 - ML-KEM encapsulation/decapsulation
 - One-time key derivation
 - Output scanning correctness
3. **Ring signatures** (`crypto/clsag.rs`)
 - Signature generation and verification
 - Key image computation
 - Ring member selection
4. **Confidential transactions** (`crypto/bulletproofs.rs`)
 - Pedersen commitment arithmetic
 - Range proof generation and verification
 - Value conservation checks
5. **Consensus** (`consensus/scp.rs`)
 - Quorum slice validation
 - Ballot protocol state machine
 - Externalization logic
6. **Cluster tags** (`economics/cluster.rs`)
 - Tag inheritance calculation
 - Fee multiplier computation
 - Decay mechanism

E.3.2 Test Vectors

Auditors should verify the following test vectors:

Mnemonic: "abandon abandon ... about" (standard 24-word)
Expected view key (hex): 0x7b3a...
Expected spend key (hex): 0x4c1f...
Expected ML-KEM public key (first 32 bytes): 0x9d2e...

Message: 0x0000...0000 (32 zero bytes)
Ring size: 20
Real index: 7
Expected key image (hex): 0x5f8c...
Signature must verify: true

Cluster wealth: 1,000,000 BTH
Steepness parameter: 100,000
Expected factor: 5.5x

E.4 Known Limitations

The following are acknowledged limitations, not vulnerabilities:

1. **Classical ring signatures:** CLSAG provides sender privacy against classical adversaries only. A future quantum computer could potentially identify signers retroactively.
2. **Metadata leakage:** Transaction size reveals approximate input/output count. Timing analysis may leak information despite Dandelion++.
3. **Ring selection bias:** The decoy selection algorithm uses a gamma distribution that may not perfectly match real spend patterns.
4. **Trusted setup:** Bulletproofs require no trusted setup, but the choice of generators G and H must be verifiably random.

E.5 Reporting

Security issues should be reported to:

- **Email:** security@botho.org (PGP key on website)
- **Severity levels:** Critical, High, Medium, Low, Informational
- **Bug bounty:** Available for critical and high severity issues