Delegation and Stake Locking in Cardano SL

© IOHK 2017

Abstract

In this article we define requirements for delegation scheme that are imposed by the real-life concerns and weren't considered in the original Ouroboros paper. We show validity of such concerns and describe an approach that addresses those, deviating as little as possible from the original proposal.

Contents

Delegation in Cardano SL

Requirements

We need a delegation scheme for Cardano SL. This scheme:

1. Should allow us to delegate/redelegate/revoke rights on stake owned by user.
2. Shouldn't require to expose public key on which money are kept to perform delegation.
3. Should be easy to integrate with HD wallets, i.e. to easily delegate from all keys of HD wallet tree/subtree to somebody.

The important concern is the fact that new address' types can be introduced via softfork in the future, and we don't know in advance about semantics of these types.

Original Scheme

The concept of delegation is simple: any stakeholder can allow a delegate to generate blocks on her behalf. In the context of our protocol, where a slot leader signs the block it generates for a certain slot, such a scheme can be implemented in a straightforward way based on proxy signatures. A stakeholder can transfer the right to generate blocks by creating a proxy signing key that allows the delegate to sign messages of the form (st, d, sl_j) (i.e., the format of messages signed in Protocol π_{DPoS} to authenticate a block). Protocol π_{DPoS} is described in Ouroboros paper, page $[33]$. In order to limit the delegate's block generation power to a certain range of epochs/slots, the stakeholder can limit the proxy signing key's valid message space to strings ending with a slot number sl_j within a specific range of values. The delegate can use a proxy signing key from a given stakeholder to simply run Protocol π_{DPoS} on her behalf, signing the blocks this stakeholder was elected to generate with the proxy signing key.

This scheme is secure due to the *Verifiability and Prevention of Misuse* properties of proxy signature schemes, which ensure that any stakeholder can verify that a proxy signing key was actually issued by a specific stakeholder to a specific delegate and that the delegate can only use these keys to sign messages inside the key's valid message space, respectively. *Verifiability and Prevention of Misuse* is described in the paper "Secure Proxy Signature Schemes for Delegation of Signing Rights", page $[2]$.

We remark that while proxy signatures can be described as a high level generic primitive, it is easy to construct such schemes from standard digital signature schemes through delegation-by-proxy. In this construction, a stakeholder signs a certificate specifying the delegates identity (e.g., its public key) and the valid message space. Later on, the delegate can sign messages within the valid message space by providing signatures for these messages under its own public key along with the signed certificate. As an added advantage, proxy signature schemes can also be built from aggregate signatures in such a way that signatures generated under a proxy signing key have essentially the same size as regular signatures.

An important consideration in the above setting is the fact that a stakeholder may want to withdraw her support to a stakeholder prior to its proxy signing key expiration. Observe that proxy signing keys can be uniquely identified and thus they may be revoked by a certificate revocation list within the blockchain.

Eligibility Threshold

Delegation as described above can ameliorate fragmentation that may occur in the stake distribution. Nevertheless, this does not prevent a malicious stakeholder from dividing its stake to multiple accounts and, by refraining from delegation, induce a very large committee size. To address this, as mentioned above, a threshold T, say 1%, may be applied. This means that any delegate representing less a fraction less than T of the total stake is automatically barred from being a committee member. This can be facilitated by redistributing the voting rights of delegates representing less than T to other delegates in a deterministic fashion (e.g., starting from those with the highest stake and breaking ties according to lexicographic order).

Suppose that a committee has been formed, $C_1, ..., C_m$, from a total of k draws of weighing by stake. Each committee member will hold k_i such votes where $\sum_{i=1}^{m} k_i = k$. Based on the eligibility threshold above it follows that $m \leq T^{-1}$ (the maximum value is the case when all stake is distributed in T^{-1} delegates each holding T of the stake).

Original Scheme Implementation

The original scheme of delegation is implemented in Cardano SL by two different delegation types: heavyweight delegation and lightweight delegation.

Heavyweight Delegation

Heavyweight delegation is using stake threshold T. It means that stakeholder has to posses not less than T in order to participate in heavyweight delegation. The value of this threshold is defined in the configuration file.

Moreover, the issuer stakeholder must have particular amount of stake too, otherwise it cannot be a valid issuer.

Proxy signing certificates from heavyweight delegation are stored within the blockchain. Please note that issuer can post only one certificate per one epoch.

Lightweight Delegation

In contrast to heavyweight delegation, lightweight delegation doesn't require that delegate posses T-or-more stake. So lightweight delegation is available for any node. But proxy signing certificates for lightweight delegation are not stored in the blockchain, so lightweight delegation certificate must be broadcasted to reach delegate.

Later lightweight PSK can be verified given issuer's public key, signature and message itself.

Why Two Delegations?

Heavyweight certificates are stored in the blockchain, so delegated stake may participate in MPC by being added to the stake of delegate. So delegate by many heavyweight delegations may accumulate enough stake to pass eligibility threshold.

On the contrary, stake for lightweight delegation won't be counted in delegate's MPC-related stake.

Revocation Certificate

Revocation certificate is a special certificate that issuer creates to revoke delegation. Both heavyweight and lightweight delegation can be revoked, but not in the same way.

The revocation certificate is just a normal one where issuer and delegate are the same (in other words, issuer delegates to himself).

To revoke lightweight delegation issuer sends revocation certificate to the network and *asks* to revoke delegation, but it cannot *enforce* this revocation, since lightweight PSKs are not the part of the blockchain.

Revocation of heavyweight delegation is handled other way. Since proxy signing certificates from heavyweight delegation are stored within the blockchain, revocation certificate will be committed in the blockchain as well. In this case the node removes heavyweight delegation certificate which was issued before revocation certificate. But there are two important notes about it.

1. If the committed heavyweight delegation certificate is in the node's memory pool yet, and revocation certificate was committed as well, the delegation certificate will be removed from the memory pool. Obviously, in this case delegation certificate will never be added to the blockchain.
2. If a user commits heavyweight delegation certificate and *after that* he loses money, he still can revoke that delegation, even if by that time he does not have enough money (i.e. less than threshold T mentioned above).

Original Scheme Drawbacks

Current implementation of delegation scheme described below uses proxy signing key scheme, which itself requires a public key being associated with stakeholder and used to sign delegation. Initially it was thought this public key to be an actual key which holds money, but this decreases security by exposing public key of address before spending money from it. We propose a solution for this concern.

Transaction Distribution

Transaction distribution is another part of Cardano SL, not directly related to delegation, but one we can exploit for its benefit.

Some addresses have multiple owners, which poses a problem of stake computation as per Follow-the-Satoshi each coin should only be counted once towards each stakeholder's stake total. Unlike balance (real amount of coins on the balance), stake gives user power to control different algorithm parts: being the slot leader, voting in Update system, taking part in MPC/SSC.

Suppose we have an address A. If it is a `PublicKey`-address it's obvious and straightforward which stakeholders should benefit from money stored on this address, though it's not for `ScriptAddress` (e.g. for $2-of-3$ multisig address implemented via script we might want to have distribution $[(A,1/3),(B,1/3),(C,1/3)]$). For any new address' type introduced via softfork in the future it might be useful as well because we don't know in advance about semantics of the new address' type and which stakeholder it should be attributed to.

Transaction distribution is a value associated with each transaction's output, holding information on which stakeholder should receive which particular amount of money on his stake. Technically it's a list of pairs composed from stakeholder's identificator and corresponding amount of money. E.g. for output $(A,100)$ distribution might be $[(B,10),(C,90)]$.

Transaction distributions are considered by both slot-leader election process and Richmen Computations.

This feature can be used in similar way to delegation, but there are differences:

1. There is no certificate(s): to revoke delegation A has to move funds, providing different distribution.
2. Only part of A's balance associated with this transaction output is delegated. This can be done in chunks per balance parts (on contrary, delegation requires you to delegate all funds of whole address at once).

Modified Delegation Proposal

Protocol Participation Keys and Spending Keys

Transaction distribution is a practical way to split spending keys and protocol participation keys. Protocol participation keys allow to control stake, associated with transaction output.

In transaction output we specify spending key data. Thus:

- for public key address we specify spending key hash,
- for script address some spending key will be used within script probably.

Let's consider basic use case. We want user U to send v coins to our address R. Then we find transaction $U \rightarrow R$ in the blockchain, which shows us money were sent. We call R a *receiving* address.

Let's assume we have two more addresses:

1. K, *keeper* address,
2. D, *delegator* address.

Next we form a new transaction $R \rightarrow K$ (sending all v coins from R to K) with $txDistr = [(D, v)]$. After this transaction will be processed, funds would be contained on address K, but the right to issue blocks and participate in slot leader election would be held by D. This way we effectively decoupled key which controls money and key which is used for protocol maintenance.

Usage with HD Wallets

For HD wallets, we reserve $(root, 0)$ key as a delegator. We use $(root, k > 1, 2*i)$ keys as receiving addresses and $(root, k > 1, 2 * i + 1)$ keys as keepers.

Delegation or redelegation of the whole HD wallet structure then is as simple as issuing a single lightweight/heavyweight certificate for an address $(root, 0)$.

Modified Delegation Proposal Analysis

As careful reader may observe, when transaction with transaction distribution is being sent, money are sent to the key K, but D is responsible for delegation. This way if even D public component will be exposed (which is case when we would like to delegate with certificate), K's public key won't be exposed till money are sent. This satisfies requrement 2.

Section **Usage with HD Wallets** descirbes how we satisfy requirement 3.

Stake Locking in Cardano SL

The Bootstrap era is the period of Cardano SL existence that allows only fixed predefined users to have control over the system. The set of such users (the bootstrap stakeholders) and propotion of total stake each of them controls is defined in genesis block.

Purpose of Bootstrap era is to address concern that at the beginning of mainnet majority of stake will probably be offline (which protocol broken at start). Bootstrap era is to be ended when network stabilizes and majority of stake is present online.

The next era after Bootstrap is called the Reward era. Reward era is actually a "normal" operation mode of Cardano SL as a PoS-cryptocurrency.

Requirements

1. During Bootstrap era stake in Cardano SL should be effectively delegated to a fixed set of keys S.
2. $|S| \leq 3$
3. Stake should be distributed among $s \in S$ in fixed predefined propotion, e.g. $2 : 5 : 3$.
4. At the end of Bootstrap era stake should be unlocked:
    1. Ada buyers should be able to participate in protocol themselves (or delegate their rights to some delegate not from S).
    2. Each Ada buyer should explicitly state she wants to take control over her stake.
        - Otherwise it may easily lead to situation when less than majority of stake is online once Reward era starts.
    3. Before this withdrawing stake action occurs, stake should be still being controlled by S nodes.
    4. *(Optional)* Stake transition during unlocking should be free for user.

Proposal

Let us now present the Bootstrap era solution:

1. Initial $utxo$ contains all the stake distributed among $gcdBootstrapStakeholders$. Initial $utxo$ consists of $(txOut, txOutDistr)$ pairs, so we just set $txOutDistr$ in a way it sends all coins to $gcdBootstrapStakeholders$ in proportion specified in genesis block.
2. While the Bootstrap era takes place, users can send transactions changing initial $utxo$. We enforce setting $txOutDistr$ for each transaction output to spread stake to $gcdBootstrapStakeholders$ in proportion specified by genesis block. This effectively makes stake distribution is system constant.
3. When the Bootstrap era is over, we disable restriction on $txOutDistr$. Bootstrap stakeholders will vote for Bootstrap era ending: special update proposal will be formed, where a particular constant will be set appropriately to trigger Bootstrap era end at the point update proposal gets adopted. System operates the same way as in Bootstrap era, but users need to explicitly state they understand owning their stake leads to responsibility to handle the node. For user to get his stake back he should send a transaction, specifying delegate key(s) in $txOutDistr$. It may be the key owned by user himself or the key of some delegate (which may also be one or few of $gcdBootstrapStakeholders$).

Multiple Nodes with Same Key

To reduce the size of transactions, we want to have set of bootstrap stakeholder as small as possible. In principle it should be as small as number of actual parties involved (e.g. IOHK, CGG, CF each holding single key).

This way it's handy to have secret key $s \in gcdBootstrapStakeholders$ can be distributed accross multiple nodes (because usual case is you want to have multiple nodes operating in different data centers to provide reliable service).

Simplest proposal to distribute key accross multiple nodes would be to run multiple nodes with the same secret key and have them participating in the algorithm in round-robin fashion, in particular:

1. Create blocks in predefined order. If key is a slot leader for $slot_1$, $slot_3$ and $slot_5$, then $node_0$ creates block at $slot_1$, $node_1$ - at $slot_3$, and $node_2$ - at $slot_5$.
2. Create MPC payloads accurately. If key is obliged to post M commitments, openings, shares, then $node_0$, $node_1$ and $node_2$ will post $\frac{M}{3}$ each.

Free Transaction for Bootstrap Era

Delegating stake back to user is done via transaction. But transactions cost money (via fees), which violates requirement 4.4 (which is marked as an optional, but yet desirable).

As a solution to this issue we could make a snapshot of utxo U at the moment Bootstrap era ends and don't require fees to be withdrawn from any transaction output, contained in U. This will effectively make delegation transition transaction free.

Follow-up on Merkle Tree Idea

Address attribute malleability issue

Designing data structures for Cardano SL, we widely adopted idea of putting attributes to various data structures, including:

- transactions,
- addresses,
- update proposals,
- block, block header.

This was introduced for leaving us an option to include additional data to these structures via soft fork update.

Most of these data structures are being signed before putting to the blockchain, only exemption for this rule is address. This doesn't seem to open significant attack surface. Addresses are open data exchanged between users off chain. They may appear on chain only via inclusion into transaction which is signed, this way addresses attributes cannot be subject for modification by adversary.

But one design flow of scheme is that if user U asks user V to send transaction to his address $A = \langle A_{pkhash}, A_{attrs} \rangle$, V can modify A_{attrs} and send funds to $A' = \langle A_{pkhash}, A_{attrs}' \rangle$. This leads to awkward situation when U actually has access to A', i.e. may spend funds from it, but in fact $A' \neq A$ and attrs may contain some attributes which are sensitive for workflow used by U. For instance, if we store delegate address as attribute, V may replace delegate and U will receive correct transaction, but funds would not be delegated to appropriate delegate.

In ideal world we would prefer to restrict such cases.

Address Structure Modification with Use of Merkle Tree

Let's consider public key addresses. Public key address is at the moment composed from:

- public key hash,
- attribute list.

As is mentioned in previous section, public key address is now subject to malleability issue.

But there is an idea on how to solve this issue. Instead of defining address as $Address = (PubKeyHash, Attributes)$ we may better define $Address' = (MerkleRoot[PubKey, Attributes], Attributes)$. By $MerkleRoot$ here we mean root of Merkle tree built upon defined list of data.

Let's consider what will change.

Changes in Workflow

Sending transaction from B to A is same. A gives B an $Address'$. B sends money to it. This $Address'$ is stored on the blockchain just in same way as old $Address$ was stored.

When A wants to withdraw these funds, he uses public key, corresponding to this $Address'$. By combining public key and attributes with $MerkleRoot$ one may build $Address'$ and check if it's same for $utxo$ entry that is to be spend. Interesting property here is that B can't change attributes without corrupting Merkle root (because B doesn't know public key hash). And if Merkle root presented in address is corrupted, then address is invalid by design and thus B sends money to nowhere (which is B's problem). This effectively solves address attribute malleability.

Application of Modification to Delegation Concerns

With having attributes as an unmodifiable part of address, we can get rid of $txOutDistr$ feature and instead have delegate key distribution field as a part of address. This way we won't need aforementioned flow with recieving money to address R, then sending $R \to K$ with $txOutDistr = [(D, value)]$. We just present $(K, Attributes\{delegateDistr = [(D, value)]\})$ as an address and then it's guaranteed that we will recieve money to K and have D as protocol participation key (which was our intention).