CONFIDENTIAL

UNITED KINGDOM INTELLECTUAL PROPERTY OFFICE

PATENT APPLICATION

Applicant

The Bitcoin Corporation Ltd

Inventor

Richard Boase

Date of Preparation

8 March 2026

Title of Invention

Verifiable Derived State Computation in Overlay Networks Using Proof of Indexing Consensus Applied to UTXO Event Streams

Field of the Invention

The present invention relates to distributed computing systems built upon blockchain-based overlay networks, and more particularly to methods and systems for verifying the correctness of derived state computations performed by indexing nodes that process unspent transaction output (UTXO) event streams. The invention addresses the problem of trustless verification in systems where raw on-chain data (UTXOs, transaction graphs) must be transformed into aggregate derived state (balances, eligibility tables, rankings) by off-chain service providers, and where no single provider can be trusted to compute that state honestly.

Background of the Invention

Problem Statement

1. **The Derived State Trust Gap**

   Blockchain overlay networks such as those defined by BRC-24 (Overlay Lookup Services) maintain two categories of state: (a) raw UTXO data synchronised between nodes via graph-aware sync protocols (e.g., GASP — Graph-Aware Sync Protocol), and (b) derived state computed locally by each node from admit and spend events in the UTXO event stream. Raw UTXO data is inherently verifiable — each output has a Merkle proof rooting to a block header. Derived state, however, is computed off-chain in databases maintained by lookup service providers. A user querying a lookup service for an aggregate balance, a staking eligibility table, or a dividend accrual figure has no cryptographic guarantee that the provider computed it correctly from the underlying UTXO events.

2. **Scalability vs Verifiability Trade-off**

   Representing every piece of derived state as an on-chain UTXO would eliminate the trust gap but creates prohibitive chain bloat. A content network processing millions of micro-sale events per day cannot mint a new UTXO for each incremental balance change. The industry standard is to derive and serve such state via lookup services (the "Meter" pattern), accepting a trust trade-off. No existing system resolves this trade-off — providing the scalability of derived state with the verifiability of on-chain settlement.

3. **Competing Indexers Without Consensus**

   In federated overlay networks where multiple providers host the same overlay code (discovered via SHIP/SLAP under BRC-88), each provider independently computes derived state from its local copy of the UTXO graph. There is no consensus mechanism to detect or penalise a provider that computes incorrect derived state — whether through bugs, data loss, or deliberate manipulation. Users must either trust a single provider or manually cross-check multiple providers, neither of which scales.

4. **Absence of Economic Incentive for Honest Computation**

   Existing overlay hosting compensation models (e.g., CARS — Cloud Automated Runtime System) pay hosts for compute resources consumed (CPU, memory, network, disk) but not for computational correctness. A host is paid the same whether its derived state is accurate or fabricated. There is no mechanism to reward nodes that maintain provably correct state or penalise those that serve incorrect state.

Prior Art Limitations

Traditional blockchain indexers (block explorers, UTXO set trackers) index raw chain data but do not compute application-specific derived state. They are observation tools, not computation engines.

Proof of Work (PoW) mining verifies transaction validity and block ordering but has no concept of overlay-specific derived state. A miner validates that a transaction's inputs are unspent; it does not validate that a lookup service correctly computed an aggregate balance from a stream of admitted transactions.

Proof of Stake (PoS) systems select block producers based on economic stake but do not address the correctness of off-chain computations derived from on-chain data.

Existing Proof of Indexing concepts (as applied in the Applicant's prior ClawMiner patent application) use indexing work as a proof-of-work function for token minting. However, the prior art uses Proof of Indexing solely as a work function — the output is a minted token reward, not a verifiable attestation of derived state correctness. The prior art does not define a consensus mechanism among competing indexers, does not specify how derived state computations are compared across nodes, and does not provide a challenge-response protocol for disputing incorrect state.

The Applicant's prior Decentralized Dividend Distribution patent defines a system where competing miners race to distribute dividends, using Proof of Indexing to earn priority. That system addresses distribution logistics (who delivers the payout) but not verification of the underlying balance computation (is the payout amount correct). The dividend distribution patent assumes correct balances; the present invention ensures they are correct.

The Graph Protocol (GRT, 2020–present) — The Graph provides a decentralised indexing protocol for querying blockchain data. Indexers stake GRT tokens, process "subgraphs" (indexing specifications), and are subject to dispute resolution. Curators signal which subgraphs are valuable. However, The Graph's dispute mechanism relies on a centralised Arbitration Council to adjudicate disputes — a disputed indexer's stake is slashed by council decision, not by an automated cryptographic verification protocol. The Graph does not define a state commitment protocol where indexers publish Merkle roots over derived state at deterministic checkpoints, does not provide a binary narrowing dispute protocol enabling automated identification of the first divergent event, and does not generate Proof of Indexing attestations linking derived state back to Merkle-proven on-chain UTXO data. The Graph also operates exclusively on account-based blockchains (Ethereum) and does not address UTXO-based overlay state. The present invention provides fully automated, cryptographically verifiable state verification without reliance on human arbitrators.

Arbitrum Bisection Protocol (Offchain Labs, 2018–present) — Arbitrum uses a bisection-based interactive fraud proof protocol to verify the correctness of layer-2 state transitions on Ethereum. When a validator disputes a state assertion, the parties engage in a binary narrowing process to identify the single instruction at which their computations diverge. The present invention adapts this binary narrowing concept to a fundamentally different domain: overlay-specific derived state computed from UTXO event streams, rather than virtual machine instruction execution. The present invention's dispute protocol narrows over UTXO event intervals (not VM instructions), uses Proof of Indexing attestations (not execution traces), operates across multiple independent indexer nodes (not two-party validator disputes), and applies to application-specific state (token balances, lookup tables) rather than generic VM state transitions. The binary narrowing mechanism is a shared technique; the application domain, the attestation structure, and the multi-node consensus model are distinct.

TrueBit (Jason Teutsch and Christian Reitwießner, 2017–present) — TrueBit provides a verification game for off-chain computations, where a solver posts a result and a verifier can challenge it through interactive bisection to identify computational errors. TrueBit addresses generic off-chain computation verification — it does not specifically address blockchain indexing, overlay-specific derived state, or UTXO event stream processing. TrueBit's verification game requires an on-chain referee contract to adjudicate each round of bisection, incurring gas costs per round. The present invention operates within an overlay network's own peer-to-peer infrastructure, uses deterministic checkpoint intervals tied to event counts (not arbitrary computation steps), and generates Proof of Indexing attestations that can be independently verified by any participant without on-chain adjudication.

IBM checkpoint patents for distributed computation (various, 2000–2020) — IBM holds multiple patents on checkpoint-based state recovery in distributed systems, including periodic snapshots for fault tolerance and rollback recovery (e.g., US 10,586,210 B2, "Blockchain Checkpoints and Certified Checkpoints", filed 2016). These systems address fault tolerance (recovering from crashes by restoring the last checkpoint) or ledger integrity verification (compressing world state into certified hash representations) rather than the specific problem of verifying correctness of application-specific derived state computations by competing indexer nodes. IBM checkpoints are internal consistency or audit mechanisms; the present invention's checkpoints are externally verifiable state commitments published to an overlay network for cross-node comparison and automated dispute resolution via binary narrowing.

INDECURE — Stateless and Verifiable Execution Layer for Meta-Protocols on Bitcoin (RiemaLabs, July 2024) — INDECURE proposes a modular indexer architecture for Bitcoin meta-protocols (BRC-20, Runes, etc.) using polynomial commitments as checkpoints to verify state transitions. This is the closest prior art to the present invention's application domain — both address the correctness of derived state computed by indexers processing

Bitcoin meta-protocol transactions. However, INDECURE uses polynomial commitments (a specific algebraic construction) as its checkpoint format, whereas the present invention uses Merkle roots over canonical byte representations. INDECURE does not define a multi-node consensus mechanism (it provides a verification method, not a competitive consensus protocol), does not specify a binary narrowing dispute protocol for isolating the first divergent event, and does not integrate Proof of Indexing attestations linking derived state to specific UTXO event sequences. INDECURE provides a verification primitive for individual state transitions; the present invention provides a complete overlay consensus system with checkpoint comparison, dispute resolution, and competitive correctness incentives.

No existing system combines: a state commitment protocol publishing Merkle roots over overlay-derived state at deterministic checkpoint intervals; cross-node verification by comparing independently computed state commitments without a central coordinator; Proof of Indexing attestations linking derived state back to Merkle-proven on-chain UTXO event sequences; a binary narrowing challenge-response dispute protocol adapted for UTXO event intervals; and competitive correctness incentives rewarding nodes that consistently match network consensus.

Summary of the Invention

The present invention provides a system and method for verifiable derived state computation in overlay networks, comprising:

a. A **State Commitment Protocol** whereby each indexing node, upon processing a batch of UTXO admit/spend events from a synchronised transaction graph, computes a deterministic state commitment (a Merkle root or hash digest) over the resulting derived state, and publishes this commitment to the overlay network;

b. A **Cross-Node Verification Mechanism** whereby multiple independent nodes processing the same UTXO event stream compare their state commitments at defined checkpoints, producing a consensus view of the correct derived state without requiring a central coordinator;

c. A **Proof of Indexing Attestation** whereby a node generates a compact cryptographic proof demonstrating that its derived state was computed by processing a specific, verifiable sequence of UTXO events — linking the derived state back to the Merkle-proven on-chain data from which it was derived;

d. A **Challenge-Response Dispute Protocol** whereby any participant (node, user, or auditor) can challenge a node's claimed derived state by requesting the Proof of Indexing Attestation, replaying the specified UTXO events, and independently verifying the computation — with economic consequences (token slashing or reputation penalties) for nodes that fail the challenge;

e. A **Competitive Correctness Incentive** whereby nodes that consistently produce state commitments matching the network consensus earn enhanced rewards (overlay-specific tokens or preferential service routing), creating an economic incentive for honest, accurate, and timely state computation.

Detailed Description of the Invention

1. System Architecture

1.1 Overlay Network Foundation

The system operates within a UTXO-based blockchain overlay network as defined by BRC-24 and related standards. The overlay comprises one or more TopicManagers that admit on-chain transaction events into the overlay, and one or more LookupServices that maintain both raw UTXO indexes and derived state computed from the event stream. Multiple independent nodes host the same overlay code, discovered via SHIP/SLAP (BRC-88), and synchronise their raw UTXO state via GASP (Graph-Aware Sync Protocol).

1.2 UTXO Event Stream

Each TopicManager produces an ordered stream of **admit events** (new UTXOs entering the overlay) and **spend events** (UTXOs being consumed). In the preferred embodiment, the overlay tracks four event types across four topics:

- **Ticket events** — creation and consumption of content-access tokens (tradeable UTXOs representing access rights to digital content)
- **Sale events** — transfer of tickets between parties for payment (revenue events)
- **Stake events** — temporary locking of tickets by holders (removing from circulation, creating dividend eligibility)
- **Claim events** — payout transactions where accumulated dividends are settled as UTXOs

Each event carries a Merkle proof linking it to a confirmed block, providing an unforgeable chain of provenance.

1.3 Derived State Engine

The LookupService maintains derived state by processing the UTXO event stream using deterministic rules. In the preferred embodiment:

- On each **sale admit event**: the engine identifies which tickets for the relevant content item are currently staked, and increments a `dividends_owed` balance for each staker proportionally to their stake.
- On each **claim spend event**: the engine decrements the staker's `dividends_owed` balance by the claimed amount.
- On each **stake admit event**: the engine records the staker's position and eligibility start time.
- On each **stake spend event** (unstake): the engine removes the staking position.

This is the "Meter" pattern — BRC-24 increment/decrement on admit/spend — extended with the verification mechanisms described below.

1.4 State Commitment Protocol

At defined **checkpoint intervals** (every N events, every T seconds, or every B blocks), each node computes a **State Commitment** over its current derived state. The State Commitment is a deterministic hash computed as follows:

1. Serialise all derived state entries (e.g., all `dividends_owed` balances for all stakers across all content items) in a canonical ordering (lexicographic by staker public key, then by content identifier).
2. Construct a Merkle tree over the serialised entries.
3. The Merkle root is the State Commitment for that checkpoint.

1.4.1 Canonical Serialisation Algorithm

Each derived state entry is serialised as a fixed-format byte sequence:

```
ENTRY := PUBKEY (33 bytes, compressed SEC) ||
         CONTENT_ID (32 bytes, SHA-256 hash of content identifier) ||
         DIVIDENDS_OWED (8 bytes, uint64 little-endian, in base token
units) ||
         STAKE_COUNT (4 bytes, uint32 little-endian) ||
         LAST_EVENT_SEQ (8 bytes, uint64 little-endian)
```

Total entry size: 85 bytes fixed.

Entries are sorted by a compound key: first by PUBKEY in unsigned byte-lexicographic order (33 bytes compared left-to-right), then by CONTENT_ID in unsigned byte-lexicographic order. This ordering is deterministic and language-independent, ensuring all compliant implementations produce identical serialisation from identical state.

1.4.2 Merkle Tree Construction

The State Commitment Merkle tree is constructed as follows:

1. Each serialised entry is hashed with SHA-256 to produce a 32-byte leaf hash: `leaf[i] = SHA-256(ENTRY[i])` .
2. If the number of leaves is odd, the final leaf is duplicated to produce an even count.
3. Leaves are paired sequentially: `(leaf[0], leaf[1])` , `(leaf[2], leaf[3])` , etc.
4. Each parent node is computed as: `parent = SHA-256(0x01 || left_child || right_child)` , where `0x01` is a single domain-separation byte distinguishing internal nodes from leaves (leaves use `0x00` : `leaf[i] = SHA-256(0x00 || ENTRY[i])`).
5. Steps 2–4 repeat on each successive level until a single root remains.
6. The root is the 32-byte State Commitment.

The domain-separation bytes (`0x00` for leaves, `0x01` for internal nodes) prevent second-preimage attacks where a crafted entry could be misinterpreted as an internal node.

1.4.3 Checkpoint Descriptor Format

The State Commitment is paired with a **Checkpoint Descriptor** with the following exact binary layout:

```
CHECKPOINT_DESCRIPTOR :=
  VERSION           (1 byte,  uint8, currently 0x01)
  CHECKPOINT_SEQ    (8 bytes, uint64 little-endian, monotonically
increasing sequence number)
  OVERLAY_ID        (32 bytes, SHA-256 hash of the overlay topic
identifier)
  STATE_ROOT        (32 bytes, the State Commitment Merkle root)
  LAST_EVENT_HASH   (32 bytes, SHA-256 hash of the last UTXO event
processed)
  EVENT_COUNT       (8 bytes, uint64 little-endian, count of events
processed since prior checkpoint)
  TIMESTAMP         (8 bytes, uint64 little-endian, Unix epoch seconds)
  NODE_PUBKEY       (33 bytes, compressed SEC public key of the
attesting node)
  SIGNATURE         (64 bytes, Schnorr signature over SHA-256(VERSION ||
CHECKPOINT_SEQ || OVERLAY_ID || STATE_ROOT || LAST_EVENT_HASH ||
EVENT_COUNT || TIMESTAMP))
```

Total descriptor size: 218 bytes fixed. The signature covers all preceding fields (excluding NODE_PUBKEY and the signature itself) to prevent tampering.

Nodes broadcast their State Commitments to the overlay network via a dedicated synchronisation channel.

1.5 Cross-Node Verification

When a node receives State Commitments from peer nodes for the same checkpoint:

1. **Agreement check**: If the received commitment matches the node’s own commitment, the peer is recorded as agreeing. No further action is required.
2. **Disagreement detection**: If the received commitment differs, the node initiates the Dispute Protocol (Section 1.7).
3. **Consensus threshold**: A State Commitment is considered **verified** when a configurable supermajority (e.g., 2/3 or 3/4) of active nodes for that overlay produce matching commitments for the same checkpoint.

Nodes that consistently agree with the consensus are awarded **Correctness Score** points, which influence their eligibility for enhanced rewards and preferential routing.

1.6 Proof of Indexing Attestation

A Proof of Indexing (PoI) Attestation is a compact cryptographic proof that a node can generate on demand, demonstrating that its derived state at a given checkpoint was computed by processing a specific sequence of UTXO events. The attestation contains:

1. **Event Range**: The first and last UTXO event hashes in the processing window.
2. **Event Merkle Proof**: A Merkle tree over all event hashes in the window, proving completeness (no events skipped or fabricated).
3. **Intermediate State Hashes**: Hash of the derived state at configurable sub-intervals within the window, allowing a challenger to narrow the point of divergence via binary search.
4. **State Commitment**: The final derived state Merkle root.
5. **Node Signature**: The node's cryptographic signature over all of the above.

The Event Merkle Proof is anchored to the underlying UTXO Merkle proofs — each event hash incorporates the UTXO's own Merkle path to a block header. This creates a chain of provenance from block headers through UTXO events to derived state.

1.6.1 Attestation Binary Format

The PoI Attestation is structured as a self-describing binary message:

```
POI_ATTESTATION :=
  VERSION                (1 byte, uint8, currently 0x01)
  CHECKPOINT_SEQ         (8 bytes, uint64 LE, the checkpoint this
attestation covers)
  FIRST_EVENT_HASH       (32 bytes, SHA-256 of the first UTXO event in
the window)
  LAST_EVENT_HASH        (32 bytes, SHA-256 of the last UTXO event in
the window)
  TOTAL_EVENTS           (8 bytes, uint64 LE, total event count in
window)
  EVENT_MERKLE_ROOT      (32 bytes, root of the Event Merkle Tree)
  INTERMEDIATE_COUNT     (4 bytes, uint32 LE, number of intermediate
state hashes)
  INTERMEDIATE_HASHES    (INTERMEDIATE_COUNT × 32 bytes, ordered state
hashes)
  STATE_COMMITMENT       (32 bytes, final derived state Merkle root)
  NODE_PUBKEY            (33 bytes, compressed SEC)
  SIGNATURE              (64 bytes, Schnorr signature over SHA-256 of
all preceding fields)
```

1.6.2 Intermediate State Hash Spacing

Intermediate state hashes are recorded at regular intervals of every `INTERVAL = max(1, ceil(TOTAL_EVENTS / 1024))` events, yielding at most 1024 intermediate hashes per attestation window. Concretely:

- For a window of 10,000 events with INTERVAL = 10: hashes are recorded after processing events at indices 10, 20, 30, …, 10000.
- For a window of 500 events with INTERVAL = 1: a hash is recorded after every event (500 intermediate hashes).
- Each intermediate hash is computed as `SHA-256(SERIALISED_DERIVED_STATE)` at that point, using the same canonical serialisation defined in Section 1.4.1.

This spacing enables binary narrowing (Section 1.7) to isolate a divergent event within `ceil(log2(1024)) = 10` rounds in the worst case for the initial phase, then `ceil(log2(INTERVAL))` additional rounds to pinpoint the exact event within the narrowed sub-interval.

1.6.3 Event Merkle Tree Construction

The Event Merkle Tree is constructed identically to the State Commitment Merkle tree (Section 1.4.2), but over event hashes rather than state entries:

1. Each UTXO event is hashed as: `event_hash[i] = SHA-256(TXID || VOUT || EVENT_TYPE || UTXO_MERKLE_ROOT)`, where TXID is 32 bytes, VOUT is 4 bytes (uint32 LE), EVENT_TYPE is 1 byte (0x01=admit, 0x02=spend), and UTXO_MERKLE_ROOT is the 32-byte Merkle root from the event's block-inclusion proof.
2. Leaves use domain separation byte `0x00`; internal nodes use `0x01`, as in Section 1.4.2.
3. The resulting root is the EVENT_MERKLE_ROOT field in the attestation.

This construction allows a verifier to confirm that a specific event was included in the attestation window by checking a standard Merkle inclusion proof of `O(log2(TOTAL_EVENTS))` hashes.

1.7 Challenge-Response Dispute Protocol

Any participant can challenge a node's derived state:

1. **Challenge Initiation**: The challenger identifies the checkpoint at which they believe the node's state diverges and requests the node's PoI Attestation for that checkpoint window.
2. **Attestation Delivery**: The challenged node provides its PoI Attestation.
3. **Event Replay**: The challenger (or any third party) replays the events listed in the attestation through the deterministic state computation rules, producing their own state

commitment.

4. **Binary Narrowing**: If the final states differ, the challenger examines the intermediate state hashes to identify the specific sub-interval where divergence occurred, then requests the detailed event list for that sub-interval.
5. **Fault Identification**: The process narrows to the specific event where the challenged node's computation diverged from the deterministic rules.
6. **Resolution**:
   - If the challenged node's computation is incorrect: the node's Correctness Score is penalised, and if the overlay uses a staking/slashing model, a portion of the node's staked tokens is slashed and awarded to the challenger.
   - If the challenged node's computation is correct (the challenger was wrong or malicious): no penalty to the challenged node. Repeated frivolous challenges may be rate-limited.

1.7.1 Binary Narrowing Algorithm — Pseudocode

The binary narrowing procedure identifies the exact event at which a node's computation diverged from the correct result. Let `IH[0..K-1]` be the K intermediate state hashes in the attestation, spaced at INTERVAL events apart.

```
function BINARY_NARROW(challenger, challenged_node, attestation):
    K = attestation.INTERMEDIATE_COUNT
    lo = 0
    hi = K - 1

    // Phase 1: Narrow to a sub-interval of INTERVAL events
    // Requires ceil(log2(K)) rounds, at most 10 for K <= 1024
    while lo < hi:
        mid = floor((lo + hi) / 2)
        challenger_hash = challenger.recompute_state_at(mid * INTERVAL)
        if challenger_hash == attestation.INTERMEDIATE_HASHES[mid]:
            lo = mid + 1    // divergence is after this point
        else:
            hi = mid        // divergence is at or before this point
    // lo == hi == the index of the first divergent intermediate hash

    // Phase 2: Request detailed event list for the sub-interval
    sub_start = (lo == 0) ? 0 : (lo - 1) * INTERVAL
    sub_end   = lo * INTERVAL
    events = challenged_node.request_events(sub_start, sub_end)

    // Phase 3: Binary search within the sub-interval
    // Requires ceil(log2(INTERVAL)) additional rounds
    elo = 0
    ehi = len(events) - 1
    while elo < ehi:
        emid = floor((elo + ehi) / 2)
        state = challenger.replay_events(events[0..emid])
        node_state = challenged_node.request_state_after_event(sub_start
+ emid)
        if state == node_state:
            elo = emid + 1
        else:
            ehi = emid

    // elo == ehi == the index of the divergent event
    return events[elo]
```

Total rounds: `ceil(log2(K)) + ceil(log2(INTERVAL))`. For a window of 1,000,000 events with K=1024 and INTERVAL=977, this is at most `10 + 10 = 20` interactive rounds to pinpoint one event out of one million.

1.7.2 Challenge/Response Message Format

Each message in the dispute protocol uses the following binary envelope:

```
DISPUTE_MESSAGE :=
  MSG_TYPE          (1 byte: 0x01=CHALLENGE_INIT,
0x02=ATTESTATION_RESPONSE,
                     0x03=NARROW_REQUEST, 0x04=NARROW_RESPONSE,
                     0x05=EVENTS_REQUEST, 0x06=EVENTS_RESPONSE,
                     0x07=FAULT_DECLARATION)
  DISPUTE_ID        (32 bytes, SHA-256 of CHALLENGER_PUBKEY ||
CHALLENGED_PUBKEY || CHECKPOINT_SEQ)
  PAYLOAD_LENGTH    (4 bytes, uint32 LE)
  PAYLOAD           (variable, message-type-specific)
  SENDER_PUBKEY     (33 bytes, compressed SEC)
  SIGNATURE         (64 bytes, Schnorr over SHA-256(MSG_TYPE ||
DISPUTE_ID || PAYLOAD))
```

NARROW_REQUEST payload contains: `ROUND_NUMBER (4 bytes) || QUERY_INDEX (8 bytes, uint64 LE)`. NARROW_RESPONSE payload contains: `ROUND_NUMBER (4 bytes) || STATE_HASH (32 bytes)`.

1.8 Competitive Correctness Incentive

The system creates economic incentives for honest state computation:

1. **Token Rewards**: Nodes with high Correctness Scores (consistent consensus agreement) earn overlay-specific tokens (e.g., $402 tokens in the preferred embodiment) at enhanced rates. Nodes with low Correctness Scores earn at reduced rates or are excluded from rewards entirely.
2. **Preferential Routing**: When users discover overlay nodes via SHIP/SLAP and must choose which LookupService to query, nodes with higher Correctness Scores are preferred — receiving more query traffic and associated service fees.
3. **Reputation Persistence**: Correctness Scores are maintained across checkpoint windows, creating long-term reputation. A node cannot game the system by being correct for one window and incorrect for the next without consequence.
4. **Slashing**: In deployments where nodes stake tokens to participate, proven state computation errors result in token slashing — direct economic loss proportional to the severity of the error.

1.8.1 Correctness Score Computation

Each node maintains a Correctness Score `S` initialised to `S_0 = 100`. The score is updated at each checkpoint as follows:

```
function UPDATE_SCORE(node, checkpoint_result):
    if checkpoint_result == CONSENSUS_MATCH:
        node.S = min(S_MAX, node.S + REWARD_INCREMENT)
    else if checkpoint_result == CONSENSUS_MISMATCH:
        node.S = max(0, node.S - MISMATCH_PENALTY)
    else if checkpoint_result == DISPUTE_LOST:
        node.S = max(0, node.S - DISPUTE_PENALTY)

    // Time-decay: applied once per checkpoint regardless of result
    node.S = node.S × DECAY_FACTOR
```

Parameter values in the preferred embodiment:

Parameter	Value	Description
`S_0`	100	Initial score for new nodes
`S_MAX`	1000	Maximum score cap
`REWARD_INCREMENT`	1	Points added per consensus match
`MISMATCH_PENALTY`	10	Points deducted per unresolved mismatch
`DISPUTE_PENALTY`	50	Points deducted when a node loses a dispute
`DECAY_FACTOR`	0.999	Multiplicative decay per checkpoint (half-life ~693 checkpoints)
`REWARD_THRESHOLD`	50	Minimum score to be eligible for token rewards

The reward distribution for a given period is proportional to each eligible node's score relative to the sum of all eligible scores:

```
node_reward = TOTAL_REWARD_POOL × (node.S / SUM(eligible_node.S for all
eligible nodes))
```

Nodes with `S < REWARD_THRESHOLD` receive zero rewards and are deprioritised in SHIP/SLAP query routing.

1.8.2 Economic Penalty and Slashing Specification

When a node loses a dispute (its computation is proven incorrect via the binary narrowing protocol):

1. **Primary slash**: 5% of the node’s total staked tokens are slashed.
2. **Challenger reward**: 60% of slashed tokens are awarded to the challenger who initiated and proved the dispute.
3. **Network pool**: 30% of slashed tokens are added to the overlay’s reward pool for distribution to high-scoring nodes.
4. **Burn**: 10% of slashed tokens are permanently burned (sent to an unspendable output), creating deflationary pressure.

Repeated offences within a rolling window of 100 checkpoints trigger escalating penalties:

Offence count (per 100 checkpoints)	**Slash percentage**
1	5%
2	10%
3	25%
4+	100% (full stake seizure + permanent ban)

Cooldown period: After a successful dispute against a node, the node enters a cooldown period of 10 checkpoints during which it may not earn rewards but continues processing events. This prevents a slashed node from immediately re-entering the reward pool. The node must demonstrate 10 consecutive consensus-matching checkpoints after cooldown before reward eligibility is restored.

Anti-griefing bond: To initiate a challenge, the challenger must post a bond of 0.1% of the challenged node’s stake. If the challenge fails (the challenged node’s computation is proven correct), the bond is forfeited to the challenged node as compensation for the computational cost of producing the attestation. This prevents denial-of-service via frivolous challenges.

2. Operational Flow

The complete operational flow of the system is as follows:

1. **Overlay Bootstrap**: Multiple independent nodes deploy the same overlay code (TopicManagers + LookupServices) via CARS or equivalent hosting infrastructure. Each node registers its topics and services via SHIP/SLAP.

2. **UTXO Synchronisation**: Nodes synchronise their raw UTXO state via GASP, ensuring all nodes process the same underlying transaction data.
3. **Event Processing**: As new UTXO events are admitted or spent, each node's LookupService processes them through the deterministic derived state computation rules.
4. **Checkpoint Reached**: At each checkpoint interval, each node computes its State Commitment and broadcasts it to peers.
5. **Cross-Verification**: Nodes compare State Commitments. Matching commitments strengthen consensus. Divergent commitments trigger the Dispute Protocol.
6. **PoI Attestation Generation**: Nodes maintain running PoI Attestation data (event Merkle trees, intermediate state hashes) as they process events, enabling efficient attestation generation when challenged.
7. **Challenge Resolution**: Disputes are resolved via the binary narrowing protocol, identifying the exact point of computational divergence.
8. **Reward Distribution**: At configurable intervals, overlay-specific tokens are distributed to nodes proportional to their Correctness Scores and the volume of events they processed.
9. **State Queries**: Users query LookupServices for derived state (e.g., "What is my dividend balance?"). The response includes the relevant State Commitment and checkpoint number, allowing the user to verify the response against the network consensus.
10. **Settlement**: When derived state indicates a user is owed a payout, the user or a node constructs a settlement transaction (a UTXO) that is admitted to the overlay, decrementing the derived balance and creating a spendable on-chain output — bridging derived state back to UTXO-native settlement.

3. Relationship to GASP Synchronisation

The present invention is complementary to, and does not replace, GASP synchronisation. GASP ensures that all nodes have the same raw UTXO data (the inputs to derived state computation). The present invention ensures that all nodes compute the same derived state from those inputs. Together, they provide end-to-end verifiability: from block headers, through UTXO Merkle proofs, through GASP-synchronised event streams, through deterministically computed derived state, to user-facing query results.

4. Application to Content Distribution Networks

In the preferred embodiment, the system is applied to a decentralised content distribution network where:

- Content creators mint limited-supply tickets (UTXOs) representing access rights to digital content.
- Tickets are tradeable between users, with each sale generating a revenue event.
- Users can stake tickets to earn dividends from future sales of tickets for the same content.
- The derived state engine computes dividend balances across all stakers and content items.
- Nodes compete to maintain accurate derived state, earning overlay tokens via Proof of Indexing.

This application demonstrates the invention's value in high-throughput scenarios: a popular content item may generate thousands of sale events per day, each requiring incremental updates to dozens of staker balances. Computing these balances on-chain (as UTXOs) would be prohibitively expensive. Computing them off-chain via the present invention's verified derived state system provides the same trust guarantees at a fraction of the cost.

Brief Description of Drawings

- **Figure 1** — System architecture diagram showing the relationship between the blockchain layer (UTXOs, Merkle proofs), the GASP synchronisation layer, the derived state computation engine, and the State Commitment broadcast channel.
- **Figure 2** — UTXO event stream processing flow, showing how admit and spend events flow from TopicManagers through the deterministic computation rules to produce derived state entries.
- **Figure 3** — State Commitment Protocol, showing the checkpoint interval trigger, canonical serialisation, Merkle tree construction, and commitment broadcast.
- **Figure 4** — Cross-node verification process, showing State Commitment comparison, consensus threshold evaluation, and Dispute Protocol initiation.
- **Figure 5** — Proof of Indexing Attestation structure, showing the chain of provenance from block headers through UTXO Merkle proofs through event Merkle trees through intermediate state hashes to the final State Commitment.

- **Figure 6** — Challenge-Response Dispute Protocol, showing the binary narrowing process from full checkpoint window to individual event-level fault identification.
- **Figure 7** — Competitive incentive structure, showing the relationship between Correctness Score, token reward rate, query routing preference, and slashing conditions.

Initial Claims

Note: These claims are provided in sketch form for the purposes of establishing a priority date. Formal claims will be drafted and filed within 12 months in accordance with UKIPO rules.

Claim 1 — Verifiable Derived State System

A system for verifying derived state computations in a blockchain overlay network, comprising:

a. a plurality of indexing nodes, each hosting overlay service code comprising at least one TopicManager and at least one LookupService, synchronising raw UTXO data via a graph-aware sync protocol;
b. a deterministic derived state computation engine operating within each LookupService, processing an ordered stream of UTXO admit and spend events to produce aggregate derived state;
c. a State Commitment Protocol whereby each node, at defined checkpoint intervals, computes a deterministic hash commitment over its current derived state and broadcasts the commitment to peer nodes;
d. a cross-node verification mechanism whereby nodes compare State Commitments at each checkpoint, identifying consensus and detecting divergence; and
e. a Proof of Indexing Attestation mechanism whereby a node generates a compact cryptographic proof linking its derived state commitment to the specific sequence of Merkle-proven UTXO events from which it was computed.

Claim 2 — Challenge-Response Dispute Method

A method for resolving disputes regarding derived state correctness in a blockchain overlay network, comprising:

a. receiving a challenge identifying a checkpoint at which a node's derived state is disputed;

b. the challenged node providing a Proof of Indexing Attestation comprising event range identifiers, an event Merkle proof, intermediate state hashes at sub-intervals within the event range, and the final state commitment;

c. the challenger replaying events in the attestation through the deterministic state computation rules to produce an independent state commitment;

d. if state commitments diverge, performing binary narrowing using the intermediate state hashes to identify the specific sub-interval containing the divergence; and

e. applying economic consequences to the node determined to have computed incorrect derived state, said consequences comprising one or more of: Correctness Score reduction, token slashing, or removal from preferential service routing.

Claim 3 — Competitive Correctness Incentive

A method for incentivising honest derived state computation in a federated overlay network, comprising:

a. maintaining a Correctness Score for each participating node based on the node's history of State Commitment agreement with network consensus;

b. distributing overlay-specific tokens to participating nodes at rates proportional to their Correctness Scores and the volume of UTXO events processed;

c. routing user queries for derived state preferentially to nodes with higher Correctness Scores; and

d. slashing staked tokens of nodes proven via the Challenge-Response Dispute Protocol to have computed incorrect derived state.

Claim 4 — End-to-End Provenance Chain

A method for establishing an unbroken chain of cryptographic provenance from blockchain block headers to user-facing derived state query results, comprising:

a. receiving UTXO events, each carrying a Merkle proof linking the UTXO to a confirmed block header;

b. incorporating the UTXO Merkle proofs into an event Merkle tree maintained by the indexing node;

c. computing derived state from the events using deterministic rules, recording intermediate state hashes at defined intervals;

d. constructing a State Commitment as the Merkle root of the canonically serialised derived state;

e. upon receiving a user query for derived state, returning the query result together with the State Commitment and checkpoint identifier, enabling the user to verify the result against the network's consensus commitment.

Claim 5 — Hybrid Settlement

A system combining verified derived state with UTXO settlement, comprising:

a. a derived state engine computing aggregate balances from UTXO event streams without creating on-chain UTXOs for each incremental state change;

b. the State Commitment Protocol and Cross-Node Verification mechanism of Claim 1 to ensure derived state correctness;

c. a settlement mechanism whereby, when a user claims a balance indicated by the verified derived state, the system constructs a settlement transaction producing a spendable UTXO on the underlying blockchain; and

d. the settlement transaction being admitted to the overlay as a spend event, decrementing the derived state balance, thereby maintaining consistency between the derived state layer and the UTXO settlement layer.

Abstract

A system and method for verifying the correctness of derived state computations performed by indexing nodes in blockchain-based overlay networks. In overlay architectures where raw UTXO (unspent transaction output) data is synchronised between nodes via graph-aware sync protocols and application-specific derived state (such as aggregate balances, eligibility tables, and accrual figures) is computed off-chain by lookup service providers, the present invention provides trustless verification through: (1) a State Commitment Protocol where nodes compute deterministic hash commitments over derived state at checkpoint intervals; (2) cross-node verification by comparing commitments across independent nodes; (3) Proof of Indexing Attestations that cryptographically link derived state to the specific Merkle-proven UTXO events from which it was computed; (4) a challenge-response dispute protocol with binary narrowing to identify specific computational faults; and (5) a competitive incentive structure rewarding nodes for consistent correctness. The invention bridges the gap between the scalability of off-chain derived state computation and the verifiability of on-chain UTXO

settlement, enabling high-throughput overlay applications such as decentralised content distribution networks to maintain trustless derived state without prohibitive chain bloat.

Document prepared for UKIPO filing. Priority date to be established upon submission. Applicant: The Bitcoin Corporation Ltd Inventor: Richard Boase Date of preparation: 8 March 2026