Paybyte Blockchain Engine (PBE)

Architecture, Consensus, Cryptography, Networking, Commerce, Security, and Operational Analysis

A source-level research paper on PBE Core 0.0.1

Release identity	Value
Core	0.0.1
Peer protocol	1.0
Chain	PBE-MAINNET / Network ID 1
Block / transaction	Block v4 / Transaction v2
Commerce protocol	PBE-PAY v1
Target block slot	15 seconds
Document date	September 2026

Source-handling note This paper was prepared from the supplied PBE source tree using only public protocol material and bundled example/default configurations. Local configuration files, credentials, environment-specific secrets, private keys, runtime bindings, lock files, logs, and raw secret values are intentionally excluded and are not reproduced anywhere in this document.

Technical analysis, not an independent formal security audit or guarantee of fitness for financial settlement.

Abstract

Paybyte Blockchain Engine (PBE) 0.0.1 is a compact Layer-1 blockchain reference implementation written primarily in PHP and designed around a 15-second account-based chain, Ed25519 signatures, SHA-256 commitments, Bech32m addresses, a deterministic memory-hard participation mechanism, peer-to-peer synchronization, and an application-facing payment layer called PBE-PAY. The architecture is deliberately small enough to be auditable at the source level while still containing the core systems expected of an independent blockchain: canonical serialization, signed transactions, block validation, state commitments, emission, peer identity, discovery, chain reorganization, mempool management, wallet key isolation, merchant payment verification, and optional signed webhooks.

The most distinctive element is the hybrid participation consensus design. Every participating wallet can derive one deterministic Argon2id ticket for a particular parent block and slot, including wallets with zero PBE. Confirmed holdings do not create extra tickets; instead, they reduce the eligibility delay through a bounded non-linear multiplier. A separate machine producer key signs the block, while the wallet cryptographically authorizes that exact producer key and remains the reward identity. Equal-height fork selection is tied primarily to the memory-hard wallet ticket rather than the machine producer key, limiting a class of cheap key-rotation grinding.

The analysis finds a coherent separation between consensus state and operational services. Consensus state consists of account balances, nonces, and total supply; optional merchant webhooks live outside consensus. PBE-PAY embeds a signed payment reference in an otherwise normal transfer and verifies payments by an exact tuple of reference, recipient, and amount. The browser wallet signs locally and encrypts private key material client-side. P2P peer hints are treated as untrusted until a signed hello is independently verified, and outbound DNS targets are pinned and filtered to reduce server-side request forgery and DNS-rebinding risk.

The paper also identifies design boundaries. PBE 0.0.1 does not provide hard finality, cumulative-work chain selection, checkpoints, or a finality gadget. The participation proof is deterministic and historically reproducible, so accelerated alternate-history construction and long-partition reorganization remain material research risks. Zero-balance participation removes a capital gate and makes identity multiplication inexpensive at the key-generation layer; practical Sybil resistance therefore derives primarily from the per-wallet Argon2id resource cost rather than scarce identity or locked stake. These properties do not invalidate the design, but they define the security questions that should dominate future protocol review.

Keywords: Paybyte, PBE, blockchain, Argon2id, Ed25519, Bech32m, account model, peer-to-peer networking, payment protocol, webhooks, canonical-chain depth, PHP blockchain.

Implementation references: docs/MAINTAINER-REFERENCE.md; src/PBE/Core/; src/PBE/Node/*; src/PBE/Network/*; src/PBE/Storage/*; src/PBE/Commerce/**

Executive Summary

PBE Core 0.0.1 is best understood as a purpose-built payment blockchain rather than a generic smart-contract platform. The consensus layer accepts one transaction type - a signed account-to-account transfer - and the merchant layer builds on that primitive without creating a second monetary system. This keeps the protocol surface small: the chain validates transfer authorization, nonce ordering, fees, balance sufficiency, block production authorization, participation eligibility, state roots, transaction roots, and supply issuance; merchant semantics remain application metadata.

At the wire-format level, PBE is strongly domain separated. Transaction signing hashes, block signing hashes, node hello signatures, state leaves, Merkle leaves/nodes, participation challenges, participation salts, and mining authorization all carry explicit PBE-specific prefixes or canonical binary encodings. This does not

replace cryptographic analysis, but it is a sound engineering pattern because it reduces ambiguity between structurally different signed or hashed objects.

The reference implementation uses an account/balance/nonce state model. A transfer increments the sender nonce exactly by one, debits amount plus fee, credits the recipient amount, and routes transaction fees to the block reward address. Block emission increases total supply; transfer fees do not. State commitment is deterministic: accounts are sorted by address, each account leaf commits to the address, balance, and nonce, and a separate leaf commits to total supply. Both state and transaction commitments use a prefixed SHA-256 Merkle construction.

Hybrid participation uses a 32-byte Argon2id proof with an operation limit of 4 and a memory limit of 64 MiB. The first four proof bytes form an unsigned score. Confirmed whole-PBE holdings map through a piecewise-linear schedule from 1.00x at zero PBE to a nominal protocol cap of 2.50x. The adjusted score maps into an eligibility delay between one and fourteen seconds of a fifteen-second slot. A zero-balance wallet therefore has a real chance to produce; holdings improve timing but are not required. Because total eventual issuance is approximately 84.096 million PBE, the published 100 million PBE schedule point cannot be reached by a single wallet from legitimate mainnet supply alone; the maximum achievable multiplier under the current emission ceiling is approximately 2.4558x, assuming one account controls the entire final supply.

P2P networking combines an initially configured seed with persistent peer gossip. The seed is not a permanent trust anchor: peers must present a signed hello containing protocol version, network ID, chain ID, frozen genesis hash, node identity, current canonical tip, timestamp, and advertised endpoint. Compatibility and signature checks precede trust. Sync decisions use the signed hello tip rather than unsigned convenience status. Direct extensions use a fast incremental path; divergent branches use a short-tail probe followed by binary common-ancestor search, then atomic suffix reorganization with state undo data. If incremental reorganization fails, the implementation can fall back to a full candidate-chain rebuild.

PBE-PAY is intentionally not a custodial payment processor. A merchant generates a request containing an exact address, amount, payment reference, label, and confirmation policy. The wallet renders the request and signs a normal transaction locally. The merchant or node verifies an exact reference-recipient-amount tuple against mempool or canonical-chain indexes. Optional webhooks add bearer-protected subscription management and HMAC-SHA256 authenticated deliveries, including reorganization events. The webhook subsystem applies HTTPS-by-default callback restrictions, DNS validation/pinning, no redirects, bounded timeouts, at-least-once delivery, and retry backoff.

The central security caveat is finality. Canonical-chain depth is a probability/risk policy, not a protocol-level irreversible state. A participant capable of recomputing deterministic historical tickets can attempt to build an alternate branch outside real-time slot progression, subject to validation timestamps and the cost of participation proofs. The implementation itself acknowledges the need for a future checkpoint/finality policy and deeper adversarial review. Accordingly, the strongest technical interpretation of PBE 0.0.1 is a coherent experimental mainnet implementation with a broad end-to-end payment stack, but not yet a finality-hardened settlement system for high-value irreversible transfers.

Contents at a Glance

1. Research Scope and Methodology

This paper is a source-level technical study of the supplied PBE Core 0.0.1 project. The primary evidence base is the implementation itself: consensus classes, transaction and block serialization, state transition logic, peer protocol, synchronization logic, storage schemas, browser-wallet cryptography, Commerce/PBE-PAY code, public OpenAPI description, and maintainer documentation. The objective is not to restate marketing copy, but to reconstruct the protocol as the software enforces it and to distinguish consensus-critical behavior from UI, deployment, and operational policy.

The analysis treats code-level validation paths as authoritative where they are more specific than prose. For example, the block validator defines the exact timestamp tolerance, the transaction class defines the exact serialization sequence and permitted payment-reference alphabet, and the peer handshake defines the signed fields and clock-skew limit. Public documentation is used to explain intent, while implementation paths are used to verify behavior.

Configuration analysis is deliberately constrained to the bundled example/default configuration files. No local production configuration is reproduced. The example configurations show the supported deployment patterns - a web/MySQL node, a primary web node without an upstream seed, and a standalone/SQLite node with separate localhost UI/RPC and public P2P ports. Password placeholders remain placeholders, optional merchant secrets remain blank, and webhook functionality is disabled by default. Runtime keys, operator bindings, setup tokens, or other local material are outside the scope of this document.

External literature is used only to situate primitive choices. Ed25519 is described by RFC 8032; Argon2 by RFC 9106; Bech32m by BIP 350; SHA-256 by NIST FIPS 180-4; and HMAC by RFC 2104. Bitcoin and Ouroboros are cited as historical reference points for longest-chain proof-of-work and proof-of-stake research respectively. These citations do not imply protocol equivalence; PBE uses its own chain-selection and participation rules.

> **Interpretation rule** When this paper says "protocol" it refers to behavior that affects block/transaction acceptance, chain selection, state transition, cryptographic identity, or frozen mainnet identity. UI defaults and merchant services are described separately unless they directly constrain consensus.

Implementation references: VERSION; docs/MAINTAINER-REFERENCE.md; config/.example.php; web/developers/openapi.json; tests/selftest.php*

2. Protocol Identity and Design Goals

PBE Core 0.0.1 fixes a narrow protocol identity: software release 0.0.1, peer protocol 1.0, chain ID PBE-MAINNET, network ID 1, Bech32m human-readable prefix pbe, block format v4, transaction format v2, and PBE-PAY v1. The genesis timestamp is frozen at Unix time 1789171200, the genesis account set is empty, and the premine is zero. Genesis validation recomputes the genesis state and block hash and compares it to a constant embedded in code, so editing the JSON genesis file alone is insufficient to create a node that silently joins the same network identity.

The protocol is intentionally non-backward-compatible with pre-release generations. Transaction::fromArray rejects versions other than v2 and types other than transfer. Block::fromArray rejects block versions other than v4. ChainStore migration checks refuse older database shapes and require a fresh release database. This is an important design choice: 0.0.1 is treated as the first public chain, not as an upgrade that reinterprets earlier test data.

PBE chooses a small ledger feature set. There is no contract VM, no validator registry, no staking ledger, no delegation record, no staking lock, and no register/exit/withdraw transaction family. Consensus participation derives from wallet identity, a memory-hard ticket, and a balance-based timing multiplier. That simplicity

makes the chain easier to reason about, but it also means features such as conditional payments, token issuance, programmable custody, or protocol-governance transactions would require future protocol extensions rather than merely application code.

The architecture has a clear payment orientation. A normal transfer can carry a signed paymentReference field. PBE-PAY builds merchant workflow around that field without creating a second transaction type. This is technically significant because merchant identity data remains inside the signed transaction payload and therefore follows the same signature, mempool, block, reorganization, and state-validation paths as ordinary value transfer.

Protocol invariant	PBE 0.0.1 value
Chain ID	PBE-MAINNET
Network ID	1
Address prefix	pbe
Peer protocol	1.0
Block format	v4 only
Transaction format	v2 transfer only
PBE-PAY	v1
Slot target	15 seconds
Base unit	1 PBE = 100,000,000 base units
Hard finality	No

Implementation references: src/PBE/Core/Genesis.php; src/PBE/Core/Address.php; src/PBE/Core/Block.php; src/PBE/Core/Transaction.php

3. System Architecture

The implementation is organized into six functional layers: Core, Node, Network, Storage, Commerce, and presentation/client surfaces. The Core layer contains serialization, cryptography, addresses, transactions, blocks, state, Merkle commitments, emission, genesis, and consensus. Node orchestrates block production, peer identity, discovery, synchronization, and operator-wallet binding. Network contains outbound HTTP peer handling and endpoint filtering. Storage provides a shared abstraction over MySQL/MariaDB and SQLite. Commerce adds payment requests and optional webhooks. Browser wallet, explorer, developer documentation, and RPC front controller sit above these layers.

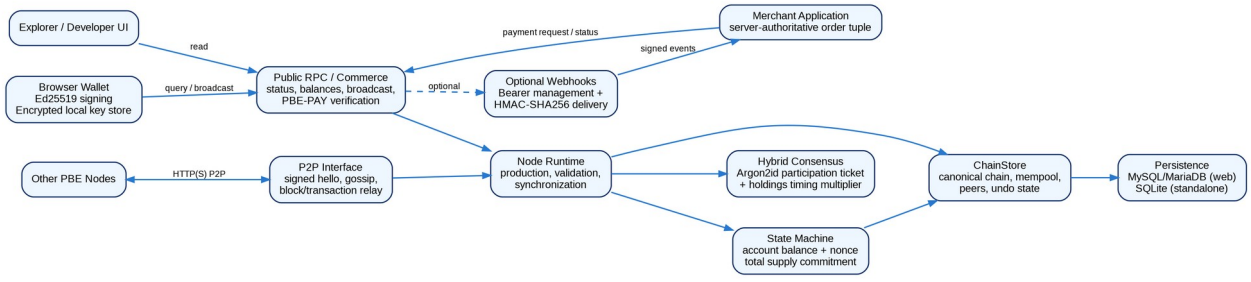

Figure 1. PBE 0.0.1 component architecture and trust boundaries.

A key architectural property is that web and standalone targets share the same canonical Core. The web deployment uses MySQL/MariaDB and exposes explorer, wallet, developer documentation, Commerce APIs, and P2P routes. The standalone deployment uses SQLite, a localhost UI/RPC listener, and a separate P2P listener. Consensus and chain logic are not duplicated per platform. This reduces the risk that a Windows or Linux package accidentally implements different validation semantics.

Runtime construction is deterministic. Runtime loads the chosen example-derived configuration, validates frozen genesis, generates a local node identity if one does not exist, connects to the selected database,

initializes the base schema, conditionally initializes merchant webhook tables only when enabled, initializes genesis, constructs operator-wallet state, constructs the Commerce service, and then instantiates the node with a peer client using the configured timeout. Optional merchant persistence therefore does not become a requirement for ordinary chain participation.

The separation between the wallet identity and machine identities is particularly important. A node has a node identity key for signed peer hello messages. A block-producing machine can have a distinct producer key. The non-custodial wallet authorizes the producer key and remains the reward address. Compromise of a producer key and compromise of the reward wallet are therefore different events, although a valid producer key plus its still-active authorization can still be operationally dangerous until disabled or rotated.

Implementation references: src/PBE/Node/Runtime.php; src/PBE/Node/Node.php; src/PBE/Node/OperatorWallet.php; src/PBE/Storage/Database.php

4. Canonical Data Model and Cryptographic Domains

Consensus systems fail when independently implemented nodes disagree about how the same logical object becomes bytes. PBE addresses this with an explicit Binary helper and fixed field ordering. Unsigned integers are encoded in network byte order: u16 and u32 use big-endian packing, while u64 is composed from high and low 32-bit words. Variable-length byte strings and text are prefixed with a 32-bit length. JSON is used as an API and persistence representation, but transaction and block signing operate on canonical binary encodings rather than raw JSON text.

Cryptographic hashing is built around SHA-256, but distinct object classes are separated by domain strings. Transaction signatures sign SHA-256("PBE:TX:V1\0" || unsignedTransactionBytes). Block signatures sign SHA-256("PBE:BLOCK:V1\0" || canonicalHeader). Node peer messages use SHA-256("PBE:NODE:V1\0" || canonicalMessage). State leaves and empty Merkle roots carry their own prefixes, as do participation challenges and salts. The practical benefit is that bytes valid in one context are not naively interpreted as a signature challenge in another context.

The Merkle construction also uses explicit node-domain separation. Each leaf is SHA-256(0x00 || leafHash), while internal nodes are SHA-256(0x01 || left || right). Odd levels duplicate the final element. An empty Merkle tree commits to SHA-256("PBE:MERKLE:EMPTY\0"). Transaction roots commit to transaction IDs. State roots commit to sorted account leaves plus a supply leaf, making total supply part of the state commitment rather than an external database statistic.

Block identity and block authentication are deliberately separate. Block::hash computes SHA-256 over the canonical header bytes and therefore does not include the producer signature itself. The producer signature authenticates a domain-separated hash of those same header bytes. Transaction identity differs: the TXID is SHA-256 of the signed transaction bytes, so changing the transaction signature changes the TXID. This distinction is internally consistent, but implementers must preserve it exactly.

Domain	Canonical operation
Transaction signing	SHA-256("PBE:TX:V1\0" || unsigned tx bytes), then Ed25519
Block signing	SHA-256("PBE:BLOCK:V1\0" || header bytes), then Ed25519
Node hello signing	SHA-256("PBE:NODE:V1\0" || canonical hello), then Ed25519
Address payload	First 20 bytes of SHA-256(public key), then Bech32m
Transaction ID	SHA-256(signed transaction bytes)
Block hash	SHA-256(canonical block header)
Merkle leaf	SHA-256(0x00 || object hash)
Merkle internal node	SHA-256(0x01 || left || right)

Implementation references: src/PBE/Core/Binary.php; src/PBE/Core/Crypto.php; src/PBE/Core/Merkle.php; src/PBE/Core/State.php

5. Addresses, Keys, and Wallet Security

PBE uses Ed25519 signing keys. The server-side reference code relies on libsodium, while the browser wallet uses Web Crypto Ed25519. A public PBE address is derived by SHA-256 hashing the 32-byte public key, truncating to 20 bytes, and encoding the payload using Bech32m with the human-readable prefix pbe. Address validation checks both the prefix for network ID 1 and the exact 20-byte decoded payload length.

The choice of Bech32m provides a human-readable network prefix and checksum while remaining independent of Bitcoin script semantics. PBE adopts the encoding mechanism, not Bitcoin witness-version rules. Because the payload is only the first 160 bits of SHA-256(public key), the address is a compact identifier rather than the public key itself. The transaction includes both sender public key and sender address; validation recomputes the address from the public key and requires an exact match.

The browser wallet is non-custodial in architecture. New Ed25519 keys are generated inside Web Crypto. Exported private PKCS#8 material is encrypted before storage using AES-256-GCM. The encryption key is derived from the wallet password using PBKDF2-SHA256 with a 16-byte random salt and 600,000 iterations; each wallet record uses a 12-byte random IV. The format validates a minimum iteration count on import, checks encoded sizes, decrypts locally, and performs a sign/verify challenge before accepting the recovered private key.

The wallet and node interact through proofs rather than private-key transfer. Operator binding uses a one-time setup token plus a wallet signature. Block-production authorization is a separate signature that binds the network, wallet address, wallet public key, and exact producer public key. Management actions can require signed operator actions. The node therefore needs the wallet public identity and signatures, but the long-term wallet private key stays in the browser.

> **Security boundary** The non-custodial design still depends on the integrity of the browser origin and JavaScript delivered to it. If wallet-page code or the execution environment is compromised, client-side key isolation can be defeated before signing. HTTPS, content-security policy, dependency discipline, and origin integrity remain essential.

Implementation references: src/PBE/Core/Address.php; src/PBE/Core/Bech32m.php; src/PBE/Core/Crypto.php; wallet/assets/pbe-crypto.js; src/PBE/Node/OperatorWallet.php

6. Transaction Model and State Transition

Transaction v2 supports a single type: transfer. A canonical transfer contains version, network ID, type, sender public key, sender address, nonce, fee, timestamp, and a transfer payload containing recipient, amount, and an optional paymentReference. The reference is limited to 64 bytes and a deliberately narrow ASCII set: letters, digits, dot, underscore, colon, and hyphen. Unknown payload keys are rejected rather than ignored, preventing unsigned or ambiguously interpreted extensions from hitchhiking inside a transfer object.

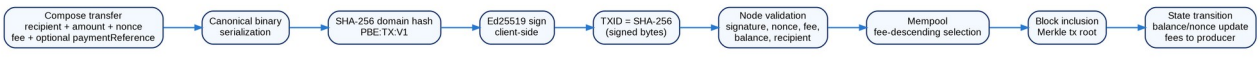

Figure 2. Signed transfer lifecycle from composition to state transition.

Nonce semantics are strict account sequencing. The sender transaction nonce must equal current account nonce plus one. This prevents replay on the canonical state and gives each account a total order of transactions. A recipient is validated as a mainnet PBE address and self-transfers are rejected. The state machine checks amount positivity, fee bounds, overflow, and sufficient balance before writing changes. The sender is debited amount plus fee; the recipient receives exactly the amount; the sender nonce becomes the transaction nonce.

Fees are deterministic from serialized transaction size. The default minimum is 1,000 base units plus 2 base units per signed byte. A minimal transfer with no payment reference is approximately 236 signed bytes, producing a minimum fee of approximately 1,472 base units, or 0.00001472 PBE. A transfer carrying a full 64-byte payment reference is approximately 300 signed bytes, producing a minimum fee of approximately 1,600 base units, or 0.000016 PBE. These are protocol-reference calculations, not market fee estimates; a future release could change fee policy only through an explicit protocol upgrade if consensus validation changes.

Transaction selection is independent of transaction validity. Mempool insertion first verifies the signature and the node applies the transaction against current state to reject invalid nonce, fee, or balance conditions. The database stores sender, nonce, fee, payment reference, recipient, and transfer amount as searchable indexes in addition to the raw transaction JSON. This indexing is what makes exact PBE-PAY matching efficient without changing transaction consensus rules.

Transfer field	Consensus meaning
version	Must be 2
networkId	Must match PBE mainnet network ID 1
type	Must be transfer type 1
senderPublicKey	32-byte Ed25519 public key
senderAddress	Must equal address derived from senderPublicKey
nonce	Exactly previous sender nonce + 1
fee	At least base + byte-size fee
timestamp	Signed metadata; block validity governs block time
recipient	Valid pbe mainnet address, not sender
amount	Positive integer base units
paymentReference	Optional signed merchant/application correlation value

Implementation references: src/PBE/Core/Transaction.php; src/PBE/Core/State.php; src/PBE/Storage/ChainStore.php; wallet/assets/pbe-crypto.js

7. Block Structure and Validation

Block v4 binds chain position, timing, producer identity, reward identity, participation credentials, transaction commitment, state commitment, and consensus constants in one canonical header. Fields include version, network ID, height, slot, previous block hash, timestamp, producer public key, reward address, mining wallet public key, mining-authorization signature, participation proof, transaction root, state root, and consensus root. The producer signature is stored alongside the header but is not included in the block hash.

A non-genesis block must be no larger than 2,097,152 serialized bytes. Validation rejects future timestamps more than two seconds beyond the verifier clock, timestamps earlier than genesis tolerance, implausibly future slots, incorrect heights, wrong parent hashes, wrong networks, non-increasing slots, and consensus-root mismatch. The consensus root commits to the Argon2 and multiplier constants; a block produced under different consensus parameters therefore cannot silently pass validation on an unmodified node.

Validation order reflects denial-of-service awareness. The node verifies reward-wallet consistency and the producer signature before paying the cost of recomputing the memory-hard Argon2id participation proof. It then verifies wallet authorization, participation proof, eligibility timing, transaction Merkle root, each transaction/state transition, reward/fee credit, and final state root. Invalid expensive proofs are therefore gated behind cheaper cryptographic header authentication when possible.

The genesis block uses the same block version and root machinery but has height and slot zero, all-zero producer and participation fields, an empty transaction set, the genesis state root, and a consensus root. Genesis has no producer reward. This avoids a special alternative block format while still clearly separating genesis validation from normal production.

Block validation stage	Representative checks
Structural	v4 only; size <= 2 MiB; fixed cryptographic field lengths
Temporal	slot plausibility; timestamp tolerance; increasing slot
Chain linkage	height increments by one; previous hash matches canonical tip
Consensus identity	network ID and consensus root match
Reward identity	reward address derives from mining wallet public key
Authentication	producer signature; wallet authorization for producer key
Participation	Argon2id proof recomputation and eligibility delay
Transactions	Merkle root, tx validity, nonce, fee, balance
State	fees + reward credit; recomputed state root

Implementation references: src/PBE/Core/Block.php; src/PBE/Core/HybridConsensus.php; src/PBE/Core/State.php

8. Hybrid Participation Consensus

Hybrid participation is the defining experimental mechanism in PBE 0.0.1. It combines a per-wallet memory-hard ticket with a holdings-based timing advantage. The protocol does not require a minimum stake and does not lock balances. A wallet with zero PBE is eligible. Holdings affect when a valid block may be emitted within a slot, not whether the wallet gets a ticket at all.

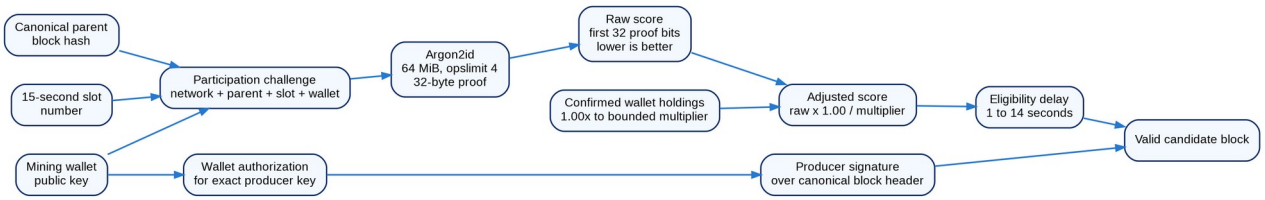

Figure 3. Hybrid participation ticket, timing multiplier, and producer authorization path.

8.1 Deterministic participation challenge

For each candidate slot, the challenge concatenates a domain tag with network ID, previous block hash, slot number, and mining wallet public key. The machine producer public key is intentionally absent. The same wallet, parent, and slot therefore produce the same challenge regardless of how many machines or producer keys the operator owns. A deterministic salt is derived from SHA-256 of a second domain tag plus the challenge. Libsodium Argon2id then derives a 32-byte proof using operation limit 4 and 64 MiB of memory.

Because the ticket is deterministic, recomputation by validators is straightforward: the verifier derives the same challenge, salt, and Argon2id output and compares the proof in constant time. It also means there is no nonce-grinding loop inside the protocol. A wallet receives one proof value per parent/slot combination. An operator cannot improve that wallet ticket by repeatedly changing a local nonce or producer key.

8.2 Score, multiplier, and eligibility delay

The raw lottery score is the first four bytes of the proof interpreted as an unsigned big-endian 32-bit integer; lower is better. Holdings are converted to whole PBE, then mapped through a piecewise-linear schedule. The base multiplier is 10,000 basis points (1.00x) and the nominal maximum is 25,000 basis points (2.50x). The adjusted score is integer-divided as rawScore x 10,000 / multiplierBps. The adjusted score is then linearly mapped onto a 1-to-14 second delay.

$$adjustedScore = floor(rawScore \times 10{,}000 / multiplierBps)$$

$$delay = clamp(1 + floor(adjustedScore \times 13 / 0xffffffff),\ 1,\ 14)\ seconds$$

Assuming the first 32 proof bits are approximately uniform, a zero-balance wallet has an average eligibility delay of about 7 seconds and can fall anywhere from 1 to 14 seconds. At the nominal 2.50x multiplier, the effective score range contracts so strongly that eligibility is always within roughly 1 to 6 seconds, with an average near 3.1 seconds. The multiplier therefore acts as a timing preference rather than a multiplicative ticket count.

Whole PBE anchor	Multiplier	Approx. average delay*	Max delay*
0	1.00x	7.00 s	14 s
100	1.10x	6.42 s	12 s
1,000	1.25x	5.71 s	11 s
10,000	1.50x	4.85 s	9 s
100,000	1.75x	4.23 s	8 s
1,000,000	2.00x	3.77 s	7 s
10,000,000	2.25x	3.40 s	6 s
100,000,000	2.50x	3.12 s	6 s

**Derived analytically/numerically from the implemented score-to-delay mapping assuming a uniform 32-bit raw score; actual ticket outcomes are deterministic per challenge.*

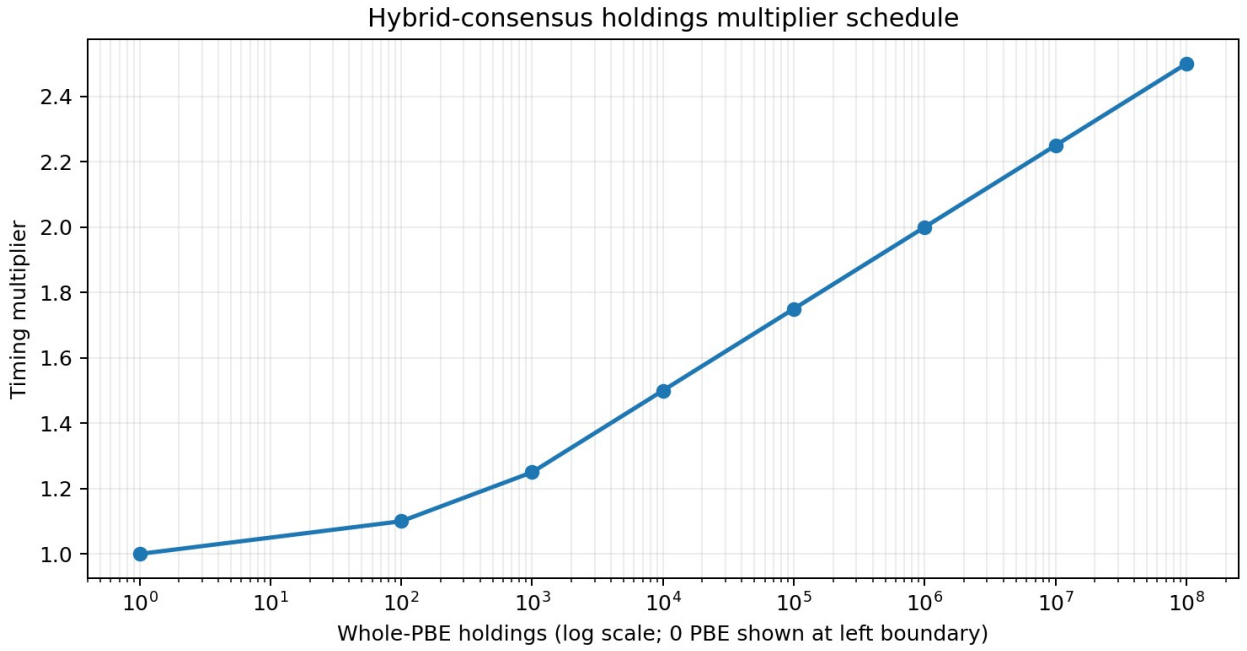

Figure 4. Holdings multiplier schedule. Interpolation is linear between listed whole-PBE anchors.

8.3 Wallet authorization and machine producer key

The wallet and block-producing machine are separated. The wallet signs a mining-authorization hash that binds network ID, wallet address, wallet public key, and producer public key. A block contains both the wallet authorization signature and the producer public key. The producer then signs the block header with the producer secret key. Validators therefore require two distinct proofs: the wallet consented to this producer identity, and the producer machine possesses the corresponding private key.

Rewards are paid to the address derived from the mining wallet public key, not to the producer-machine key. This enables operational key rotation without redefining the economic identity. It also prevents a machine producer from redirecting rewards by altering only the block header; changing the reward address would break the wallet-public-key relationship and header signature.

8.4 Sybil characteristics

The one-ticket-per-wallet rule prevents one wallet from obtaining additional tickets by cloning the same wallet across many machines. It does not, by itself, prevent an adversary from generating many distinct wallets. Since zero-balance wallets are eligible and key creation is cheap, identity scarcity is not the Sybil defense. Instead, each distinct wallet-slot-parent ticket requires a separate Argon2id computation with the protocol memory/time cost. The effective Sybil cost is therefore computational and memory bandwidth rather than stake acquisition or permissioned enrollment.

This distinction matters for security modelling. If an attacker can evaluate N independent wallet tickets per slot, the minimum delay among those tickets falls as N grows. The Argon2id cost makes this scaling non-free, but the protocol currently contains no identity admission control, no stake minimum, and no cumulative-resource score across history. Future analysis should quantify ticket throughput on commodity CPUs, GPUs, and memory-rich parallel hardware and compare that cost to honest node population and expected network latency.

> **Consensus research priority** The protocol has a clear local rule for ticket generation and eligibility, but network security depends on aggregate participant economics and alternate-history cost. Formal modelling should treat wallet multiplicity, parallel Argon2 throughput, network propagation, equal-height forks, and deep historical branch construction together.

Implementation references: src/PBE/Core/HybridConsensus.php; src/PBE/Core/Block.php; src/PBE/Node/Node.php; src/PBE/Node/OperatorWallet.php

9. Fork Choice, Reorganization, and Finality

PBE first prefers greater valid canonical height. When two valid tips have equal height, it compares forkChoiceKey values lexicographically. For non-genesis blocks the key is slot || participationProof || miningWalletPublicKey. Lower sorts first. Producer public key and producer signature are excluded from the primary tie-break so a participant cannot cheaply rotate machine keys or signatures to search for a better equal-height result. If two blocks have identical ticket keys - effectively equivocation using the same wallet ticket - the final fallback is the lexicographically lower block hash so all nodes still have a deterministic convergence rule.

Chain height is not cumulative work. This is the most important difference between PBE and Nakamoto-style proof-of-work. The memory-hard ticket is evaluated per candidate slot, but the chain selector does not sum ticket difficulty or work across a branch. A longer valid branch wins regardless of how much wall-clock time or aggregate resource consumption the competing branches required to produce.

Reorganization support is implemented as a first-class storage operation. When a remote signed tip differs from the local tip, the node first determines whether the remote fork wins. For divergent winning branches it finds a common ancestor, downloads only the suffix, validates that suffix against reconstructed historical state, then atomically deletes the old divergent tail, inserts the winning blocks and transactions, rewrites touched accounts, recreates undo records, updates tip metadata, and removes newly confirmed mempool entries. A full-chain rebuild path remains as a recovery fallback.

Finality is explicitly probabilistic and operational. Commerce confirmations count canonical blocks after the payment block. Twelve confirmations are the default merchant policy and thirty are documented for higher-value cases, but neither number creates protocol-level irreversibility. A reorganization can reduce confirmation depth, replace a transaction, or remove it from the canonical chain; the webhook service explicitly emits payment.reorged when this happens.

The known deep-reorganization concern follows from deterministic historical participation. Proofs are tied to parent, slot, and wallet, and validators can recompute them after the fact. The current design has no cumulative-work rule, hard checkpoint, weak-subjectivity checkpoint, or BFT-style finality gadget. A sufficiently capable actor attempting an alternate historical branch is constrained by proof computation and block timestamp/slot validation, but not necessarily by the original real-time pace of honest history. Newly syncing nodes and nodes returning after long partitions are the most obvious contexts in which stronger finality assumptions would be valuable.

> **Do not overstate confirmations** In PBE 0.0.1, "confirmed" means the exact payment is on the current canonical chain at or above the requested depth. It does not mean mathematically irreversible.

Implementation references: src/PBE/Core/Block.php::forkChoiceKey; src/PBE/Node/Node.php::remoteForkWins; src/PBE/Storage/ChainStore.php::reorganizeFromAncestor; docs/MAINTAINER-REFERENCE.md

10. Monetary Policy and Emission

PBE uses integer base units with eight decimal places: 100,000,000 base units per PBE. The initial block reward is 5 PBE. The reward epoch is 8,409,600 blocks. At a nominal 15-second slot this corresponds to approximately 3.997 years per epoch. For each new epoch, the base-unit reward is integer-divided by two until it reaches zero.

Because halving is performed on integer base units, eventual supply is not exactly the infinite geometric-series value of 84,096,000 PBE. Summing every positive base-unit reward produces approximately 84,095,998.906752 PBE over 29 positive-reward epochs. The final positive reward is one base unit (0.00000001 PBE) per block during epoch 28; the next epoch rewards zero. At continuous 15-second slots, positive issuance spans approximately 115.9 years from genesis.

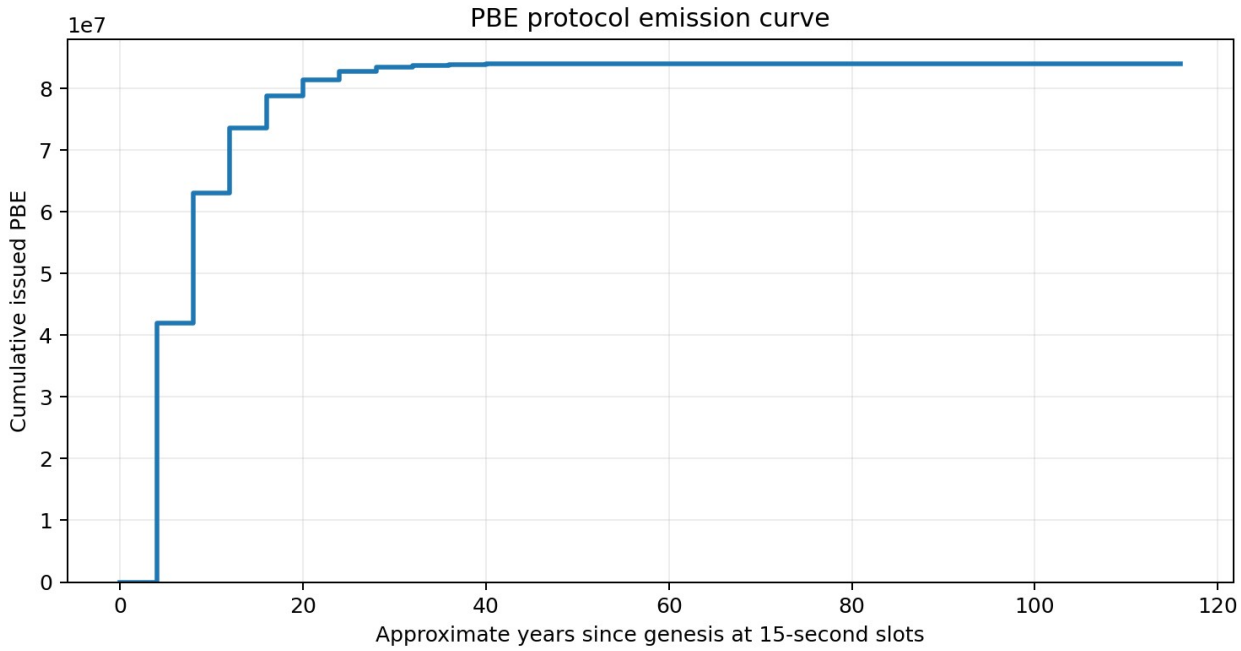

Figure 5. Cumulative protocol issuance under uninterrupted 15-second slots.

Fees are not new issuance. State::creditProducer adds fees plus block reward to the producer reward address but increases total supply only by the reward component. Transfer activity therefore redistributes existing PBE through fees while emission expands supply according to height. This separation is also committed by the state root because total supply is a dedicated state leaf.

A subtle interaction exists between emission and the holdings multiplier schedule. The nominal maximum multiplier is anchored at 100,000,000 PBE, but the maximum possible mainnet supply is lower. If one address eventually held the entire final supply, the implemented piecewise interpolation would yield approximately 24,558 basis points, or 2.4558x, rather than the nominal 2.50x cap. Thus 2.50x is a protocol bound, not an economically reachable mainnet state under the current issuance rules.

Epoch	Reward (PBE)	Cumulative supply after epoch (PBE)	Approx. end year
0	5	42,048,000	4.00
1	2.5	63,072,000	7.99
2	1.25	73,584,000	11.99
3	0.625	78,840,000	15.99
4	0.3125	81,468,000	19.99
5	0.15625	82,782,000	23.98
6	0.078125	83,439,000	27.98
7	0.0390625	83,767,500	31.98
...	integer base-unit halvings continue	...	...
28	0.00000001	84,095,998.906752	115.92

Implementation references: src/PBE/Core/Emission.php; src/PBE/Core/State.php; src/PBE/Storage/ChainStore.php

11. Peer-to-Peer Networking and Discovery

PBE uses HTTP/HTTPS JSON endpoints rather than a custom binary socket protocol. That choice makes the protocol easy to inspect and deploy behind conventional web infrastructure, but it also places importance on HTTP request hardening, endpoint normalization, response caps, and reverse-proxy rate limits. The peer protocol version is 1.0 and compatibility requires exact protocol version, network ID, chain ID, and frozen genesis hash.

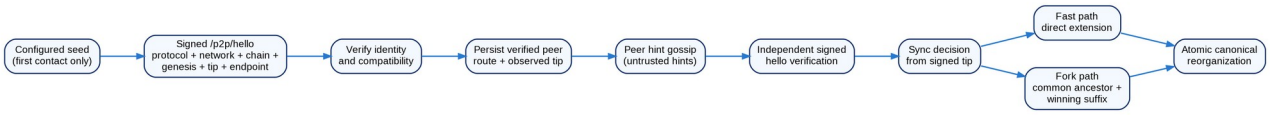

Figure 6. Bootstrap, signed peer verification, gossip, and synchronization decision flow.

Each node has an Ed25519 node identity key. The public node ID is pbenode_ followed by the first 32 hexadecimal characters of SHA-256(nodePublicKey), yielding a 128-bit displayed identifier while the hello still carries the full 32-byte public key. A hello signs software version, protocol version, network ID, chain ID, genesis hash, node ID, height, tip hash, timestamp, and advertised endpoint. A receiver verifies the signature, recomputes node ID, checks a two-minute timestamp tolerance, and rejects heights that exceed the maximum plausible slot-derived height.

Peer discovery treats addresses as hints, not trust statements. After first contact with a configured seed, a node can learn peer endpoints through gossip. Hinted endpoints are normalized and deduplicated, and private/loopback/reserved endpoints are rejected by default unless explicitly allowed. A hinted peer becomes meaningful only after an independent signed hello is verified. Self-peers are rejected by both node identity and endpoint.

Outbound hostname handling includes DNS-rebinding protection. Hostnames are resolved once; peer-supplied names are rejected if any resolved address is private or reserved. The outbound URL then connects to a pinned resolved address while retaining the original host for Host/TLS verification. Redirects are disabled. Responses are capped at 64 MiB. These controls materially reduce the chance that peer gossip can be used to coerce the node into probing localhost or private infrastructure.

Seeds are special only for bootstrap. Once compatible peers exist, the node uses its persisted peer table and gossip. If the cache later contains no usable compatible peers, bootstrap can reopen. The primary-node example intentionally allows an empty seed list, enabling the first public node to start without an upstream dependency, while ordinary node examples seed from the public Paybyte endpoint.

Implementation references: src/PBE/Network/PeerClient.php; src/PBE/Network/PeerDiscovery.php; src/PBE/Node/Node.php::hello; src/PBE/Node/Node.php::bootstrapIfNeeded

12. Synchronization and Canonical-Chain Replacement

Synchronization begins with a signed peer hello. The node explicitly bases remote height/hash decisions on the signed hello rather than an unsigned status object. If the remote tip exactly matches local height and hash, no work is needed. If the remote chain is longer and the block at the local height matches the local tip, the node takes a direct-extension fast path and downloads blocks in batches of at most sixteen.

Divergent branches use a more sophisticated path. Equal-height competition first runs deterministic fork choice. A winning remote branch triggers common-ancestor discovery. The algorithm optimizes for live shallow forks by probing up to the last seven heights first. If no match is found in that short tail, it binary-searches the remaining height range, yielding logarithmic remote block lookups for deep divergence.

After the common ancestor is found, only the winning suffix is downloaded. Each returned block must appear at the expected height and the final downloaded block must match the tip hash advertised in the signed peer hello. Before storage mutation, the suffix is validated against state reconstructed at the ancestor. This avoids trusting the remote peer simply because its branch is longer or wins the tie-break.

ChainStore persists state-undo records for accounts touched by each block. During incremental reorganization, it reconstructs historical state, validates the incoming branch, computes new undo data, and then performs deletion/insertion/state rewrite inside a database transaction. If incremental history is corrupt or missing, Node can download and validate the entire candidate chain and call replaceCanonicalChain, which rebuilds canonical blocks, transactions, accounts, undo rows, tip metadata, and supply atomically.

This design has a practical UI benefit as well as a correctness benefit: routine shallow forks do not require visible full-chain replay. The implementation treats reorganization as an expected chain event rather than an exceptional manual repair procedure, which is necessary because PBE explicitly lacks hard finality.

Implementation references: src/PBE/Node/Node.php::syncOnce; src/PBE/Node/Node.php::findCommonAncestor; src/PBE/Storage/ChainStore.php::reorganizeFromAncestor; src/PBE/Storage/ChainStore.php::replaceCanonicalChain

13. Storage Architecture

The storage layer supports MySQL/MariaDB for web deployments and SQLite for standalone deployments behind a common DbConnection abstraction. The canonical node schema contains chain_meta, blocks, transactions, accounts, mempool, peers, and state_undo. Optional Commerce webhook tables are initialized only when the feature is enabled. This prevents merchant infrastructure from becoming a hidden prerequisite for consensus operation.

Blocks store both indexed consensus fields and raw canonical JSON. Transactions likewise store core fields plus searchable transfer indexes. Accounts store balance, nonce, and the height at which they were last updated. Mempool rows index sender nonce, fee, payment reference, recipient, and transfer amount. Peer rows track node identity, endpoint, public key, protocol version, last height/hash, last seen time, failures, and temporary bans. Undo rows store previous balance and nonce by block height/address.

Database initialization also enforces release hygiene. The 0.0.1 ChainStore checks that the expected release-era columns exist and rejects incompatible schema revisions. MySQL startup refuses blank or unchanged

CHANGE_ME-style passwords. SQLite enables foreign keys, WAL journal mode, NORMAL synchronous behavior, and a short busy timeout. These are implementation-level operational choices rather than consensus rules, but they reduce accidental deployment failure.

The canonical chain and state are updated transactionally. applyBlock inserts the block, writes undo values for every dirty account, inserts transactions, upserts accounts, removes obsolete mempool rows, updates tip metadata and total supply, then commits. A failure rolls the entire database transaction back. This atomicity is important because a partial block/state update would otherwise create a local state root that no longer corresponds to the stored tip.

Base table	Purpose
chain_meta	Schema revision, chain identity, tip metadata, supply, network settings
blocks	Canonical block headers, roots, rewards, fees, raw representation
transactions	Confirmed transaction index and payload data
accounts	Current account balance and nonce state
mempool	Validated unconfirmed transactions
peers	Persisted verified routes and peer status
state_undo	Per-block previous account state for reorganization

Implementation references: database/schema.mysql.runtime.sql; database/schema.sqlite.sql; src/PBE/Storage/Database.php; src/PBE/Storage/ChainStore.php

14. Mempool and Block Assembly

Mempool admission verifies the transaction signature and applies the transaction to a clone/current state path before insertion, so invalid balances, nonces, fees, or recipients are rejected before storage. The mempool enforces unique transaction ID and unique sender-address/nonce combinations at the database layer. Selection order is fee descending, then arrival time ascending.

The local block producer requests at most 1,000 mempool transactions for a candidate block. It applies each candidate sequentially to a scratch state and skips any transaction that becomes invalid in the chosen ordering. It also estimates serialized block size and refuses additions that would exceed the 2 MiB block maximum. After block creation, the block is validated again through the normal validator before being committed, providing a useful self-consistency check between production and validation logic.

The 1,000-transaction producer selection limit is an implementation throughput ceiling distinct from the protocol block-size ceiling. At exactly one 15-second canonical block per slot, the current local producer can include at most about 66.7 transactions per second from its own mempool selection, even though the 2 MiB wire format could theoretically contain several thousand minimal transfers. A peer-provided valid block is not subject to the producer's 1,000-candidate selection cap; it is subject to block size and transaction validity.

Confirmed transactions are removed from the mempool by deleting entries for each sender with nonce less than or equal to the confirmed nonce. This naturally clears earlier superseded sequence positions for that account. Reorganizations do not automatically reconstruct every displaced transaction back into the mempool; the system focuses on canonical state correctness. Application-level rebroadcast policy therefore remains relevant after a transaction is reorged out.

Implementation references: src/PBE/Storage/ChainStore.php::addMempool; src/PBE/Storage/ChainStore.php::mempool; src/PBE/Node/Node.php::produceBlock

15. PBE-PAY Merchant Payment Protocol

PBE-PAY v1 is a merchant-facing convention built on the existing transfer transaction. A request specifies recipient, amount, reference, optional label, and required confirmation depth. The URI form is pbe:<address>?amount=<PBE>&ref=<reference>&confirmations=<n>&label=<label>. The protocol does not create custody, escrow, or a merchant-controlled signing flow. The payer still signs a normal PBE transaction locally.

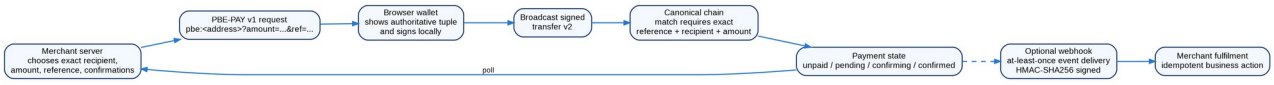

Figure 7. Server-authoritative PBE-PAY checkout and optional webhook fulfilment.

Payment matching is intentionally exact. ChainStore queries for the three-part tuple paymentReference + recipient + transfer amount. A reference by itself is not sufficient, and a matching amount sent to the wrong recipient is not sufficient. This protects merchant code from treating browser-controlled display data as the source of truth. Secure checkout examples therefore generate authoritative order values server-side and let browser JavaScript render and hand off the already-created request.

The status model distinguishes unpaid, pending, confirming, and confirmed. A matching mempool transaction is pending and non-canonical. Once included, canonical is true and confirmations are computed as tipHeight - blockHeight + 1. The state becomes confirmed when that depth reaches the merchant requirement. Because the lookup is against the current canonical transaction index, a reorganization can cause a previously visible payment to disappear or lose depth.

Default required confirmations are 12. The protocol helper exposes 30 as a high-value recommendation and caps requested confirmations at 1,000. These values are merchant risk parameters only. The response explicitly advertises hardFinality=false and settlementModel=canonical-chain-depth so applications do not confuse depth policy with irreversible consensus.

The signed payment reference is one of the strongest design choices in the Commerce layer. The reference is inside the transaction payload and therefore protected by the sender signature; changing it invalidates the signature. This closes the basic gap that exists when a merchant correlates payments using only unsigned browser metadata or a mutable client-side order identifier.

Implementation references: src/PBE/Commerce/PaymentProtocol.php; src/PBE/Storage/ChainStore.php::paymentStatus; src/PBE/Core/Transaction.php; sdk/paybyte-pay.js

16. Commerce Webhooks

Webhooks are explicitly operational data, not consensus state. A merchant-controlled node can enable two additional tables for payment subscriptions and queued events. Management endpoints require a bearer token, while outbound event bodies are authenticated with a distinct HMAC-SHA256 signing secret. The service requires both configured secret values to meet a minimum length before management is considered active.

Subscriptions bind the same exact payment tuple used by polling: reference, recipient, amount, confirmation count, callback URL, and expiry. The service periodically checks canonical payment status. It emits payment.pending, payment.confirming, payment.confirmed, payment.reorged, and payment.expired events when state changes. A reorg event can be triggered by loss of canonical status, a changed transaction ID, a drop in confirmations, or loss of previously confirmed depth.

Delivery authentication follows a simple canonical scheme: HMAC-SHA256(secret, timestamp + "." + exact_raw_body). The HTTP request carries event type, event ID, timestamp, and a v1-prefixed signature.

Verification rejects stale timestamps outside a default five-minute tolerance and compares the expected MAC with hash_equals. This is a conventional and understandable construction consistent with HMAC guidance, assuming the secret remains server-side and has adequate entropy.

Callback validation is deliberately defensive. HTTPS is required by default. URLs containing credentials or fragments are rejected. Localhost is blocked, private/reserved addresses are blocked unless explicitly permitted, DNS names are resolved and pinned to a validated address, redirects are disabled, connect and total timeouts are bounded, and callback length is capped. These measures directly address SSRF-style abuse from merchant-provided callback URLs.

Delivery is at-least-once, so merchant fulfilment must be idempotent. Successful 2xx responses mark events delivered; failures remain queued with a bounded retry schedule of 5, 15, 30, 60, 120, 300, 600, 900, 1800, then 3600 seconds for later attempts. Event IDs provide a natural merchant-side deduplication key.

Webhook property	Behavior
Management authentication	Bearer token, server-side only
Delivery authentication	HMAC-SHA256 over timestamp + "." + exact body
Timestamp replay window	Default verification tolerance 300 seconds
Delivery semantic	At-least-once; merchant must be idempotent
Default callback transport	HTTPS
Redirects	Disabled
Private targets	Blocked by default
Retry backoff	5 s -> 1 h capped schedule
Reorg awareness	Explicit payment.reorged event

Implementation references: src/PBE/Commerce/WebhookService.php; database/schema.commerce-webhooks.mysql.sql; web/developers/webhooks.php

17. RPC and Developer Interface

The public API surface is compact and can be divided into chain reads, transaction submission, Commerce, and P2P. The OpenAPI document exposes status, consensus parameters, block lookup, balance/nonce, address transaction history, transaction lookup, signed-transaction broadcast, Commerce capability/payment routes, optional webhook management, signed peer hello, transaction/block relay, canonical block batches, and peer hints.

The front controller distinguishes public browser-friendly routes from privileged management actions. Public reads and broadcast endpoints can expose permissive CORS, while setup/operator/network-management routes rely on localhost restrictions, one-time setup codes, wallet signatures, origin checks, Fetch Metadata checks, and JSON content-type requirements as appropriate. Hosted operator status is privacy-redacted unless the bound wallet proves identity with a signed action.

This privacy distinction is subtle but valuable. A public hosted node does not expose its operator wallet address, deterministic hash of the address, producer identity, current participation ticket, or local mining state through the ordinary public status view. A deterministic address hash would not provide meaningful privacy because public blockchain addresses could be enumerated and hashed offline. Instead, the wallet proves it is the bound operator and then receives the private management view.

Method	Path	Purpose
GET	/rpc/status	Node status
GET	/rpc/consensus	Consensus parameters
GET	/rpc/block/{height}	Block by height
GET	/rpc/balance/{address}	Account balance and nonce
GET	/rpc/address/{address}/transactions	Address transactions
GET	/rpc/transaction/{txid}	Transaction by TXID

Method	Path	Purpose
POST	/rpc/broadcast	Broadcast signed transaction
GET	/rpc/commerce/catalog	Commerce endpoint catalog
GET	/rpc/commerce/info	Commerce capabilities
POST	/rpc/commerce/payment-request	Create PBE-PAY request
GET	/rpc/commerce/payment/{reference}	Verify exact PBE-PAY payment
POST	/rpc/commerce/webhooks	Register merchant webhook
GET	/rpc/commerce/webhooks/{id}	Inspect merchant webhook
POST	/rpc/commerce/webhooks/{id}/cancel	Cancel merchant webhook
GET	/p2p/status	P2P node status
POST	/p2p/hello	Signed peer handshake
POST	/p2p/transaction	Peer transaction relay
POST	/p2p/block	Peer block relay
GET	/p2p/blocks	Canonical block batch
GET	/p2p/peers	Verified peer hints

Implementation references: web/developers/openapi.json; public/index.php; web/developers/rpc.php; web/developers/commerce.php; web/developers/p2p.php

18. Deployment Models and Example Configuration

The project ships example configuration files rather than a production secret-bearing configuration. This is the correct basis for a public technical description because it captures supported topology without exposing local credentials. All values below are taken from example/default configurations and should be interpreted as templates rather than production secrets.

18.1 Web / server deployment

The generic web example selects MySQL, a local database endpoint, a PBE database/user placeholder, a five-second network timeout, public-peer filtering, the frozen mainnet genesis file, a public node endpoint placeholder, local node/producer key-file locations, and a public Paybyte seed. Commerce webhooks are disabled and their management/signing values are blank. The primary-node example is the same general model but uses the public Paybyte endpoint and an empty seed list because a bootstrap origin does not require an upstream seed.

Sanitized shape of config/config.example.php. Secret fields remain placeholders/blank by design.

```
return [
  'deployment' => ['mode' => 'web'],
  'database'   => ['driver' => 'mysql', 'host' => '127.0.0.1',
                   'database' => 'pbe', 'user' => 'pbe',
                   'password' => 'CHANGE_ME'],
  'network'    => ['timeoutSeconds' => 5,
                   'allowPrivatePeers' => false,
                   'seeds' => ['https://paybyte.org']],
  'commerce'   => ['webhooks' => ['enabled' => false,
                   'manageToken' => '', 'signingSecret' => '']]
];
```

18.2 Standalone deployment

The standalone example selects SQLite, localhost UI/RPC on port 8024, a P2P listener on port 8025, publicRpc=false, private-peer filtering, and the public Paybyte seed. Its public node endpoint initially points to the local P2P listener and can later be replaced or automatically observed according to network settings. This split allows the browser UI to remain local while the peer listener is separately exposed through router/firewall configuration.

Sanitized shape of config/standalone.example.php. Local key/database paths are described generically.

```
return [
  'deployment' => [
    'mode' => 'standalone',
    'localHost' => '127.0.0.1', 'localPort' => 8024,
    'p2pHost' => '0.0.0.0',    'p2pPort' => 8025,
    'publicRpc' => false
  ],
  'database' => ['driver' => 'sqlite', 'path' => '<local pbe.sqlite>'],
  'network' => ['timeoutSeconds' => 5,
                'allowPrivatePeers' => false,
                'seeds' => ['https://paybyte.org']],
  'commerce' => ['webhooks' => ['enabled' => false,
                'manageToken' => '', 'signingSecret' => '']]
];
```

The example configuration establishes an important operational principle: Commerce webhooks are opt-in. A normal node can relay transactions, synchronize, produce blocks, run a wallet, and expose payment verification without holding merchant management credentials or webhook signing secrets. This minimizes secret-bearing surface for participants who do not need server-to-server merchant callbacks.

Implementation references: config/config.example.php; config/paybyte.org.example.php; config/standalone.example.php; src/PBE/Storage/Database.php

19. Security and Threat Analysis

The security posture of PBE 0.0.1 is a combination of conventional cryptographic primitives, explicit validation, operational hardening, and an experimental consensus model. The conventional pieces - Ed25519 signatures, SHA-256 commitments, HMAC-SHA256 webhooks, Bech32m checksums, PBKDF2/AES-GCM wallet encryption, prepared SQL, and DNS/redirect restrictions - are familiar technologies. The principal research uncertainty sits in consensus economics and alternate-history resistance rather than in the syntax of those primitives.

19.1 Key compromise domains

PBE separates three key domains: wallet key, node identity key, and producer key. Wallet compromise enables value transfer and authorization changes. Producer-key compromise can authorize blocks only while paired with a still-valid wallet authorization; it does not reveal the wallet private key. Node-identity-key compromise can impersonate that peer identity in handshakes but does not directly sign transfers or producer blocks. This separation reduces blast radius, although operational response procedures should still support prompt producer-key rotation, operator disablement, and peer-key regeneration.

19.2 Transaction and replay security

Sender signatures bind network ID, sender identity, nonce, fee, timestamp, recipient, amount, and payment reference. Nonces enforce strict sequence and prevent canonical replay of an already-applied sender operation. Address/public-key binding prevents a transaction from claiming an unrelated sender address. Unknown payload fields are rejected, closing an extension ambiguity that could otherwise lead different clients to sign or display different semantic content.

19.3 Consensus and Sybil risk

The main consensus risk is not that one wallet can generate multiple tickets - it cannot for the same parent and slot - but that distinct zero-balance wallets are cheap to create. The Argon2id requirement turns identity scaling into a hardware-resource problem. Security therefore depends on the ratio between adversarial ticket throughput and honest ticket throughput/propagation, not on ownership of a scarce stake credential. That

ratio has not been formally characterized in the source tree and should be measured empirically before strong decentralization or attack-cost claims are made.

A related issue is deterministic historical recomputation. If an attacker can build an alternate branch faster than elapsed historical wall time, longest-valid-height selection may favor that branch once it is longer. Timestamp checks limit obviously future blocks but do not constitute cumulative work or finality. A future checkpoint/finality design should make trust assumptions explicit rather than relying on merchant confirmation depth to solve a chain-selection problem.

19.4 Network and API abuse

P2P and webhook HTTP clients include strong SSRF-oriented controls, but public endpoints still need rate limiting. Candidate block validation includes a 64 MiB Argon2id recomputation after header authentication; an attacker able to send many distinct, correctly signed but invalid candidate headers could consume resources. The application code can reject malformed structures early, but network-layer connection limits, request body limits, IP/ASN controls where appropriate, and reverse-proxy rate limits remain necessary operational defenses.

The peer response cap of 64 MiB is much larger than the 2 MiB block limit because batch and JSON overhead must be accommodated. That cap prevents unbounded reads but is still substantial. Operators should treat peer endpoints as hostile input and enforce process memory, concurrency, and HTTP server limits in addition to in-process validation.

19.5 Merchant threat model

The strongest merchant guarantee is exact tuple matching against canonical data. Browser modification cannot turn an arbitrary wallet/amount pair into a paid order if the merchant server holds the authoritative recipient, amount, and reference and queries those exact values. However, a merchant that incorrectly trusts client-supplied order values can still defeat the model at the application layer. The supplied secure checkout pattern therefore keeps authoritative tuple construction on the server.

Webhook authenticity protects event transport, not blockchain finality. A valid HMAC proves that the configured merchant node emitted the event body. It does not prove that the payment can never be reorganized. Merchant fulfilment policy should therefore combine signature verification, idempotency, explicit event-state handling, and a confirmation depth appropriate to the merchant's own risk tolerance.

Threat	Existing control	Residual issue / research need
Transaction forgery	Ed25519 + address/public-key binding	Endpoint/browser key security remains critical
Replay / sequencing	Strict account nonce	Reorg may change canonical sequence context
Machine key grinding	Fork key excludes producer key/signature	Wallet multiplicity still possible
Peer spoofing	Signed hello + node-ID derivation	DDoS and route availability remain operational
DNS rebinding / SSRF	Resolve/pin; reject private targets; no redirects	Resolver/hosting policy still matters
Webhook forgery	HMAC-SHA256 + timestamp tolerance	Secret rotation and merchant idempotency needed
Deep reorganization	Validation + confirmations + deterministic fork choice	No hard finality/checkpoint/cumulative work
Sybil identities	Per-wallet 64 MiB Argon2id work	No stake/identity admission gate

Implementation references: src/PBE/Core/; src/PBE/Network/*; src/PBE/Commerce/WebhookService.php; public/index.php; docs/MAINTAINER-REFERENCE.md*

20. Performance and Scalability Analysis

PBE separates protocol limits from reference-implementation limits. The protocol block maximum is 2 MiB. A minimal signed transfer is approximately 236 bytes and is length-prefixed by four bytes inside block serialization. Ignoring block overhead, this suggests an upper packing bound on the order of 8,700 minimal transfers in a 2 MiB block, or roughly 580 transactions per second at one block every fifteen seconds. A maximum-length payment reference increases transfer size to roughly 300 bytes and lowers the packing bound to roughly 6,900 transfers, or about 460 per second. These figures are serialization ceilings, not measured throughput.

The current producer implementation is more conservative: Node::produceBlock fetches at most 1,000 mempool transactions for block assembly. At perfect 15-second cadence, this creates a local production ceiling of about 66.7 transactions per second before CPU, database, propagation, or signature-validation cost is considered. Raising that limit is an implementation change if block validation rules remain unchanged, but it should be tested carefully because state application and JSON/database writes scale with transaction count.

Argon2id is intentionally memory intensive. A participant ticket allocates 64 MiB and uses operation limit 4. A normal producer caches the proof for the current parent/slot/wallet tuple, avoiding redundant recomputation within that slot. Validators recompute the proof for every candidate block they validate. Consequently, block-validation throughput under adversarial load is constrained by memory bandwidth and Argon2 latency, not only signature verification or database speed.

State root generation currently iterates all accounts, sorts all addresses, hashes each account leaf, and builds a Merkle root. This is simple and deterministic but asymptotically O(A log A) for sorting plus O(A) hashing per root, where A is the number of accounts. At small chain size this is attractive for clarity. At very large account populations, incremental authenticated-state structures or cached subtrees may be needed to avoid full-state root recomputation on every block.

Persistence also stores raw JSON for blocks and transactions in addition to indexed columns. This simplifies explorer/RPC reconstruction and debugging at the cost of storage duplication. SQLite WAL is suitable for a single local standalone process; high-concurrency public deployments are better matched to MySQL/MariaDB. The abstraction keeps consensus semantics shared, but the performance characteristics of the two backends should be benchmarked independently.

Metric	Derived/reference value	Interpretation
Slot target	15 s	Nominal block opportunity cadence
Protocol block size	2,097,152 bytes	Validation hard limit
Minimal signed transfer	~236 bytes	No payment reference
Max-ref signed transfer	~300 bytes	64-byte reference
Wire-format packing ceiling	~8.7k minimal tx/block	Not a benchmark
Wire-format rate ceiling	~580 tx/s	Assumes one max-filled block every slot
Current producer selection	<=1,000 tx/block	Reference implementation behavior
Current producer selection rate	~66.7 tx/s	Before CPU/DB/network limits
Participation work	64 MiB Argon2id, opslimit 4	Per wallet/parent/slot proof

Implementation references: src/PBE/Core/Block.php; src/PBE/Core/Transaction.php; src/PBE/Core/State.php; src/PBE/Node/Node.php; src/PBE/Storage/ChainStore.php

21. Reliability and Operational Behavior

The codebase contains several recovery-oriented design choices. Chain mutations are transactional, state undo is persisted, divergent sync validates before commit, and full-chain replacement remains available when incremental reorganization cannot safely proceed. Bootstrap can reopen when a previously completed node

loses all usable compatible peers. Producer tickets are cached per parent/slot/wallet to avoid repeated memory-hard work. These features collectively make ordinary restart and fork recovery less dependent on manual intervention.

The node also distinguishes synchronization from production. Public status reports best-known peer height and a catching-up/synced state. A well-operated node should avoid treating local tip height as authoritative when known peers advertise a higher compatible tip. The implementation's signed-hello synchronization basis reduces the risk that an unsigned status response alone can trick the node into a chain-download decision.

Standalone deployment is designed around a private local control surface and a separate public peer surface. The example keeps publicRpc false and binds the local UI/RPC to loopback. Public endpoint advertisement is controlled separately and can use configured, manual, or automatically observed addresses. Automatic observation only accepts public IPs and is limited to standalone mode.

Operational reliability ultimately depends on multi-node soak testing. The maintainer roadmap correctly emphasizes long partition/rejoin tests, multi-node duration tests, and broader adversarial review. Consensus correctness in unit/self-tests is necessary but not sufficient to reveal timing races, simultaneous block production, NAT asymmetry, peer churn, database lock behavior, or extended reorganization edge cases.

Implementation references: src/PBE/Node/Node.php; src/PBE/Storage/ChainStore.php; config/standalone.example.php; docs/MAINTAINER-REFERENCE.md

22. Verification and Release Discipline

The project includes a mainnet-oriented self-test covering release identity, address format, amount conversion, transaction signing and tamper resistance, self-transfer rejection, genesis freezing, mining authorization, participation proof verification, zero-balance eligibility, multiplier caps, block signing, fork-key resistance to producer-key rotation, reward issuance, legacy-format rejection, PBE-PAY request behavior, webhook HMAC verification, endpoint normalization, signed peer hello validation, future-slot rejection, schema invariants, operator privacy boundaries, request-body limits, and documentation accessibility.

A notable strength is that tests include negative compatibility assertions. Older transaction and block generations are expected to be rejected rather than silently accepted. Likewise, schemas are checked for removed validator/staking concepts. This supports the release model in which 0.0.1 is a clean mainnet generation rather than a permissive superset of earlier experiments.

The maintainer process additionally calls for PHP linting, JavaScript syntax checks, JSON parsing, schema validation, merchant-example inspection, secret scanning, frozen-genesis verification, checksum generation, and ZIP-integrity testing. For a project distributed as runnable source, these release-hygiene controls are as important as protocol tests because accidental keys, credentials, databases, or logs can create immediate operational compromise even if consensus code is correct.

> **Paper scope** This document does not publish or reproduce any local secret-bearing artifacts from the supplied working snapshot. Configuration examples shown here retain only placeholders and public/default topology values.

Implementation references: tests/selftest.php; docs/MAINTAINER-REFERENCE.md; web/developers/errors.php

23. Comparative Design Context

PBE sits between familiar consensus categories rather than mapping exactly onto one of them. Like proof-of-work systems, participation requires repeated resource expenditure - Argon2id memory/time - and a lower

derived score is advantageous. Like proof-of-stake systems, current holdings influence leader timing. Unlike conventional proof of work, the protocol does not search a nonce until a global difficulty target is met, and chain selection does not sum work. Unlike conventional proof of stake, there is no stake lock, validator registry, delegation, or minimum economic bond.

The resulting mechanism can be described as resource-costed, balance-biased timed participation. Every wallet has one deterministic ticket per slot/parent. The resource cost is attached to evaluating each identity ticket. Holdings then transform the score into earlier eligibility. This is conceptually elegant because it permits zero-balance bootstrap and avoids manual staking, but it also means security arguments from either Nakamoto consensus or classical PoS cannot be imported unchanged.

PBE's use of an account model is closer to many modern smart-contract chains than to Bitcoin's UTXO model, but PBE intentionally omits a VM. That produces straightforward nonce/replay semantics and efficient balance lookup at the expense of parallel transaction independence: two transactions from one sender are strictly ordered and a missing nonce blocks later sequence positions.

The payment layer resembles URI/payment-request conventions seen in other cryptocurrencies, but its exact signed payment reference and merchant-webhook integration are first-class project concerns. This makes PBE notably application-oriented for a very small Layer-1. The tradeoff is that confirmation semantics remain only as strong as the underlying canonical-chain finality model.

Dimension	PBE 0.0.1	Classical PoW reference	Classical PoS reference
Leader opportunity	One deterministic wallet ticket per slot	Hash search / difficulty target	Stake-weighted selection
Resource signal	Argon2id per wallet ticket	Compute/energy	Stake/economic capital
Balance effect	Timing multiplier, bounded	Usually none	Usually direct selection weight
Zero-balance eligibility	Yes	Yes if miner has hardware	Usually no
Stake lock / validator registry	No	No	Common
Chain selection	Height, then deterministic ticket tie-break	Cumulative work	Protocol-specific; often finalized checkpoints
Hard finality	No	No native hard finality	Varies; many add finality

This comparison is intentionally high level. Bitcoin and Ouroboros are cited as historical reference classes, not as security-equivalent systems. PBE requires its own formal model because its ticket determinism, timing window, identity multiplicity, and fork-choice rule create a distinct adversarial landscape.

24. Known Limitations and Protocol Evolution

The source tree itself identifies hard/deep-reorganization policy as the leading post-0.0.1 priority. That ordering is technically justified. A finality upgrade could take several forms - periodic hard checkpoints, weak-subjectivity checkpoints distributed out of band, cumulative participation-resource scoring, a separate voting/finality layer, or a protocol-defined irreversible checkpoint cadence - but each option changes trust assumptions and must be designed explicitly rather than patched into local node policy.

A second priority is adversarial consensus measurement. The project should benchmark Argon2id ticket generation across representative CPUs, integrated/discrete GPUs where applicable, and high-memory parallel systems. Measurements should report tickets per second, memory bandwidth, cost per concurrent identity, and validation throughput. Those numbers can then inform whether 64 MiB/opslimit 4 provides the intended barrier against mass zero-balance identities.

State scaling is another likely future concern. The current full-account state-root computation is excellent for transparency and small networks but may become the dominant per-block cost at large account counts. Potential evolution paths include incremental Merkle trees, sparse Merkle structures, authenticated database

indexes, or state snapshot/checkpoint schemes. Any change that alters stateRoot computation would be consensus breaking and would require an explicit versioned network upgrade.

Protocol-version negotiation is also intentionally minimal in 0.0.1: peers require exact protocol 1.0. That is safe for a first release because incompatible nodes fail closed, but future upgrades need a transition procedure for new block/transaction versions, activation heights, mixed-version peer discovery, and rollback. The maintainer rule that serialization, consensus constants, emission, genesis, and fork choice must not change silently is the correct foundation for such a process.

Finally, independent security review should be prioritized before positioning PBE for high-value settlement. Source clarity and self-tests reduce implementation mistakes, but formal consensus analysis, network simulation, fuzzing of binary/JSON boundaries, adversarial P2P testing, and wallet-origin security review provide different kinds of assurance that unit tests cannot substitute for.

Implementation references: docs/MAINTAINER-REFERENCE.md section 16; src/PBE/Core/HybridConsensus.php; src/PBE/Core/State.php

25. Conclusion

PBE Core 0.0.1 is a surprisingly complete small blockchain implementation. It is not merely a ledger table with a mining loop: it defines frozen mainnet identity, canonical binary serialization, domain-separated cryptography, deterministic state and transaction commitments, account nonces, fee and emission rules, signed peer identity, discovery, fork choice, incremental reorganization, a non-custodial browser wallet, merchant payment requests, exact chain verification, optional authenticated webhooks, and dual MySQL/SQLite deployment targets.

Its strongest engineering quality is coherence across layers. The same signed transfer used by a user wallet is the object indexed for merchant payment matching. The same canonical chain used by the explorer drives confirmation depth and webhook reorg events. The same Core validates server and standalone deployments. The wallet authorizes a producer without surrendering its private key, and the peer layer authenticates chain identity before synchronization. These are good system-design properties because they reduce hidden parallel sources of truth.

Its most important unresolved property is consensus finality under a capable adversary. The hybrid participation mechanism is novel enough that security cannot be inferred from proof-of-work or proof-of-stake literature by analogy. Zero-balance eligibility, deterministic historical tickets, memory-hard per-identity cost, holdings-based timing, and height-first fork choice must be evaluated as one system. Until that work is complete, canonical depth should be treated as a practical risk heuristic rather than irreversible settlement.

As a research and engineering artifact, PBE 0.0.1 establishes a clear baseline from which such work can proceed. Its protocol surface is small enough to model, its constants are explicit, its mainnet identity is frozen, and its application layer already demonstrates a realistic end-to-end payment use case. The next major improvement is therefore not feature proliferation; it is stronger evidence about consensus security, partition behavior, and finality.

Appendix A. Protocol Constants and Derived Metrics

Constant / metric	Value	Notes
Core version	0.0.1	Reference release
Peer protocol	1.0	Exact compatibility
Network ID	1	Mainnet only
Chain ID	PBE-MAINNET	Frozen
Address HRP	pbe	Bech32m
Block version	4	Only supported format
Transaction version	2	Only supported format
Transaction type	1	Transfer
PBE-PAY version	1	Merchant request protocol
Base units	100,000,000 / PBE	8 decimal places
Slot length	15 s	Consensus timing unit
Initial reward	5 PBE	Height > 0
Reward epoch	8,409,600 blocks	~3.997 years at target slots
Approx. terminal supply	84,095,998.906752 PBE	Integer base-unit halving
Positive reward epochs	29	Epochs 0 through 28
Max block bytes	2,097,152	~2 MiB
Max transaction bytes	65,536	Validation limit
Min fee defaults	1,000 + 2 x signed bytes	Base-unit fee formula
Argon2 proof bytes	32	Participation ticket
Argon2 ops limit	4	Consensus constant
Argon2 memory	67,108,864 bytes	64 MiB
Eligibility window	1-14 s	Within 15-second slot
Multiplier base/cap	1.00x / 2.50x	2.50x nominal cap
Economically reachable max multiplier	~2.4558x	At theoretical entire terminal supply in one wallet
P2P response cap	64 MiB	HTTP body hard cap
P2P block batch	16	Sync batch size
Default Commerce confirmations	12	Risk policy, not finality
High-value documented confirmations	30	Risk policy, not finality
Max Commerce confirmations	1,000	Request clamp
Webhook default TTL	86,400 s	1 day
Webhook max TTL	604,800 s	7 days

Appendix B. Canonical Wire-Format Field Maps

B.1 Transaction v2 unsigned serialization

Order	Field	Encoding / constraint
1	version	u16; must be 2
2	networkId	u32; mainnet 1
3	type	u16; transfer 1
4	senderPublicKey	32 raw bytes
5	senderAddress	u32-length-prefixed text
6	nonce	u64
7	fee	u64
8	timestamp	u64
9	payload	u32-length-prefixed transfer payload
9a	payload.recipient	u32-length-prefixed text
9b	payload.amount	u64
9c	payload.paymentReference	u32-length-prefixed text, may be empty
10	signature	64 bytes appended to unsigned bytes

B.2 Block v4 header serialization

Order	Field	Encoding / constraint
1	version	u16; 4
2	networkId	u32
3	height	u64
4	slot	u64
5	previousBlockHash	32 bytes
6	timestamp	u64
7	producerPublicKey	32 bytes
8	rewardAddress	length-prefixed text
9	miningWalletPublicKey	32 bytes
10	miningAuthorizationSignature	64 bytes
11	participationProof	32 bytes
12	transactionRoot	32 bytes
13	stateRoot	32 bytes
14	consensusRoot	32 bytes
15	producerSignature	64 bytes stored outside header hash/signing bytes

Appendix C. Public Endpoint Catalog

Method	Endpoint	Role
GET	/rpc/status	Node status
GET	/rpc/consensus	Consensus parameters
GET	/rpc/block/{height}	Block by height
GET	/rpc/balance/{address}	Account balance and nonce
GET	/rpc/address/{address}/transactions	Address transactions
GET	/rpc/transaction/{txid}	Transaction by TXID
POST	/rpc/broadcast	Broadcast signed transaction
GET	/rpc/commerce/catalog	Commerce endpoint catalog
GET	/rpc/commerce/info	Commerce capabilities
POST	/rpc/commerce/payment-request	Create PBE-PAY request
GET	/rpc/commerce/payment/{reference}	Verify exact PBE-PAY payment
POST	/rpc/commerce/webhooks	Register merchant webhook
GET	/rpc/commerce/webhooks/{id}	Inspect merchant webhook
POST	/rpc/commerce/webhooks/{id}/cancel	Cancel merchant webhook
GET	/p2p/status	P2P node status
POST	/p2p/hello	Signed peer handshake
POST	/p2p/transaction	Peer transaction relay
POST	/p2p/block	Peer block relay
GET	/p2p/blocks	Canonical block batch
GET	/p2p/peers	Verified peer hints

Appendix D. Persistence Schema Map

Table	Selected fields / indexes	Role
chain_meta	meta_key, meta_value	Chain identity, tip, supply, settings
blocks	height, hash, slot, previous_hash, roots, producer, reward, raw_json	Canonical chain
transactions	txid, block_height, sender, nonce, fee, payload, payment indexes	Confirmed transaction index
accounts	address, balance, nonce, updated_height	Current state
mempool	txid, sender, nonce, fee, tx JSON, payment indexes	Validated unconfirmed tx
peers	node_id, endpoint, key, protocol, tip, last_seen, failures	Peer cache
state_undo	block_height, address, previous	Incremental reorg state

Table	Selected fields / indexes	Role
	balance/nonce	
commerce_webhook_subscriptions	exact payment tuple, callback, state, expiry	Optional merchant watch state
commerce_webhook_events	event ID/type/body/attempts/delivery status	Optional at-least-once queue

Appendix E. Example Configuration Matrix

Setting	Generic web example	Primary web example	Standalone example
deployment.mode	web	web	standalone
database.driver	mysql	mysql	sqlite
database credential	CHANGE_ME placeholder	CHANGE_ME placeholder	Not applicable to SQLite
network timeout	5 s	5 s	5 s
allowPrivatePeers	false	false	false
bootstrap seeds	Public Paybyte seed	none	Public Paybyte seed
local UI/RPC	web server surface	web server surface	127.0.0.1:8024
P2P listener	web endpoint	public Paybyte endpoint	0.0.0.0:8025
publicRpc	web routing policy	web routing policy	false
Commerce webhooks	disabled	disabled	disabled
Webhook secret fields	blank	blank	blank

Only bundled example/default configuration values are shown. No local production credentials or secret values are included.

Appendix F. Emission Schedule

Epoch	Start height	End height	Reward PBE	Epoch issuance	Cumulative PBE	Approx. end yr
0	1	8,409,600	5	42,048,000	42,048,000.000000	4.00
1	8,409,601	16,819,200	2.5	21,024,000	63,072,000.000000	7.99
2	16,819,201	25,228,800	1.25	10,512,000	73,584,000.000000	11.99
3	25,228,801	33,638,400	0.625	5,256,000	78,840,000.000000	15.99
4	33,638,401	42,048,000	0.3125	2,628,000	81,468,000.000000	19.99
5	42,048,001	50,457,600	0.15625	1,314,000	82,782,000.000000	23.98
6	50,457,601	58,867,200	0.078125	657,000	83,439,000.000000	27.98
7	58,867,201	67,276,800	0.0390625	328,500	83,767,500.000000	31.98
8	67,276,801	75,686,400	0.01953125	164,250	83,931,750.000000	35.98
9	75,686,401	84,096,000	0.00976562	82,124.957952	84,013,874.957952	39.97
10	84,096,001	92,505,600	0.00488281	41,062.478976	84,054,937.436928	43.97
11	92,505,601	100,915,200	0.0024414	20,531.19744	84,075,468.634368	47.97
12	100,915,201	109,324,800	0.0012207	10,265.59872	84,085,734.233088	51.97
13	109,324,801	117,734,400	0.00061035	5,132.79936	84,090,867.032448	55.96
14	117,734,401	126,144,000	0.00030517	2,566.357632	84,093,433.390080	59.96
15	126,144,001	134,553,600	0.00015258	1,283.136768	84,094,716.526848	63.96
16	134,553,601	142,963,200	0.00007629	641.568384	84,095,358.095232	67.95
17	142,963,201	151,372,800	0.00003814	320.742144	84,095,678.837376	71.95
18	151,372,801	159,782,400	0.00001907	160.371072	84,095,839.208448	75.95
19	159,782,401	168,192,000	0.00000953	80.143488	84,095,919.351936	79.95
20	168,192,001	176,601,600	0.00000476	40.029696	84,095,959.381632	83.94
21	176,601,601	185,011,200	0.00000238	20.014848	84,095,979.396480	87.94
22	185,011,201	193,420,800	0.00000119	10.007424	84,095,989.403904	91.94
23	193,420,801	201,830,400	0.00000059	4.961664	84,095,994.365568	95.94
24	201,830,401	210,240,000	0.00000029	2.438784	84,095,996.804352	99.93
25	210,240,001	218,649,600	0.00000014	1.177344	84,095,997.981696	103.93
26	218,649,601	227,059,200	0.00000007	0.588672	84,095,998.570368	107.93
27	227,059,201	235,468,800	0.00000003	0.252288	84,095,998.822656	111.93
28	235,468,801	243,878,400	0.00000001	0.084096	84,095,998.906752	115.92

Appendix G. Source Module Map

Source file	Class	Selected methods	Layer
src/PBE/Commerce/PaymentProtocol.php	PaymentProtocol	createReference, normalizeReference, normalizeConfirmations, normalizeLabel, buildUri, request	Merchant payment / webhook layer
src/PBE/Commerce/	WebhookService	__construct, enabled,	Merchant payment / webhook layer

Source file	Class	Selected methods	Layer
WebhookService.php		managementConfigured, assertManagementToken, createSubscription, subscription, cancel, process ...	
src/PBE/Core/Address.php	Address	fromPublicKey, validate, hrpForNetwork	Consensus/data/cryptographic primitive
src/PBE/Core/Amount.php	Amount	parsePbe, formatPbe	Consensus/data/cryptographic primitive
src/PBE/Core/Bech32m.php	Bech32m	encode, decode, hrpExpand, polymod, createChecksum, verifyChecksum, convertBits	Consensus/data/cryptographic primitive
src/PBE/Core/Binary.php	Binary	u8, u16, u32, u64, varBytes, text, hex, assertUnsignedIntegerString ...	Consensus/data/cryptographic primitive
src/PBE/Core/Block.php	Block	__construct, createHybrid, genesis, headerBytes, forkChoiceKey, serializedSize, hash, hashHex ...	Consensus/data/cryptographic primitive
src/PBE/Core/Config.php	Config	__construct, fromFile, get, path	Consensus/data/cryptographic primitive
src/PBE/Core/Crypto.php	Crypto	keypairFromSeed, generateKeypair, sha256, sha256Hex, sign, verify, txSigningHash, blockSigningHash ...	Consensus/data/cryptographic primitive
src/PBE/Core/Emission.php	Emission	rewardForHeight	Consensus/data/cryptographic primitive
src/PBE/Core/Genesis.php	Genesis	__construct, fromFile, networkId, chainId, timestamp, state, block, toArray ...	Consensus/data/cryptographic primitive
src/PBE/Core/HybridConsensus.php	HybridConsensus	consensusRoot, multiplierSchedule, authorizationHash, verifyAuthorization, challenge, proof, verifyProof, rawScore ...	Consensus/data/cryptographic primitive
src/PBE/Core/KeyFile.php	KeyFile	save, load	Consensus/data/cryptographic primitive
src/PBE/Core/Merkle.php	Merkle	root	Consensus/data/cryptographic primitive
src/PBE/Core/State.php	State	__construct, totalSupply, account, setAccount, setTotalSupply, applyTransaction, assertSenderNonce, creditProducer ...	Consensus/data/cryptographic primitive
src/PBE/Core/Transaction.php	Transaction	__construct, createUnsignedTransfer, normalizePaymentReference, sign, verifySignature, unsignedBytes, signedBytes, txid ...	Consensus/data/cryptographic primitive
src/PBE/Network/PeerClient.php	PeerClient	__construct, get, post, pinnedUrl, resolveHost, isPublicIp, request	P2P endpoint and HTTP networking
src/PBE/Network/PeerDiscovery.php	PeerDiscovery	normalizeEndpoint, isPrivateOrLocalEndpoint, isPublicEndpoint, endpointsFromHints	P2P endpoint and HTTP networking
src/PBE/Node/Node.php	Node	__construct, status, miningStatusForWallet, ensureProducerIdentity, participationProof, helloSigningBytes, hello, acceptHello ...	Node orchestration / operator runtime
src/PBE/Node/OperatorWallet.php	or	__construct, nodeId, status, configured, address, miningEnabled, setMiningEnabled, walletPublicKey ...	Node orchestration / operator runtime
src/PBE/Node/Runtime.php	Runtime	__construct	Node orchestration / operator runtime
src/PBE/Storage/ChainStore.php	ChainStore	__construct, migrateSchema, initializeGenesis, tip, loadState, addMempool, mempool, applyBlock ...	Persistence and canonical chain state
src/PBE/Storage/Database.php	Database	__construct, driver, isSqlite, isMysql, begin, commit, rollback, ensureSchema ...	Persistence and canonical chain state
src/PBE/Storage/DbConnection.php	DbConnection	__construct, fromMysqli, fromPdo, mysqli, pdo, driver, begin, commit ...	Persistence and canonical chain state
src/PBE/Storage/DbResult.php	DbResult	__construct, fromMysqli, fromPdo, fetch_assoc, fetch_row, fetch_all	Persistence and canonical chain state

Source file	Class	Selected methods	Layer
src/PBE/Storage/DbStatement.php	DbStatement	__construct, fromMysqli, fromPdo, bind_param, execute, get_result	Persistence and canonical chain state

References

[1] Paybyte Blockchain Engine (PBE) Core 0.0.1 source tree. Primary source archive supplied for this research paper; includes consensus, node, network, storage, Commerce, wallet, schemas, OpenAPI, tests, and example configurations.

[2] RFC 8032 - Edwards-Curve Digital Signature Algorithm (EdDSA). https://www.rfc-editor.org/rfc/rfc8032.html

[3] RFC 9106 - Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. https://www.rfc-editor.org/rfc/rfc9106.html

[4] BIP 350 - Bech32m format. https://github.com/bitcoin/bips/blob/master/bip-0350.mediawiki

[5] NIST FIPS 180-4 - Secure Hash Standard (SHS). https://csrc.nist.gov/pubs/fips/180-4/upd1/final

[6] RFC 2104 - HMAC: Keyed-Hashing for Message Authentication. https://www.rfc-editor.org/rfc/rfc2104.html

[7] Satoshi Nakamoto - Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/en/bitcoin-paper

[8] Kiayias, Russell, David, and Oliynykov - Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. https://eprint.iacr.org/2016/889.pdf

Research Notes on External Literature

RFC 8032 is used only to contextualize the Ed25519 primitive selected by PBE. RFC 9106 contextualizes Argon2 as a memory-hard construction and provides parameter-selection guidance; PBE's exact use and consensus security remain specific to PBE. BIP 350 is cited for the Bech32m checksum mechanism. FIPS 180-4 is cited for SHA-256. RFC 2104 is cited for HMAC. The Bitcoin and Ouroboros papers provide historical comparison points for proof-of-work and proof-of-stake chain design; neither paper is evidence that PBE inherits their security properties.