# A Confidential Payment Scheme with BFTKV

Ryuji Ishiguro

– Very Preliminary (ver 0.1) –

**Abstract**

This is an extension of bftkv [1]. We show that our Byzantine fault-tolerant key-value store can be an efficient and scalable solution for Bitcon like trasactions. On top of that, we develop a confidental payment scheme. In our scheme, anonymity and confidentiality are built-in, not an add-on. We use a ZKP technique not only to make the transaction confidential but also to ensure the validity of transactions, which is usually done using digital signatures. The double spending problem is reduced to the equivocation problem that is solved with bftkv.

## 1 Introduction

All transactions are stored in the global ledger in the PoW type blockchains to make PoW efficient. Even if the addresses are pseudonym it is easy to track the transactions. Also each transaction includes the amount of coin that is transferred from an address to another address so even 3rd parties can know exactly how much coin is spent or kept by an address. This is the way to check that someone is not cheating in blockchains like Bitcoin. Even non PoW type blockchains, most of which are based on BFT state machine replica (such as PBFT [2]), follow the same mechanism when it comes to transactions. We take an advantage of Byzantine fault-tolerant KV store. First of all, your transaction is known by only you and someone you are trading with, and it does not have to be published to all over the world. Payers and payees verify a transaction themselves and stores it in the KV store with a "Proof of Balance". The KV slots are "write once" so any other transactions cannot overwrite it. This feature of bftkv can be equivalent to the global ledger but unlike PoW or state machine replica it does not require absolute constency among participants. This makes it easy to scale the system.

As above, we introduce a new consensus mechanism called Proof of Balance or PoB to add transactions to the global ledger (which is, in our case, the KV store). Once a pair of key (balance) - value (transaction) is in the leger (KV store) we consider the system reaches a consensus on the validity of the transaction with the balance. We use a Zero Knowledge Proof technique to construct PoB. This scheme does not require any long term key such as signing keys and the transaction does not include any linkable information such as signatures or public keys. We discuss anonymity later. The balance is encrypted and never decrypted in any way. All we have to consider is the total balance before and after a transaction. We do not have to know the actual balance of others. ZKP makes it possible to calculate the balance on encrypted data.

We do not have a hash chain. Instead, the KV relation forms a DAG where the vertices are encrypted balances and the edges are transactions, which we call Chain of Balance. PoW bonds the balances so without PoW no vertix can be inserted. We can verify the soundness of the payment system by tracing the graph. (See Audit.)

## 2 Recap of bftkv

A quick reqcap of the building blocks of bftkv.

### Quorum Cliques

bftkv follows the idea of byzantine quorum [3]. Our quorum system is constructed by the trust graph (Web of Trust) and "Quorum Clique" is the fundamental building block. The quorum clique is a sub-graph, that is $G_C = (V, E)$, where

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, an attempt of sybil attacks ends up splitting a clique.

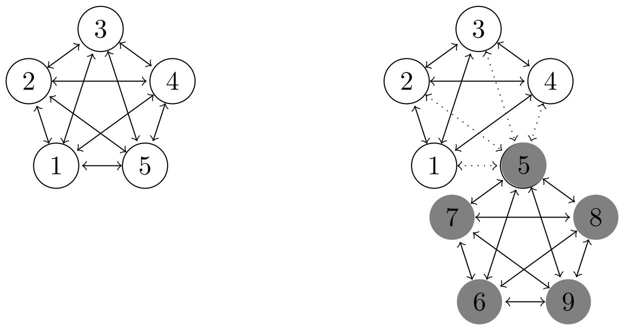

Figure 1: Quorum Clique

The left clique is a legitimate one. When the node 5 is compromised and the attacker attempts to outnumber the legitimate nodes with colluding nodes (node 6..9). Since the node 5 cannot be a memeber of another clique the attacker has no choice but severing the trust links. The end result is we have two separated cliques and it does not affect the legitimate cliques.

**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 and it represents a trust graph rooted by the node.

**Key-value Quorum**

Once a triple $\langle x, t, v\rangle$ is signed by a quorum clique, it will be stored in other nodes ($\notin QC$). These nodes form another quorum system, called "KV quorum". In our system this quorum system plays a role of the auditor as well. Each node checks equivocation before storing the triple and if it finds a malicious action it revokes the member and propagates a proof of malicious actions. Once a member is revoked there is no way to re-join the network.

**Revocation**

Each legitmate node must check equivication. Malicious nodes could sign a different value 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 it's stored in KV quorum. Another type of equivocation is to show other nodes a different view of the trust graph. To detect such attack, each node should keep its own view of the trust graph and whenever it receives the quorum certificate it checks if it's consitent with its own graph and udpates it if it finds an updated nodes in the QC.

## 2.1 Read / Write Protocols

The transport security is out of scope of this proposal. Without any additional security scheme the protocol has to be secure by itself. But in a real world situation we need to consider additional security measures. Imagine that you are connecting a public WiFi at a cafe and this network is totally controled by a malicious attacker. Even if the attacker cannot forge the message they can do some common attacks such as reply attack and relay attack. If we only rely on the quorum system this situation looks like a case where all nodes are faulty. To mitigate this kind of situation we add a simple challenge-response protocol with a nonce.

In addition to the transport level security, we add a simple gossip among nodes in the KV quorum. Gossip is voluntary and the system does not rely on how gossip propagate the message, but this is a good way to keep the whole network sound and secure. We use gossip to extend the write protocol and propagate the revocation message.

## Write

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

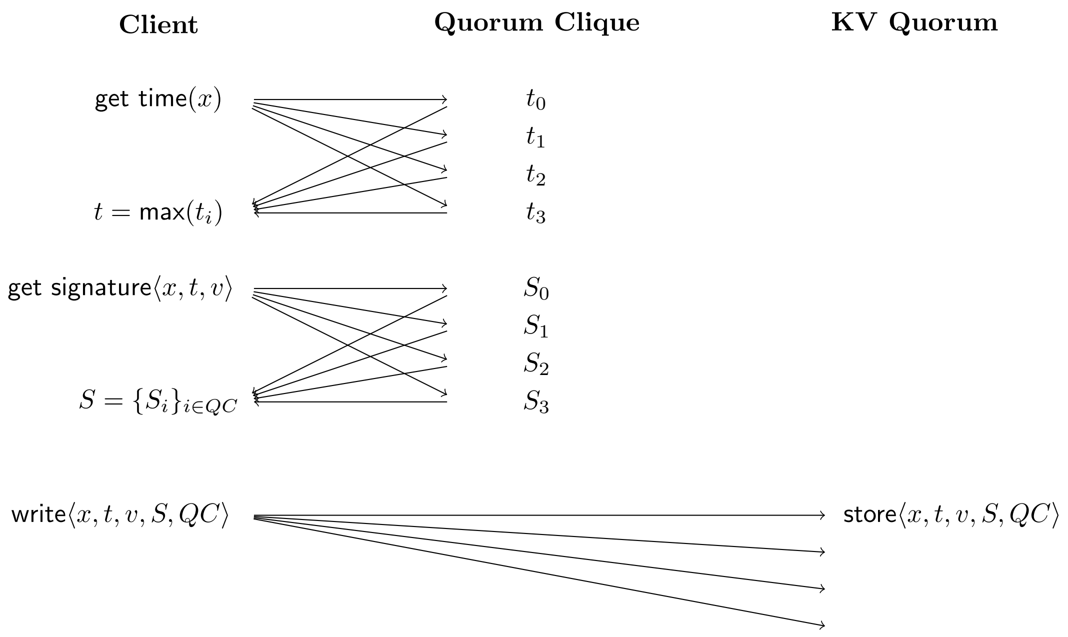

Figure 2: Write Protocol

Each member of KV quorum does the equivocation check. If the check fails it will propagate a revocation message instead of the signed triple to other members. The system does not provide absolute or eventual consistency unlike state machine replication. The communication complexity is $\mathcal{O}(N)$ or $\Theta(|Q|)$ at the time of read / write. The system asynchronously gossips the signed triple and revocation state. The client does not wait for the reply from replicas to conclude a transaction. See the risk analysis. This way we can scale the system without increasing communication complexity beyond the limitation of the quorum threadhold.

Each member of Quorum Clique checks if $\langle x, t\rangle$ has been already used to sign. If a client makes such a request that is attempting to make a equivocation the client will be revoked, though this kind of malicious action is too obvious. Typical "malicious" actions will be found by KV quorum (5. Security Analysis [1]).

## Read

The read operation is done between the client and KV quorum.

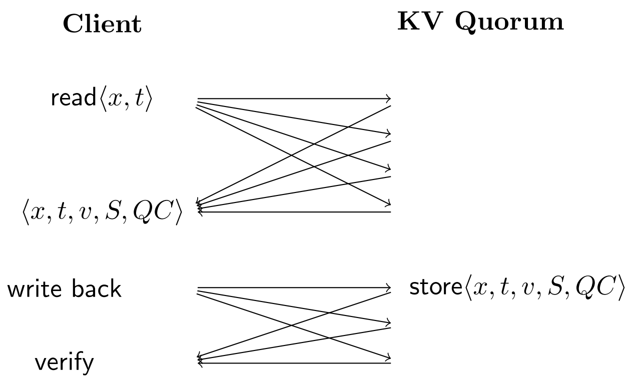

Figure 3: Read Protocol

## 3 Payment Scheme

As stated in Introduction our payment scheme is strictly peer-to-peer and only the result of transactions, which is a new balance, is written in the KV quorum. An image of a trasaction is like this:

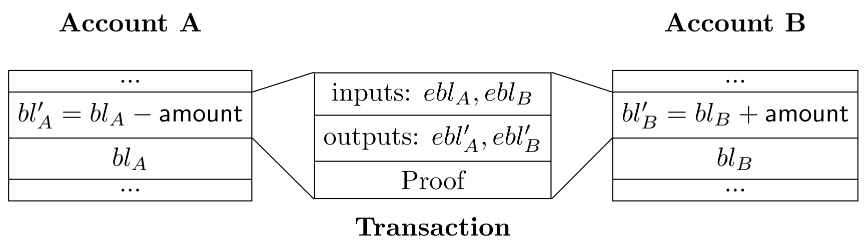

Figure 4: Balance based Transaction

**A** pays amount to **B** "out of band" or in a peer-to-peer manner. This payment protocol is discussed later in this section. Proof is a Zero Knowlege Proof of the existance of $bl$ and $bl'$. We discuss the ZK Proof of the balance in detail below. The total balance is calculated over $ebl$ which is an encryption of $bl$, without decrypting it. By verifying the proof we know the total balance has not changed before and after the transaction and any balance is not negative, i.e.,

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

The quorum system is used to prevent double spending. Specifically we use the "write once" feature of bftkv. We also add an anonymous write feature to write once. Since the variable is never modified it can be written without the owner's quorum certificate.

### 3.1 Zero Knowledge Proof of Balance (zkPoB)

As above, each entity does not have to know the balance of other parties. They only need to know that the total balance has not changed before and after a transaction. We use a ZKP technique to calculate the total balance without letting others know the actual value. Also we need to make sure that each balance is in a certain range so that a negative balance is detected. Putting them together the relation can be:

$$\begin{aligned}\mathcal{R} := \{((bl_i, bl'_i, \beta_i, \beta'_i), (u, e_i, e'_i)) :\\ e_i = u^{\beta_i} g^{bl_i}, e'_i = u^{\beta'_i} g^{bl'_i}, 0 \leq bl'_i < 2^d, \text{ and } \sum_i bl_i = \sum_i bl'_i\}.\end{aligned}$$

With this relation, we have:

$$(u, e_i, e_i') \in L_{\mathcal{R}} \iff \prod_i e_i / \prod_i e_i' = \prod_i u^{\beta_i} u^{-\beta_i'} = \prod_i u^{\beta_{\triangle i}},$$
$$\text{where } \beta_{\triangle i} = \beta_i - \beta_i',$$

if and only if $\sum_i bl_i = \sum_i bl_i'$. Therefore all we need to do is to run a sigma protocol to prove that each party has $\beta_{\triangle i}$, i.e.,

$$\mathcal{R}' := \{((\beta_{\triangle i}, bl_i', \gamma_i), (u, w_i, b_{ij}, c_{ij})) : w_i = u^{\beta_{\triangle i}}, c_{ij} = u^{\gamma_i} g^{b_{ij}}\}$$

that satisfies:

$$(\prod_i e_i / \prod_i e_i' = \prod_i w_i) \wedge (bl_i' = \sum_{j=0..d-1} 2^j b_{ij}).$$

The Sigma protocol of PoB will look like the Figure 5.

$P((\beta_{\triangle i}, bl_i', \gamma_i, \beta_i', b_{ij}), (u, c_{ij}, w_i))$ $\qquad\qquad$ $V(u, c_{ij}, w_i)$

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

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

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

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

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

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

Figure 5: Sigma Protocol for PoB

After finishing the protocol with all parties, check if $\prod_i e_i / \prod_i e_i' \stackrel{?}{=} \prod_i w_i$.

## 3.2 Payment Protocols

We define the transaction as follows:

$$PoB_i := ((u, \{c_{ij}\}_j, w_i), (\{v_{1ij}\}_j, v_{2i}, \{v_{3ij}\}_j, v_{4i}, v_{5i}, \{z_{1ij}\}_j, z_{2i}, \{z_{3ij}\}_j, z_{4i}, z_{5i}))\ (j = 0..d-1)$$
$$Tx := \{ebl_i, ebl'_i, PoB_i\}_i$$

$ebl_i, ebl'_i$ for some $i$ can be nil which is treated as $bl = 0$. We use the Fiat-Shamir transform to convert the Sigma Protocol 5 to $PoB$.

The Figure 3.2 is a payment example: A and B jointly pay 2 and 3 coins 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 does not change.

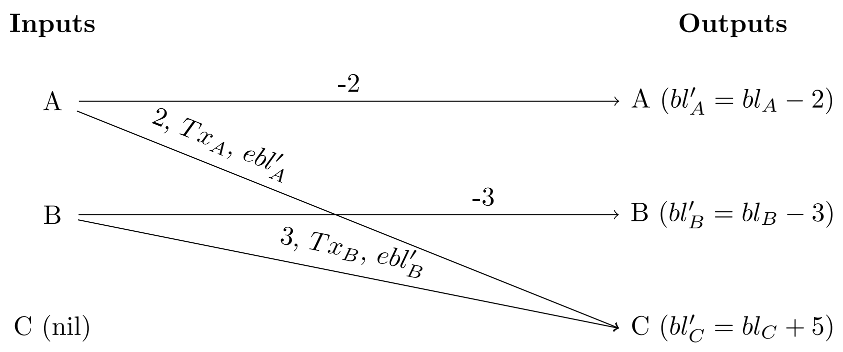

Figure 6: Payment Example

The payment protocol between two parties is done as the Figure 7.

$\mathsf{Tx}_A$ is a transaction that has 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 perform $PoB$ on the new balance ($bl'_B$). The payer constructs a new transaction combining the encrypted balances and PoBs of both parties, then writes it to the quorum. (See the following section for wo.) Tx is verified during the protocol as the 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 $read(Q, Tx.ebl_i) \neq H(Tx)$ then
            return $false$
        end
    end
    return $(\prod_i PoB_i.e)/(\prod_i PoB_i.e') \overset{?}{=} \prod_i PoB_i.w$
```

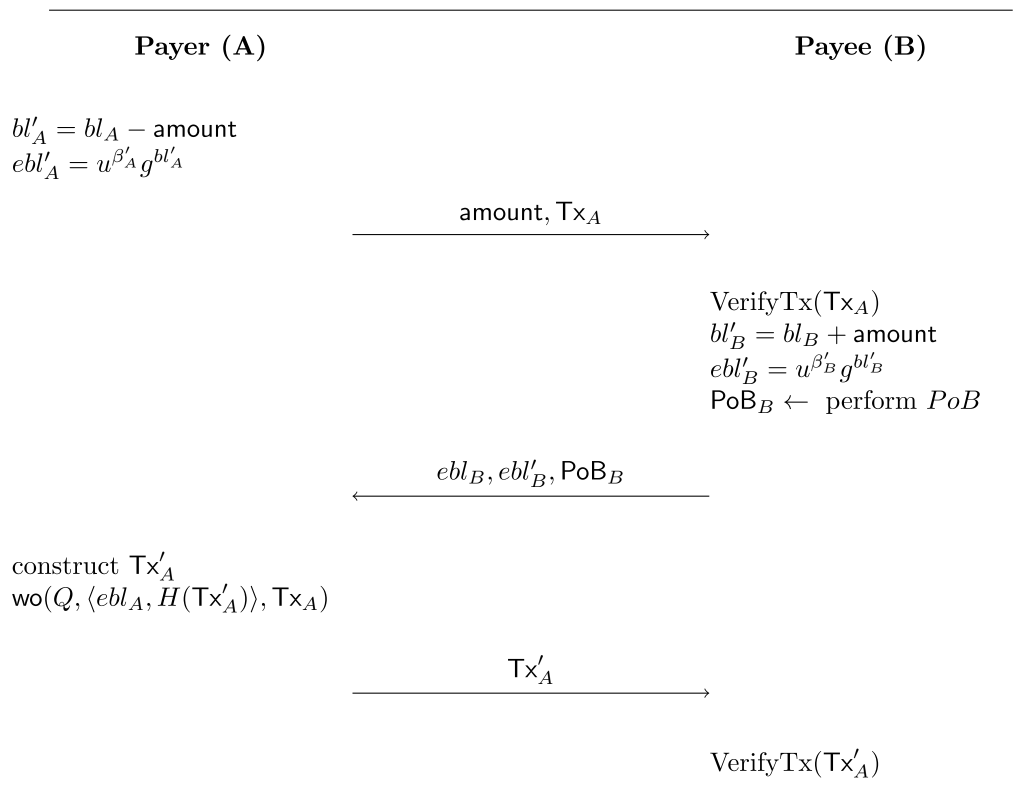
Payer (A)
Payee (B)
$bl'_A = bl_A - \mathsf{amount}$
$ebl'_A = u^{\beta'_A} g^{bl'_A}$
$\mathsf{amount}, \mathsf{Tx}_A$
VerifyTx$(\mathsf{Tx}_A)$
$bl'_B = bl_B + \mathsf{amount}$
$ebl'_B = u^{\beta'_B} g^{bl'_B}$
$\mathsf{PoB}_B \leftarrow$ perform $PoB$
$ebl_B, ebl'_B, \mathsf{PoB}_B$
construct $\mathsf{Tx}'_A$
$\mathsf{wo}(Q, \langle ebl_A, H(\mathsf{Tx}'_A)\rangle, \mathsf{Tx}_A)$
$\mathsf{Tx}'_A$
VerifyTx$(\mathsf{Tx}'_A)$

Figure 7: Payment Protocol

## 3.3 Write Once with PoB

In bftkv, to write a variable to a quorum the client signs the tuple with its own Quorum Certificate so it enforces the TOFU policy. In other words, the client becomes the owner of the variable. But with the Write Once property the variable will never be modified in any way therefore such ownership does not make much sense. We extend the write function for Tx. Instead of signing the tuple with QC we use PoB to guarantee the validity of the key ($ebl$) - value (Tx'). The Quorum Clique runs the Algorithm 1 over Tx and if it verifies it signs the tuple:

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

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

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

The KV quorum checks equivocation the same manner as the usual case so double spending will be detected with the possibility analyzed in bftkv [1]. Not only that, the quorum system prevents replay attacks as well. The key $ebl$ is used only once for a specific Tx. If someone copies ($ebl, ebl', \mathsf{PoB}$) and makes up a new Tx' it will be different than the original Tx that has consumed $ebl$, so the attack will be detected.

## 3.4 Audit

The key ($ebl$) / value ($Tx$) pairs form a DAG. The topological order is guaranteed – a new balance must be created from existing balances, except the genesis. In the Figure 8, when $ebl_4$ is spent, $Tx_2$ is verified as $\mathsf{VerifyTx}(Tx_2)$. It is not necessary to track back beyond $ebl_3$ or $ebl'_1$ as when they are spent (i.e., written to the quorum) the past transactions must have been verified in the same manner.

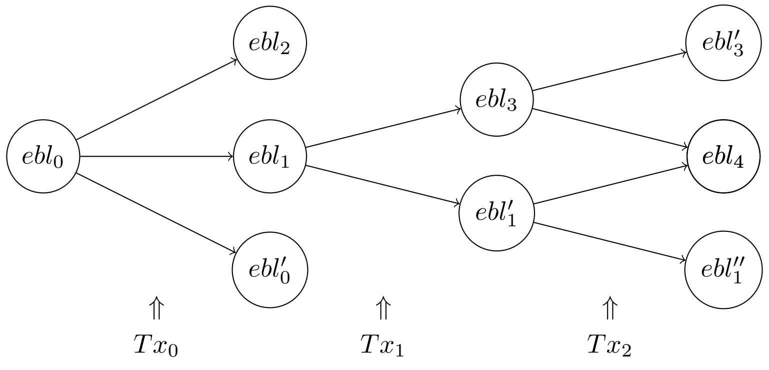

Figure 8: Balance Chain

But, what if the quorum system is corrupted or the payer is somehow able to trick the quorum? In this case his balance might have been created out of thin air. As the security analysis of bftkv, even if faulty nodes outnumber the threshold it is still possible to detect the equivocation and in the transaction case, it will be immediately invalidate the balance and abort the transaction. Even if the probability is low there must be some fail-safe mechanism.

When an balance (*ebl*) is written to a quorum each member verifies PoB of the transaction that has created *ebl*, then store the hash value of the new transaction that spends *ebl*. Once Tx is verified it can be disposed. Since Tx is verified anyone who reads the variable (*ebl*) knows the value ($H$(Tx')) is valid without backtracking the balance to the genesis, but because of the above concern it would be good to have a way to validate it without relying on the quorum. We propose to have an archive service of Tx for this purpose. It is possible to keep Tx of all transaction in each quorum member but it can be too big and it is not necessary to keep the replication. The archive service is a simple key value store that takes *ebl* and returns Tx. The client checks if ReadArchive(*ebl*) = $H$(Tx) and verifies each PoB in Tx. The audit can be done as the Algorithm Audit.

```
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 $read(Q, Tx.ebl'_i) \neq H(Tx) \vee$ VerifyTxDeeply($ReadArchive(Tx.ebl_i)$) $= false$ then
              return false
          end
      end
      return $(\prod_i PoB_i.e)/(\prod_i PoB_i.e') \overset{?}{=} \prod_i PoB_i.w$
```

## 3.5 Anonymity

Tx does not include any information that can be linkable to users or devices. The balance is encrypted so it cannot be a clue to track transactions. However, the payment protocol is peer to peer, which means the payers and payees know each other's identity. $ebl$ can be linked to such identity even if it is pseudonym and we cannot anonymize them who are involed in the payment. The question is $ebl$ can be traceable? Basically $\mathsf{Tx}$ has $n$ pairs of input / output. When you get $\mathsf{Tx}$ from the archive service, you know $ebl$ is one of the $\mathsf{outputs}$ but you do not know what input is corresponding to it. (Also it can be newly created with that transaction, where the input is $\mathsf{nil}$.) The same can be said for forward direction. By getting $\mathsf{Tx}$ of $ebl'$ you can trace it forwardly but you do not know what output is corresponding to the input. In theory if someone wants to trace a balance he will need to keep tracking $\prod_i^{\mathsf{gen}} \mathsf{Tx}_i.n$ balances for $\mathsf{gen}$ generations, which is $\Omega(2^{\mathsf{gen}})$.

If the transaction chain is short and simple it would be easy to trace the transactions. It is possible to add "dummy" balances that have 0 value to both inputs and outputs just to obscure the actual transaction. Such dummy balances can be consumsed in other transactions without affecting other balances to make the trace difficult. This makes Audit more inefficient as well though.

## 4 Wallet

The payment scheme does not require any long term keys such as signing keys. Instead, each balance has a corresponding witness $(\beta_\triangle, bl, \gamma)$. The balance $(bl, \gamma)$ can be outside of HW and the range proof can be calculated in AP. Only $\beta_\triangle$ has to be stored in the HW. In the ZK protocol only $(t_5, v_{5i}, z_{5i})$ is calculated inside the HW.

## 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"