Extended Abstract: Proof of Balance (PoB), an Anonymous Payment Scheme on BFTKV

Ryuji Ishiguro

– Very Preliminary (ver 0.4) –

Abstract

This paper extends bftkv [1] and presents Proof of Balance (PoB), an anonymous payment scheme built on a Byzantine fault-tolerant key-value store. bftkv provides a scalable write-once substrate for balance updates without requiring a globally replicated transaction log. On top of it, PoB uses zero-knowledge proofs to show that balances are conserved and remain non-negative while keeping balances, identities, and transaction metadata confidential. PoB acts as a deterministic proof-sealing mechanism: a transaction becomes admissible once its cryptographic proof and quorum checks succeed, without any separate mining or staking phase. The paper also describes a parallel EVM execution layer, PoBEVM, and a Bulletproofs-family range proof for the PoB statement.

1 Introduction

Traditional blockchains keep all transactions in a global ledger so that everyone can verify that no participant is cheating. This design works for systems such as Bitcoin, but it also makes transaction history globally visible. Even when addresses are pseudonymous, balances and transfer graphs remain highly linkable. Most non-PoW blockchains, including BFT state machine replication systems such as PBFT [2], inherit the same basic model: the transaction log is globally replicated, and transaction validity is derived from globally visible state transitions.

PoB starts from a different observation. A payment is usually relevant only to the parties involved and to the nodes that must prevent equivocation. We therefore use a Byzantine fault-tolerant key-value store as the consensus substrate. Payers and payees verify a transaction locally and store only the resulting balance transition in the key-value quorum together with a proof that the transition is valid. The key-value slots are write-once, so once a balance has been consumed, no later transaction can overwrite the fact that it was spent. Unlike PoW or total-order state machine replication, this design does not require absolute global consistency of a single transaction stream.

We call this validity proof Proof of Balance, or PoB. A balance-value pair is accepted once the corresponding transaction is verified, proof-sealed, signed by the relevant quorum, and written to the ledger. The proof is zero-knowledge: balances are committed, not revealed, and the verifier learns only that the total balance is conserved and that every output balance lies in the valid range. PoB therefore serves as a deterministic proof-sealing mechanism rather than a probabilistic block-discovery process. The scheme does not require long-term signing keys in each transaction, and the transaction object itself contains no address-like identifier that must later be hidden.

PoB also changes the underlying data model. Instead of a hash chain, the ledger forms a DAG in which vertices are encrypted balances and edges are transactions. We call this structure the Chain of Balance. A new vertex can be inserted only if it is backed by a valid PoB and accepted by the quorum. Soundness can therefore be audited by tracing the graph itself; see Audit.

2 Recap of bftkv

A quick recap of the building blocks of bftkv.

Quorum Cliques

bftkv follows the idea of Byzantine quorum systems [3]. Our quorum system is constructed from a trust graph (a Web of Trust), and the fundamental building block is a "Quorum Clique". A quorum clique is a subgraph $G_C = (V, E)$ such that

1. For $\forall v_i, v_j \in V, (v_i, v_j)$ and $(v_j, v_i) \in E$,
2. $|V| \geq 4$,
3. $\forall v \in V$ is not a member of any other quorum clique.

With these conditions, a Sybil attack tends to split a clique rather than silently capture it.

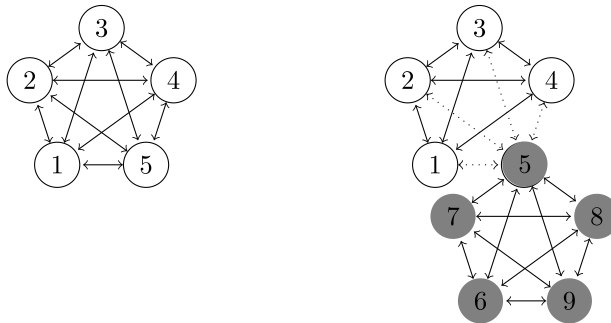

Figure 1: Quorum Clique

The clique on the left is legitimate. If node 5 is compromised and the attacker tries to outnumber the legitimate nodes with colluding nodes (nodes 6–9), node 5 cannot simultaneously belong to another clique. The attacker therefore has no choice but to sever trust links. The result is two separated cliques rather than a silent takeover of the legitimate clique.

Quorum Certificates

Quorum certificates are collective certificates signed by the members of quorum cliques. When a node joins the network, a quorum certificate is issued; it represents a trust graph rooted at that node.

Key-value Quorum

Once a triple $\langle x, t, v \rangle$ is signed by a quorum clique, it is stored on other nodes ($\notin QC$). These nodes form another quorum system, called the "KV quorum". In our system, this quorum also serves as an auditor. Each node checks for equivocation before storing the triple. If it finds malicious behavior, it revokes the member and propagates a proof of the malicious action. Once a member is revoked, there is no way to rejoin the network.

Revocation

Each legitimate node must check for equivocation. A malicious node could sign different values for the same variable and timestamp, i.e., $\langle x, t, v \rangle$ and $\langle x, t, v' \rangle$ where $v \neq v'$. Such equivocation can be detected when the tuple reaches the KV quorum. Another form of equivocation is to show different peers inconsistent views of the trust graph. To detect that attack, each node keeps its own view of the trust graph; whenever it receives a quorum certificate, it checks whether the certificate is consistent with its own graph and updates the graph if the certificate contains new information.

2.1 Read / Write Protocols

The read and write protocols follow Phalanx [4] with several practical extensions. See Section 4 of [1] for details. We sketch the protocols here only as a recap.

Transport security is out of scope for this proposal. The protocol should remain safe even without additional channel protection, but in practice we still need defensive measures. For example, if a client uses a malicious public WiFi network, the attacker may not be able to forge messages but can still attempt replay and relay attacks. If we relied only on the quorum system, this situation could look indistinguishable from a failure scenario in which all contacted nodes are faulty. To mitigate this, we add a simple nonce-based challenge-response step.

In addition to transport-level security, we add simple gossip among nodes in the KV quorum. Gossip is voluntary and the correctness of the system does not depend on its exact propagation pattern, but it helps keep the network sound by spreading signed triples and revocation messages.

Write

The write operation is carried out among the client, the Quorum Clique, and the KV quorum.

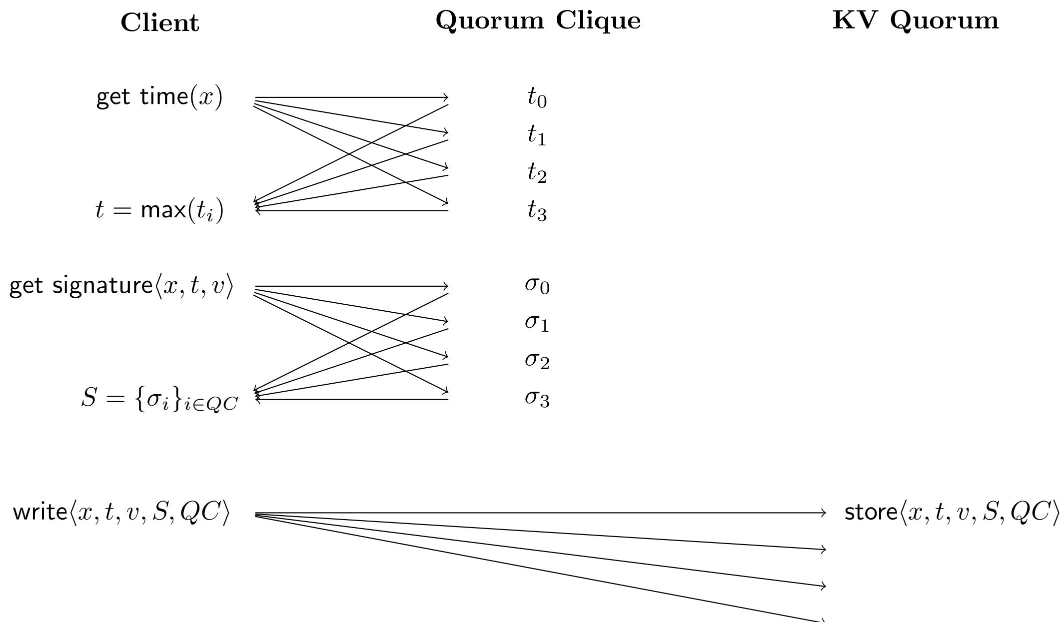

Figure 2: Write Protocol

Each member of the KV quorum checks for equivocation. If the check fails, it propagates a revocation message instead of the signed triple. Unlike state machine replication, the system does not provide absolute or eventual consistency over a single totally ordered log. The communication complexity at read and write time is $\Theta(|Q|)$. Signed triples and revocation state are then gossiped asynchronously. The client does not wait for every quorum member to reply before concluding the transaction. This is the key scalability advantage: communication does not grow beyond the quorum threshold itself.

Each member of the Quorum Clique also checks whether $\langle x, t \rangle$ has already been used in a prior signature. If a client explicitly requests such a conflicting signature, the client can be revoked immediately, although in practice more subtle malicious behavior is more likely to be detected by the KV quorum; see the security analysis in [1].

Read

The read operation is carried out between the client and the KV quorum.

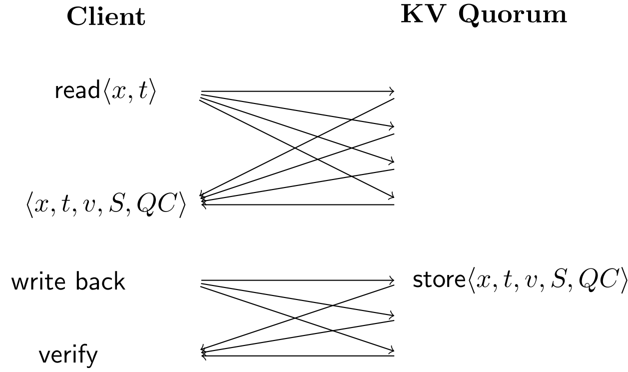

Figure 3: Read Protocol

3 Payment Scheme

As stated in the introduction, our payment scheme is strictly peer-to-peer. Only the result of a transaction, namely a new balance commitment, is written to the KV quorum. A transaction can be viewed as follows:

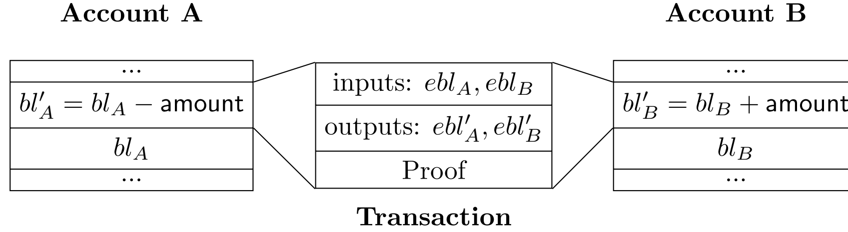

Figure 4: Balance based Transaction

A pays amount to **B** out of band, or more generally in a peer-to-peer payment protocol. Proof is a zero-knowledge proof that binds the old and new balances without revealing them. The total balance is checked over ebl, which is a commitment to bl, without decrypting it. By verifying the proof, we learn that the total balance is conserved across the transaction and that no resulting balance is negative, i.e.,

$$(\sum \mathsf{bl}_i = \sum \mathsf{bl}'_i) \wedge (\mathsf{bl}'_i \geq 0).$$

The quorum system prevents double spending through the write-once property of bftkv. We also add an anonymous write variant for transaction outputs. Because the variable is never modified after creation, it can be written without binding it to a long-lived owner identity.

3.1 Zero Knowledge Proof of Balance (zkPoB)

Each participant does not need to know the balance of any other party. It is enough to know that the total balance is unchanged before and after the transaction. We therefore use a zero-knowledge proof to check balance conservation without revealing any concrete value. We must also ensure that every output balance lies in a valid range so that underflow or negative balances are impossible. Putting these together, the relation can be written as

$$\mathcal{R} := \{((bl_i, bl'_i, \beta_i, \beta'_i), (ebl_i, ebl'_i)) :$$
$$ebl_i = h^{\beta_i} g^{bl_i}, ebl'_i = h^{\beta'_i} g^{bl'_i}, 0 \leq bl'_i < 2^d, \text{ and } \sum_i bl_i = \sum_i bl'_i\}.$$

g and h are generators in $\mathbb{Z}_q$. With this relation, we obtain

$$(ebl_i, ebl'_i) \in L_{\mathcal{R}} \Longleftrightarrow \prod_i ebl_i / \prod_i ebl'_i = \prod_i h^{\beta_i} h^{-\beta'_i} = \prod_i h^{\beta_{\triangle i}},$$
$$\text{where } \beta_{\triangle i} = \beta_i - \beta'_i,$$

if and only if $\sum_i bl_i = \sum_i bl'_i$. Intuitively, it is therefore enough to prove knowledge of the blinding deltas together with a valid range decomposition for each output balance. As a conceptual derivation, we can write the relation as a sigma protocol:

$$\mathcal{R}' := \{((\beta_{\triangle i}, bl'_i, \gamma_i), (u_j, w_i, b_{ij}, c_{ij})) : w_i = h^{\beta_{\triangle i}}, c_{ij} = u_j^{\gamma_i} g^{b_{ij}}\},$$
$$\text{where } u_j = g^{\alpha_j}, \alpha_j \xleftarrow{\mathcal{R}} \mathbb{Z}_q,$$

subject to

$$(\prod_i ebl_i / \prod_i ebl'_i = \prod_i w_i) \wedge (bl'_i = \sum_{j=0..d-1} 2^j b_{ij}).$$

The sigma protocol corresponding to this relation is shown in Figure 5. It is best viewed as an explanatory derivation of the statement that PoB must prove.

$P((\beta_{\triangle i}, bl'_i, \gamma_i, \beta'_i, \{b_{ij}\}), (\{u_j\}, \{c_{ij}\}, w_i))$ $\qquad$ $V(\{u_j\}, \{c_{ij}\}, w_i)$

$$t_{1j}, t_2, t_{3j}, t_4, t_5 \xleftarrow{\mathcal{R}} \mathbb{Z}_q$$
$$v_{1ij} = c_{ij}^{t_{1j}},\ v_{2ij} = u_j^{t_2},\ v_{3ij} = u_j^{-t_{3j}},$$
$$v_{4i} = h^{t_4},\ v_{5i} = h^{t_5}, v_{1i*} = \prod g^{t_{1j}} 2^j$$

$$\xrightarrow{\{v_{1ij}\}, \{v_{2ij}\}, \{v_{3ij}\}, v_{4i}, v_{5i}, v_{1i*}}$$

$$C \xleftarrow{\mathcal{R}} \mathbb{Z}_q$$

$$\xleftarrow{C}$$

$$z_{1ij} = t_{1j} + b_{ij}C,\ z_{2i} = t_2 + \gamma_i C,$$
$$z_{3ij} = t_{3j} + \gamma_i b_{ij} C,\ z_{4i} = t_4 + \beta'_i C,$$
$$z_{5i} = t_5 + \beta_{\triangle i} C$$

$$\xrightarrow{\{z_{1ij}\}, z_{2i}, \{z_{3ij}\}, z_{4i}, z_{5i}}$$

$$c_{ij}^{z_{1ij}} u_j^{z_{2i}} u_j^{-z_{3ij}} \stackrel{?}{=} v_{1ij} v_{2ij} v_{3ij} c_{ij}^C$$
$$(\prod_j g^{z_{1ij} 2^j}) h^{z_{4i}} \stackrel{?}{=} v_{1i*} v_{4i} ebl_i'^C$$
$$h^{z_{5i}} \stackrel{?}{=} v_{5i} w_i^C$$

Figure 5: Conceptual Sigma Protocol for PoB

After all parties finish the protocol, the verifier checks whether $\prod_i ebl_i / \prod_i ebl'_i \stackrel{?}{=} \prod_i w_i$.

In this paper, the range statement is instantiated with a Bulletproofs-family proof: specifically, a BP++-style reciprocal argument over an arithmetic circuit, together with a WNLA inner argument used by the proof system. The prover constructs a commitment to the output balance, proves that the committed value lies in the valid range, and binds that proof to the encrypted balance commitment carried in the transaction. The interactive derivation above therefore explains the algebraic relation, while the proof object used by PoB is a compact non-interactive construction in the Bulletproofs family [5].

3.2 Payment Protocols

We define the transaction abstractly as follows:

$$PoB_i := (ebl_i, ebl'_i, \pi_i)$$
$$Tx := \{ebl_i, ebl'_i, PoB_i\}_i$$

ebl_i and ebl'_i for some i may be nil, which is treated as $bl = 0$. π_i denotes the non-interactive PoB proof attached to the balance transition. Conceptually it is obtained from the relation above via Fiat-Shamir; concretely, it is a commitment-bound PoB proof in the Bulletproofs family.

Figure 3.2 shows a payment example in which A and B jointly pay \$2 and \$3 to C. This transaction creates a brand-new balance for C. Whatever bl_A and bl_B are, the total balance of A, B, and C remains unchanged.

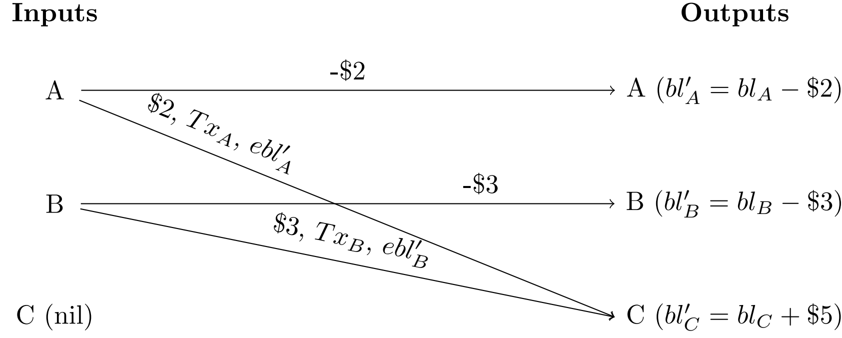

Figure 6: Payment Example

The payment protocol between two parties is shown in Figure 7.

Tx_A is the transaction that created bl_A. The payer subtracts the amount from the current balance (bl_A) and creates a new encrypted balance (ebl'_A). The payee adds the amount to the current balance (bl_B) and performs PoB on the new balance (bl'_B). The payer then constructs a new transaction that combines the encrypted balances and PoBs of both parties and writes it to the quorum. (See the next section for wo.) Tx is verified during the protocol as in Algorithm 1.

Algorithm 1: VerifyTx

Input: Tx: Transaction of ebl
Output: true/false
Function `VerifyTx(`Tx`)`**:**
  foreach $i \in Tx$ **do**
    if $PoB.Verify(Tx.PoB_i) = failed$ **then**
      return *false*
    end
    if $\mathit{read}(Q, Tx.ebl_i) \neq H(Tx)$ **then**
      return *false*
    end
  end
  return *Conserved*(Tx)

3.3 Write Once with PoB

In bftkv, writing a variable to a quorum normally requires the client to sign the tuple with its own Quorum Certificate, thereby enforcing a TOFU-style ownership policy. With a write-once balance object, however, that notion of ownership is less useful because the variable is never modified after creation. We therefore extend the write function for Tx. Instead of signing the tuple with a long-lived QC identity, we use PoB to guarantee the validity of the key (ebl) and value (Tx'). The Quorum Clique runs Algorithm 1 over Tx; if verification succeeds, it signs the tuple

$$\mathsf{wo}(QC, \langle ebl, \mathsf{Tx}' \rangle, \mathsf{Tx}) \longrightarrow \sigma_i = Sign(\langle ebl, H(\mathsf{Tx}')\rangle).$$

The client collects the signatures ($S = \{\sigma_i\}_{i \in QC}$) from the QC and writes them to the KV quorum:

$$\mathsf{write}(KVQ, \langle ebl, H(\mathsf{Tx}') \parallel H(\mathsf{Tx})\rangle, S).$$

The KV quorum checks equivocation in the same way as in the ordinary case, so double spending is detected with the same probability bounds analyzed in bftkv [1]. The quorum system also prevents replay

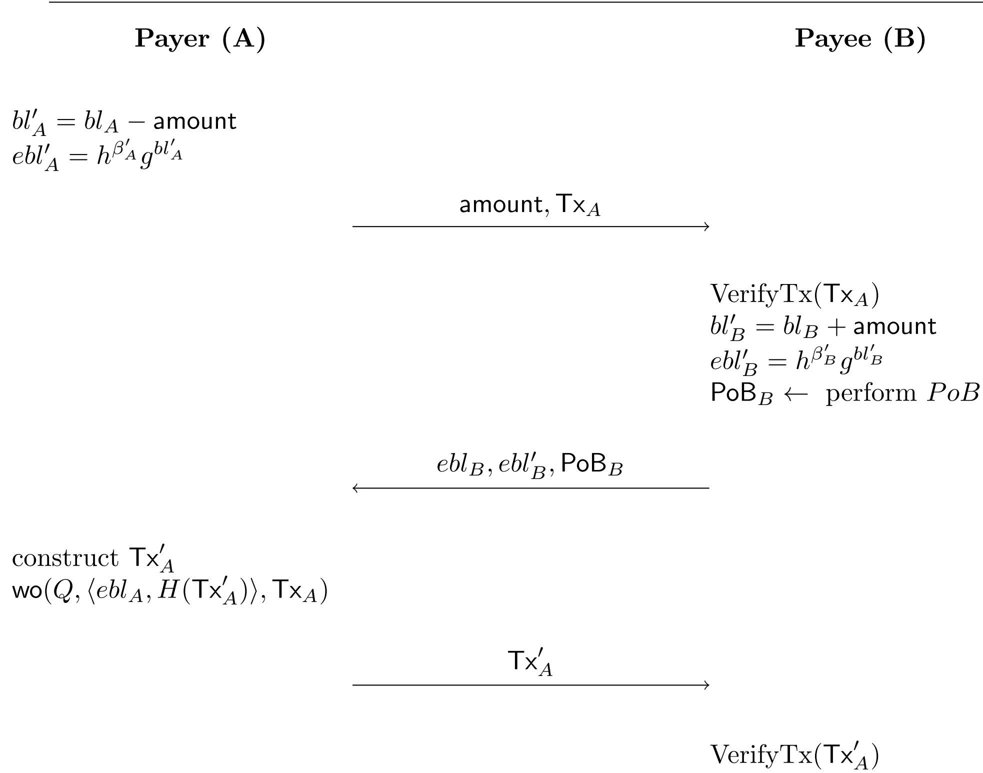

Figure 7: Payment Protocol

attacks. The key ebl is consumed only once for a specific Tx. If someone copies (ebl, ebl', PoB) and fabricates a different Tx', it will differ from the original Tx that consumed ebl, and the attack will be detected.

3.4 Audit

The key-value pairs (ebl, Tx) form a DAG. Topological order is guaranteed: every new balance must be created from existing balances except for the genesis. In Figure 8, when ebl_4 is spent, Tx_2 is verified as $\mathsf{VerifyTx}(Tx_2)$. It is not necessary to backtrack beyond ebl_3 or ebl'_1 because when those balances were spent, the earlier transactions had already been verified in the same way.

What if the quorum system is corrupted, or the payer somehow tricks the quorum? In that case, a balance might appear to have been created out of thin air. As analyzed in bftkv, even if faulty nodes outnumber the threshold it is still possible to detect equivocation; in the transaction setting, detection immediately invalidates the balance and aborts the transaction. Still, even a low-probability failure mode deserves a fail-safe mechanism.

When a balance (ebl) is written to a quorum, each member verifies the PoB of the transaction that created ebl, then stores the hash of the new transaction that spends ebl. Once Tx has been verified, the full object can be discarded locally. Anyone who later reads the variable (ebl) can treat the stored value (H(Tx')) as valid without tracing all the way back to genesis. However, because of the failure mode above, it is useful to have a way to validate the chain without relying entirely on the quorum. We therefore propose an archive service for Tx. Keeping every full transaction on every quorum node is possible, but it is expensive and unnecessary. The archive service is a simple key-value store that takes ebl and returns Tx. The client checks whether the hash of Tx $\leftarrow$ ReadArchive(ebl) matches the value read from the quorum and then verifies each PoB in Tx. The audit can be done as the Algorithm Audit.

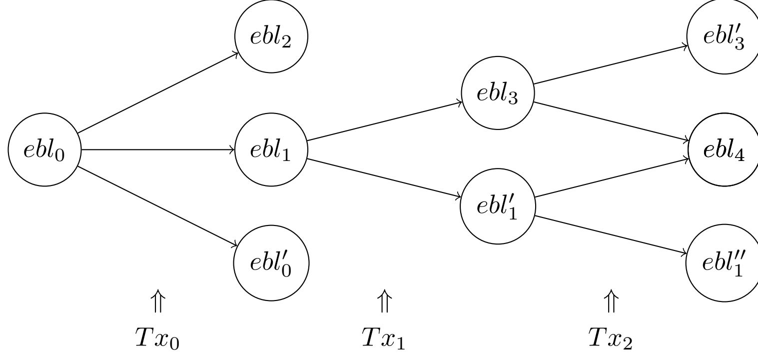

Figure 8: Balance Chain

Algorithm 2: Audit

Input: Tx: Transaction of *ebl*
Output: true/false
Function `VerifyTxDeeply(`Tx`):`
  foreach $i \in Tx$ **do**
    if $Tx.ebl_i$ *is genesis* **then**
      continue
    end
    if $PoB.Verify(Tx.PoB_i) = failed$ **then**
      return *false*
    end
    if $\mathit{read}(Q, Tx.ebl'_i) \neq H(Tx) \lor$ `VerifyTxDeeply`$(\mathit{ReadArchive}(Tx.ebl_i)) = \mathit{false}$ **then**
      return *false*
    end
  end
  return *Conserved*(Tx)

3.5 Anonymity

Tx does not include information that is directly linkable to users or devices. Balances are committed, so they do not themselves reveal an obvious tracing signal. However, the payment protocol is still peer-to-peer: payers and payees know each other's identity during the payment itself. An *ebl* can therefore be linked to an identity at the edge, even if the system does not publish that identity globally.

The more interesting question is whether *ebl* is traceable through the ledger. In general, Tx has n input-output pairs. When an observer obtains Tx from the archive service, they know that a specific *ebl* is one of the outputs, but they do not know which input corresponds to it. (The output may also be newly created, in which case the corresponding input is nil.) The same ambiguity applies in the forward direction: when following Tx for ebl', the observer cannot tell which output corresponds to which prior input. In the worst case, tracing one balance requires following $\prod_i^{\mathsf{gen}} \mathsf{Tx}_i.n$ candidates over gen generations, which is $\Omega(2^{\mathsf{gen}})$.

If the transaction graph is short and simple, it may still be possible to guess the path of a specific payment. A simple mitigation is to add dummy balances with zero value on both the input and output sides in order to obscure the actual transaction path. Such dummy balances can be consumed in later transactions without affecting real balances, although this also makes auditing less efficient.

4 Wallet

The payment scheme does not require long-term address-signing keys in each transaction. Instead, each balance is associated with witness material such as $(\beta_\triangle, bl, \gamma)$. In practice, the wallet can separate secret storage from proving work: the committed balance and most of the Bulletproofs-family range-proof computation can run on an application processor, while the most sensitive witness material remains inside hardware. The exact split depends on the wallet design, but the important point is that PoB is witness-based rather than signature-based.

4.1 OPAQUE-protected wallet storage

PoB wallets can also use OPAQUE as the password-authenticated front end for secure wallet storage [6, 7]. In this model, the user's password is used only inside the OPAQUE exchange to recover an export key, while the wallet state itself remains encrypted at rest under that recovered key. This is a good fit for PoB because the wallet may need to store not only PoB witness material but also other long-lived secrets such as an Ethereum private key used by an EVM-facing account.

The key enabling mechanism is OPAQUE's oblivious pseudorandom function (OPRF). The client evaluates the PRF on its password with help from the server, but the server does not learn the password itself, and the client does not learn the server's OPRF secret. The resulting pseudorandom output can then be stretched into the export key that protects the wallet envelope and encrypted storage. In effect, the wallet can make password-based key recovery possible without exposing the password to the server or storing the wallet secret directly under a reusable password hash.

The important property is that password authentication and secret storage are separated cleanly. The server stores an OPAQUE record and encrypted wallet state, but it does not learn the password and does not directly receive the wallet secret. After successful OPAQUE authentication, the client recovers the export key and uses it to open the encrypted wallet, retrieve the stored secret material, and continue with PoB or EVM operations. In this way, a PoB wallet can act as secure storage for auxiliary keys, including an ETH private key, without reducing the security model to simple server-side password storage.

Wallets can also participate in the KV quorum. Each member keeps hashes only for the leaves of the Balance Chain. Intermediate nodes are needed for deep audit, but ordinary wallets do not have to store them all. This can make payments faster because a payee may be able to verify the payer's current balance from a local KV view before final settlement reaches the quorum. The payee still has to confirm that the payer eventually wrote the transaction to the quorum, but this design offers flexibility in settlement timing. More importantly, wallet participation can greatly enlarge the KV quorum and make the overall system more robust; see the security analysis of bftkv. Wallets are also the most trustworthy edge entities because they are controlled by the end user rather than by arbitrary third parties.

5 PoBEVM: Parallel EVM Execution

PoB is not limited to direct peer-to-peer balance updates. It can also serve as the sealing layer for a parallel EVM. PoBEVM keeps the EVM programming model while replacing Ethereum's Merkle Patricia Trie with a Redis-backed logical state model. The goal is to preserve familiar smart-contract execution and JSON-RPC entry points while removing the strictly sequential execution bottleneck of standard Ethereum.

PoBEVM executes transactions with optimistic concurrency control. Each transaction starts from a snapshot identified by a visible read version. Accounts and storage slots are versioned independently in Redis, so multiple transactions can execute concurrently against the same head block. At commit time, the engine performs an atomic compare-and-write step over every touched account and storage slot. If any touched object changed since the transaction's read version, the commit fails with a live-state conflict and the transaction is retried. This gives snapshot isolation and fine-grained conflict detection rather than coarse global serialization.

The important point is that PoB acts as a deterministic proof seal. A transaction does not wait for a miner, a proposer auction, or a staking-based validation phase in order to become valid. Once execution succeeds, the balance-conservation statement is proven, and the transaction passes the admission checks, it is already proof-sealed. Publication is therefore not a second validity phase.

Instead, successful executions are appended to a pending queue, and a separate publisher simply collects the already proof-sealed transactions that arrive within a publication window. It then merges their state patches, derives a deterministic logical state root, writes the resulting block, and advances the canonical head. In other words, PoBEVM separates deterministic validity from block publication.

This design preserves EVM bytecode execution semantics, but it does not preserve every Ethereum data-structure invariant. In particular, the state root is a logical hash of the published execution set rather than a Merkle Patricia Trie root, and transaction receipts become canonical only after the publish phase. The efficiency gain comes from the same design choice: validity is established cryptographically and deterministically before publication, so the publisher performs only lightweight batching and head movement rather than resource-intensive consensus work.

6 Bootstrap from Bitcoin

This is a payment scheme, not a standalone cryptocurrency. There is no mining process that mints new value. Instead, the system relies on an external digital currency system to create genesis balances. Using Bitcoin as the currency model, a genesis balance can be created as follows:

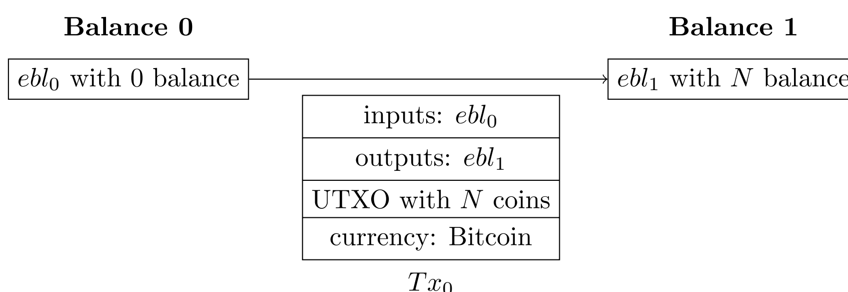

Figure 9: Bootstrap from Bitcoin and Fiat Currency

The UTXO sends N coins to a public address controlled by the quorum. To convert the balance back into Bitcoin, the user creates a transaction that brings the PoB balance back to 0, and the system creates a UTXO that sends the corresponding amount from the quorum-controlled address to the user's address. The key invariant is balance conservation. The total amount of bitcoin deposited to the public address must always equal the total balance represented in the Balance Chain, so withdrawals must be processed without shortfall. We also use the distributed signing feature of bftkv to manage the private key of the public address. The signing key is distributed across the quorum with threshold cryptography and is never fully revealed during signing. See Section 3.4, "Distributed Signing", in [1].

6.1 Currency

As shown in Figure 6, the currency field indicates the origin currency of the balance. The unit of balance depends on the currency. For example, $bl = 1$ in USD is not the same as $bl = 1$ in Bitcoin. All transactions derived from a genesis balance must inherit the same currency. Transactions between balances with different currencies are forbidden. This preserves the invariant that the total balance of each currency can be redeemed without changing value. Currency exchange is out of scope for this scheme.

The currency can be encoded in the balance as $\mathsf{currency} \times 2^d + bl$, where currency is mapped to a small integer such as 1..127. This makes it easy to verify that all balances in a transaction were created with the same currency by dividing ebl by $g^{\mathsf{currency} \times 2^d}$.

References

[1] Ryuji Ishiguro, "A Byzantine Fault-tolerant KV Store for Decentralized PKI and Blockchain" https://github.com/yahoo/bftkv/blob/master/docs/bftkv.pdf

[2] Miguel Castro and Barbara Liskov, "Practical Byzantine Fault Tolerance"

[3] Malhi, D. and Reiter, M. (1998) "Byzantine Quorum Systems"

[4] Malhi, D. and Reiter, M. (1998) "Secure and Scalable Replication in Phalanx"

[5] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. (2018) "Bulletproofs: Short Proofs for Confidential Transactions and More"

[6] Jarecki, S., Krawczyk, H., and Xu, J. (2018) "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks" https://eprint.iacr.org/2018/163

[7] Bourdrez, D., Krawczyk, H., Lewi, K., and Wood, C. A. (2025) "The OPAQUE Augmented Password-Authenticated Key Exchange (aPAKE) Protocol", RFC 9807 https://www.rfc-editor.org/rfc/rfc9807.html