Qbit: Consensus Specification

Consensus, Validation, and Protocol Surface

www.qbit.org

June 2, 2026

Contents

Abstract

Qbit is a Bitcoin Core v30.2-derived network [BCORE30] that relocates spend authorization onto a post-quantum footing. Its distinguishing property is not merely the use of a bounded SLH-DSA signature profile [FIPS205], but the way that cryptographic choice propagates through the entire protocol: the spendable output model, witness accounting, validation budget, block envelope, mining cadence, and archival model are all parameterized around large post-quantum witnesses. In that sense, Qbit is not a classical UTXO chain with a post-quantum patch; it is a system designed around the operational consequences of post-quantum authentication from the outset.

By removing classical discrete-log signatures from the public spend path on public networks, Qbit directly addresses the principal quantum vulnerability in conventional Bitcoin-like authorization models. Its security claim is therefore structural rather than cosmetic: the protocol does not merely prefer post-quantum keys at the wallet layer, it encodes post-quantum spend authorization into the consensus surface itself.

This document specifies Qbit at the protocol level. It focuses on consensus rules first, and then records the wallet, signing, and node-behavior surface where those behaviors are materially relevant to implementers and operators. It is a technical specification rather than a design whitepaper. The intended audience includes implementers, auditors, exchanges, custodians, wallet authors, mining infrastructure operators, and technically literate researchers.

Unless a section is explicitly marked non-consensus, the normative statements in Sections 3 through 8 describe the canonical protocol behavior of Qbit. Named public-network parameter sets are not normative in this pre-launch specification until a final launch-parameter freeze assigns their genesis, network identity, bootstrap, and checkpoint values.

1. Conformance and Terminology

The key words MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY are to be interpreted in their usual RFC 2119 and RFC 8174 sense [RFC2119] [RFC8174].

A conforming Qbit full node MUST enforce the consensus rules described in Sections 3 through 8.

A conforming wallet or signing implementation that claims P2MR compatibility SHOULD implement the non-consensus wallet and signing behavior in Section 9 or clearly document any deviations.

Qbit Core is the reference full-node implementation of the protocol. Where future discrepancies arise between explanatory prose and deployed network behavior, deployed consensus governs until an erratum or protocol upgrade resolves the difference.

This document uses the following terms:

- permissionless block: a Qbit block mined directly against Qbit without an AuxPoW payload
- AuxPoW block: a Qbit block whose proof of work is supplied through the merged-mining AuxPoW payload [MM]
- P2MR: Pay-to-Merkle-Root, Qbit's witness-v2 spendable output model
- Qbit bit: the smallest unit of QBT; Qbit bits are equivalent in denomination to Bitcoin satoshis (`1 QBT = 100,000,000 bits`), and the term is distinct from binary bit positions such as header signaling bits or compact target fields like `nBits`
- lane: one cadence track, either permissionless or AuxPoW
- public launch profile: the settled protocol profile intended for public deployment once concrete network identity, genesis, bootstrap, and checkpoint parameters are frozen
- local testing profile: a configurable implementation profile used for regression tests, compatibility tests, and private development networks

2. Reference Basis

This specification is implementation-grounded. Normative statements were derived from the reference full node and cross-checked across the main consensus, script, mining, wallet, and networking subsystems, especially:

- `src/consensus/consensus.h`
- `src/consensus/amount.h`
- `src/consensus/params.h`
- `src/kernel/chainparams.cpp`
- `src/primitives/pureheader.h`
- `src/pow.cpp`
- `src/validation.cpp`
- `src/script/script.h`
- `src/script/interpreter.cpp`
- `src/auxpow_validation.cpp`

Two practical consequences follow from that approach:

- this document intentionally preserves a clean split between consensus and non-consensus behavior
- parameter values in this document are taken from the current reference implementation
- behavior not explicitly replaced or narrowed here should be read as inherited from the Bitcoin Core v30.2 baseline [BCORE30]

3. Protocol Summary

Relative to Bitcoin Core v30.2 [BCORE30], Qbit differs in the following high-level ways:

- aggregate target cadence is approximately 60 seconds
- cadence mining uses two block classes with lane-local ASERT tracks [ASERT]
- public launch profile spendable outputs are constrained by restricted-output mode rather than allowing the full inherited set of Bitcoin spendable outputs
- public launch profile spend authorization removes reliance on ECDSA and Schnorr [BIP340] for ordinary spend paths; the live P2MR path uses witness version 2, script-path-only Merkle commitments, and `OP_CHECKSIGPQC`, while reserved future witness outputs remain consensus-valid for forward compatibility
- the live P2MR signature path is bounded SLH-DSA-SHA2-128s [FIPS205] with fixed-size 32-byte public keys and 3,680-byte signatures
- witness discount is removed by setting `WITNESS_SCALE_FACTOR = 1`
- monetary issuance uses a compound-floor subsidy schedule with a 210 QBT initial reward, 598/625 stepdown ratio, and 210,000,000 QBT `MAX_MONEY` cap
- witness compaction is supported, but it is an explicit operator choice through `-prunewitnesses`; full witness retention remains the default node mode

The architectural point is central. Qbit does not present post-quantum signing as an isolated cryptographic substitution. It constrains the spend surface, prices witness bytes directly, budgets validation explicitly, and narrows long-run storage assumptions so that the protocol remains coherent under large hash-based signatures. The result is a concrete demonstration that a Bitcoin-like UTXO ledger can move its public spend path away from discrete-log assumptions without hiding the cost model or weakening consensus clarity.

3.1 Principal Divergences from Bitcoin Core v30.2

The table below is the shortest useful map of the protocol fork surface.

Area	Bitcoin Core v30.2 [BCORE30]	Qbit	Why it matters
Target cadence	600-second blocks	60-second aggregate cadence	Faster block production changes retarget sensitivity, relay pressure, and miner timing assumptions
Difficulty adjustment	Epoch retargeting	Per-block ASERT [ASERT], lane-local under cadence	Difficulty responds continuously instead of waiting for epoch boundaries
Mining model	Single PoW block class	Permissionless plus AuxPoW block classes on one most-work chain [MM]	Native mining and merged mining contribute to the same chain without height-based fork choice
Spendable outputs	Broad inherited Bitcoin output set	Restricted-output mode centered on P2MR	The public spend surface is narrowed around the post-quantum authorization model
Spend authorization	ECDSA, Schnorr [BIP340], and inherited script forms	`OP_CHECKSIGPQC` inside P2MR script-path spends, with reserved future witness outputs kept consensus-valid	Ordinary public-network spends remove reliance on discrete-log signatures while preserving forward-compatible witness space
Witness accounting	`WITNESS_SCALE_FACTOR = 4`	`WITNESS_SCALE_FACTOR = 1`	Large PQC witnesses are priced directly rather than discounted
Block envelope	4,000,000 weight, 4,000,000-byte serialized cap	2,000,000 weight, 2,000,000-byte serialized cap	Resource limits are tuned to one-minute blocks and large signatures
Monetary policy	50 BTC initial subsidy with 210,000-block halvings	210 QBT initial subsidy with 43,200-block compound-floor stepdowns in the public launch profile	Issuance is parameterized around Qbit's 60-second cadence and 210,000,000 QBT money-range cap
Validation budget	Tapscript-style validation weighting only [BIP342]	Explicit P2MR validation budget with 3,730-unit live-sig cost	CPU exposure from post-quantum verification is bounded per input

Area	Bitcoin Core v30.2 [BCORE30]	Qbit	Why it matters
Historical witness model	Classical archival assumption	Full retention by default, optional witness compaction	Long-run storage is treated as a first-class protocol operations concern

3.2 How to Read This Specification Relative to Bitcoin Core v30.2

This document is intentionally delta-aware. It should not be read as claiming that every Qbit rule is novel; much of the inherited Bitcoin consensus machine remains intact. The point of the specification is instead to identify where Qbit deliberately changes that machine and why those changes cohere.

For implementers already familiar with Bitcoin Core v30.2 [BCORE30], the shortest correct reading rule is:

- if a rule is not explicitly replaced, tightened, or marked non-consensus in this document, treat it as inherited from the Bitcoin Core v30.2 baseline
- the principal consensus deltas are concentrated in four surfaces: cadence and difficulty adjustment, mining and header semantics, spend authorization and witness accounting, and long-horizon node operation under large post-quantum witnesses
- the security claim is therefore architectural rather than cosmetic: Qbit removes the dominant discrete-log dependency from the public spend path and then adjusts validation, resource pricing, and block construction so that the post-quantum path is the native path rather than an exceptional one

4. Consensus Parameter Set

Compared with Bitcoin Core v30.2 [BCORE30], this section changes the consensus envelope most visibly: the chain runs on a one-minute aggregate cadence, uses a smaller block envelope, removes witness discounting, and adopts a different issuance and retarget environment around those choices.

Parameters not restated here should be treated as inherited from the Bitcoin Core v30.2 baseline.

4.1 Block Timing and Cadence

Qbit treats the chain-wide target cadence as one block every approximately 60 seconds.

On cadence-active networks, the two lane targets are:

Lane	Target spacing
Permissionless	75 seconds
AuxPoW	300 seconds

These lane spacings define an exact 4:1 permissionless-to-AuxPoW ratio at a 60-second aggregate schedule.

In the public launch profile, cadence is active from genesis. In local testing profiles, cadence defaults to active and can be overridden by test configuration.

4.2 Block Size, Weight, and Sigops

Consensus block and transaction size parameters are:

Parameter	Value
Maximum serialized block size	2,000,000 bytes
Maximum block weight	2,000,000
Witness scale factor	1
Maximum block sigops cost	80,000
Minimum transaction weight	60
Minimum serializable transaction weight	10

Because `WITNESS_SCALE_FACTOR = 1`, a conforming implementation MUST treat block and transaction weight as byte-aligned rather than discounting witness bytes.

4.3 Coinbase Maturity and Monetary Policy

The current subsidy schedule is compound-floor rather than halving-based:

Parameter	Value
Initial subsidy	210 QBT
Slow start	none
Public launch-profile subsidy step interval	43,200 blocks
Local-test subsidy step interval	150 blocks
Subsidy stepdown numerator	598
Subsidy stepdown denominator	625
Subsidy stepdown ratio	598/625
Per-step reduction	4.32%
Coinbase maturity	1,000 blocks
`MAX_MONEY` consensus cap	210,000,000 QBT
Tested total subsidy emission	209,999,997.61876800 QBT
First zero-reward step index	480
Public launch-profile first zero-reward height	20,736,000

`MAX_MONEY` is a consensus-critical money-range sanity cap, not the exact emitted supply. The implemented subsidy sum is slightly below that cap because each step floors integer arithmetic.

`GetBlockSubsidy()` behaves as follows:

- return `0` for negative heights
- require `nSubsidyStepInterval > 0`
- require `nSubsidyStepdownNumerator >= 0`
- require `nSubsidyStepdownDenominator > 0`
- require `nSubsidyStepdownNumerator <= nSubsidyStepdownDenominator`
- require `MoneyRange(nSubsidyInitial)`
- initialize subsidy to `nSubsidyInitial`
- compute `step_count = height / nSubsidyStepInterval`
- repeat `step_count` times, stopping if subsidy reaches zero: `subsidy = floor(subsidy * nSubsidyStepdownNumerator / nSubsidyStepdownDenominator)`

In the public launch profile, `nSubsidyInitial = 210 * COIN`, `nSubsidyStepInterval = 43,200`, `nSubsidyStepdownNumerator = 598`, and `nSubsidyStepdownDenominator = 625`. The local testing profile uses the same initial subsidy and stepdown ratio with `nSubsidyStepInterval = 150` to keep step-boundary tests short.

4.4 Proof-of-Work Anchors and Launch Calibration

Qbit derives ASERT anchor state from genesis. The `ASERTAnchor` structure stores `nHeight`, `nBits`, `nBitsLegacy`, `nBitsAuxPow`, `nAuxPow`, and `nBlockTime`: respectively the anchor height, fallback single-track target, permissionless-lane target, AuxPoW-lane target, cumulative AuxPoW count at the anchor, and anchor timestamp. The genesis block header itself contains one `nBits` value. On cadence-active networks, post-genesis ASERT reads `ASERTAnchor.nBitsLegacy` for permissionless blocks and `ASERTAnchor.nBitsAuxPow` for AuxPoW blocks, while `ASERTAnchor.nBits` remains the single-track fallback.

The public launch profile uses a `60`-second aggregate spacing, `75`-second permissionless spacing, `300`-second AuxPoW spacing, and `7,200`-second ASERT halflife. Concrete genesis timestamps, nonces, header bits, lane anchor bits, and proof-of-work limits are launch parameters and are intentionally not assigned by this pre-launch specification.

Launch calibration uses two distinct models. The permissionless lane is calibrated from a reference QBT price and an external SHA-256 hashprice:

$$P_{QBT} = \frac{\mathrm{FDV}}{S_{QBT}}, \qquad V_p = R_p \times P_{QBT}, \qquad h_{hash} = \frac{\text{hashprice}}{10^{15} \times 86400}, \qquad D_p = \frac{V_p}{h_{hash}\, 2^{32}}$$

where `FDV` is the launch fully diluted value input, S_{QBT} is terminal supply, R_p is the permissionless block reward, and `hashprice` is the selected SHA-256 production-cost input. The final launch-calibration inputs are an FDV of `$100,000,000 USD`, S_{QBT} = 210,000,000, R_p = 210 QBT, and a placeholder hashprice of 32.56 USD/PH/day. These inputs imply a permissionless initial launch difficulty of approximately `61.8 billion`.

The AuxPoW lane is calibrated from an assumed share of global Bitcoin hashrate:

$$H_{aux} = H_{BTC,global} \times share_{aux}, \qquad D_{aux} = \frac{H_{aux}\, T_{aux}}{2^{32}}$$

where $share_{aux}$ is the selected merge-mining participation assumption, T_{aux} is the AuxPoW lane target spacing, and $H_{BTC,global}$ is the selected global Bitcoin hashrate assumption. The final launch-calibration inputs are $share_{aux} = 1\%$, $T_{aux} = 300$ s, and $H_{BTC,global} = 1{,}000$ EH/s.

These models are intentionally different: the permissionless lane is anchored to a reference QBT price and SHA-256 production cost, while the AuxPoW lane is anchored to an assumed merge-mined security share. The FDV input defines one reference QBT price and is not split across lanes. A higher difficulty implies a lower proof-of-work target. Each derived difficulty is converted into compact `nBits` form and clamped to the profile's final `powLimit` if necessary before being written into the corresponding anchor field. These launch-calibration figures are external assumptions used to derive the initial anchor bits; they are not forecasts or commitments about post-launch QBT valuation, SHA-256 hashprice, merge-mining participation, or Bitcoin hashrate.

5. Transaction and Script Rules

Compared with Bitcoin Core v30.2 [BCORE30], the most consequential transaction-layer change is that Qbit does not merely add another script template. It restructures the public spend path around P2MR and post-quantum script-path authorization, while sharply narrowing which spendable outputs are acceptable on public networks.

5.1 Restricted Output Mode

On networks where `fRestrictedOutputMode` is enabled, every transaction output accepted in a block, including coinbase outputs, MUST be one of:

- a P2MR output
- an `OP_RETURN` output
- a Pay-to-Anchor output
- a reserved future witness output if `nOuterReservedWitnessHeight` is active at that height

In the current implementation:

- the public launch profile enables restricted-output mode from genesis
- the local testing profile enables restricted-output mode by default and exposes `-p2mronly=0` as the explicit unrestricted opt-out for compatibility tests
- in the public launch profile, `nOuterReservedWitnessHeight = 0`, so reserved future witness outputs are consensus-allowed from genesis
- reserved future witness outputs remain non-standard in current policy

This narrowing of the consensus spend surface is security-relevant. It keeps ordinary spend authorization inside the protocol's post-quantum transaction model instead of leaving multiple inherited classical spend paths available by default.

5.2 P2MR Output Program

P2MR is encoded as a witness version 2 program with an exactly 32-byte witness program.

A canonical P2MR `scriptPubKey` is:

`OP_2 <32-byte merkle_root>`

`P2MRHeight` defaults to `0`, so P2MR rules are active from genesis in the public launch profile. Local testing profiles can override that activation height for tests.

5.3 P2MR Spend Structure

P2MR spends are script-path only. There is no key-path spend.

After optional annex stripping, a P2MR witness MUST provide:

- zero or more script arguments
- the leaf script
- the P2MR control block

The P2MR control block differs from Taproot [BIP341]:

- it omits the Taproot internal key entirely
- its size is `1 + 32*m` bytes for `m` Merkle path elements
- bit 0 of the control byte MUST be set

The executing leaf code is `control[0] & 0xfe`.

The live P2MR leaf code is `0xc0`. The reserved future P2MR leaf-code range is `0xc2`, `0xc4`, …, `0xfe`; those leaves remain consensus-valid until later activation assigns semantics to them. The current implementation names the reserved subranges as production script versions (`0xc2` through `0xde`), staged signature-system versions (`0xe0` through `0xee`), experimental or deployment-staging versions (`0xf0` through `0xfc`), and an extension-envelope leaf version (`0xfe`).

5.4 OP_CHECKSIGPQC

OP_CHECKSIGPQC uses opcode value 0xb3.

Inside P2MR execution it behaves like a signature-checking opcode with stack shape:

[sig] [pubkey] -> [result]

Outside P2MR execution, byte 0xb3 is invalid. It MUST fail in base, P2SH, witness-v0, and Tapscript execution [BIP342] rather than behaving as an inherited OP_NOP4 or other no-op upgrade hook.

In P2MR v1 execution:

- a 32-byte public key selects the current PQC public-key format
- the opcode computes the P2MR sighash using Schnorr-style digest construction [BIP340] adapted for SigVersion::P2MR
- it then verifies the supplied signature against the supplied public key using the current bounded SLH-DSA path [FIPS205] under live leaf code 0xc0

The launch-target P2MR v1 signature item is the raw 3,680-byte PQC signature plus an optional non-default sighash byte. It does not carry a witness-level signature-algorithm selector.

Future spend semantics require a future leaf code, witness version, opcode, or public-key version. They are not selected through an in-band selector inside the live v1 witness.

5.5 P2MR Opcode Surface

The Qbit-specific P2MR opcode surface covered here is:

Opcode	Scope	Stack shape	Purpose
OP_CHECKSIGPQC	P2MR only	[sig] [pubkey] -> [bool]	transaction-sighash PQC authorization
OP_CHECKTEMPLATEVERIFY	P2MR only	[template_hash] -> [template_hash]	transaction-template constraint
OP_CHECKDATASIGPQC	P2MR only	[sig] [msg_hash] [pubkey] -> [bool]	PQC signature over explicit data
OP_CHECKDATASIGADDPQC	P2MR only	[sig] [msg_hash] [num] [pubkey] -> [num + success]	threshold-style PQC data signatures

Within P2MR execution, OP_CHECKSIG and OP_CHECKSIGVERIFY MUST fail. The canonical single-key P2MR authorization form is <32-byte pqc pubkey> OP_CHECKSIGPQC; verify-style behavior is expressed as OP_CHECKSIGPQC OP_VERIFY; threshold forms use the P2MR OP_CHECKSIGADD / multi_a path.

The opcode value for OP_CHECKSIGPQC is 0xb3. New P2MR opcode assignments MUST be Qbit-specific and MUST NOT conflict with 0xb3. In particular, Qbit MUST NOT reuse Bitcoin BIP-119's proposed OP_NOP4 / 0xb3 assignment for OP_CHECKTEMPLATEVERIFY [BIP119].

OP_CHECKTEMPLATEVERIFY, OP_CHECKDATASIGPQC, and OP_CHECKDATASIGADDPQC are current P2MR v1 opcodes in the reference implementation. Their current opcode assignments are 0xbb, 0xbc, and 0xbd, respectively, and their activation follows the live P2MR v1 script rules rather than a separate deployment.

5.6 OP_CHECKTEMPLATEVERIFY

OP_CHECKTEMPLATEVERIFY constrains the spending transaction to a 32-byte template hash.

The P2MR stack effect is:

[template_hash] -> [template_hash]

Under the live semantics:

- the opcode is valid only during P2MR execution
- an empty stack fails consensus

- if the top stack item is exactly 32 bytes, the opcode succeeds only if the supplied hash equals the template hash computed for the spending transaction
- a matching 32-byte argument remains on the stack
- a non-32-byte argument is a consensus no-op, but is nonstandard when the discourage-OP-SUCCESS policy flag is active

Unless Qbit deliberately specifies a different construction, the template hash follows BIP-119-style default template semantics [BIP119] and commits to the transaction fields needed to bind the spend structure, including version, locktime, input count, current input index, sequence commitments, output count, outputs hash, and scriptSig commitments where applicable.

Scripts that combine template verification with later checks use `OP_DROP` when they want to discard the checked template hash before continuing. Standardness policy SHOULD reject inactive or non-standard template-verification uses before relay.

5.7 P2MR Data-Signature Opcodes

`OP_CHECKDATASIGPQC` verifies a post-quantum signature over an explicit 32-byte message hash rather than over the spending transaction sighash.

The P2MR stack effect is:

`[sig] [msg_hash] [pubkey] -> [bool]`

Under the live semantics:

- the opcode is valid only during P2MR execution
- `msg_hash` MUST be exactly 32 bytes
- `pubkey` MUST be exactly 32 bytes under the live P2MR public-key format
- an empty signature evaluates false without signature verification
- a non-empty signature MUST be exactly the live PQC signature size
- invalid non-empty signatures fail deterministically

`OP_CHECKDATASIGADDPQC` provides threshold-style accumulation.

The P2MR stack effect is:

`[sig] [msg_hash] [num] [pubkey] -> [num + success]`

A valid non-empty signature contributes `1`; an empty signature contributes `0`; an invalid non-empty signature fails.

Data-signature verification uses Qbit-specific domain separation:

`TaggedHash("QbitDataSigPQC", msg_hash)`

Each non-empty data-signature verification consumes the fixed `3,730` validation-budget charge before deeper validation, matching `OP_CHECKSIGPQC`.

5.8 Live Cryptographic Parameters

The current live P2MR cryptographic parameter set is:

Parameter	Value
Signature family	bounded SLH-DSA-SHA2-128s-bounded30 [FIPS205]
Public key size	32 bytes
Secret key size	64 bytes
Signature size	3,680 bytes
Per-key signature cap	`2^30`
Witness version	`2`
P2MR leaf code	`0xc0`

5.9 P2MR v1 Resource Limits

P2MR v1 enforces resource limits on the initial witness stack before script execution. After removing the optional annex, leaf script, and control block, the remaining initial stack arguments MUST satisfy:

Parameter	Value
Maximum initial stack items	`1,000`
Maximum initial stack item size	`16,384` bytes
Maximum aggregate initial stack bytes	`131,072` bytes

These limits are separate from the live PQC signature size. They allow larger template and attestation scripts while keeping the initial stack bounded. The implementation also reserves `MAX_P2MR_V1_CAT_RESULT_SIZE = 16,384` bytes for a future CAT-style result-size rule; `OP_CAT` is not active in P2MR v1.

Within P2MR v1, OP_SUCCESS-style opcodes cannot bypass the initial-stack resource checks because those checks happen before script execution. Reserved future P2MR leaf versions do not execute P2MR v1 and keep their own future resource model until activation defines one.

5.10 Validation Budget

For a P2MR input, initialize the validation budget to:

```
serialized witness stack size + 50
```

Then apply these rules to each live PQC signature-check opcode:

- a non-empty signature item consumes `3,730` budget before verification
- a present-but-empty signature item consumes `0` budget and evaluates as an unsuccessful signature check

If the remaining validation budget becomes negative, script execution MUST fail. Under the live `0xc0` leaf and 32-byte pubkey path, malformed or otherwise unsupported non-empty signatures still consume the full `3,730` budget before failing consensus.

6. Difficulty Adjustment and Chain Selection

Compared with Bitcoin Core v30.2 [BCORE30], Qbit replaces epoch retargeting with ASERT [ASERT] and then extends it under cadence so that the permissionless and AuxPoW lanes each follow their own target schedule while still competing inside one cumulative-work ordering.

6.1 ASERT

ASERT is the active difficulty-adjustment algorithm on ASERT-enabled networks [ASERT]. Qbit uses the `aserti3-2d` family with:

- per-block adjustment
- halflife `7,200` seconds
- genesis height and time as ASERT anchor coordinates, with anchor target bits supplied by the active profile

On non-cadence ASERT paths, the expected schedule is the usual:

```
target_spacing * height_diff
```

On cadence-active networks, the schedule is lane-local rather than chain-height-local.

6.2 Dual ASERT Under Cadence

When cadence is active, Qbit computes next-work requirements against the previous block of the same lane.

The implementation does this in two steps:

1. it walks back to the previous same-type block
2. it computes the effective same-lane height difference relative to the anchor

The lane-local reference target is selected from the ASERT anchor fields:

- for permissionless blocks, use `ASERTAnchor.nBitsLegacy`
- for AuxPoW blocks, use `ASERTAnchor.nBitsAuxPow`
- on single-track or fallback paths, use `ASERTAnchor.nBits`

The same-lane height rule is:

- for AuxPoW blocks, use the cumulative `nAuxPow` count since the anchor
- for permissionless blocks, use total height growth minus cumulative AuxPoW growth since the anchor

This means the permissionless and AuxPoW lanes each track their own schedule while still contributing to one accumulated-work chain. The genesis header remains single-valued at the header level, but cadence-active ASERT uses the lane-specific anchor bits to define the two post-genesis starting targets.

6.3 Timestamp Validity

A valid block timestamp MUST satisfy all of the following:

- it is strictly greater than the median time past of the previous eleven blocks
- it is not more than two hours in the future relative to local adjusted time
- on the inherited non-ASERT BIP94 paths, `MAX_TIMEWARP` remains `60` seconds [BIP94]

Min-difficulty behavior is keyed off the active target spacing. When min-difficulty is enabled for a profile, a block more than `2 * active_spacing` late may use the `powLimit` target.

6.4 Chain Selection

Qbit selects the valid chain with the greatest accumulated work.

Cadence does not introduce a height-based override. The presence of two block lanes does not change the most-work rule.

7. Mining, Header Encoding, and AuxPoW

Compared with Bitcoin Core v30.2 [BCORE30], Qbit introduces a materially different mining surface: canonical header encoding changes, AuxPoW [MM] is a native block class with explicit signaling and payload-validation rules, and block-class distinction feeds into lane-specific target selection before contributing to the same accumulated-work chain.

7.1 Block Classes

A valid Qbit block is one of:

- a permissionless block with no AuxPoW payload
- an AuxPoW block with a valid AuxPoW payload and AuxPoW-signaling version field

Permissionless blocks that carry an AuxPoW payload are invalid. AuxPoW-signaling headers that omit the AuxPoW payload are invalid.

7.2 Version Field Layout

The canonical Qbit block version layout is:

`[top_bits:3][chain_id:16][reserved:4][auxpow_flag:1][version_bits:8]`

In that layout:

Bits	Meaning
0-7	deployment signaling range
8	AuxPoW flag
9-12	reserved
13-28	chain ID
29-31	fixed BIP9-shaped top bits [BIP9]

Cadence header validation accepts either:

- the Qbit canonical BIP9-shaped layout [BIP9], or
- a zero-layout legacy-compatible version where all cadence layout bits are clear

Those three top bits are not the deployment namespace. Deployment signaling lives in bits `0-7`, giving Qbit an 8-bit signaling range inside the canonical layout.

Once cadence is active:

- canonical headers with BIP9 top bits MUST keep the reserved bits clear
- AuxPoW signaling headers MUST set the AuxPoW flag and encode the active chain ID
- permissionless headers MUST clear the AuxPoW flag
- when the AuxPoW flag is clear, the chain-ID field is not interpreted as an AuxPoW chain identity

7.3 AuxPoW Chain ID

The public launch profile MUST define a single AuxPoW chain ID before launch. This pre-launch specification does not assign that value.

Any AuxPoW header whose encoded chain ID does not match the active profile's configured chain ID MUST be rejected.

The active chain ID is consensus-critical and should be treated as immutable after launch. It is a coordination constant, not a protocol-level collision-proof namespace: avoiding clashes with other merged-mined chains depends on ecosystem coordination and pool software configuration.

For permissionless mining templates, the implementation exposes the chain-ID bit range as the BIP310-compatible version-rolling mask `0x1fffe000` [BIP310]. Pool software may intersect miner-requested masks with this value. Rolling those bits does not create an AuxPoW block as long as the AuxPoW flag remains clear and the reserved bits remain clear.

7.4 AuxPoW Payload Semantics

An AuxPoW payload contains [MM]:

- the parent coinbase transaction
- the parent coinbase Merkle branch
- the parent coinbase branch index
- the auxiliary chain Merkle branch
- the auxiliary chain index
- the parent block header

Validation rules include all of the following:

- the parent coinbase transaction MUST exist and MUST be a coinbase
- the parent coinbase branch index MUST be exactly `0`

- the auxiliary chain branch length MUST NOT exceed `30`
- the parent coinbase Merkle branch must reconstruct the parent block Merkle root
- the parent header hash MUST satisfy the Qbit child target when proof-of-work checking is enabled

The parent coinbase commitment rule is:

- if the merged-mining header `0xfa 0xbe 0x6d 0x6d` is present, it MUST appear exactly once and MUST be immediately followed by the committed auxiliary Merkle root
- if that header is absent, the committed auxiliary Merkle root MUST appear within the legacy first-20-byte commitment window of the parent `scriptSig`

After the committed chain root, the parent coinbase script must also contain:

- a 4-byte Merkle-tree size footer
- a 4-byte nonce footer

The chain index MUST match the slot derived from the nonce, the configured chain ID, and the auxiliary Merkle-branch height.

7.5 Mining Payout Policy

Restricted-output consensus allows P2MR, `OP_RETURN`, Pay-to-Anchor, and active reserved future witness outputs in coinbase transactions. On restricted-output profiles, address-based mining RPC payout paths are narrower: `createauxblock` and related payout-address construction require a P2MR payout destination. Pay-to-Anchor remains consensus-valid as an output form but is not accepted as a miner-controlled payout address.

8. Activation and Network Parameter Profiles

Compared with Bitcoin Core v30.2 [BCORE30], Qbit is conservative about inherited soft-fork behavior but not about activation posture. Much of the buried ruleset is carried forward, while SegWit [BIP141], Taproot [BIP341] [BIP342], cadence, and P2MR are treated as native assumptions of the protocol rather than optional late additions.

8.1 Buried and Always-Active Rule Sets

The public launch-profile activation posture is:

- BIP34 active from genesis [BIP34]
- BIP65 active from genesis [BIP65]
- BIP66 active from genesis [BIP66]
- CSV family active from genesis [BIP68] [BIP112] [BIP113]
- SegWit active from genesis [BIP141]
- Taproot is configured as `ALWAYS_ACTIVE` [BIP341] [BIP342]
- P2MR defaults to height `0`

Local testing profiles may expose shorter activation heights or explicit override hooks for compatibility tests.

8.2 Versionbits Windows and Thresholds

Concrete versionbits windows and thresholds are network parameters and are not assigned by this pre-launch specification. They MUST be frozen with each final public network parameter set.

The canonical Qbit signaling range is version bits `0` through `7`.

Taproot is configured as `ALWAYS_ACTIVE` [BIP341] [BIP342].

8.3 Network Identity

Message-start bytes, default ports, Bech32 HRPs, Base58 prefixes, challenge-network commitments, DNS seeds, and fixed seeds are network-identity or bootstrap parameters. They are intentionally not assigned by this pre-launch specification and MUST be frozen with each final public network parameter set.

Base58 prefixes may remain populated in the implementation, but they are not the public spendable output model under restricted-output Qbit profiles.

8.4 Effective Consensus Flags by Profile

Profile	ASERT	Cadence from genesis	Restricted outputs	Outer reserved witness active	P2MR height
Public launch	yes	yes	yes	yes	0
Local testing	configurable	default yes	default yes, opt-out	configurable	configurable

Min-difficulty behavior, BIP94 enforcement [BIP94], and challenge-network rules are profile-specific settings. They are not assigned for named public networks by this pre-launch specification.

8.5 Launch-Parameter Freeze Items

This pre-launch specification intentionally omits concrete values for reset-bound public-network parameters, including:

- genesis `nTime`, `nNonce`, `nBits`, Merkle root, and block hash
- `powLimit` and concrete ASERT anchor bits
- message-start bytes, default ports, address HRPs, Base58 prefixes, and challenge-network commitments
- AuxPoW chain ID if not already frozen before final launch parameterization
- versionbits windows and thresholds
- checkpoints, `nMinimumChainWork`, and `defaultAssumeValid`
- DNS seeds, fixed seeds, and other bootstrap assumptions

Those values MUST be assigned by a final launch-parameter freeze before a public network is treated as launched. Once frozen, full asserted hashes and network parameter constants belong in the reference implementation's chain-parameter source.

9. Wallet and Signing Surface (Non-Consensus)

Compared with Bitcoin Core v30.2 [BCORE30], the wallet surface is materially re-centered around P2MR. Classical address and descriptor machinery still exists where inherited code paths require it, but the visible default user path on public networks is a post-quantum script-path spend model rather than a menu of classical spend types.

This section is non-consensus. It records the wallet and signing behavior exposed by the reference implementation because integrators will need it even though consensus does not require wallet interoperability.

9.1 Addresses

P2MR addresses use witness version 2 with Bech32m encoding [BIP350]. Concrete address HRPs are network-identity parameters and are not assigned by this pre-launch specification. For any final HRP, the visible P2MR address form is `<hrp>1z...`.

9.2 Descriptors

The canonical P2MR descriptor family is `mr(...)`.

The raw-root form is `rawmr(...)`.

Representative current forms are:

- `mr(pk(PUBKEY))`
- `mr({pk(PUBKEY1),pk(PUBKEY2)})`
- `mr(multi_a(2,PUBKEY1,PUBKEY2,PUBKEY3))`
- `rawmr(HEX_MERKLE_ROOT)`

Unlike Taproot `tr(...)` [BIP341], `mr(...)` does not imply or require an internal key.

9.3 Current Key Derivation Surface

Current wallet derivation uses `DerivePQCKey()` in `src/crypto/pqc.cpp`.

The implementation derives PQC keys from master seed material with:

- HKDF-SHA256-style extract and expand logic
- salt `qbit-sphincs-v1`
- info prefix `qbit/sphincs+/1`
- little-endian `(account, change, index)` concatenated after that prefix

Current wallet-generated P2MR descriptors use path family `.../87h/...` and, in the default wallet descriptor generation path, pin `account = 0`.

Imported exact-size PQC secret material is validated for internal consistency before being accepted into wallet key state. Signing paths also reject inconsistent SLH-DSA secret keys [FIPS205] before attempting to produce a signature. These checks affect wallet import and signing failure behavior; they do not change consensus verification of already-formed signatures.

9.4 Default Wallet Behavior

Under the public launch profile and the default local testing profile:

- the wallet output-type set is effectively P2MR-only
- fresh descriptor-wallet generation uses P2MR descriptors
- the default address type becomes P2MR when a P2MR script-pubkey manager is present

Local testing can still opt out of that launch-like behavior with `-p2mronly=0`, which re-enables unrestricted legacy descriptor and output-type coverage for compatibility tests.

Wallet signing reserves PQC per-key counters before slow signature generation and may perform the expensive signing operation outside the wallet lock using signing-provider snapshots. This preserves counter uniqueness for concurrent signing without changing the consensus signature format.

Wallet send and funding RPCs reject reserved future witness destinations before returning a transaction, PSBT, or txid. Manually constructed transactions can still contain those outputs, but current policy rejects them from standard relay.

9.5 Watch-Only and Pubkey Database RPCs

The current wallet RPC surface for explicit watch-only P2MR workflows includes:

- `exportpubkeydb`
- `importpubkeydb`
- `getnextpubkeydbaddress`
- `listpubkeydbstatus`

The present implementation also explicitly rejects importing public P2MR `pqc(...)` descriptors when the import path lacks the private key material needed to derive the necessary PQC public keys.

On wallet-signing and address-inspection RPC surfaces, the current implementation also reports PQC usage state for local keys. The exposed state fields include `pqc_key_states`, `pqc_signature_count`, `pqc_signature_limit`, `pqc_signatures_remaining`, `pqc_limit_state`, and `pqc_overall_limit_state`. Wallet-signing and funding responses also include human-readable `warnings` where applicable; address-inspection responses omit those warning strings.

9.6 PSBT

Qbit extends PSBT [BIP174] for current P2MR script-path spends. The live encoding is not proprietary-only. The current implementation serializes two dedicated PSBT input types and one proprietary field in the `qbit` namespace:

Wire key	Meaning
`0x19` (`PSBT_IN_P2MR_SCRIPT_SIG`)	P2MR script-path signature, keyed by `pubkey || leaf_hash`
`0x1d` (`PSBT_IN_P2MR_LEAF_SCRIPT`)	P2MR leaf script, keyed by control block; value is `script || leaf_ver`
`qbit:0x01`	P2MR merkle root

The proprietary namespace identifier is `qbit`.

For compatibility, the parser also accepts `qbit:0x02` for P2MR leaf scripts and `qbit:0x03` for P2MR script-path signatures, but current serialization writes the standard `0x19` and `0x1d` encodings instead.

`decodepsbt` exposes these P2MR fields as `p2mr_script_path_sigs`, `p2mr_scripts`, and `p2mr_merkle_root`.

PSBT signing and finalization currently target the live P2MR spend surface: witness-v2 P2MR outputs, leaf code `0xc0`, and the current 32-byte PQC public-key format. The signing path auto-populates the P2MR merkle root from the prevout when needed and preserves P2MR `SIGHASH_DEFAULT` semantics rather than aliasing `SIGHASH_DEFAULT` to `SIGHASH_ALL`.

Current P2MR PSBT handling normalizes cached script-path signatures by `(pubkey, leaf_hash)`. Conflicting duplicate standard/proprietary script-signature encodings are rejected during parsing, and cached P2MR script signatures are verified before being treated as usable signing candidates. Leaf scripts accepted through the standard and compatibility encodings are merged by script, leaf version, and control block rather than treated as conflicting duplicates.

This differs from Bitcoin Core v30.2 [BCORE30], which does not define P2MR PSBT input types, does not define the `qbit` P2MR proprietary namespace, does not expose `p2mr_*` fields through `decodepsbt`, and does not carry a P2MR-specific PSBT signing/finalization path.

Reserved future leaf codes or unknown nonzero P2MR pubkey sizes may still appear in explicit custom workflows, but ordinary wallet signing/finalization does not synthesize witnesses for them.

10. Node Behavior and Policy Defaults (Non-Consensus)

Most node behavior remains recognizably Bitcoin-like. The important departures from Bitcoin Core v30.2 [BCORE30] are concentrated in witness retention, bootstrap assumptions, and policy values shaped by the byte cost of large post-quantum witnesses.

10.1 Witness Retention and Archive Behavior

Witness compaction is supported, but the default node mode retains full witness history.

Current behavior is:

- `-prunewitnesses=1` enables witness pruning beyond the witness-prune depth
- the witness-prune depth defaults to `COINBASE_MATURITY`, which is `1,000`

- archive-capable nodes advertise `NODE_ARCHIVE`
- once witness compaction has actually completed, the node advertises `NODE_WITNESS_PRUNED`
- once witness compaction has actually completed, the node withdraws `NODE_ARCHIVE`
- a historical `MSG_WITNESS_BLOCK` request for a witness-pruned block returns `NOTFOUND`

The reference implementation uses archive behavior by default and exposes witness pruning as an explicit operator choice via `-prunewitnesses`.

10.2 Reindex and Bootstrap Constraints

For a witness-pruned node, the supported in-place rebuild path is constrained.

Current implementation behavior includes:

- `-reindex-chainstate` on a witness-pruned node requires a non-null assumed-valid block, either from explicit `-assumevalid=<hash>` or from the active chain parameters
- without any assumed-valid block, the node fatally rejects `-reindex-chainstate`
- the assumed-valid hash should be above the witness-pruned range; otherwise validation may need temporary witness recovery and archive-peer availability
- `-reindex-chainstate` remains incompatible with `-prune`
- explicit `-prunewitnesses=1` is incompatible with `-txindex`
- a datadir that has already pruned witnesses cannot later start with `-txindex`
- `-connectarchive` requests peers that advertise `NODE_NETWORK`, `NODE_WITNESS`, and `NODE_ARCHIVE`, are not limited to `NODE_NETWORK_LIMITED` block service, and do not advertise or later imply `NODE_WITNESS_PRUNED`

10.3 Archive Peer Visibility RPC

The current network RPC surface includes `getarchivepeers` for operator visibility into archive-relevant peer state. This RPC is observational only: it does not perform DNS lookups, probes, connection attempts, or peer-selection changes.

Its supported views are:

- `all`
- `summary`
- `connected`
- `configured`

The response distinguishes currently connected peers that advertise `NODE_ARCHIVE`, peers opened through `-connectarchive`, configured `-connectarchive` targets, and configured archive targets that are currently connected. Observed `NODE_ARCHIVE` service bits and configured `-connectarchive` targets are reported as facts, not as proof that a peer can serve every historical block and witness.

10.4 Relay Defaults

Current relay timing and compact-block settings are:

Parameter	Value
Compact-block direct-fetch depth	10
`getblocktxn` depth	20
Inbound inventory broadcast average interval	1 second
Outbound inventory broadcast average interval	1 second
Headers response timeout	30 seconds
Fee-filter rebroadcast average interval	2 minutes

Inventory and fee-filter rebroadcast timers are randomized around the stated averages rather than firing at exact fixed intervals. The headers response timeout is a fixed `30`-second timeout.

10.5 Fee Policy Defaults

Current default fee-policy values are:

Parameter	Value
Default minimum relay fee	250 bits/kvB
Dust relay fee	750 bits/kvB
Default incremental relay fee	250 bits/kvB
Default block minimum transaction fee	1 bit/kvB
Maximum standard transaction weight	400,000
Maximum package weight	404,000
Ancestor count limit	25
Ancestor size limit	404 kvB
Descendant count limit	25
Descendant size limit	404 kvB
Reduced `-blocksonly` mempool size floor	17 MB
Maximum standard P2MR stack item size	16,384 bytes
Maximum standard P2MR initial stack bytes	131,072 bytes
TRUC maximum transaction vsize	50,000 vbytes
TRUC child maximum vsize	45,500 vbytes
Maximum `OP_RETURN` relay size	200,000 bytes

Reserved future witness outputs remain non-standard in current policy even where consensus would accept them.

P2MR dust, dummy-input sizing, PSBT size estimation, and wallet input-weight floors use P2MR-specific spend-size estimates rather than inherited P2WPKH or Taproot-sized placeholders. Caller-provided `input_weights` cannot understate a known P2MR prevout below the P2MR spend-size floor.

11. Specification Notes and Open Items

This section highlights implementation-snapshot caveats. It should be read narrowly: it does not weaken the consensus statements above, but it does identify where launch-state parameter freeze, active launch-target PRs, or later editorial cleanup would improve the document.

11.1 Parameter Precedence

This specification uses the current reference-implementation values for:

- full witness retention is the default; witness pruning is opt-in through `-prunewitnesses`
- cadence spacing is `75 / 300` seconds
- the implemented opcode value for `OP_CHECKSIGPQC` is `0xb3`

11.2 Provisional Public-Network Parameters

The following public-network values remain outside this pre-launch specification and require final launch signoff:

- genesis blocks and regenerated Merkle roots
- `powLimit` values and concrete ASERT anchor bits
- message-start bytes, ports, address HRPs, and challenge-network commitments
- AuxPoW chain ID if not separately frozen
- versionbits windows and thresholds
- `defaultAssumeValid`, `nMinimumChainWork`, and synchronization checkpoints

- DNS seeds, fixed seeds, and other bootstrap assumptions

Those items do not prevent this specification from describing the intended public launch profile, but a named public network should not be treated as launched until they are frozen.

11.3 P2MR Signature-Item and Resource-Limit Alignment

The live P2MR v1 signature item remains the 3,680-byte PQC signature plus an optional non-default sighash byte. That produces a maximum ordinary signature item of 3,681 bytes.

That signature-item size is no longer the P2MR stack-item resource ceiling. Current P2MR v1 resource rules allow initial stack items up to 16 KiB and cap aggregate initial stack bytes at 128 KiB so template and attestation scripts can fit without changing the live signature format or validation-budget charge.

Normative References

- [RFC2119] RFC Editor, RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels," March 1997, accessed June 2, 2026. RFC 2119
- [RFC8174] RFC Editor, RFC 8174, "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," May 2017, accessed June 2, 2026. RFC 8174
- [BCORE30] Bitcoin Core, "Bitcoin Core v30.2 source tree," accessed June 2, 2026. Bitcoin Core v30.2 source tree
- [FIPS205] NIST, FIPS 205, "Stateless Hash-Based Digital Signature Standard," August 13, 2024, accessed June 2, 2026. FIPS 205 DOI
- [BIP9] Bitcoin Improvement Proposal 9, "Version bits with timeout and delay," accessed June 2, 2026. BIP-9
- [BIP34] Bitcoin Improvement Proposal 34, "Block v2, Height in Coinbase," accessed June 2, 2026. BIP-34
- [BIP65] Bitcoin Improvement Proposal 65, "OP_CHECKLOCKTIMEVERIFY," accessed June 2, 2026. BIP-65
- [BIP66] Bitcoin Improvement Proposal 66, "Strict DER signatures," accessed June 2, 2026. BIP-66
- [BIP68] Bitcoin Improvement Proposal 68, "Relative lock-time using consensus-enforced sequence numbers," accessed June 2, 2026. BIP-68
- [BIP112] Bitcoin Improvement Proposal 112, "CHECKSEQUENCEVERIFY," accessed June 2, 2026. BIP-112
- [BIP113] Bitcoin Improvement Proposal 113, "Median time-past as endpoint for lock-time calculations," accessed June 2, 2026. BIP-113
- [BIP119] Bitcoin Improvement Proposal 119, "CHECKTEMPLATEVERIFY," accessed June 2, 2026. BIP-119
- [BIP141] Bitcoin Improvement Proposal 141, "Segregated Witness (Consensus layer)," accessed June 2, 2026. BIP-141
- [BIP174] Bitcoin Improvement Proposal 174, "Partially Signed Bitcoin Transaction Format," accessed June 2, 2026. BIP-174
- [BIP310] Bitcoin Improvement Proposal 310, "Stratum protocol extensions," accessed June 2, 2026. BIP-310
- [BIP340] Bitcoin Improvement Proposal 340, "Schnorr Signatures for secp256k1," accessed June 2, 2026. BIP-340
- [BIP341] Bitcoin Improvement Proposal 341, "Taproot: SegWit version 1 spending rules," accessed June 2, 2026. BIP-341
- [BIP342] Bitcoin Improvement Proposal 342, "Validation of Taproot Scripts," accessed June 2, 2026. BIP-342
- [BIP350] Bitcoin Improvement Proposal 350, "Bech32m format for v1+ witness addresses," accessed June 2, 2026. BIP-350

- [BIP94] Bitcoin Improvement Proposal 94, "Testnet 4," accessed June 2, 2026. BIP-94
- [ASERT] Bitcoin Cash Node, "2020-NOV-15 ASERT Difficulty Adjustment Algorithm (aserti3-2d)," version 0.6, August 12, 2020, accessed June 2, 2026. Bitcoin Cash ASERT upgrade specification
- [MM] Bitcoin Wiki, "Merged mining specification," accessed June 2, 2026. Merged mining specification

Appendix A. Implementation Reference Map

Topic	Primary implementation files
Consensus constants	`src/consensus/consensus.h`, `src/consensus/params.h`, `src/consensus/amount.h`
Per-network parameters and genesis	`src/kernel/chainparams.cpp`
Block version layout	`src/primitives/pureheader.h`
Mining template and payout RPC behavior	`src/rpc/mining.cpp`
ASERT and cadence work calculation	`src/pow.cpp`
Subsidy calculation and restricted-output enforcement	`src/validation.cpp`
P2MR opcodes and constants	`src/script/script.h`
P2MR execution and validation budget	`src/script/interpreter.cpp`
P2MR spend sizing and policy limits	`src/script/p2mr_sizing.h`, `src/policy/policy.h`, `src/policy/truc_policy.h`
AuxPoW payload validation	`src/auxpow_validation.cpp`, `src/auxpow.cpp`
Wallet output-type defaults	`src/wallet/walletutil.cpp`, `src/wallet/wallet.cpp`
PQC key derivation	`src/crypto/pqc.cpp`
PSBT extensions	`src/psbt.h`, `src/psbt.cpp`, `src/node/psbt.h`, `src/node/psbt.cpp`
Node retention and bootstrap behavior	`src/init.cpp`, `src/node/chainstate.cpp`, `src/validation.cpp`, `src/net_processing.cpp`, `src/node/blockstorage.cpp`, `src/rpc/net.cpp`

This appendix is included so later revisions can replace prose assertions with even tighter file-level or line-level traceability as the specification matures.

Appendix B. Enumerated Consensus Deviations from Bitcoin Core v30.2

This appendix restates the highest-value consensus deltas in one place. Items not listed here should be treated as inherited from Bitcoin Core v30.2 [BCORE30] unless another section narrows them.

- Timing and difficulty: a 600-second single-lane schedule with epoch retargeting becomes an approximately 60-second aggregate cadence with lane-local ASERT adjustment.
- Block resource accounting: a 4,000,000-weight, witness-discounted block model becomes a 2,000,000-byte-aligned model with `WITNESS_SCALE_FACTOR = 1`.
- Issuance and maturity: launch subsidy becomes `210 QBT`, coinbase maturity becomes `1,000`

blocks, and Bitcoin-style halvings are replaced by a compound-floor schedule with a `43,200`-block public launch-profile step interval, `150`-block local-test step interval, and `598/625` stepdown ratio under a `210,000,000 QBT` `MAX_MONEY` cap.

- Spendable outputs: broad inherited spendability is narrowed by restricted-output mode in the public launch profile.
- Spend authorization: classical public spend paths give way to P2MR witness-v2 script-path spends validated through `OP_CHECKSIGPQC`.
- P2MR opcode surface: Qbit scopes post-quantum authorization, template verification, and data-signature opcodes to P2MR, while legacy `OP_CHECKSIG` and `OP_CHECKSIGVERIFY` are invalid inside P2MR.
- Live cryptographic profile: the active public spend path uses bounded SLH-DSA-SHA2-128s with 32-byte public keys and 3,680-byte signatures.
- Script resource bounds: P2MR execution introduces an explicit per-input validation budget and P2MR v1 initial-stack limits to cap post-quantum verification and witness-resource exposure.
- Mining model: a single native proof-of-work block class becomes a two-lane permissionless-plus-AuxPoW system that still resolves by cumulative work.
- Header semantics: canonical header encoding now carries an AuxPoW flag, chain ID field, reserved field, and 8-bit signaling range; permissionless templates may expose the chain-ID field as a version-rolling mask while the AuxPoW flag remains clear.
- AuxPoW identity: merged-mined blocks are consensus-bound to the active profile's configured chain ID.