

White Paper

Quantum Monetary System

An Introduction to the Features of the QMS Blockchain

Abstract

Quantum Monetary System (QMS) is a layer-1 blockchain purpose-built for the quantum era. It is quantum-resistant by design and accommodates quantum computers in its miner network. Every building block is optimized for this future, from consensus mechanism to fast finality to tokenomics.

QMS achieves this by using *proof-of-useful-work* as its consensus mechanism combined with a post-quantum finality layer. By replacing wasteful hashing with computation suited to quantum hardware, QMS becomes a decentralized cloud computing platform that quantum miners can power.

This is the "useful" in proof-of-useful-work. Commercial clients pay to have their optimization problems solved, giving miners a second source of income on top of standard block rewards. That second income reduces the inflationary cost of securing the chain while preserving the physical security of proof-of-work.

Version 1.1 · October 2026

© QMS Network

Contents

1 Introduction **4**

2 An Overview of QMS **5**
- 2.1 A Decentralized Computing Platform . . . 5
- 2.2 Built for Quantum Hardware . . . 6
- 2.3 Post-Quantum Security . . . 6
- 2.4 EVM Compatibility . . . 6
- 2.5 Explicit Finality . . . 6

3 Architecture **7**
- 3.1 Overview . . . 7
- 3.2 Network Layer . . . 7
- 3.3 Consensus Layer . . . 8
- 3.4 Finality Layer . . . 8
- 3.5 Execution Layer . . . 9

4 Proof-of-Useful-Work **9**
- 4.1 Why Proof-of-Useful-Work . . . 9
- 4.2 How It Works . . . 9
- 4.3 Chain Selection . . . 11
- 4.4 Security of PoUW . . . 11
- 4.5 QUBO Problems . . . 12
- 4.6 Integrating Quantum Miners . . . 12

5 Finality Layer **13**
- 5.1 How It Works . . . 13
- 5.2 Why Explicit Finality . . . 13
- 5.3 Security of QMS . . . 14
- 5.4 Progressive Decentralization . . . 15

6 Post-Quantum Security **15**
- 6.1 Consensus Layer . . . 16
- 6.2 Finality Layer . . . 16
- 6.3 Execution Layer . . . 16

7 Incentives **17**
- 7.1 Transaction Fees . . . 18
- 7.2 Solution Rewards . . . 18
- 7.3 Block Rewards . . . 18
- 7.4 Parameters . . . 20

8 Portfolio Optimization **20**
8.1 Problem Lifecycle 21
8.2 The Optimization Problem 21
8.3 Reduction to QUBO 22
8.4 Problem Classes 22

9 Governance **23**

10 Progressive Roll-Out **23**
10.1 Releases 23
10.2 Directions 24

References **25**

Glossary **26**

1 Introduction

Blockchain technology has matured. Proof-of-work (PoW) has secured Bitcoin's monetary system for more than fifteen years. Proof-of-stake (PoS), introduced at scale by Ethereum, replaces equipment and energy costs with validator collateral and has brought sub-second block times to a new generation of chains. The Ethereum Virtual Machine (EVM) [8] has become the de facto standard for smart contract execution, supported by a vast ecosystem of wallets, tooling, and infrastructure. Underpinning all of this is a single, elegant invention: public-key cryptography. Fast, practical, and proven across decades, it is what makes the entire stack possible.

Quantum computing puts that foundation at risk. Elliptic-curve signatures will be broken by Shor's algorithm, allowing funds to be stolen. Classical zero-knowledge (ZK) proofs will be forged. And for privacy chains, the entire history of encrypted transactions will be exposed. Established networks face multi-year, ecosystem-wide migrations to post-quantum (PQ) cryptography, complicated by accumulated protocol versions, deep tooling dependencies, and dormant accounts whose owners may never apply the upgrade. The PQ algorithms that solve the problem carry costs of their own: larger signatures and meaningful computational overhead. Sub-second block times, the cornerstone of PoS's performance narrative, are not compatible with PQ validator signatures.

But the same hardware that threatens existing cryptography is also a tool. The blockchain industry has organized its response around the threat alone, missing what it can do for chains themselves. Many of the optimization problems quantum hardware is natively designed to tackle (e.g., portfolio optimization, risk selection, clustering) already have paying commercial clients. Miners whose work solves these problems and secures the chain earn from both the chain and these clients. Two revenue streams from a single computation.

Quantum hardware is no longer hypothetical. Public roadmaps from major research labs and venture-backed startups now point to systems capable of running Shor's algorithm against deployed cryptography. The horizon is short. Any blockchain still relying on non-PQ cryptography when those systems arrive will be a target. Privacy chains are already exposed through harvest-now-decrypt-later (HNDL) attacks. The migration window is closing, and migrating a live multi-year ecosystem is fundamentally harder than starting from a clean slate.

But the flip side is that quantum prototypes are also already here. Quantum annealers and gate-based machines are accessible over standard cloud APIs for small tasks. Miners delegating part of the work to these quantum prototypes are toy examples today; they will be standard tomorrow. The QMS protocol is designed so that as quantum hardware matures, the miners who run it produce better solutions and earn larger shares of client payments.

Advances in quantum computing directly benefit the chain.

This is what QMS does. It is built from the ground up for that world: quantum-resistant by design, with a PQ-secure *proof-of-useful-work* (PoUW)[1] consensus protocol, a PQ-secure finality layer, and prototype quantum computers working alongside classical ones in its miner network.

2 An Overview of QMS

QMS, the Quantum Monetary System, is a layer-1 blockchain built for the quantum era. Its custom consensus layer, implemented by a client named Casimir,[2] is quantum-resistant by design. Block production uses no public-key cryptography, the component of conventional blockchains that quantum computers can break. Its consensus mechanism, PoUW, spends the effort of securing the chain on optimization problems that commercial clients pay to have solved, turning the miner network into a decentralized cloud computing platform. Quantum hardware is suited to these problems, so quantum miners can join the network and claim rewards for the solutions they find. A PQ-secure finality layer marks blocks as final shortly after they are produced, serving applications that require fast finality.

This white paper describes QMS in its complete, target form. QMS is launched in stages. The first releases deliver a secure, functioning chain, and further capabilities are introduced as commercial demand and technological progress direct their priority.

2.1 A Decentralized Computing Platform

Under PoUW, miners run randomized solvers on real optimization problems in place of computing hashes. Clients post problems to an on-chain marketplace and lock funds to pay for good solutions, so the same solver run that earns a miner the chance at a block reward can also claim a client payment. Standard PoW expends energy on hashing that serves only to secure the chain; PoUW spends that effort on problems whose solutions carry independent commercial value. Earning from both block rewards and client payments is designed to support a reinforcing cycle: more miners add computational capacity, greater capacity attracts more clients, and their payments draw more miners. Like any market-driven mechanism, the strength of this cycle depends on uptake.

[1] The term proof-of-useful-work was introduced by Ball et al. [2]. The authors subsequently showed their definition to be inadequate and dropped the framing from the published version; QMS adopts the formal treatment of [12].

[2] Casimir was initially forked from Lighthouse [20].

2.2 Built for Quantum Hardware

The initial class of problems QMS solves, quadratic unconstrained binary optimization (QUBO), is the class that quantum annealing hardware and the quantum approximate optimization algorithm (QAOA) [10] are natively designed to tackle. Clients pay for solutions that meet their criteria, so a miner whose hardware reaches those criteria more often earns a larger share, regardless of how the solutions are produced. Quantum hardware can therefore join the miner network and earn as soon as it produces competitive solutions, without modification to the consensus protocol. As the technology matures, the miners who run it can produce stronger solutions and earn more, aligning advances in quantum computing with the network rather than against it.

2.3 Post-Quantum Security

QMS's core protocol is designed to be PQ-secure. The consensus layer is quantum-resistant by construction. PoUW secures block production with hash functions, which remain secure against Shor's algorithm [19]. The finality layer uses PQ signatures to mark blocks as final. The execution layer, inherited from Ethereum, is the exception. It uses standard Ethereum signatures for EVM compatibility. Security-conscious users can upgrade their accounts to PQ signatures.

2.4 EVM Compatibility

The execution layer runs a standard Ethereum execution client, making QMS compatible with existing wallets, block explorers, and developer tooling from launch. Smart contracts deployed on Ethereum migrate without code changes. This compatibility also means QMS arrives with the capabilities expected of a modern layer-1: native settlement, programmable smart contracts, and the standard developer tooling of the EVM ecosystem.

2.5 Explicit Finality

Alongside the consensus protocol, a PQ-secure finality layer marks blocks as final through the votes of *finalizers*, who stake the native token. QMS builds it on Afgjort [7], a partially synchronous finality protocol. Once a block is finalized, it and its ancestors cannot be rolled back. This gives bridges, exchanges, and other applications a settled state to build on, and it protects the developing chain. While mining power is still modest, a permissioned finalizer set guards against an opportunistic 51% attack and acts as a safety net as PoUW parameters are calibrated on a live network. The finalizer set is known at launch and opens to staked participants as the network matures and token distribution broadens.

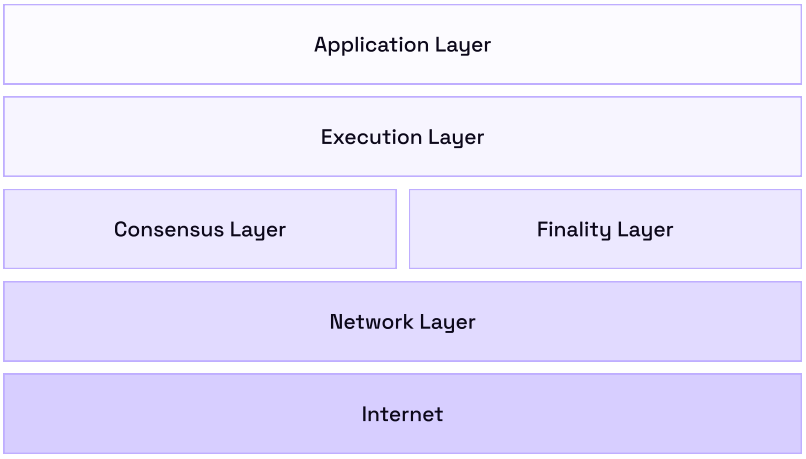

Figure 1. The QMS protocol stack. The four QMS layers — network, consensus, finality, and execution — sit between the public internet below and the application layer above.

3 Architecture

3.1 Overview

The QMS blockchain is organized into four layers, each handling a distinct responsibility (Figure 1):

1. **Network Layer**: peer-to-peer communication and catch-up.
2. **Consensus Layer**: block production via PoUW and chain selection.
3. **Finality Layer**: explicit finality through a leaderless stake-based finalizer set.
4. **Execution Layer**: EVM-compatible transaction processing and smart contract execution.

QMS takes the network layer and execution layer mostly untouched from Ethereum, inheriting a mature codebase and EVM compatibility. The consensus layer and finality layer are custom-made to support useful work, quantum miners, and PQ security.

3.2 Network Layer

The network layer handles peer discovery, peer-to-peer message propagation, block dissemination, and catch-up for nodes that join or have fallen behind.

A standard Ethereum execution client handles transaction propagation at the execution layer, and the consensus client, Casimir, handles peer-to-peer communication on the consensus side. Both inherit networking from clients that have run for years on Ethereum mainnet, refining peer scoring, rate

limiting, and resistance to denial-of-service attacks. This avoids rebuilding well-understood infrastructure.

Transport and peer discovery carry over unchanged. The messages transported are defined by the layers above: blocks by the consensus layer, transactions by the execution layer, finality votes by the finality layer. The sync mechanisms that bring new joiners up to date are adapted as well. A node can either replay from genesis or bootstrap from a recent finalized block, then continue from the live tip via gossip.

3.3 Consensus Layer

The consensus layer orders transactions into blocks and resolves which chain is canonical when miners produce competing ones. QMS's consensus client, Casimir, secures the chain with PoUW — a PoW protocol in which miners run a solver on real-world optimization problems rather than computing hashes.

The initial problem class is QUBO, which covers many practical applications in finance and data analysis. The modular PoUW implementation allows further problem classes to be added as the network matures.

Block production rights are awarded by a lottery over valid solver outputs, independently of solution quality, so pre-solving submitted problems gives an adversary no advantage. Above the last finalized block, the canonical chain is the heaviest chain, the one with the most cumulative work. Block production and chain selection both rely on computational work rather than signatures and staking, making the consensus layer quantum-resistant by design.

3.4 Finality Layer

The finality layer uses a PQ signature algorithm to mark blocks as final, guaranteeing they will never be rolled back. Finalizers stake the native token and vote on which blocks to finalize. Once a block is finalized, all of its ancestors are finalized with it. The finality layer runs alongside the consensus protocol. Under ideal conditions, blocks are finalized shortly after they are produced.

The PoUW consensus is leaderless, and QMS's finality layer preserves this property. No single finalizer drives a given round of finalization. This provides security against targeted distributed denial-of-service (DDoS) attacks, where an attacker can stall consensus by disrupting successive round leaders. Finalization happens asynchronously to block production, so the finality layer runs at its own pace. Finalization time, including the overhead of the PQ signatures, does not affect block times or throughput.

3.5 Execution Layer

The execution layer processes transactions and runs smart contracts. QMS uses a standard Ethereum execution client [9],[3] making the chain EVM-compatible. Existing wallets, block explorers, and developer tooling work without modification, and smart contracts deployed on Ethereum can be migrated with no code changes. Transactions are priced in the native token, but the EIP-1559 fee structure [5] is inherited from Ethereum.

QMS adopts a hybrid model for PQ security of the execution layer. Users can choose standard Ethereum-style signatures, which preserve EVM-tooling compatibility, or PQ signatures, which protect against future quantum computers but may require an additional wallet. Accounts can be upgraded from standard to PQ signatures with a dedicated transaction.

4 Proof-of-Useful-Work

4.1 Why Proof-of-Useful-Work

PoW's security is grounded in physical reality. The cost of hardware and energy determines a miner's relative computational power in the consensus protocol. Alternatives such as PoS reduce that expenditure but introduce trade-offs of their own, including a tendency toward centralization (more rewards for the larger validators), protocol-level vulnerabilities (long-range attacks, targeted validator DDoS), and biases resulting from loans, delegation, and off-chain deals.

PoW also has a property that has become particularly relevant now that blockchains are racing to migrate to PQ cryptography. There are no signatures in block production, so it remains secure against Shor's algorithm [19].[4]

PoUW is a special type of PoW protocol that fixes the main drawback of standard PoW while preserving its advantages. It replaces the hash-based PoW with work that has independent commercial value. If the work needs to be done anyway for commercial applications, then the remaining cost for the miners is the proof that it has been done, which is the genuine, irreducible cost of securing the chain. The physical security and PQ readiness are inherited from the PoW paradigm.

4.2 How It Works

The QMS implementation follows Ofelimos [11, 12]. Miners solve problems proposed by clients and get paid for finding good solutions. In parallel,

[3] QMS currently uses a fork of Reth [18].

[4] While Grover's algorithm [14] offers a quadratic asymptotic speedup, recent work [1] has shown that in non-asymptotic settings this is too small to compensate for the constant overhead and slower clock speed of quantum computers. A quantum miner is in itself not a threat, but rather the next evolution in mining hardware.

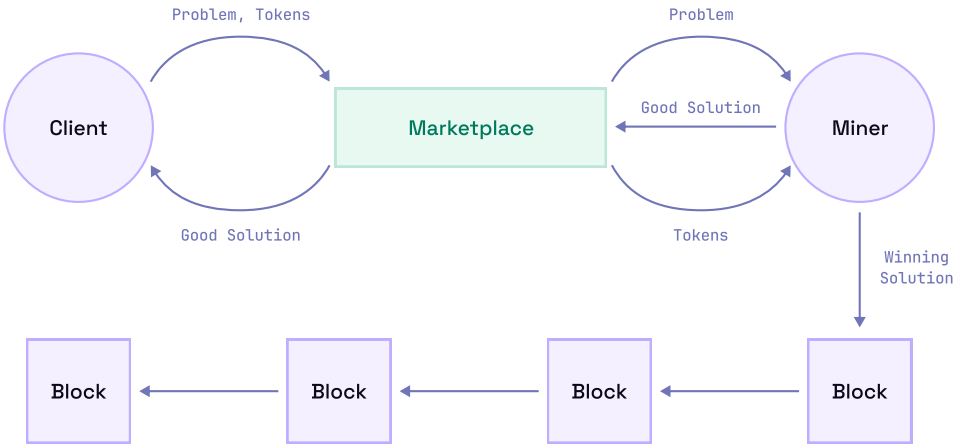

Figure 2. The PoUW marketplace. Clients post problems to the marketplace and lock tokens to pay for good solutions. Miners take problems from the marketplace, run solvers against them, and submit good solutions, which the marketplace pays for in tokens and returns to the client. A solver run that wins the block-production lottery produces the next block in the chain.

miners prove that they are doing the work, which earns them the right to produce blocks and collect block rewards. For security reasons, the right to produce blocks is decoupled from finding good solutions. This is covered in more detail in Section 4.4.

More specifically, clients post problems to an on-chain marketplace and lock funds into a smart contract to pay for good solutions. Miners choose problems from the marketplace and run a solver against them. Good solutions are submitted back on-chain, and the smart contract distributes the locked payments to the miners that produced them (Figure 2).

Ofelimos [11, 12] relies on randomized heuristics to solve an optimization problem. The algorithm gradually finds better solutions until it gets stuck in a local optimum. At this point the miner may restart from another point, or decide that the solution is good enough and choose another problem to solve. At regular intervals, the current solution is hashed to determine if the miner wins the right to produce a block. This follows the same principle as Bitcoin [16]. If the hash falls below a difficulty target, the miner wins the lottery. By adjusting the difficulty target, the expected block rate can be tuned.[5] If the block makes it to the main chain, the miner receives the block rewards and their share of transaction fees.

To add a block to the chain, in addition to winning the lottery, the miner has to prove that they performed the useful work. QMS verifies this by chunking the computation. The solver hashes its state at short intervals

[5]The same principle keeps the block rate steady as miners join or leave the network. A difficulty adjustment algorithm periodically computes a new target to compensate for these fluctuations in mining power.

during its run, so verification need only re-run a single chunk rather than the full computation. A miner cannot skip work without producing an inconsistent hash. A succinct-argument variant (a hash-based SNARK [3] proving the work was performed correctly) is a possible alternative: heavier to generate but lighter to verify, and able to certify the work without revealing the specific problem solved.

4.3 Chain Selection

When competing forks exist, miners extend the heaviest chain, the one with the most cumulative work. A block's weight is determined by the hash difficulty at the time of production, as in Bitcoin [16], but also by how much solver work each lottery attempt requires, i.e., the size of a chunk of computation. The protocol fixes how much work a chunk represents, independently of the problem its miner is solving. This requires calibration for every solver implemented and every problem size.

A miner that has not yet seen a competing fork builds on the chain it knows. As chains propagate across the network, whichever fork has accumulated the most work eventually prevails.

4.4 Security of PoUW

A major security challenge of any PoUW system is preventing a party from submitting problems they have pre-solved offline, then mining with pre-computed answers. QMS adopts the Ofelimos approach [11, 12], which combines three mechanisms to remove the advantage of pre-computation:

1. **Randomized solver.** The solver is required to be a randomized algorithm, e.g., simulated annealing, tabu search, or evolutionary search. This means that the solver's output depends on a random seed, not just on the problem.
2. **Seed bound to the parent block.** The miners are required to use a seed derived from a hash of the parent block. Nodes check that the miner's solution was produced with the required seed, preventing any pre-computation of the output.
3. **Lottery over outputs, not quality.** Block production rights are awarded by a lottery over the solver's valid outputs, decoupling block production from solution quality. Any prior knowledge of good starting points or solutions does not help in producing blocks.

The security of a PoUW consensus protocol is defined in the same way as for any other PoW protocol. The framework of Garay, Kiayias, and Leonardos [13] establishes two core properties: *persistence* (once a transaction is sufficiently deep in the chain, no honest party will see it removed in any

future view of the chain) and *liveness* (any valid transaction broadcast on the network will eventually be included in the chain at a sufficient depth). Both hold under bounded network delay, a majority of honest mining power, and (for PoUW specifically) a hardness assumption on the miner's computation.[6] Ofelimos [12] proves these properties for the PoUW construction described above.

4.5 QUBO Problems

QMS's initial PoUW implementation uses QUBO problems. A QUBO problem asks for the binary assignment that minimizes a quadratic objective function. It is simple to state, NP-hard in general, and applicable to a broad range of practical problems. Concretely, a QUBO problem is given by a symmetric matrix Q of real numbers with n rows and columns, and asks for the assignment $x \in \{0,1\}^n$ that minimizes $x^\mathsf{T} Q x$. Because $x_i^2 = x_i$ for a binary variable, the diagonal of Q carries the linear part of the objective function, and no separate linear term is needed. Solving a QUBO is mathematically equivalent to finding the ground state of an Ising Hamiltonian, the lowest-energy configuration of an associated physical system. This is precisely the task that quantum annealing hardware and QAOA [10] are designed to tackle.

Because many practical problems in finance and data analysis (including portfolio optimization, risk selection, and clustering) can be formulated as QUBOs, the QMS network can be rented to solve real commercial problems. The modular design of the PoUW implementation allows new problem classes to be added as the network evolves, either as a new reduction to a QUBO problem or with a different solver altogether (see Section 8.4). Security does not depend on which problem is solved, only on the solver being a randomized local search with the moderate hardness property [11, 12].

4.6 Integrating Quantum Miners

The PoUW architecture creates two pathways for quantum hardware to participate in the network.

The first pathway is built into the incentive structure from the start. Since the marketplace pays for solutions that meet a client's criteria, a miner whose hardware reaches those criteria more often earns a larger share of marketplace payments, regardless of how those solutions were produced. As soon as a quantum solver can outperform a classical one, it can pass an intermediate solution to a classical miner as a starting point for its seeded local search. The result is a hybrid quantum-classical miner with an edge over purely classical ones. The security argument still holds. The miner

[6] Formally, in Ofelimos, this is a *moderate-hardness* condition: the local-search algorithm cannot be shortcut, and each step contributes computational work in expectation [12].

proves that the work of the seeded classical search was done, regardless of its starting state. It earns the right to produce blocks and collect payments from the marketplace.

The second pathway is a longer-term research direction. The Ofelimos security argument depends on a property classical randomized solvers have but quantum ones do not. A classical solver, seeded from the parent block, produces a deterministic trajectory that a verifier can rerun and check. Quantum solvers are inherently probabilistic. Their outputs are samples drawn from the state the computation produces, not the output of a deterministic seeded procedure. Extending Ofelimos-style security to quantum solvers therefore requires a different mechanism for proving that the work has not been pre-computed. QMS is working with academic partners on such a framework.

5 Finality Layer

5.1 How It Works

QMS's finality layer is built on Afgjort [7], a partially synchronous finality protocol developed by Dinsdale-Young et al. Finalizers stake the native token and vote to mark blocks as final. Once a block is finalized, it and all its ancestors cannot be rolled back, regardless of subsequent mining activity. Finalization runs asynchronously alongside block production. The finality layer processes blocks at its own pace without gating block production or affecting transaction throughput. If it falls behind consensus, finalizing a recent block catches the whole chain up at once.

The finality layer divides the chain into two regions. Below the last finalized block, history is permanent — that portion of the chain will never be rewritten. Above it, the selection rule is unchanged. Miners extend the heaviest chain, the one with the most cumulative work. As the finality layer advances, it progressively converts more of this chain into permanent history.

Afgjort is *Byzantine-fault-tolerant* (BFT): it tolerates finalizers that behave arbitrarily (Byzantine faults), and stays secure as long as fewer than one-third of them are dishonest. One-third is the optimal resilience threshold for Byzantine agreement under partial synchrony.[7] The original Afgjort protocol has been refined in subsequent work; the variant deployed will reflect the state of the art.

5.2 Why Explicit Finality

PoUW, like standard PoW, provides probabilistic finality. The deeper a block sits in the chain, the less likely it is to be reversed, but reversal never

[7] Under partial synchrony, message delays are bounded, but the bound is unknown and holds only after some point in time.

becomes formally impossible. In practice, reversal is managed by waiting for enough block confirmations that the probability of a deep reorganization is negligible, but for any application that requires the chain's state to be settled, this waiting period can be substantial.

The finality layer converts this into an absolute guarantee for a growing prefix of the chain. Cross-chain bridges, exchanges, and any protocol that requires fast finality can rely on finalized blocks rather than waiting for probabilistic confidence.

The finality layer also provides a safety net against a calibration challenge specific to PoUW. The protocol fixes the solver work required before each lottery attempt, and that requirement must be calibrated so that it is equivalent across the problems miners solve. If the calibration is imperfect, some problems yield lottery attempts more cheaply than others, giving the miners that select them a disproportionate share of block production. The finality layer prevents this from becoming a critical vulnerability. Finalized blocks cannot be rolled back even if miscalibrated work requirements temporarily distort block production.

A new chain with modest total mining power is also vulnerable to opportunistic 51% attacks. PoUW runs on general-purpose hardware, which can be rented in quantity for the length of an attack. Explicit finality removes this window. Finalized blocks cannot be reversed even if an attacker temporarily dominates PoUW mining.

5.3 Security of QMS

Composing the heaviest-chain consensus (Ofelimos) with the BFT finality layer (Afgjort) determines the security of the QMS chain — the network conditions it tolerates and the honesty it requires of its participants. *Safety*, that a finalized block is never rolled back, is inherited from Afgjort; it holds even under full network asynchrony and requires only that more than two-thirds of the finalizers, weighted by stake, be honest. *Liveness*, that transactions are eventually finalized, requires both layers to make progress: a majority of honest mining power to extend the chain, more than two-thirds honest finalizers to finalize it, and a partially synchronous network. The QMS chain is thus secure in the partially synchronous model, with safety resting on the finalizers alone and liveness on miners and finalizers together.

Many BFT-style finality protocols designate a single round leader who proposes the next block to finalize. This creates a concentrated target. Disrupting the leader stalls finalization. Afgjort runs every round without a designated proposer [7], eliminating this attack surface. QMS's PoUW consensus is already leaderless, as any miner can produce a block at any time, and Afgjort preserves this advantage at the finality layer, making the whole protocol resistant to targeted DDoS attacks.

5.4 Progressive Decentralization

The finality layer launches with a known, permissioned finalizer set, operated initially by entities accountable to QMS Network. This arrangement is deliberate and time-bounded. A controlled finalizer set is easier to correct and coordinate while the chain is being calibrated against live conditions.

A trusted finalizer set also admits a simpler fault model. Where Afgjort is BFT, the launch protocol need only be *crash-fault-tolerant* (CFT): finalizers may inadvertently crash or the network may partition and messages get lost, but all online finalizers are assumed to run the protocol honestly. In this model, only a majority of finalizers need to be online for the protocol to keep progressing.

QMS implements this layer with Raft [17], a widely deployed CFT consensus algorithm. Additionally, the finalizers use a PQ signature algorithm to sign blocks as final, which provides the desired PQ security.[8]

As the network matures and token distribution broadens, the finality layer migrates to Afgjort so that the finalizer set may open to unknown participants who stake the native token.

6 Post-Quantum Security

Quantum computing does not threaten all cryptographic primitives equally. Shor's algorithm [19], which solves integer factorization and discrete logarithm problems in polynomial time, breaks RSA and all elliptic-curve cryptography. Conventional digital signatures fail, since a quantum adversary can derive a private key from the corresponding public key. Classical ZK proofs whose soundness rests on discrete-log or factoring hardness also become forgeable.

The broken primitives have replacements. Cryptographers have developed public-key algorithms whose security rests on problems believed to be hard even for quantum computers, such as finding short vectors in high-dimensional lattices, decoding random linear codes, or inverting hash functions. Several have been standardized[9] and can secure most of the same applications as the primitives they replace, at a cost: PQ public keys, signatures, and ciphertexts are substantially larger than their elliptic-curve counterparts, and some operations are more computationally expensive.

Grover's algorithm [14] presents a different and more limited threat. It searches an unstructured space of size N in roughly $\sqrt{N}$ quantum operations, an asymptotically quadratic speedup that erodes the security margins of hash-based cryptographic primitives, including hash-based signature algorithms

[8] Since the finalizers are trusted, a single finalizer signature suffices, avoiding the many large PQ signatures that Afgjort, or any BFT consensus protocol, would require.

[9] These include the lattice-based ML-KEM for key establishment and ML-DSA for signatures, and the hash-based SLH-DSA for signatures.

and proof systems. However, hash-based constructions maintain adequate PQ security with appropriately sized parameters.

Block production on QMS's consensus layer requires no public-key cryptography; it relies on the hardness of inverting hash functions. The finality layer uses PQ signatures and is PQ-secure from launch. The execution layer, inherited from Ethereum, uses elliptic-curve signatures for EVM compatibility and requires migration to PQ signatures. Dormant accounts that miss the upgrade window face a targeted residual risk after Q-Day, the point at which quantum computers become powerful enough to break deployed elliptic-curve cryptography. Such accounts are frozen to contain this risk.

6.1 Consensus Layer

Block production with PoUW requires no digital signatures. Its security rests on hash functions, the moderate hardness of the local search, and the unpredictability of the randomness seed, which all remain secure against quantum adversaries. The consensus layer is quantum-resistant by design.

Miners can include finality information in the blocks they produce. Processing this information runs in parallel with the PoUW solver and hash lottery, so PQ overhead at the finality layer adds to finalization latency rather than to block production time. In BFT-style consensus protocols, by contrast, a child block cannot be produced until signatures on the parent have been verified. Thus signature overhead extends directly to block time. QMS's parallel design removes that dependency.

6.2 Finality Layer

The security proof for the QMS chain relies on the signatures in the finality layer being unforgeable (see Section 5.3). For this reason, QMS launches with PQ signatures on this layer.

In the BFT model, each finalization proof requires the signatures of a quorum of finalizers. Rather than transmit all of them, the proof can include a hash-based SNARK [3] that proves they exist.[10] As noted in Section 6.1, this added cost falls on finalization latency rather than block production time. In the CFT model used at launch, a single signature per finalization proof suffices, and the Raft protocol [17] determines which finalizer signs it.

6.3 Execution Layer

For EVM compatibility, the execution layer initially uses standard Ethereum signatures for signing transactions, and introduces PQ signatures gradually (see Section 10). Unlike the consensus and finality layers, where quantum

[10] Conventional finality and BFT consensus protocols typically aggregate BLS signatures [4] to save space, but BLS is not PQ-secure.

vulnerabilities threaten the entire chain, on the execution layer it is the responsibility of individual users to choose their level of security.

Account abstraction [6] enables smart contracts to act as primary accounts (*smart accounts*), replacing standard ones. A smart account whose code requires a PQ signature on every transaction is effectively a PQ-secure wallet, even if the execution layer offers only non-PQ signatures for transactions.[11] Verifying a PQ signature in smart contract code is considerably more expensive than a native check. A precompile for PQ signature verification reduces that cost and is among the directions in Section 10.2.

A particular concern for any chain with optional PQ account security is that some account owners will not upgrade. Once Q-Day arrives, those accounts become vulnerable. A quantum adversary can derive the private key from the public key using Shor's algorithm [19] and steal the funds, which is potentially harmful to the whole chain.

QMS addresses this by freezing non-upgraded accounts once wallets, bridges, and exchanges support PQ accounts, which can be well before Q-Day. Legitimate owners can recover their funds by providing a PQ-secure ZK proof that they know the seed from which their private key was derived. The reverse operation, recovering a seed from a private key, requires inverting a cryptographic hash function. Even with Grover's quadratic speedup [14], the seed remains computationally infeasible to recover, so the recovery path remains secure against a quantum adversary. This is possible, however, only if a seed exists and the user still has access to it. The earlier the migration to PQ accounts, the easier this problem becomes. On an incumbent network, whose accounts are older and spread across far more wallet software, that condition is much harder to satisfy.

7 Incentives

QMS uses a single native token to collect payments from users and reward the miners and finalizers that maintain the security and stability of the chain. Three sources provide the tokens that fund these rewards:

1. Chain users, who submit transactions and pay gas fees.
2. Clients, who pay for solutions to their optimization problems.
3. The protocol itself, which mints[12] new tokens as block rewards.

[11] The user signs each operation with a PQ key. A bundler wraps the operation in an ordinary transaction signed with the bundler's own elliptic-curve key, and submits it to the chain. The protocol verifies the outer non-PQ signature, and the smart account's code verifies the inner one. Within the same transaction, the smart account reimburses the bundler's fee.

[12] Minting creates tokens rather than transferring them, so the cost of block rewards falls on existing token holders through dilution.

Tokens from these sources are either burned, paid to miners for finding solutions and producing blocks, or paid to finalizers for finalizing blocks.

Parameters govern how these tokens are shared between miners and finalizers and how much of them is burned (Figure 3). The exact parameter values are deferred to a later specification; the mechanisms below hold regardless of the numbers eventually chosen.

7.1 Transaction Fees

QMS follows Ethereum's fee model [5]. Each transaction pays a base fee, which is burned, and an optional priority fee, which goes to the miner that includes the transaction. Because transactions on QMS behave as they do on any EVM chain, keeping this structure both reuses proven code and keeps the cost of transacting familiar to existing users and tooling.

7.2 Solution Rewards

Clients pay miners for the solutions they find, a source of revenue that standard PoW does not provide. A client posts an optimization problem to the on-chain marketplace and locks tokens in its smart contract to pay for good solutions. Miners that produce qualifying solutions submit them on-chain, and the contract releases the locked payment. The same solver run that earns a miner the chance at a block reward can therefore also earn a client payment.

The client sets what counts as a qualifying solution and how large the reward is, within the rules the marketplace contract enforces. This keeps the reward criteria close to the commercial problem. A client optimizing a portfolio and a client routing a fleet of vehicles can value solutions on their own terms. The marketplace contract also determines what share of a client's payment is burned rather than paid to miners.

Because each solution that meets the client's criteria is paid, the miners who reach those criteria most often collect the most, regardless of how the solutions are produced, classical or quantum.

7.3 Block Rewards

Producing a block mints new tokens. The reward M for a new block is set by the protocol and may vary with block height or time. This reward is split three ways:

1. For making a valid block, $B = \alpha_B M$ is given to the miner.
2. For including a finalization proof in the block, $P = \alpha_P M$ is made available.
3. For signing the finalization proof, $F = \alpha_F M$ is made available to the finalizers, where $\alpha_F = 1 - \alpha_B - \alpha_P$.

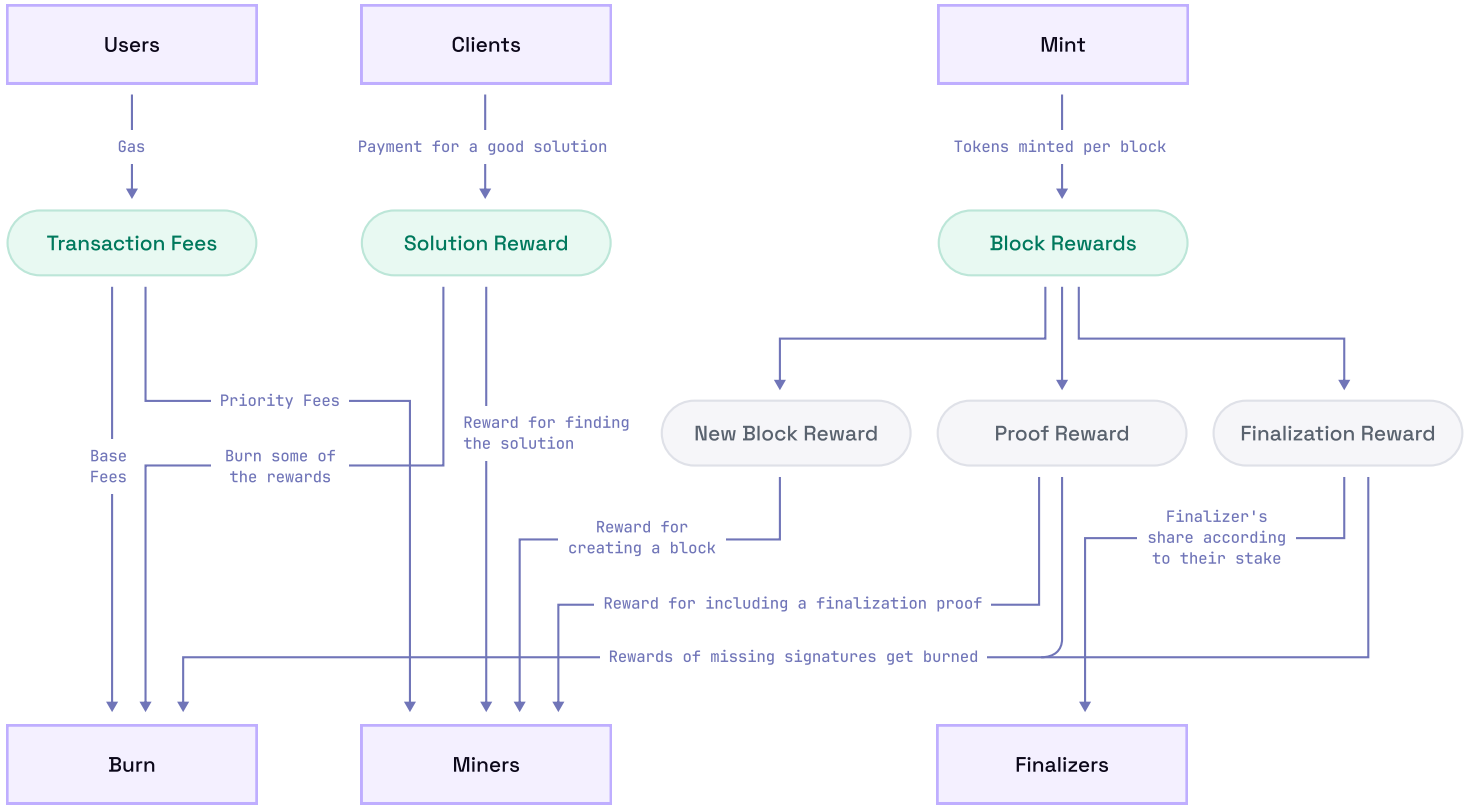

Figure 3. Token flows in QMS. Tokens enter from three sources: chain users paying gas, clients paying for good solutions, and the protocol minting tokens as block rewards. The minted reward is split three ways, for making the block (New Block Reward), for including a finalization proof (Proof Reward), and for signing it (Finalization Reward). Parameters determine how much reaches miners and finalizers, and how much is burned.

New block reward. B is paid directly to the miner that produced the block.

Finalization proof reward. If no finalization proof is included in the block, the proof reward P is added to an accrual account for the next block. If a proof is present, let $P_{\text{total}} = P_{\text{accrual}} + P$ be the sum of the accrual account and the current proof reward. The miner who included the proof in their block receives $w_{\text{total}} P_{\text{total}}$, where $w_{\text{total}} = \sum_i w_i$ is the combined relative weight of all signatures in the proof and $w_i = \frac{\texttt{stake}_i}{\texttt{total stake}}$ is the weight of finalizer i's signature. The remaining $(1 - w_{\text{total}})P_{\text{total}}$ is burned, and the accrual account is reset to 0.

Finalization reward. The finalization reward is distributed in the same way. If no proof is included, the finalization reward F is added to an accrual account for the next block. If a proof is present, let $F_{\text{total}} = F_{\text{accrual}} + F$ be the sum of the accrual account and the current finalization reward. Each finalizer i whose signature is included in the proof receives $w_i F_{\text{total}}$. The remaining $(1 - w_{\text{total}})F_{\text{total}}$ is burned, and the accrual account is reset to 0.

7.4 Parameters

These mechanisms are governed by a small set of parameters:

1. The minted block reward M.
2. The block, proof, and finalization shares of the block reward M, namely α_B, α_P, and α_F, which sum to one.
3. The parameters governing solution rewards, which the marketplace contract enforces.

The exact parameter values are not yet fixed; they will be set in a later specification, informed by commercial pilots and ecosystem feedback.

8 Portfolio Optimization

The first commercial problem class on the QMS marketplace is portfolio optimization: dividing a fixed amount of capital across a set of assets so as to balance expected return against risk. Markowitz [15] gave the formulation that remains standard, measuring the return of a portfolio by its expected value and its risk by the variance of that value. Correlation between assets is part of that risk. Holdings that move together reinforce one another instead of offsetting, which raises the variance of the portfolio. A portfolio can keep its risk low and accept a lower expected return, or reach for a higher expected return and accept more risk. The balance between them is set by the investor's risk profile. A portfolio optimization problem reduces to a QUBO problem, which QMS miners can solve.

8.1 Problem Lifecycle

The client is the party that submits the problem and pays for solutions. It may be a firm optimizing its own capital, or a platform submitting problems on behalf of its users and passing through their risk profiles and constraints.

A submitted problem states the assets to consider, estimates of the expected return and the risk of those assets, the amount to invest, a risk profile, and the constraints the resulting portfolio must satisfy, such as the maximum number of positions, bounds on the size of each position, and whether short positions are permitted. Producing the estimates is a modeling exercise performed by the client.

Once the submitted problem is in the marketplace, it can be picked up by miners, who run their solvers and submit the solutions they find. The solutions are evaluated against the client's criteria, and the marketplace contract releases payment on that basis. Anyone can check that the evaluations and payments follow those criteria. A client can add to the reward on a problem to draw more effort to it, and post a new problem when its data changes.

8.2 The Optimization Problem

Consider n assets and let w_i denote the fraction of the capital allocated to asset i. Write μ for the vector of expected returns and Σ for the covariance matrix of returns, both supplied with the problem. The portfolio then has expected return $\mu^{\mathsf{T}} w$ and variance $w^{\mathsf{T}} \Sigma w$. A risk-aversion coefficient $q \geq 0$, fixed by the client's risk profile, weighs the second against the first, resulting in the objective function $q w^{\mathsf{T}} \Sigma w - \mu^{\mathsf{T}} w$ to be minimized. This is the mean-variance criterion of Markowitz [15] in its risk-aversion form.

Minimizing this objective function on its own can produce an impractical portfolio, with the capital divided into very small holdings across a large number of assets. Additional constraints therefore restrict the solutions to admissible portfolios. For the capital to be fully allocated, the weights must satisfy $\sum_i w_i = 1$. An indicator $y_i \in \{0, 1\}$ records whether asset i is held at all, which allows the number of positions to be capped at K by $\sum_i y_i \leq K$. Each held position is kept between a floor ℓ and a ceiling u by $\ell y_i \leq w_i \leq u y_i$, which also binds w_i and y_i by making $y_i = 1$ whenever $w_i > 0$. Putting these together gives

$$\begin{aligned} \min_{w \in \mathbb{R}^n,\ y \in \{0,1\}^n} \quad & q w^{\mathsf{T}} \Sigma w - \mu^{\mathsf{T}} w \\ \text{subject to} \quad & \sum_i w_i = 1, \quad \sum_i y_i \leq K, \quad \ell y_i \leq w_i \leq u y_i. \end{aligned}$$

The continuous weights and the binary indicators together make this a *mixed-integer quadratic program* (MIQP), which is NP-hard in general.

The formulation above is long-only, with $\ell \geq 0$. Short positions can be captured by splitting each weight into a long and a short component, together with an optional bound on total gross exposure. A reduction similar to the one explained below is carried out when short positions are included.

8.3 Reduction to QUBO

A QUBO is a single quadratic objective function over binary variables, with no constraints. The MIQP has constraints and continuous variables. The reduction discretizes the weights and moves the constraints into the objective function.

In their discrete form, weights are represented as a whole number of units, where one unit is a fixed fraction of the capital (e.g., a unit of 1% allows any weight $x\%$ with $x \in \{0, \ldots, 100\}$). These integers are then written in binary, which maps each weight to a set of binary variables. The constraints and the objective function are adapted accordingly. Because each weight is a linear function of its binary variables, the objective function remains quadratic.

Each constraint is then added to the objective function as a penalty term that is zero when the constraint holds and positive when it does not, scaled by a weight chosen so that violating the constraint never lowers the objective function. What remains is an unconstrained quadratic objective function over binary variables, which is a QUBO.

In the MIQP a constraint is hard, and an assignment that violates it is not a portfolio at all. In the QUBO the same constraint is only expensive to violate, so a minimizer of the QUBO is not automatically a valid portfolio. Any assignment returned by a solver is checked against the original constraints before it is accepted.

8.4 Problem Classes

The same reduction can be used for a variety of other problem classes, where each candidate has an expected return and variance that can be derived from past data, for example:

- Allocating capital across AI trading agents.
- Distributing stake across pools.
- Dividing a treasury across reserves, staking, lending, and hedges.
- Selecting which risks an insurance protocol underwrites, and how much capital to allocate to each.

In all these examples, the risk of correlations between the assets, agents, or pools is captured by Markowitz's formulation of the optimization problem. Different optimization problem classes such as clustering or feature selection can also be expressed as QUBOs, but each requires a reduction of its own

before it can be offered on the marketplace. Other problem classes such as vehicle routing and scheduling are more efficiently solved by a solver of their own than by a reduction to QUBO.

9 Governance

In the early phases, protocol parameters and upgrades are managed by QMS Network. This is a deliberate and time-bounded arrangement, intended to allow rapid iteration while the network is still being calibrated against real-world conditions. Decentralization of governance proceeds in parallel with the decentralization of the finalizer set and the maturation of the broader ecosystem.

The detailed governance mechanism is still under design. Voting rules, proposal thresholds, vote weighting, and the relationship between QMS Network and the on-chain governance process will be specified in a future revision of this document.

10 Progressive Roll-Out

The QMS blockchain described in this white paper is launched in stages, each adding new features to the one before. Releasing incrementally allows each component to be rapidly tested and validated, so that each new version builds on a solid basis. It also leaves the order open to revision as commercial demand, market feedback, and the surrounding technology change.

The releases described below reflect current priorities. Client requirements differ: a trading firm using QUBO for portfolio optimization needs tools that connect the solver to wallets and exchanges, while a logistics company needs a solver of its own for much larger problems at high accuracy, reached through a conventional, non-blockchain interface. The PQ timeline depends on how fast quantum hardware advances, how PQ standards settle, and when Ethereum ships the execution-layer changes QMS inherits. The same advances in quantum hardware also determine when quantum miners can join the network.

10.1 Releases

QMS v0.1. The first release is a preview, a minimal blockchain that showcases the work in progress, with no expectation that its state is preserved in the next version. It runs a PoUW consensus protocol solving randomly generated problems, connected to an EVM execution layer, so transactions can be submitted and smart contracts run. Some key features of a consensus layer, e.g., the difficulty adjustment algorithm and block rewards, are deferred to the next release, as is the finality layer.

QMS v1. By the second release, QMS has everything needed to be a functional, secure blockchain fit for applications that hold and move real value: a feature-complete PoUW consensus protocol, a PQ finality layer, and an EVM execution layer. The network processes transactions and runs smart contracts like any EVM chain, except that fees are paid in the native token. Some advanced features, e.g., blobs, are deactivated.

QMS v2. The third release focuses on the useful-work marketplace and on problems submitted by clients. A back-end service converts a client's data into QUBO form, and the resulting problems are posted to the marketplace for miners to pick up. Solution rewards are also implemented.

QMS v1 implements a PoUW consensus protocol in the academic sense. The construction is proven to be secure even if the problems solved by the miners are adversarially chosen [12]. This is the distinguishing security feature between standard PoW and PoUW. However, from a commercial perspective, one cannot claim that the work is useful. This release does not require significant mining power. The finality layer ensures security in this setting. Energy expenditure for mining is expected to grow with commercial demand and solution rewards, introduced in QMS v2.

In QMS v2 the finality layer is still run by a permissioned finalizer set, possibly a single operator with redundancy. Staking is not implemented yet, so the finalization rewards are all burned. Decentralizing the finality layer is one of the directions described below.

10.2 Directions

Beyond QMS v2, several directions of development are open. The order of the list below is not a ranking.

- **Useful work.** QMS v2 implements the reduction from portfolio optimization to QUBO. There are many directions in which the useful work can develop: more features for portfolio optimization, problems with the same structure (e.g., allocating capital across AI trading agents), or further reductions and solvers for problems of other structures.
- **Quantum miners.** Research can establish how quantum miners prove that they have done the required work. Integrating them into the network alongside their classical and hybrid quantum-classical counterparts is a second step.
- **PQ security.** Adding a precompile for PQ signatures can make PQ applications such as smart accounts more efficient. Implementing PQ signatures on the execution layer makes accounts PQ-secure natively rather than through a smart account, with protections for dormant accounts to follow.

- **Finalizer decentralization.** Opening the finality layer to a permissionless set of finalizers, who stake the native token to participate, removes the dependence on trusted operators. This means running the Afgjort finality protocol [7] and building the staking mechanism.

References

[1] Divesh Aggarwal, Gavin K. Brennen, Troy Lee, Miklos Santha, and Marco Tomamichel. "Quantum Attacks on Bitcoin, and How to Protect Against Them". In: *Ledger* 3 (2018). DOI: 10.5195/ledger.2018.127. arXiv: 1710.10377.

[2] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. *Proofs of Useful Work*. 2017. Cryptology ePrint Archive, Paper 2017/203.

[3] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, Transparent, and Post-Quantum Secure Computational Integrity*. 2018. Cryptology ePrint Archive, Paper 2018/046.

[4] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Journal of Cryptology* 17.4 (2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9.

[5] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. *EIP-1559: Fee market change for ETH 1.0 chain*. 2019. URL: https://eips.ethereum.org/EIPS/eip-1559.

[6] Vitalik Buterin, Yoav Weiss, Dror Tirosh, Shahaf Nacson, Alex Forshtat, Kristof Gazso, and Tjaden Hess. *ERC-4337: Account Abstraction Using Alt Mempool*. 2021. URL: https://eips.ethereum.org/EIPS/eip-4337.

[7] Thomas Dinsdale-Young, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. "Afgjort: A Partially Synchronous Finality Layer for Blockchains". In: *Security and Cryptography for Networks – SCN 2020*. Vol. 12238. LNCS. Springer, 2020, pp. 24–44. DOI: 10.1007/978-3-030-57990-6_2. Cryptology ePrint Archive, Paper 2019/504.

[8] ethereum.org. *Ethereum Virtual Machine (EVM)*. URL: https://ethereum.org/developers/docs/evm/ (visited on 09/20/2026).

[9] ethereum.org. *Nodes and clients*. URL: https://ethereum.org/developers/docs/nodes-and-clients/ (visited on 09/20/2026).

[10] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. arXiv: 1411.4028.

[11] Matthias Fitzi, Aggelos Kiayias, Laurent Michel, Giorgos Panagiotakos, and Alexander Russell. *Efficient and Proof-of-Useful-Work Friendly Local-Search for Distributed Consensus*. 2025. Cryptology ePrint Archive, Paper 2025/2091.

[12] Matthias Fitzi, Aggelos Kiayias, Giorgos Panagiotakos, and Alexander Russell. "Ofelimos: Combinatorial Optimization via Proof-of-Useful-Work". In: *Advances in Cryptology – CRYPTO 2022*. Vol. 13508. LNCS. Springer, 2022, pp. 339–369. DOI: 10.1007/978-3-031-15979-4_12. Cryptology ePrint Archive, Paper 2021/1379.

[13] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *Advances in Cryptology – EUROCRYPT 2015*. Vol. 9057. LNCS. Springer, 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10. Cryptology ePrint Archive, Paper 2014/765.

[14] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the 28th Annual ACM Symposium on Theory of Computing – STOC '96*. ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[15] Harry Markowitz. "Portfolio Selection". In: *The Journal of Finance* 7.1 (1952), pp. 77–91. DOI: 10.2307/2975974.

[16] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[17] Diego Ongaro and John Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *Proceedings of the 2014 USENIX Annual Technical Conference – USENIX ATC '14*. USENIX Association, 2014, pp. 305–320. URL: https://dl.acm.org/doi/10.5555/2643634.2643666. Full text at https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf.

[18] Paradigm. *Reth: A modular, contributor-friendly Ethereum execution client written in Rust*. URL: https://github.com/paradigmxyz/reth.

[19] Peter W. Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172.

[20] Sigma Prime. *Lighthouse: An Ethereum consensus client written in Rust*. URL: https://github.com/sigp/lighthouse.

Glossary

block weight The work a single block contributes to its chain's cumulative total. In QMS it is determined by two protocol parameters: the hash difficulty at the time of production, and the work that one chunk of computation represents. 11, 27

Byzantine fault tolerance (BFT) A fault model in which a protocol's guarantees hold even when some participants behave arbitrarily, deviating from the protocol through faults, compromise, or deliberate attack rather than merely halting. Under *partial synchrony*, Byzantine agreement tolerates fewer than one-third of participants being dishonest, which is the optimal threshold. 13–16, 27

crash fault tolerance (CFT) A fault model in which a protocol's guarantees hold as long as each participant either follows the protocol or stops, by crashing or becoming unreachable, with no defense against participants that behave arbitrarily. Weaker than *BFT*, but adequate when the participants form a known, accountable set unlikely to act maliciously. 15, 16

difficulty adjustment algorithm The rule that periodically resets the difficulty target so that the expected block rate stays close to its intended value as mining effort joins or leaves the network. 10, 23

distributed denial-of-service (DDoS) An attack that overwhelms a target with traffic from many sources, preventing it from responding to legitimate requests. 8, 9, 14

Ethereum Virtual Machine (EVM) The runtime environment for smart contracts on Ethereum, and the execution model used by QMS for compatibility with existing tooling. 4, 6, 7, 9, 16, 18, 23, 24, 29

finalizer A participant in the finality layer that stakes the native token and signs votes to finalize blocks, marking them as irreversible. When a miner adds a finalization proof to a block, each finalizer whose signature is in that proof earns a reward proportional to their stake. 6–8, 13–20, 23–25

harvest-now-decrypt-later (HNDL) An attack in which an adversary captures ciphertext today with the intention of decrypting it once sufficiently powerful quantum hardware arrives. A concern for any protocol that relies on long-term confidentiality of encrypted data, including privacy chains that protect on-chain transaction data using elliptic-curve encryption or key exchange. 4

heaviest chain The chain with the most cumulative work, the sum of its *block weights*. Honest miners always extend the heaviest chain, making it the canonical chain above the last finalized block. 8, 11, 13, 28, 29

Ising Hamiltonian The energy function of a system of binary spins with pairwise interactions. Its ground state, the configuration that minimizes the energy, is mathematically equivalent to the optimal solution of a *QUBO* problem. 12, 29

liveness A security property of a consensus protocol: if all honest parties broadcast a transaction for sufficiently many rounds, every honest party will eventually report it in the chain at sufficient depth. For protocols with explicit finalization, liveness is amended to require the transaction to be finalized. 12, 14

miner A participant that secures the chain by running randomized solvers on client optimization problems and producing blocks under PoUW. Miners earn block rewards and transaction fees for the blocks they produce, and client payments for the solutions they find. 4–14, 16–21, 23, 24

mixed-integer quadratic program (MIQP) An optimization problem with a quadratic objective function and linear constraints in which some variables are continuous and others are restricted to integers. Cardinality-constrained mean-variance portfolio selection is an MIQP. 21, 22

moderate hardness A condition on a PoUW protocol's search problem: progress in the local-search algorithm requires computational work in expectation, with no available shortcut. Analogous to the puzzle-hardness assumption in standard PoW. 12, 16

NP-hard A class of problems at least as hard as any problem whose solutions can be verified in polynomial time. No polynomial-time algorithm is known for any NP-hard problem, and none is expected to exist. Many practical optimization problems, including QUBO, are NP-hard. 12, 21

partial synchrony A network timing model lying between full synchrony, where every message arrives within a known time bound, and full asynchrony, where delays are unbounded. Under partial synchrony a bound on message delay exists but is unknown, and holds only after some unknown point in time. It is the timing model under which the QMS chain is proven secure. 13, 27

persistence A security property of a consensus protocol: once an honest party reports a transaction at sufficient depth in the chain, every honest party will continue to report it at the same position. In *heaviest-chain* consensus this holds only probabilistically; protocols with explicit finalization replace it with *safety*, which is absolute. 11, 29

post-quantum (PQ) Cryptographic algorithms designed to remain secure against attacks from a sufficiently capable quantum computer. 4–9, 15–17, 23, 24, 30

precompile A function built into the EVM and callable from contracts at a fixed gas cost, used for operations too expensive to run as contract bytecode, such as signature verification. 17, 24

proof-of-stake (PoS) Consensus mechanism in which security comes from validators locking tokens as collateral. 4, 9

proof-of-useful-work (PoUW) Consensus mechanism in which miners produce blocks by solving computationally useful problems rather than performing pure hashing. 5–14, 16, 23, 24, 28

proof-of-work (PoW) Consensus mechanism in which security comes from expending physical computational effort. Used, in its hashing form, by Bitcoin. 4, 5, 8, 9, 11, 13, 18, 24

Q-Day Hypothetical point in time at which quantum computers become powerful enough to break currently deployed elliptic-curve cryptography in practice. 16, 17

quadratic unconstrained binary optimization (QUBO) A class of optimization problems over binary (0/1) variables with a quadratic objective function. Mathematically equivalent to finding the ground state of an *Ising Hamiltonian*. 6, 8, 12, 20, 22–24, 28

quantum annealing A quantum computing paradigm specialized for finding low-energy configurations of physical systems and, equivalently, for solving optimization problems expressible in QUBO form. 6, 12

quantum approximate optimization algorithm (QAOA) A hybrid quantum-classical algorithm designed for combinatorial optimization problems. 6, 12

safety A security property of a consensus protocol: honest parties never disagree on the finalized history, so a finalized transaction is never reverted. This contrasts with the probabilistic *persistence* of *heaviest-chain* consensus. 14, 28

Shor's algorithm A quantum algorithm that factors integers and computes discrete logarithms in polynomial time, breaking RSA and elliptic-curve cryptography. 4, 6, 9, 15, 17

smart account An account implemented as a smart contract, enabled by account abstraction, which is standardized on Ethereum as ERC-4337 and requires no change to the protocol. Its transaction-validation logic is programmable, so it can accept signatures of any scheme rather than

only the one the protocol verifies natively. Used by QMS to offer PQ-secure wallets. 17, 24

zero-knowledge (ZK) Property of a cryptographic proof of a statement that reveals nothing beyond the statement's truth. Used by QMS to allow the recovery of accounts frozen because they were not upgraded to PQ signatures. 4, 15, 17