Design Specification for Delegation and Incentives in Cardano

Deliverable SL-D1

Philipp Kant <philipp.kant@iohk.io>

Lars Brünjes <lars.bruenjes@iohk.io>

Duncan Coutts <duncan.coutts@iohk.io>

Project: Shelley Ledger

Type: Deliverable
Due Date: 23th July 2020

Responsible team: Formal Methods Team
Editor: Philipp Kant, IOHK
Team Leader: Philipp Kant, IOHK

Version 1.21
23th July 2020

Dissemination Level		
PU	Public	√
CO	Confidential, only for company distribution	
DR	Draft, not for general circulation	

Contents

1 Purpose **1**

2 Requirements **2**
- 2.1 Functional Requirements . . . 2
  - 2.1.1 Proof of Eligibility . . . 2
  - 2.1.2 Visibility of Delegation on the Blockchain . . . 2
  - 2.1.3 Restricting Chain Delegation . . . 2
  - 2.1.4 Cheap Re-Delegation . . . 2
  - 2.1.5 Neutral Addresses . . . 2
  - 2.1.6 Multi-Signature Addresses . . . 2
  - 2.1.7 Multi-Signature Delegation . . . 3
- 2.2 Security Requirements . . . 3
  - 2.2.1 Sybil Attack Protection at Stake Pool Level . . . 3
  - 2.2.2 Address Non-malleability . . . 3
  - 2.2.3 Public Payment Keys Should not be Disclosed Prematurely . . . 3
  - 2.2.4 Mitigate Key Exposure . . . 3
  - 2.2.5 Handle Inactive Stake Pools . . . 4
  - 2.2.6 Avoid Hard Transition . . . 4
  - 2.2.7 Change Delegation Without Payment Key . . . 4
- 2.3 Non-functional Requirements . . . 4
  - 2.3.1 Asymptotic space and time complexity . . . 4
  - 2.3.2 Minimise economic attacks . . . 4
- 2.4 Requirements to Preserve Existing Features . . . 4
  - 2.4.1 Master Recovery Key . . . 4
  - 2.4.2 Address Recognition . . . 5
  - 2.4.3 Wallet should be Runnable on Independent Devices . . . 5
  - 2.4.4 Maintain Privacy . . . 5
  - 2.4.5 Short Addresses . . . 5
  - 2.4.6 No lookup of old blocks . . . 5
- 2.5 Design Goals . . . 5
  - 2.5.1 No Special Wallet for Stake Pool Operators . . . 5

3 Design of Delegation **6**
- 3.1 Overview of Delegation . . . 6
- 3.2 Addresses and Credentials . . . 6
  - 3.2.1 On Pointer Addresses . . . 8
  - 3.2.2 On Enterprise Addresses . . . 9
  - 3.2.3 Reward Accounts . . . 9
  - 3.2.4 On Byron Addresses . . . 9
  - 3.2.5 HD Wallet Structure in Shelley . . . 10
  - 3.2.6 Address Recognition . . . 10
- 3.3 Certificates and Registrations . . . 10
  - 3.3.1 Certificates on the Blockchain . . . 10
  - 3.3.2 Certificate Replay Prevention . . . 11
  - 3.3.3 Stake Address Registration Certificates . . . 12
  - 3.3.4 Stake Pool Registration Certificates . . . 12
  - 3.3.5 Single Operator, Possibly Multiple Owners . . . 14
  - 3.3.6 Delegation Certificates . . . 14
  - 3.3.7 Operational Key Certificates . . . 15

3.3.8 Certificate Precedence and Validity 16
3.4 Delegation Relations 17
3.4.1 Address Delegation Relation 17
3.4.2 Stake Pool Delegation Relation 17
3.4.3 Overall Stake Distribution 17
3.4.4 Chain Delegation 17
3.5 State Tracking for delegation 18
3.5.1 Stake Addresses 18
3.5.2 Reward Accounts 18
3.5.3 Stake Pools 18
3.5.4 Active Delegation Certificates 19
3.5.5 Stake per Stake Address 19
3.6 Slot Leader Schedule and Rewards Calculation 19
3.7 Block Validity and Operational Key Certificates 19
3.8 Transition to Decentralization 20
3.8.1 Motivation 20
3.8.2 Proposal 20
3.8.3 Rewards during the Transition Phase 21
3.8.4 Transition Plan 21
3.9 Rewards 22
3.9.1 Distributing Rewards 23
3.10 Fees 24
3.10.1 Transaction fees 24
3.10.2 Deposits 24
3.11 Time to Live for Transactions 25
3.12 Robustness at the Epoch Boundary 25
3.12.1 Calculating the Leader Schedule 25
3.12.2 Calculating and Distributing Rewards 25
3.13 Wallet Recovery Process 25
3.13.1 Trees of Depth 1 26
3.13.2 Taller Trees 27
3.13.3 Maximal Address Gap 27

4 Delegation Scenarios **28**
4.1 Stake Pool Registration 28
4.2 Stake Pool Metadata 28
4.3 Display of Stake Pools in the Wallet 29
4.4 Basic Delegation 30
4.5 Delegation of Cold Wallets 31
4.6 Self Delegation 31

5 Design of Incentives **32**
5.1 Overview of Incentives 32
5.2 Parameters 33
5.3 Reminder: Stake Pool Registration 33
5.4 Epoch Rewards 34
5.4.1 Transaction Fees 34
5.4.2 Deposits 34
5.4.3 Monetary Expansion 34
5.4.4 Treasury 35
5.5 Reward Splitting 35
5.5.1 Relative Stake: Active vs Total 35

5.5.2 Stake, Performance, and Block Production . . . 35
5.5.3 Pool Rewards . . . 37
5.5.4 Reward Splitting inside a pool . . . 38
5.6 Non-Myopic Utility . . . 38
5.6.1 Pool Desirability and Ranking . . . 38
5.6.2 Non-Myopic Pool Stake . . . 39
5.6.3 Non-Myopic Pool Operator Rewards . . . 39
5.6.4 Non-Myopic Pool Member Rewards . . . 39
5.6.5 Replacing Apparent Performance . . . 39
5.7 Utility . . . 39
5.8 Claiming Rewards . . . 40
5.9 Information in Daedalus . . . 40
5.10 Deciding on Good Values for the Parameters . . . 40
5.10.1 k . . . 40
5.10.2 a_0 . . . 40
5.10.3 ρ . . . 42
5.10.4 τ . . . 44

6 Satisfying the Requirements 44

A Assessment of Rewards Sharing Mechanisms 45
A.1 General Considerations . . . 45
A.1.1 Hierarchy of desirability of reward distribution . . . 46
A.1.2 Summary of key points of when rewards are calculated . . . 46
A.2 Approaches that are Ruled Out . . . 47
A.2.1 Manual Sharing . . . 47
A.2.2 Automatically Issue Transactions Each Epoch . . . 47
A.2.3 Let Members Collect Rewards . . . 47
A.3 Feasible Approaches . . . 47
A.3.1 Automatic UTxO Updates . . . 47
A.3.2 Lotteries per Stake Pool . . . 48
A.3.3 Reward accounts per stake address . . . 49

B Deposits 50
B.1 Motivation . . . 50
B.2 Mechanism . . . 50

C Design Option: Stale Stake 51

D FAQ 52
D.1 Why will stake pools accept new stake pool registration certificates? . . . 52
D.2 Won't stake pools reject delegation certificates that delegate away from them? . . . 52

E Transaction Metadata 53
E.1 Motivation and design goals . . . 53
E.2 Detail . . . 53
E.3 Explanation and use . . . 54
E.4 Binary schema . . . 55

References 55

List of Figures

1 Positioning of this Deliverable (outlined in red). vii
2 Addresses and Credentials 7
3 Relationships between the keys, addresses, and certificates 11
4 Effect of different choices for a_0 43

Change Log

Rev.	Date	Who	Team	What
1	2018-12-18	PK, LB, DC	FM (IOHK)	First version that is considered stable enough to warrant V1. Some things still need to be pinned down.
2	2019-01-07	PK, LB, DC	FM (IOHK)	Changes after the first day of the Berlin workshop. TTL for transactions; stakepool registration.
3	2019-01-08	PK, LB, DC	FM (IOHK)	Changes after the second day of the Berlin workshop. Avoid Contention at Epoch Boundary. Refunds after stake pool retirement. Clarifications and corrections.
4	2019-03-01	PK, LB, DC	FM (IOHK)	Incorporating further input from the workshop in Berlin, and following discussions, into the document. Transactions have to have at least one UTxO style input; stake pool metadata formats; choice of KES scheme; deposits information; clarify certificate replay protection; fix rewards to treasury for unregistered stake pool key; update block validity to require operational key; additions to Operational Key section.
5	2019-04-05	PK, LB, DC	FM (IOHK)	Rewrote the chapter on rewards.
6	2019-04-08	PK, LB, DC	FM (IOHK)	General review of the document. Mostly small things. Consistent wording, spelling, readability, removed some obsolete things.
7	2019-04-11	PK, LB, DC	FM (IOHK)	Some subtle corrections in the rewards chapter after review by Aikaterina. First version officially published on the IOHK blog.
8	2019-05-17	PK, LB, DC	FM (IOHK)	Some clarifications in response to review by the auditors.
9	2019-06-07	PK, LB, DC	FM (IOHK)	Update section on script addresses.
10	2019/10/09	Kevin Hammond	FM (IOHK)	Added standard cover page.
11	2020-02-28	PK	FM (IOHK)	Clarify when to use active/total stake.
12	2020/03/11	DC	FM (IOHK)	Document the metadata feature.
13	2020-06-12	PK	FM (IOHK)	Rewrite chapter on addresses. Now includes multi-sig, and is clearer about the distinction of payment addresses, stake addresses, credentials.
14	2020-06-15	PK	FM (IOHK)	Ensure consistent wording, after the change in terminology in the last edit.

Rev.	**Date**	**Who**	**Team**	**What**
15	2020-07-06	PK	FM (IOHK)	Correct sentence about enforcing pledge, to be consistent with implementation; pools do not receive rewards, but can still create blocks when they fail to meet their pledge.
16	2020-07-06	PK	FM (IOHK)	Minor changes after audit. Nothing that affects the implementation.
17	2020-07-23	PK	FM (IOHK)	Change: undistributed rewards go to the reserves, not to the treasury.
18	2020-10-08	J. Corduan	FM (IOHK)	Change: include member stake in the non-myopic stake calculation. Change: replaced average apparent performance usage with references to the stake pool ranking document.

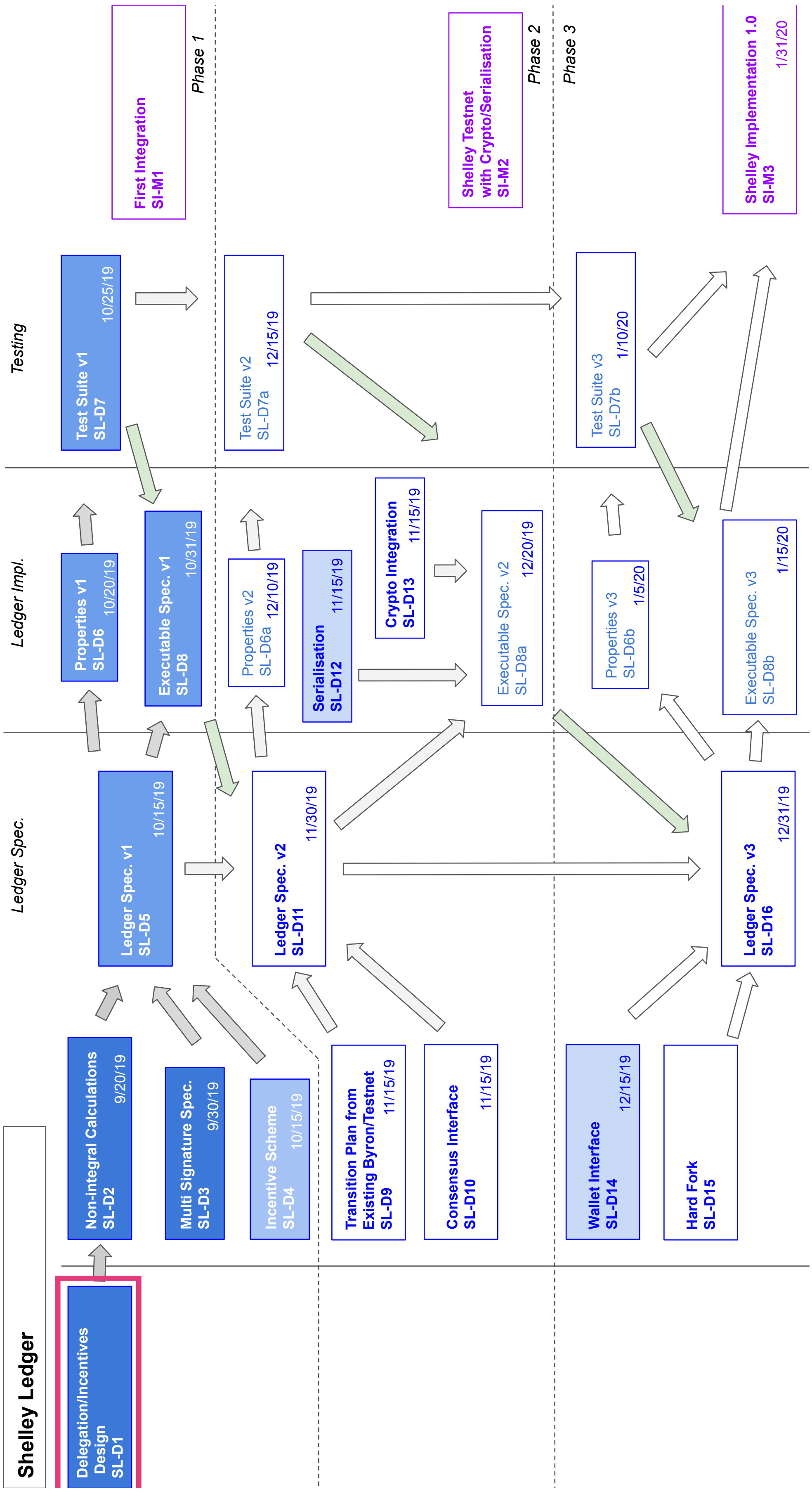

Figure 1: Positioning of this Deliverable (outlined in red).

Engineering Design Specification for Delegation and Incentives in Cardano–Shelley

An IOHK technical report

Philipp Kant
philipp.kant@iohk.io

Lars Brünjes
lars.bruenjes@iohk.io

Duncan Coutts
duncan@well-typed.com
duncan.coutts@iohk.io

February 28, 2023

Abstract

This document describes the requirements and design for a delegation and incentives mechanism to be used in the Shelley release of Cardano.

List of Contributors

Lars Brünjes, Jared Corduan, Duncan Coutts, Matthias Güdemann, Philipp Kant, Dimitris Karakostas, Aggelos Kiayias, Elias Koutsoupias, Mario Larangeira, Damian Nadales, Aikaterini-Panagiota Stouka.

1 Purpose

Delegation will allow holders of ada to transfer their rights to participate in the proof of stake (*PoS*) protocol to *stake pools*. Stake pools are run by *stake pool operators* (sometimes also called *pool leaders*, though we try to avoid the term in this document to avoid confusion with slot leaders), and a person delegating to a stake pool is called *delegator*, *member*, or *participant* of a stake pool.

Introducing delegation is important to increase the stability and performance of the system:

- We cannot expect every holder of ada to continuously run a node that is well-connected to the rest of the network, in order to write a block on rare occasions. Some users might lack the expertise to do so. Most users will not have enough stake to warrant running their own node. Delegation allows all holders of ada to participate in the protocol, regardless of their technical abilities and the amount of stake that they hold. Thus, we expect less stake to be offline, making the system faster and more resilient against an adversary.
- Even if every user were to run a node that was online all the time, it would be hard to keep all those nodes well enough in sync to avoid forks and still keep a short slot length. Our delegation design is aimed at keeping the number of nodes that produce a significant amount of blocks reasonably small (about 100 or 1000 nodes), so that effective communication between them is feasible.

This document covers the design of necessary additions to Cardano in order to support and incentivise delegation.

2 Requirements

The delegation mechanism should meet a number of requirements. They can be grouped into:

- functional requirements that the delegation system should provide;
- requirements to the security (both of the overall system and the funds of individual users);
- non-functional requirements; and
- existing features that should not be impeded when we add delegation to the system.

Requirements specific to the rewards distribution mechanism are discussed separately in Section 3.9.

2.1 Functional Requirements

2.1.1 Proof of Eligibility

Any slot leader – and in particular stake pool operators, who are elected through stake that is delegated to them – should be able to prove when they are eligible to produce a block in a given slot.

2.1.2 Visibility of Delegation on the Blockchain

We enable stake pools to automatically share their rewards with the delegators. In order to do this, there must be evidence for the delegation happening. Furthermore, we want the sharing of rewards to be enforced by the protocol, so the evidence must be recorded on the blockchain.

2.1.3 Restricting Chain Delegation

We do not want to allow stake to be re-delegated along a chain arbitrarily. We can admit some level of indirection, but not more than necessary to meet the rest of the requirements.

One reason that we do not want arbitrary chain delegation is that it makes it harder for delegators to figure out who is ultimately controlling their stake. Another is that unlimited chain delegation could open up a Denial-of-Service (DoS) attack vector on the system, where the attacker posts long delegation chains in order to slow down processes that depend on delegation, such as leader election or rewards sharing.

We must also have a mechanism to prevent cycles (such as A delegates to B, and B delegates to A) which would introduce ambiguity to the question of who manages stake in the end.

2.1.4 Cheap Re-Delegation

Changing delegation preferences should be as cheap as possible (while still using appropriate fees to prevent a denial of service attack on the blockchain).

2.1.5 Neutral Addresses

We should provide addresses that can hold value, but do not contribute to the PoS protocol. Those might be appropriate for use by exchanges, which will hold large amounts of value, without legally owning it.

2.1.6 Multi-Signature Addresses

We should provide addresses that can hold value that are owned by multiple people, such that the signatures from a specified subset are required to spend from those addresses. This needs to enable addresses where signatures are required from any N of a pre-determined set of M keys.

2.1.7 Multi-Signature Delegation

We should provide the ability to declare that delegating the stake rights for certain funds should require multiple signatures.

This should be as expressive as the multi-signature support for addresses. It should be an independent choice: a multi-signature address can use a single signature for its stake rights, or a different choice of multi-signature threshold N and key set M. Similarly, it should be possible to require multi-signature only for delegation, not for spending from a given address.

2.2 Security Requirements

2.2.1 Sybil Attack Protection at Stake Pool Level

It is conceivable that an adversary might try to take over the network by registering a large number of stake pools, hoping to accumulate enough stake to mount an attack just by people randomly delegating to them.

This Sybil attack on the level of stake pools should be made infeasible, by requiring stake pool operators to allocate a finite resource to each individual pool they register. In particular, this resource cannot be the cost of operating a node, since it is possible to run multiple pools with one node, so that cost would be constant in the number of pools an adversary is registering.

2.2.2 Address Non-malleability

The system should provide protection against the following attack:

Changing Delegation through Address Malleability Suppose that Alice makes a payment to Bob. In preparation, Bob transmits an address belonging to his wallet to Alice, and expects Alice to pay to that address. If his wallets later on shows that his balance is increased by the expected amount, he considers that transaction to be successful. An attacker that wants to increase their influence on the PoS protocol changes the address that Bob sends in such a way that funds in that address are delegated to the attacker, but the funds still show up in Bob's wallet.

The attack is considered successful if the stake rights for the transferred money belong to the attacker after the transaction, without Alice and Bob noticing the attack.

Note that the system should still allow for deliberately separating spending rights and the right to delegate, just not in the covert way described above.

2.2.3 Public Payment Keys Should not be Disclosed Prematurely

Delegation of stake should not involve revealing the public payment key (other than the public key hash, which is already visible from the address itself). The public payment key should only be revealed once the funds that are controlled by the corresponding private key are actually transferred to another address.

2.2.4 Mitigate Key Exposure

A node run by a stake pool will need to have some key that controls all the delegated stake, in order to sign blocks. In case of an incident where the node is compromised, it should be possible for the stake pool operator to revoke the key, and replace it with a new one. This should not require any action by the delegators.

2.2.5 Handle Inactive Stake Pools

We anticipate that some participants might not contribute to the proof-of-stake protocol – whether they lost their keys, lost interest, etc. We want to minimise the effect of this to the security and liveness of the system.

Note that this does not only concern large stakeholders or pool operators. The cumulative effect of a large number of small stakeholders having their stake be inactive also has to be considered.

2.2.6 Avoid Hard Transition

When we make the switch from Byron (where all stake is delegated to the nodes controlled by the Cardano Foundation, Emurgo, and IOHK) to Shelley (where ada holders have the freedom to control their stake), we should avoid a scenario where a significant amount of stake is suddenly offline.

This could happen if we automatically revoked the automatic delegation to the core nodes of the Byron network.

2.2.7 Change Delegation Without Payment Key

Users of a cold wallet, such as a paper wallet or a hardware wallet, should be able to delegate the stake corresponding to the funds in the cold wallet without using its payment key.

2.3 Non-functional Requirements

2.3.1 Asymptotic space and time complexity

All the changes to delegation are changes in the rules that define what it means to be a valid Cardano blockchain. These rules must be computable, and must be computable with reasonable space and time complexity.

2.3.2 Minimise economic attacks

An economic attack on a system arises where the costs incurred by the operators of a system are not covered by fees on the users of the system. Such situations allow users to impose costs on operators without paying that full cost themselves. In severe cases this can lead to operators dropping out and the system collapsing.

Cardano currently has transaction fees which are intended to cover the processing and long term storage cost of transactions. There are no fees however for the memory cost of tracking the current accumulated chain state, in particular the UTxO. In addition, the new mechanisms introduced for delegation add additional state that must be tracked. Moving from federated operation to fully decentralised operation may increase the incentive to exploit economic attacks, so it is important to address the existing unaccounted operator costs as well as new costs.

2.4 Requirements to Preserve Existing Features

2.4.1 Master Recovery Key

The whole wallet should be recoverable from one single key (without any additional information, such as the delegation preferences of the wallet).

The computational complexity of the recovery process should not be worse than logarithmic in the number of addresses appearing on the blockchain, and linear in the number of addresses in the wallet.

2.4.2 Address Recognition

An HD wallet should be able to recognise its addresses in the UTxO, so that it can report balances and transaction histories to the user.

2.4.3 Wallet should be Runnable on Independent Devices

Different user interfaces, running on different devices, should be able to access and control the same wallet, without transferring state between them.

We will accept some degradation of behaviour when running the wallet on different devices:

- Both copies might generate the same fresh addresses
- There can be differences in the reported balance while there are transactions in flight that only one of the two copies has knowledge of. In particular, when one copy sends a transaction, that transaction will only affect the balance reported by the other wallet once it is recorded on the blockchain.
- If the wallets use different delegation preferences, funds sent to the wallet might end up being delegated to different pools.

2.4.4 Maintain Privacy

HD Wallets maintain some level of privacy by using multiple addresses that are not obviously and publicly tied to the same wallet. Delegating stake should not necessarily link the addresses in the wallet of a delegator.

2.4.5 Short Addresses

It is beneficient to have short addresses, for two reasons: addresses are user-facing, and overly long addresses are burdensome for users. Also, every UTxO entry contains an address, so short addresses reduce the memory footprint of the UTxO and the whole ledger state.

Adding delegation to the system should not increase the length of addresses more than necessary. Ideally, we should use the opportunity of having to modify the address scheme to come up with an address length that is even shorter than in Byron.

2.4.6 No lookup of old blocks

The current Cardano design allows, in principle, an implementation of a node that discards blocks after a period of time so that it only needs to keep a limited number of recent blocks. This is true in part because nothing in the existing validation rules requires looking up arbitrary old blocks. All information necessary for validation can be accumulated in a running state, in a `foldl` style. This is a useful design property to retain.

2.5 Design Goals

2.5.1 No Special Wallet for Stake Pool Operators

If possible, we would like to avoid a situation where stake pool operators are required to use a special kind of wallet. Apart from registering their pool and running their own nodes, they should be able to use the same wallet as anyone else, without any additional or restricted features.

We expect that following this design goal will lead to less engineering effort, better maintainability, and a better user experience for stake pool operators.

3 Design of Delegation

3.1 Overview of Delegation

Delegation is a separation of the control over the movements of funds and the rights (and obligations) in the PoS protocol that are associated with those funds. We achieve this separation by modelling it in the address structure. We distinguish between *payment addresses* that determine how funds can be spent, and *stake addresses* that define if and how the stake rights of those funds take part in the PoS protocol. Coins belong to payment addresses. Each payment address (optionally) refers to a stake address. This delegates the stake rights of any funds held at the payment address to the corresponding stake address. The stake address delegates to a stake pool that participates directly in the PoS protocol. Thus overall there are two steps to the delegation of stake rights: a payment address refers to a stake address; and the stake address delegates to a stake pool.

We support multi signature (multi-sig) schemes, for payments as well as for delegations. We do this by allowing value addresses and stake addresses to use either keypairs, or scripts for authorisation, and implementing a simple scripting language to describe multi-sig schemes. Introducing multi-sig in this way has the benefit of naturally generalising when we will later introduce more powerful scripting languages, such as Plutus and Marlowe.

Participating in the PoS protocol requires two steps:

Using a registered stake address Users can post certificates to the chain to register a stake address. This will allow them to delegate funds associated with that address, and also automatically set up a corresponding *reward account*, where the system will accumulate rewards for delegating funds from that stake address.

Delegating from that stake address to a registered *stake pool* All blocks in Cardano-Shelley will be produced by a set of stake pools that need to be registered on the chain, by posting an appropriate certificate. Individual stakeholders can *delegate* funds from each of their registered stake addresses to a pool of their choosing. Stake in an address that delegates to a pool counts to the stake of that pool in the leader election. The pool will be rewarded for block production, and those rewards will automatically be distributed to the apropriate reward addresses.

Note that this does not restrict an individual stakeholder wanting to use their own stake to produce blocks ("self delegation"). Such users should register a *private stake pool*, and delegate their own funds to that pool. This uniform architecture, not distinguishing between those stakeholders that are using their stake directly and those that are delegating, reduces the overall complexity of the system significantly.

Registration of stake addresses and delegation are optional, but it is the only way to exercise the stake rights to take part in the PoS protocol and to earn rewards.

Rewards to pools and their members follow the scheme described in Section 5. In designing the rewards system, we were careful to avoid incentivising selfish behaviour, and to encourage cooperative behaviour instead. The rewards that a pool will get for producing blocks depend on how much stake they control, and on how well they perform when producing blocks. Stake pool operators can influence how the rewards for the pool are split amongst its members, by setting parameters in the stake pool registration certificate. Wallets will assist users in making rational choices and delegating to pools that are expected to give the best rewards.

3.2 Addresses and Credentials

In Shelley, an address has to provide information on two things: how tokens can be spent, and how the associated stake is controlled. To separate those two concerns, we distinguish between *payment addresses* $\mathcal{A}_p$ and *stake addresses* $\mathcal{A}_s$.

Addresses are objects that have a user-facing binary representation (they appear in the UTxO, and users can inspect them using a wallet or explorer). They contain *credentials* that govern access rights; using an address (such as spending from a payment address, or delegating funds associated with a stake address) requires a witness for the credential (which is specific to the particular transaction). There are two different kinds of credentials:

Key Credential A credential can be constructed from a pair (sk, vk) of a *signing key* sk and corresponding *verification key* vk. The credential is a cryptographic hash $\mathcal{H}(vk)$ of the verification key.

A witness for a key credential consists of the verification key vk, and a signature of the transaction from the signing key sk.

Script Credential Tokens and stake can also be controlled by a *validator script*, which can either succeed or fail to validate on a given input. In this case, the credential is the hash of the script.

A witness for a script credential is the script itself, as well as input to the script that makes it validate.

In future releases, we will add multiple languages to Cardano, with Plutus being the most prominent example. In Shelley, script credentials will only be used for the purpose of requiring signatures from multiple parties, a process known as multi signature, or *multi-sig*. For this, Cardano-Shelley will feature a minimalistic scripting language capable of expressing the requirement of having a specified subset of a given set of keys provide a signature. Examples include *M of N* schemes, where a transaction can be authorised if at least M distinct keys, from a set of N keys, sign the transaction. By introducing multi-sig script credentials, in Shelley, it will be possible to require single or multiple signatures both for the spending of funds and for the delegation of stake, independently.

For the case of multi-sig scripts, a witness contains the validator script matching the hash in the script credential, and a set of witnesses for individual key credentials. The validator script will determine whether those witnesses are sufficient for the funds to be spent. For details, and an example of a multi signature scripting language, see [CG19].

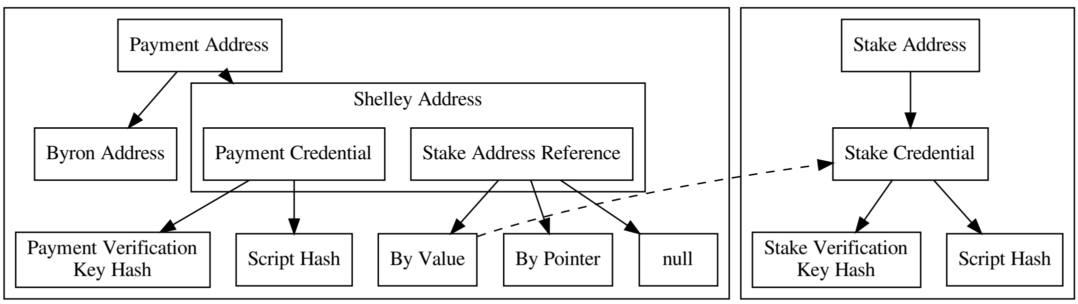

Figure 2: Addresses and Credentials

Figure 2 shows the anatomy of both payment and stake addresses. Let us first consider a *Stake Address*: it contains a stake credential (which can be a key credential or script credential). A stake address is used for two purposes: delegating stake (see Section 3.3.6), and spending rewards accumulated in a *reward account* (see Section 3.5.2).

Payment Addresses have more structure: first of all, we have the two cases of Byron and Shelley addresses. The purpose of Byron addresses is backwards compatibility. Byron addresses have no notion of stake, so it is not possible to delegate from a Byron address; users will have to transfer funds to a Shelley address first.

A Shelley address contains a payment credential (again, either a key or script credential). A transaction that consumes a UTxO entry with a Shelley address will need a witness for its payment credential in order to be validated. In addition, a Shelley address also contains a stake address reference. When calculating the stake distribution (see Section 3.4 for details), the system uses this reference to decide where to count the stake corresponding to the tokens in this address.

There are three options for the stake address reference in a Shelley address: it can be provided *by value*, i.e., just be the hash of a verification key or validator script. Shelley addresses that provide their stake address reference by value are sometimes also called *base addresses*.

There is also a more compact way of representing a stake address reference: since stake addresses need to be registered on the chain in order to be considered for the stake distribution (see Section 3.3.3), we can also include them *by pointer*, pointing to the certificate that registered it. Since the blockchain orders transactions, this pointer is quite small, containing only three numbers (slot index, transaction index within the block, and certificate index within the transaction). Shelley addresses that contain their stake by pointer are also called *pointer addresses*.

The third possibility is to not have a stake address reference at all. In this case, the stake corresponding to the tokens in the address is not considered by the system at all (just as for a Byron address). In particular, it will not be counted towards the active stake (Section 3.4.3), and so will not slow down chain growth. Users who do not wish to contribute to the PoS protocol should use this option. Such addresses are also called *enterprise addresses*.

3.2.1 On Pointer Addresses

Allowing the stake address reference to be included in a payment address via pointer allows for shorter addresses, which is a requirement (Section 2.4.5). However, there are also some subtleties to consider.

3.2.1.1 Invalid Refereces

First, we need to consider the case that the pointer does not point to an *active* stake address registration. This covers the case that the key was unregistered after (or indeed before) the transaction, and also covers pointers to targets that are plainly invalid. The system will allow transactions to and from such addresses, but their stake will not be considered for leader election and rewards.

Note that in particular, when a pointer address becomes invalid because the stake address it points to is deregistered, registering the same stake address again does not "restore" the stake in the pointer address; the tokens have to be moved to another address in order to use the stake. This minor limitation allows the system to not remember unregistered stake addresses.

Maybe add a diagram here, about the lifetime of pointer addresses.

So, to exercise the stake rights of a pointer address, the stake address must be registered in advance of using the pointer address in the output of a transaction, and the stake address must remain registered while the pointer address holds funds. This is a difference compared to addresses that contain their stake address reference by value, where the stake address can be registered after the value address is used in a transaction.

3.2.1.2 Pointer Addresses and Rollback

A special case of an invalid pointer is a rollback: when the block containing a stake address registration certificate gets rolled back, addresses containing the stake address by pointer to that certificate will lose their stake rights. Since the addresses will remain valid for payments, though, the stake rights can be restored by moving the funds to another address. Wallets can try to avoid this situation, as described in Section 4.4.

3.2.2 On Enterprise Addresses

Enterprise addresses carry no stake rights whatsoever and thus using them allows completely opting out of participation in the proof of stake protocol. Exchanges or other organisations that control large amounts of ada – but hold it on behalf of other users – may wish to follow a policy of not exercising stake rights. By using enterprise addresses, exchanges can demonstrate that they follow this policy. Since enterprise addresses are not associated with any stake address, they are automatically excluded from the mechanisms that influence the slot leadership schedule.

Note that using addresses with no stake rights effectively decreases the total amount of stake, which plays into the hands of the adversary. But unless we want the exchange to control the stake, it is unavoidable to ignore it, since there is no way to determine whom the stake ″really″ belongs to. Also note that it is generally considered bad practice to leave funds on exchanges, and in Cardano-Shelley, there will also be a monetary incentive to withdraw funds from exchanges in order to earn rewards.

3.2.3 Reward Accounts

Reward accounts are used to distribute rewards for participating in the PoS protocol, as described in 3.9.1. They have a number of interesting properties:

- They use account-style accounting, not UTxO-style.
- They can not receive funds via transactions. Instead, their balance is only increased when rewards are distributed.
- A reward account is not related to a value address, but to a stake address. Thus, spending from a reward account requires a witness for a stake credential, rather than a payment credential.

Rewards can be "withdrawn" from a reward account, by using the reward account as an input to a transaction. Note that we will still require at least one UTxO style input in this transactions for replay protection, as explained in Section 3.3.2. Stake associated with funds in a reward account will contribute to the stake of the stake address, so there is no incentive to frequently withdraw rewards.

3.2.4 On Byron Addresses

In Byron, all addresses were interpreted as having stake rights, but those stake rights were always delegated to a fixed set of keys specified in the genesis block, controlled by the Cardano Foundation, Emurgo, and IOHK.

Byron addresses continue to exist in Shelley, but their interpretation is changed subtly and their use is disincentivised. Their interpretation is changed from having stake rights with forced delegation, to having no stake rights whatsoever. Their use is disincentivised, since owners have the option to move their funds into the new base or pointer addresses that have stake rights, which can be exercised to receive rewards.

It is worth noting that initially, Byron addresses and enterprise addresses have essentially identical behaviour. This might change in the future, if new features are added to enterprise addresses.

3.2.5 HD Wallet Structure in Shelley

Shelley addresses with a verification key hash as payment credential support hierarchical deterministic wallets, as per BIP-32 [Wui12]. For value addresses with a multi-sig script credential, we can use a slight generalisation of BIP-45 [ACG14].

Lay out the necessary generalisation of BIP-45

In particular, this kind of wallet scheme allows implementations that can do wallet restoration from seed in time that is logarithmic in the total number of addresses on the blockchain. For details, see Section 3.13.

3.2.6 Address Recognition

Wallets will recognise addresses (other than reward addresses) that belong to them just as they would without delegation, by looking only at the payment credential (see Section 3.13 for how to find addresses efficiently).

Finding the stake credentials of a wallet (and in particular the corresponding reward accounts) can be done either by just reading off all the stake credentials of addresses found via their payment credentials, or by performing a second search, this time over all the registered stake addresses[1]. The former option is quicker and easier to implement. However, it is conceivable to construct addresses where the payment credential belongs to one wallet, but the stake credential belongs to another. The stake credential of such addresses would only be found by the wallet that controls it via a dedicated search on the registered stake addresses.

Once a wallet recognises an address via its payment credential, it will read its stake credential, and check whether it is set according to the current delegation preference of the wallet. If there is a discrepancy, it will alert the user, and ask them whether they want to re-delegate according to their current delegation preferences.

This check protects against the malleability attack in Section 2.2.2. It does so not by making it impossible, but by ensuring that the users are aware of it. This design also covers the case of users simply changing their delegation choice but subsequently receiving payments to addresses they handed out previously that use the previous delegation choice.

3.3 Certificates and Registrations

3.3.1 Certificates on the Blockchain

The registering of stake addresses and stake pools, and delegating, involves posting appropriate signed registration and delegation certificates to the blockchain as part of the set of certificates included in transactions. This makes the certificates part of the blockchain, which means that they are are publicly announced to all participants.

Certificates will remain valid until explicitly overwritten or revoked, as an automatic expiry would likely increase the amount of undelegated, offline stake. The following certificates can be posted to the blockchain:

- Stake address registration certificate
- Stake address de-registration certificate
- Delegation certificate
- Stake pool registration certificate
- Stake pool retirement certificate

There is one form of certificate which is not posted to the blockchain in advance, but is presented when it is used:

[1] As explained in Section 3.3.3, stake addresses need to be registered on-chain.

- Operational key certificate

Although this last kind is similar to delegation certificates in that it uses one key to grant the right to sign blocks to another key, it is quite different from the other certificates which are used to define the delegation relation. Operational key certificates are used by stake pool operators as a safety measure to mitigate key theft (see Section 3.3.7), not to delegate stake rights between different entities.

Figure 3 shows the relationships between the different types of certificates and keys. Dashed arrows represent relationships between the different keys and addresses: the owner(s) of a stake address delegates their stake rights to the owner of a stake pool cold key, using a delegation certificate. The owner of a cold and hot key grants the stake rights of their cold key to their hot key, using an operational key certificate. Incoming solid arrows represent the components of a certificate. For instance, a delegation certificate contains the stake address of the delegator, and the (hash of the verifying part of the) stake pool key.

Stake pools use a number of keys: a stake pool is controlled using the *pool key* pair $(sk_{\text{pool}}, vk_{\text{pool}})$. As explained in Section 3.3.7, this should be a *cold* key, kept on a secure machine, and only used to issue stake pool registration and operational key certificates.

Producing a valid block will require an operational key certificate, signed by sk_{pool}, and a witness for the block from the operational key specified in the certificate. Since operational keys use key evolving signatures (KES), an operational key is also referred to as a KES key.

In addition, the leader election process in Ouroboros Praos requires a verifiable random function, or VRF key pair $(sk_{\text{VRF}}, vk_{\text{VRF}})$, which is needed to prove that a pool has won its private lottery for a given slot. The hash of the verification part $\mathcal{H}(vk_{\text{VRF}})$ of this key is contained in the pool registration certificate. Subsequent sections provide more details.

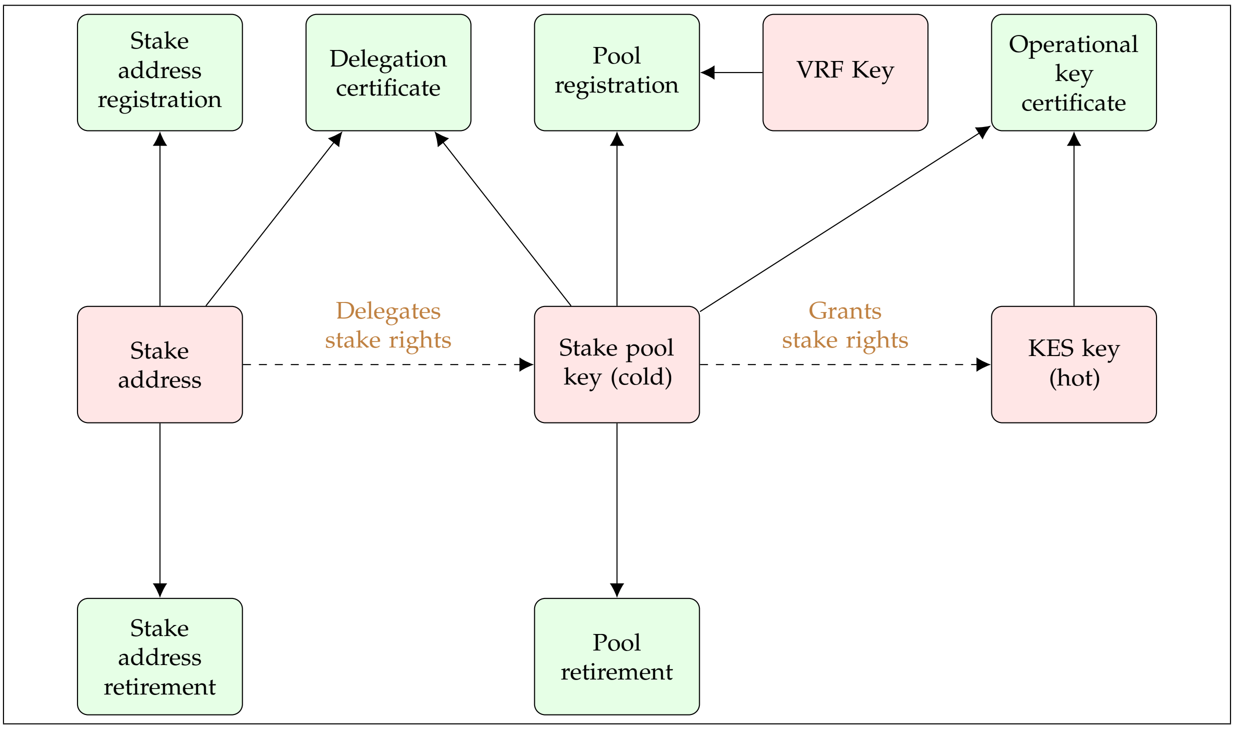

Figure 3: Relationships between the keys, addresses, and certificates

3.3.2 Certificate Replay Prevention

Unlike transactions, which inherently cannot be replayed due to the nature of the UTxO accounting model, the certificates need an explicit mechanism to prevent replays. The danger otherwise would include examples such as when a user switches away from a stake pool by posting a new delegation certificate, the old stake pool reposts the original delegation certificate, effectively thwarting the user's attempt to delegate to another stake pool.

The solution we employ is to borrow an idea from UTxO witnesses and to piggy-back on the inherent replay protection of the rest of the transaction. A UTxO witness for an input is a signature on the entire transaction body, which includes all inputs and outputs (but not witnesses, which would be circular). This means the signature can only be reused on the same transaction, and due to the nature of UTxO accounting, the same transaction cannot be included in the ledger again. For certificates, we do essentially the same thing: the witness for a certificate is a signature (or script input) for not just the certificate, but for the entire transaction body. This means that, provided the transaction spends at least one input, we inherit the inherent replay protection of UTxO accounting. The validity rules for transactions will explicitly require each transaction to have at least one UTxO entry as an input, so that certificates will be protected from replay in this way.

Why do we need to explicitly require a UTxO input? Won't the need to pay transaction fees always implicitly require this anyway? There are two additional sources of funds that could pay for the transaction fees: refunds and the contents of a reward account. So it is possible to create transactions that contain enough value to pay transaction fees even without consuming a UTxO entry. Such a transaction, if valid, could be replayed at a later point in time. In order to have all transactions (and by extension, all certificates), be protected from replay attacks, we will explicitly require at least one UTxO entry as an input to any transaction.

3.3.3 Stake Address Registration Certificates

Users wishing to exercise their rights of participation in the PoS protocol can register a stake address by posting a *stake address registration certificate* to the blockchain.

Stake address registration certificate This certificate contains a stake address. The credential can either be a key credential, or script credential, as explained in Section 3.2.

We do not require a witness to register a stake address (besides, of course, any witnesses needed for the transaction that is used to post the certificate).

Stake address de-registration certificate This certificate contains the stake address that should be de-registered.

The certificate requires a witness for the stake address that should be de-registered. As stated in Section 3.2, this will either be a key or script witness, depending on the type of credential of the address.

Registering a stake address introduces a corresponding reward account. The account is deleted when the stake address is de-registered. See Appendix A.3.3 for details on reward accounts.

In addition to a transaction fee, registering a stake address requires a deposit, as explained in Section 3.10.2 and Appendix B. The deposit is to account for the costs of tracking the stake address and maintaining the corresponding reward account. It also incentivises de-registering stake addresses that are no longer required, so that the corresponding resources can be released.

3.3.4 Stake Pool Registration Certificates

A person planning to operate a stake pool (including a private pool) can declare their intent by posting a *stake pool registration certificate* to the blockchain.

Stake pool Registration Certificate The certificate contains the following:

- The hash of the verification part of the (cold) pool key, $\mathcal{H}(vk_{\text{pool}})$.

- The hash of the verification part of the VRF key, $\mathcal{H}(vk_{\text{VRF}})$.
- A stake address $\mathcal{A}_{s,\text{reward}}$, called the *reward address* for the pool. Usually, this will be a *registered* stake address, controlled by the pool operator. The rewards for the pool operator will be paid to the reward account of $\mathcal{A}_{s,\text{reward}}$[2].

  If a pool operator wants to donate their rewards to a charity, they can do so by using a stake address that is controlled by that charity as the reward address. They can then advertise to other stakeholders that they are doing so (although evidence that the stake address does indeed belong to the charity has to be provided out of band).

  Should the reward address be unregistered, the stake pool operator will be unable to receive rewards. In that case, any rewards that they would be due are instead sent back to the reserves (but the stake pool members would still get their usual rewards).
- A list of stake addresses controlled by the pool owner(s), $\mathcal{A}_{s,\text{owner}}$. If any of these *owner stake addresses* delegate to this pool, the stake that they delegate will be counted towards the stake pledged to the pool by the owner(s), see Sections 4.1, 5.1 and 5.5. Note that adding a stake address to the set of owner stake addresses in itself does not actually delegate the stake controlled by that address to the pool – this requires posting an ordinary delegation certificate to the chain.

  During reward distribution, there will be no rewards paid to the reward accounts of the owner stake addresses. Instead, the stake delegated by all owner stake addresses will be counted as the stake contributed by the pool owner(s), and their reward will be paid to the reward account of the reward address.
- The parameters that specify the reward sharing function of the stake pool: cost, margin, and amount of stake pledged to the pool by the owner(s), see Sections 4.1 and 5.3.
- A list of IP addresses and/or DNS Names of public relay nodes that the stake pool operator provides to support the Cardano network.
- Optionally, a URL and content hash for additional metadata about the pool, for display in the wallet. The URL is restricted to a length of 64 bytes. It is the obligation of the stake pool operator that this URL points to a JSON object containing the metadata of the pool, as described in Section 4.2. The content hash of that JSON object should match the content hash in the registration certificate.

  If no URL and content hash is provided, the stake pool will not be listed in wallets. Private pools (Section 4.6) will use this option.

  Also, if there is a mismatch in the content hash, the pool will not be displayed. If a stake pool operator changes the metadata, they have to post a new stake pool registration certificate with the new content hash.

Validating the certificate requires witnesses from all owner stake addresses, as well as a witness for the pool key.

If a stake pool can foresee that it will cease to operate, it can announce this intent by posting a *stake pool retirement certificate*.

Stake pool Retirement Certificate It contains

- The public key hash $\mathcal{H}(vk_{\text{pool}})$ of the pool.

[2] We do *not* want to associate the reward account with the key of the pool itself. An important reason for this is that for security reasons, stake pool operators are required to use operational keys as described in Section 3.3.7, while storing the key of the pool securely offline. Requiring the pool key for withdrawing rewards would be detrimental to this.

- The epoch number, starting from which the stake pool will cease to operate.

It requires a witness for the pool key sk_{pool} (particularly, it need not be signed by any of the pool owners).

After the retirement epoch, any stake that is delegated to this stake pool will be disregarded for the PoS protocol. It will not take part in the leader election process (similarly to how stake in an enterprise address is not considered during the election process).

Stakeholders who delegated to this pool should be notified and asked to redelegate by their wallet the next time they are online.

3.3.5 Single Operator, Possibly Multiple Owners

Note that there is a conceptual difference between the stake pool *operator* and stake pool *owners*:

Stake Pool Operator This is the person who operates the pool – owns or rents a server, holds the key of the pool, runs and monitors the node.

Stake Pool Owner This is a person pledging stake to the pool, increasing its rewards and desirability. The ability for the owner to pledge stake is providing protection against the pool-level Sybil attack (requirement 2.2.1, see also Section 4.1).

Usually, the stake pool operator and owner will be the same person, but a stake pool can also have multiple owners. This is to allow people to coordinate and form a stake pool even if none of them had enough stake on their own to make a pledge that would make the stake pool competitive.

Still, there will only be one operator, the person holding the key of the stake pool itself. In addition to signing blocks, this key also holds the power to retire the stake pool, or to post updated registration certificates without the keys of all owners (in which case some of the owners are kicked off the stake pool). Also, the rewards for all owners will be paid to the reward account associated with the reward address of the pool, and it will be the responsibility of the person(s) owning that address (usually the pool operator) to distribute the rewards amongst the owners. That is a conscious design decision: collaborating to form a stake pool should require significant trust between the owners. Otherwise, everyone could choose to become a co-owner of a stake pool instead of delegating, which would render the mechanism of pledging stake ineffective. Allowing the operator to shut down the pool, kick other owners off, or fail to distribute rewards raises the threshold of necessary trust that owners of the pool must have in the operator.

3.3.6 Delegation Certificates

Users can transfer the rights of participation in the PoS protocol from their stake address to a stake pool, by posting a *delegation certificate* to the blockchain.

Delegation Certificate A delegation certificate is a tuple containing

- the stake address delegating its stake rights, $\mathcal{A}_{s,\text{source}}$
- the stake pool verification key hash to which stake is delegated, $\mathcal{H}(vk_{\text{pool}})$

Posting a delegation certificate requires a witness for the delegating stake address $\mathcal{A}_{s,\text{source}}$.

Note that there is no corresponding delegation revocation certificate. If a user wishes to change their delegation choice to a different stake pool (which might be their own private stake pool), they can simply post a new delegation certificate. Also, the delegation certificate is revoked automatically when the source stake address is de-registered.

3.3.7 Operational Key Certificates

Stake pool operators must use a *hot/cold* key arrangement to mitigate key exposure (see Section 2.2.4). A *hot*, or operational, key is kept online, and is used to sign blocks, while the *cold* key is kept securely offline. This requires an *operational key certificate* to create a (1-link) chain of trust from the cold key to the hot key, allowing this hot key to participate in the *PoS* protocol. Should the hot key become compromised, the stake pool operator should immediately create a new operational key certificate, and switch to a new key.

If the operational key certificates would be included in the ledger, as the other certificates are, this would present a problem for block validation. Consider the following example: suppose that the network layer wants to validate a batch of 10 block headers from the current epoch, before deciding if it wants to also download the block bodies. Can it tell if those block headers are signed by the right actors?

In this situation the network layer has the state of the ledger up to but not including these next 10 block headers. So it cannot rely on any information in the 10 corresponding block bodies.

Without this information, the validity of the hot key that signed the headers cannot be verified. If the network layer sees one of the 10 new blocks signed by a hot key it doesn't recognise, it might be because there is a delegation certificate in the block bodies (which the network layer has not seen yet), that shows that the key is valid because some stake pool key deferred its stake rights to it. Similarly, if the network layer sees a known hot key, how can it know that it is still valid? There could be a new certificate in the block bodies that defers the rights of this key to a different one (which would invalidate the key the network layer saw).

To address this problem, a new type of certificate is introduced: *operational key certificates*. These certificates are provided in the block header itself, as part of the witness, thus solving the problem of determining the precedence of hot keys. In other words, operational key certificates are **not included in the ledger**, but instead they are **included in a witness** by stake pools at the point of exercising stake rights including:

- signing blocks
- signing votes for protocol update proposals (once the update and voting system is in place)

An operational key certificate, signed by the stake pool's cold key, delegates to the hot key that is used to sign messages in the protocols (block header or vote). This operational key certificate is included in the message so that all other nodes can verify that the message is signed by a legitimate delegate of the owner of the cold key[3].

Specifically, an operational key certificate specifies that the stake rights are transferred from a cold stake pool key vk_{pool} to a hot key vk_{hot}. They are included in the message (e.g. block header) and the message itself is signed with sk_{hot}.

Operational keys will use key evolving signatures (KES). To be precise, we will use keys according to the MMM scheme [MMM01], regular evolutions after a number of slots that correspond to one day, and a key lifetime of $2^7 = 128$ days, a little over three months.

Operational key certificates will have a lifetime of 90 days after which they become invalid, to encourage pool operators to regularly rotate their operational key. The certificate will specify a slot from which it will be considered to be valid for 90 days.

In detail, the hot/cold key setup is as follows:

- The stake pool operator registers their stake pool, using a cold stake pool key vk_{pool}. This *cold key* is kept securely and off-line.

[3]This is much the same setup as with TLS certificates: there are known root certificates, but the server's operational certificate is presented inband.

- The stake pool operator uses sk_{pool} to sign an operational key certificate C, transferring the stake rights to a *hot key* vk_{hot}.
- The stake pool operator keeps sk_{hot}, as well as C, on a node that is on-line, and can sign blocks. A block signed with sk_{hot} will be considered valid, provided that C is included in its header.
- Should the node get hacked, and the hot key compromised, the stake pool operator will create a new operational key certificate C', delegating the stake rights to a new hot key $vk_{\text{hot}'}$.

In order to render sk_{hot} useless, it must be established that C' takes precedence over C. For this purpose, the operational key certificate will have an additional integer field, and certificates with a larger value for this field will take precedence.

3.3.8 Certificate Precedence and Validity

The following rules determine precedence and validity of certificates. In particular, they describe what happens when multiple certificates are issued for a given stake pool key.

The ordering of blocks and transactions induces a canonical ordering amongst certificates. Thus, the terms older/newer certificate are well defined and are used below.

3.3.8.1 Stake Pool Registration and Retirement Certificates

- There can be at most one active stake pool registration certificate for any given stake pool key. A newer certificate will override an older one.

  This will allow stake pool operators to update their costs and margin if they need to[4].
- A revocation certificate is only valid if there is an older registration certificate.

3.3.8.2 Delegation Certificates

Newer delegation certificates override older delegation certificates. This allows delegators to move from one stake pool to another.

3.3.8.3 Operational Key Certificates

For operational key certificates, we cannot rely on the ordering induced by the blockchain. But we do have the counter field, which serves the purpose of establishing precedence:

- An operational key certificate with a higher counter overrides one with a lower counter.
- Also, we require that within any given chain, if there are two blocks A and B signed using operational key certificates issued by the same cold key, if A is an older block than B, the counter in the operational certificate in the cert in the header of B must be at least as large as the one in the counter in the operational certificate in the header of A.

[4]Stake pool members should be notified of such changes by their wallet the next time they are online, if this makes the pool less desirable, see Section 4.3.

3.4 Delegation Relations

As stated in the delegation overview: delegating stake rights involves two indirections: from payment addresses to stake addresses, and from stake addresses to stake pools.

Equivalently, there are two relations: a relation between payment addresses and stake addresses, and a relation between stake addresses and stake pools. The first relation can be read off payment addresses, by looking at their stake address references. The second relation consists of registered stake addresses, registered stake pools and delegation certificates as the entries relating the two.

3.4.1 Address Delegation Relation

The address delegation relation is a relation between payment addresses and registered stake addresses.

This relation can be defined in terms of the current UTxO and the current set of registered stake addresses. For all Shelley addresses in the UTxO, the stake address is determined by the stake address reference – either directly by value, via a pointer to a stake address registration certificate, or as null. This needs to be filtered by the current set of registered stake addresses.

3.4.2 Stake Pool Delegation Relation

The stake pool delegation relation is a relation between registered stake addresses and stake pools.

The relation is defined by the active set of delegation certificates, filtered by the set of active stake pools. The active delegation certificates already exclude those where the source stake address has been de-registered (since a delegation certificate is revoked automatically when the stake address is de-registered).

3.4.3 Overall Stake Distribution

Ouroboros [KRDO17] requires a stake distribution to use as the basis of defining the slot leader schedule for the next epoch.

The overall stake distribution is the set of all registered stake pools and their aggregate stake from all addresses that are delegated to them.

This can be defined by taking the composition of the address delegation relation and the stake pool delegation relation, giving the relation between payment addresses and stake pools. The final distribution is formed by taking the transaction outputs from the UTxO and selecting all the payment addresses related to each stake pool and aggregating all the coins.

Note that defining the stake distribution in this way is in contrast to using the Follow the Satoshi algorithm. This definition automatically excludes all addresses that hold no stake, and excludes addresses that have stake rights, but which have not correctly registered their stake address or delegation certificate.

We will call stake that is correctly delegated to an existing pool *active stake*, sometimes contrasted to the *total stake*, which includes stake that is not delegated (or delegated to pools that are already retired).

3.4.4 Chain Delegation

Chain delegation is the notion of having multiple certificates chained together, so that the source key of one certificate is the delegate key of the previous one.

While the delegation research paper in principle allows a significant degree of flexibility with delegation, our chosen design is quite restrictive and uses a fixed pattern of delegation.

We will only allow a very simple form of chain delegation, where we have the following, in order:

1. a stake address
2. a delegation certificate; and
3. an operational key certificate.

This restricted pattern of chain delegation allows us to satisfy all requirements, but avoids problematic cycles in the graph of delegation certificates, and makes it simple for nodes to track the delegation.

3.5 State Tracking for delegation

It is not sufficient for certificates to be posted to the blockchain. Nodes need ready access to certain parts of previously posted information as part of the protocol execution or ledger validation. For example, since nodes need to validate signatures on new blocks in a timely manner, they need ready access to information about the registered stake pools (including operational key certificate validity).

One of the design goals is to avoid having to look up old entries on the blockchain, since we want to allow implementations that forget old blocks. Instead, we want a `foldl` design where nodes keep track – as local state – of all the information they will later need.

The following sections describe the local state that nodes must maintain as they process transactions in blocks.

3.5.1 Stake Addresses

The set of active stake addresses must be tracked. This contains the stake credentials from each stake address registration certificate. The set is uniquely indexed by the hash itself (of either the verification key, or script, depending on the type of credential). It is also uniquely indexed by the location on the blockchain of the stake address registration certificate, using the same location type as pointer addresses.

This set is updated when stake addresses are registered and de-registered. This state is consulted when validating and applying transactions that withdraw from reward accounts, to retrieve the stake credential.

3.5.2 Reward Accounts

For each stake address, there is an associated reward account. The lifetime of these accounts follows exactly those of their associated stake address.

The reward accounts are a mapping from a stake address to their current balance. The stake address is the unique index for the mapping.

The accounts are updated in bulk following the end of an epoch. They are consulted and updated when validating and applying transactions that withdraw from reward accounts. See Section 3.2.3 and Section 3.9.1 for details.

3.5.3 Stake Pools

The set of active stake pools must be tracked, uniquely indexed by the public key hash of the stake pool.

In addition, a small amount of state needs to be maintained to validate operational key certificates. The state tracked for each stake pool includes an integer representing the highest counter field seen so far in a valid certificate. This is consulted to validate operational key certificates, and updated when larger counter values are presented in a valid certificate.

3.5.4 Active Delegation Certificates

Active delegation certificates are tracked, as a finite map from stake address to pool verification key hash.

3.5.5 Stake per Stake Address

For the purpose of leader election and reward calculation, the system needs to know how much stake each registered and delegating stake address actually controls. The total stake of a stake address is calculated as the sum of all funds that are

- in value addresses referring to this stake address in their stake address reference (by value, or by pointer, as long as the pointer points to a stake address registration certificate that was already valid when the value address received funds, and has not been de-registered)
- in the reward account of that stake address

3.6 Slot Leader Schedule and Rewards Calculation

The process of leader election has to be modified to take delegation into account.

While adding blocks to their chain, nodes will keep track of the pieces of state listed above. When it is time to prepare the slot leader schedule for the upcoming epoch, they will look at a past snapshot of that state, from slot $s_{\text{stakedist}}$, the slot from which the stake distribution should be used to compute the slot leader schedule and rewards for the next epoch[5].

The nodes will use the state from slot $s_{\text{stakedist}}$ to

- compute the stake distribution (i.e., the amount of stake per stake pool)
- create the leader schedule by sampling the stake distribution (i.e., sampling the stake pools, weighted by the stake they control)
- retain this state, to use it for the reward calculation at the end of the epoch[6].

3.7 Block Validity and Operational Key Certificates

Stake pool operators will use operational key certificates in order to protect the key to which their members delegated. A block for a slot where the VRF key vk_{VRF} has been elected as leader (the proof of which is to be constructed using sk_{VRF}, and to be included in the block header) will be considered valid by all nodes if

- there is a stake pool with (cold) pool key vk_{leader}, which list $\mathcal{H}(vk_{\text{VRF}})$ as the VRF key hash in its pool registration certificate
- the block is signed by sk_{hot} and contains, in its header, an operational key certificate from sk_{leader} to sk_{hot}.
- The counter of the operational key certificate must not be smaller than the counter of the operational key certificate used to sign the last block of vk_{leader}.

[5] The detail of which slot is used as $s_{\text{stakedist}}$ depends on the variant of the Ouroboros protocol that is used. It needs to be deep enough in the chain to be stable. It also needs to be before the point in time at which the random seed for the slot leader election is determined, to prevent a grinding-type of attack. Note that $s_{\text{stakedist}}$ does not necessarily have to be in the current epoch.

[6] They cannot calculate those rewards immediately, because they depend on the efficiency of the pools during the following epoch. However, they could, alternatively to retaining the whole delegation state, calculate the rewards up to the factor that depends on the efficiency.

In case there are more than one block for the current slot, for the same pool, each of which are signed using an operational key certificate, the newest certificate (as per the included counter) takes precedence.

Note that nodes take the precedence amongst operational key certificates into account only *after* comparing the length of the chains. When the node is already up to date and receives two conflicting blocks that add to its current chain, the length will of course always be the same. But this rule is important: if we did not compare the lengths of the chains before giving preference to the block with the newer operational certificate, it would be possible to force a node to do a rollback of arbitrary length, by sending it a block from a past slot, signed using a newer certificate than the block that the node already has in its chain for that slot. This would open up an attack where a stake pool operator could force nodes to do arbitrary rollbacks.

3.8 Transition to Decentralization

In order to guarantee system stability, we must be sure that stake pool operators are "doing their job" sufficiently well before relinquishing control to them. Instead of having a simple "switch" from a centralized system controlled by a handful of bootstrap keys to a fully decentralized one, we propose a *transition phase*.

3.8.1 Motivation

Cardano *chain growth quality* is only guaranteed when for all time windows of $2k$ slots, a block has been created for at least k slots, where k is the security parameter of the protocol. At the moment, the bootstrap nodes are responsible for block creation, but in a fully decentralized system, this will be the pool operators' responsibility.

In the beginning, there might be technical problems or other issues preventing the pool leaders from creating sufficiently many blocks, so we want to make the transition gradual, monitoring system performance and being able to temporarily delay or even revert decentralization in case of an emergency.

Another consideration is the amount of stake that is necessary to mount a 51% attack on the system. Since participating in the PoS protocol requires an action on behalf of the stakeholders – registering a stake address and delegating – it is not unreasonable to expect that it will take some time until a significant fraction of the overall stake becomes active and starts contributing to the protocol. An attacker might use this window of opportunity to attack the system. A gradual handover of the protocol from the initial core nodes to the actual stakeholders will protect the integrity of the blockchain.

3.8.2 Proposal

We propose to introduce a new parameter $d \in [0, 1]$, which controls the ratio of slots created by the bootstrap keys – all other slots will follow the rules outlined in this specification. So $d = 1$ corresponds to the present "bootstrap era" state, whereas $d = 0$ corresponds to full decentralization as described in this document. Starting with $d = 1$ and gradually going down to $d = 0$ allows for a smooth transition period.

For a given value of d, the system will perform two steps to create the leader schedule for the next epoch:

- Perform leader election amongst the new nodes, according to Ouroboros Praos, for all n_s slots in the epoch.
- Randomly select dn_s slots of the epoch. Let us call those the slots the *OBFT slots* of that epoch. For the OBFT slots, the Praos leader schedule will be overridden, and the old core nodes will be responsible for creating blocks for theses slots.

In order to keep the block frequency constant, we will select a fraction f of the OBFT slots, where f is the active slots coefficient from Praos[7] (Equation (1) in [DGKR17]), and call those the *active OBFT slots* of the epoch.

For OBFT slots, we will modify the behaviour of all nodes as follows:

– No stake pool node shall create a block for an OBFT slot.

– For non-active OBFT slots, no node shall produce a block at all – neither one of the old core nodes, nor one a stake pool node.

– Also, for the non-active OBFT slots, no block shall be considered valid by any node.

– For the active OBFT slots, the old core nodes will create blocks, in a round-robin fashion as per OBFT.

– Blocks produced according to this schedule for the active OBFT slots shall be considered valid by all nodes.

3.8.3 Rewards during the Transition Phase

We do this soft transition as a de-risking strategy, so that we can intervene in case we observe any difficulties in the decentralised system. But we do not want this to have an effect on the rewards that pools get. Operational difficulties of the overall system that cause us to slow down the transition should not reduce the rewards that individual pools get.

In order to minimise the influence of the transition on pool rewards, we have to alter the way we measure the apparent performance for pools (see Section 5.5.2) during the transition phase:

1. For determining the apparent performance of any pool, we will take the total number of blocks in the epoch – $\overline{N}$ in Equation (1) – to be the number of blocks produced in *non-OBFT slots*.

2. As long as we have $d >= 0.8$, we set the apparent performance of any pool to 1.

The reason for Item 2 is that when only a small fraction of blocks are produced by stake pools, the measurement of the performance will be dominated by the statistical aspect of the leader election, and pools might frequently get a performance of 0 by no fault of their own.

As an example[8], consider a pool A with 1% of stake. In the fully decentralized case $d = 0$, A would be elected slot leader for $0.01 \cdot 21600 = 216$ slots per epoch on average. For $d = 0.9$, A would only be elected for $0.01 \cdot 0.1 \cdot 21600 = 21.6$ slots per epoch on average, so A would only have a tenth of the work (create 21.6 blocks instead of 216 blocks), but get the same rewards.

3.8.4 Transition Plan

The parameter d can be changed on an epoch-per-epoch basis, following the plan we will outline.

We plan to start with $d = 0.9$ and then decrease d by 0.1 each epoch, *provided pool leader block creation is sufficient to guarantee chain growth quality, and a sufficient fraction of active stake*.

If block creation is insufficient, we will halt lowering d (or even increase d again) until we have reason to believe that the problem has been understood and fixed.

In order to decide whether block creation is sufficient, we will estimate the probability that at least k out of every $2k$ blocks would be created. If this probability is high enough (for example greater than $1 - 10^{-10}$), block creation will be deemed sufficient.

[7] In Ouroboros Praos, a large fraction of slots will deliberately be empty, which makes it easier to treat network delays in the adversarial model, and to still give guarantees of liveness and persistence when some blocks are not propagated within a single slot. Note that it is still possible to achieve the same block frequency as in Ouroboros Classic, by choosing a shorter slot length.

[8] In this example, we assume $n_s * f = 21600$, to match the block frequency of Byron.

For the estimation, we use the Beta-Binomial Distribution: Given the number of slots a that have been faithfully created and the number b of slots that have been missed (counting from the beginning of the transition period) and using Bayes' Prior $B(1,1)$, the probability in question is $P(X \geq k)$, where X is drawn from the Beta-Binomial distribution with parameters $(a+1)$, $(b+1)$ and $2k$.

For example, in the very first transitional epoch, 10% of active slots, i.e. 2160 active slots[9] will be given to pool leaders. If at least 1261 out of these 2160 slots are properly created, above estimation (with $a \geq 1261$ and $b \leq 2160 - 1261 = 899$) leads to $P(X \geq 2160) \geq 1 - 10^{-10}$, so we will proceed with $d = 0.8$ in the second epoch. If however at least 900 slots are missed, we will keep d at 0.9 for the time being.

In addition to monitoring the number of missed blocks, we will also look at the fraction of stake that is active (i.e., is stored in addresses which belong to a registered stake address that is delegating to a stake pool). The lower this ratio, the less stake is required to launch a 51% attack on the system. This can be offset by increasing d. For example, if $d \geq 0.5$, it is impossible to launch a 51% attack. We can specify an amount of stake controlled by an adversary that we want the system to be resilient against, and delay reducing d in order to meet this level of resistance.

3.9 Rewards

For the smooth operation of the system, it is beneficial to have a large portion of the stake delegated to a set of reliable stake pools. Thus, we should incentivise delegating stake to reliable stake pools. One way to do this is to have stake pools share their rewards with their participants.

The reward sharing mechanism should satisfy the following requirements:

1. Sharing rewards should be an automatic process that does not require an action, neither by the stake pool operator nor the participants. This requirement is not only meant to ensure that participants get their share reliably. The share of the rewards that are given to a particular participant depends on the amount of stake that that participant delegated in a particular epoch. Thus, any node that verifies a transaction that transfers the rewards for a given epoch needs to access the stake information for that epoch. While this information is archived on the blockchain indefinitely, looking it up for arbitrary past epochs might be too costly. Making the sharing of rewards an automatic process in the following epoch circumvents this problem.

2. Sharing rewards should not lead to an excessive growth of the UTxO. In particular, it should avoid creating dust entries.

3. Sharing rewards should not lead to a burst of transactions that risks pushing the system to the limits of its predictable region of operation.

4. Sharing rewards should not increase the linkability of addresses of a wallet.

5. The reward sharing policy of the stake pool should be transparent to potential participants.

Coming up with a solution that satisfies all of those requirements is less straightforward than one might think. We did an exhaustive assessment of possible approaches, documented in Appendix A, and finally opted for the mechanism described in A.3.3, which compromises somewhat on Item 4, but satisfies all the other requirements.

---

[9] assuming $k = 2160$ and an epoch length of $10k$ active slots, as in Byron

3.9.1 Distributing Rewards

One of the difficult problems we had to solve during the design of the reward distribution mechanism was UTxO explosion and dust creation: since rewards occur in every epoch, and all the entries in the UTxO will generate rewards, a naive approach would lead to an exponential growth of the UTxO, which is clearly not sustainable. Furthermore, individual rewards would be small, so most of the UTxO entries created for reward distribution would be dust.

We have explored several approaches to circumvent this problem (see Appendix A for a summary), and ended up with using *reward accounts* (Section 3.2.3). Here, UTxO growth is prevented by using addresses that do not use UTxO style accounting at all. Instead, every registered stake address has an associated account, using account-style book-keeping. That way, the rewards from multiple epochs can be pooled, and stakeholders can withdraw them manually. Note that this has two advantages over the superficially simpler approach of having stakeholders claim their rewards directly:

- Updating the total rewards a stakeholder is due happens frequently, avoiding the need for all nodes to hold on to the state that is needed to calculate rewards from old epochs.
- Rewards that are accumulated in reward accounts can be delegated before they are withdrawn, eliminating an incentive for frequent withdrawals (which again would lead to an unnecessary growth of the UTxO set).

After the end of each epoch, rewards for stake pool operators and members are calculated, using the formulae in Section 5.6. The calculation will be based on

- The active stake pools, in particular their cost and margin parameters, pledged stake, owner key hashes, and reward accounts for stake pool owners.
- The finite map giving the total stake for each registered stake address, *taken at the point in time that was relevant for creating the leader schedule for that epoch*.
- The stake pool delegation relation.
- The leader schedule and list of empty slots for that epoch.

For each registered stake address, the rewards thus calculated are added to the balance of the associated reward account.

Note that all the information that is relevant for the calculation of the rewards is publicly available on the blockchain, so there is no need to explicitly write the balance of each reward account to the chain. Instead, it suffices for all the nodes to store the reward accounts and their current balance locally.

3.9.1.1 Collecting Rewards

Once a sizeable amount of funds has accumulated in a given reward account, the owner of that account will want to withdraw those funds, and move them to an ordinary address of their wallet. This withdrawal from an account to a UTxO can be done via a transaction, using the reward account and its current balance as an additional input. In order to validate, it needs a witness for the stake address associated with the reward account.

The transaction is protected against replay by the requirement of having at least one UTxO input, as described in Section 3.3.2.

3.9.1.2 Handling of Byron Addresses

All funds in Byron addresses will be ignored by the PoS system – there is no stake address associated with Byron addresses. Consequently, there will be no rewards, and stakeholders will be incentivised to stop using Byron addresses. Our transition plan, laid out in Section 3.8, prevents a situation where the system would be vulnerable to a 51% attack because only a small fraction of the total stake is active yet, by allowing for a period where the original nodes from the Byron phase are still eligible to sign some of the blocks.

3.10 Fees

To prevent economic attacks, fees or refundable deposits should be charged where operators incur costs. In particular we will have refundable deposits corresponding to the state that has to be tracked for certificates[10].

3.10.1 Transaction fees

The basic transaction fee covers the cost of processing and storage. The formula is

$a + bx$

With constants a and b, and x as the transaction size in bytes.

The fixed component is to cover per-transaction overheads. The component linear in the size of the transaction reflects the processing and storage cost of transactions.

This aspect remains unchanged with delegation except to the extent that there are additional objects that can appear in transactions relating to delegation. These simply increase the size of the transaction and so are covered by the existing fee formula.

In principle different fees could be charged for different things appearing in a transaction, to reflect their different processing costs. This is a future direction, but will not be introduced as part of delegation.

3.10.2 Deposits

In addition to ordinary (non-refundable) fees, actions that require resources on the nodes should require a deposit, as described in Appendix B. In particular,

- registering a stake address
- registering a new stake pool (but not updating the registration certificate of a stake pool that already exists)
- creating a new UTxO entry (in a future release)

should all require making a deposit. This deposit should be released when

- a stake address is de-registered
- a stake pool is retired – there is a subtlety here, however, since the retirement certificate states an epoch in the future where the pool will cease operation. The refund should depend on that epoch, and we will delay paying out the refund until that epoch.
- a UTxO entry is removed by using it as an input to a transaction (in a future release)

Note that posting a delegation certificate does *not* require a deposit; delegation certificates need a stake address registration certificate in order to be valid, so any deposit that we would require for a delegation certificate can instead be included in the deposit for the associated stake address registration certificate.

[10]We plan to extend this to also cover UTxO entries in the future.

3.11 Time to Live for Transactions

For multiple reasons, we will require that transactions that are submitted to the system include a *Time to Live* (ttl), a slot number[11] after which the transaction can not be included in the ledger any more. An obvious advantage is that this gives users the certainty that a transaction that failed to be added to the chain can not be replayed in the future, so that they are safe to re-send funds.

In the context of deposits and refunds, a ttl also proves to be useful: When a transaction that releases a resource is created and submitted, it is not known when it will be effective, and thus the refundable part of the deposit – which depends on the time at which the resource is freed – can not be computed. But if the transaction does include a ttl, the latest slot in which the transaction can be added to the ledger can be used to calculate the refund.

3.12 Robustness at the Epoch Boundary

As described, there is a lot of work to be done by the nodes as the system progresses from epoch to epoch: the stake distribution and slot leader schedule need to be calculated, the fees accumulated during the epoch need to be summed up, performance of the pools evaluated, deposits decayed, and rewards determined and distributed.

Doing all that *at* the transition between two epochs is far from ideal. It creates a time period where all the nodes will need additional resources. It also requires that all nodes finish those calculations within a single slot length, and failure to do so will result in missed blocks and temporary forks. Worse, since this period of increased activity is perfectly predictable, any attacker of the system can leverage this, and time their attack appropriately to maximise impact. Effectively, a predetermined breaking point is introduced at the epoch boundary.

It is thus desirable to spread this work out over a longer period when possible. And it turns out that this is entirely achievable.

3.12.1 Calculating the Leader Schedule

The stake distribution and randomness used to determine the leader schedule for epoch e need to be available at the start of epoch e. In the case of a public leader schedule, it is also convenient to publish the schedule itself at the start of the epoch. But we can start calculating those *before* the end of epoch $e - 1$. The details of when the stake distribution has to be taken for the leader election, and when the randomness has to be agreed on, depends on the choice of consensus protocol.

3.12.2 Calculating and Distributing Rewards

The rewards for epoch e depend on the contents of that epoch, so it is not feasible to start calculating them during that epoch. However, there is no hard constraint to actually distribute those rewards at the beginning of epoch $e + 1$. If we instead defer that payout by one epoch, and pay rewards for epoch e at the beginning of $e + 2$, we will have a whole epoch for that calculation.

3.13 Wallet Recovery Process

Wallet recovery is the process of reconstructing a wallet from the root key. In order to reconstruct a wallet, all addresses belonging to that wallet which appear on the blockchain need to be identified.

[11] We use the term "slot number" to refer to an absolute slot number, i.e., specifying both the epoch and the slot throughout this document.

In the Byron implementation, this is done by traversing the blockchain, and for each address, checking whether it belongs to the wallet. Unfortunately, this is linear in the size of the blockchain, leading to a very poor user experience.

To speed this up, we will reverse the strategy. Instead of going through the addresses on the blockchain, checking for each whether it belongs to the wallet, we go through the possible addresses of the wallet, and search whether they appeared on the blockchain.

In order for this to be efficient, we need to maintain an index, where we can look up addresses in the blockchain by some key, and we need to have a way of generating the key for an arbitrary range of addresses in the wallet, using only the root key as input.

Recall from Section 3.2 that a payment address contains a payment credential, as well as a stake address reference, where only the stake address reference depends on the delegation for that address. The key payment credential is derivable from the root key (in particular, it does not depend on the delegation preferences of the wallet), and is a suitable key for the lookup of addresses[12].

Of course, we cannot search for *all* possible addresses of the wallet. Instead, we utilise the tree structure of the HD wallet. We will require that the wallet software populates this tree in a specified way that will allow us to do a kind of exponential search for the addresses of the wallet[13].

3.13.1 Trees of Depth 1

To simplify, let us consider a wallet where the HD wallet tree is of depth 1, so that each address has an index $i \in \mathbb{N}$. We will require that the wallet creates addresses in order, and that there is a *maximal address gap* $\bar{i}$, such that the address α_i will not be generated unless there is an address $\alpha_{i'}$, with $\exists i' \in [i - \bar{i} - 1, i - 1]$ already appearing on the blockchain.

The first step in restoring a wallet is to find an upper bound on the number of addresses of the wallet, i_{up}. This can be done by consecutively looking at the intervals

$$I_n = [2^n + i | i \in [0, \bar{i}]], n \in \mathbb{N}$$

and checking whether any of the addresses in α_i for $i \in I_n$ appears on the blockchain. This check is performed by creating the corresponding payment key, hashing it, and doing a look-up in the index. For some n, this will fail, and we will have found $\bar{i}$ consecutive indices for which there are no addresses of this wallet on the blockchain. Because $\bar{i}$ is the maximal address gap, no address larger than 2^n has been created for the address, and we have $i_{\text{up}} = 2^n$.

Afterwards, we can perform a binary search for the maximal address i_{max}, in the interval $[2^{n-1}, 2^n]$. In each step of this binary search, we will probe for $\bar{i}$ consecutive addresses, starting from an offset i. If none of them exist, we know that $i_{\text{max}} < i$, otherwise $i_{\text{max}} \geq i$.

Finally, we will create all payment key hashes in the range $[0, i_{\text{max}}]$, and look up the corresponding addresses.

3.13.1.1 Early Finish and Memoisation

The above process will perform more lookups than necessary. The binary search can be aborted once the search window gets smaller than $\bar{i}$. In addition, we should consider memoising the payment keys and/or lookups.

[12] Depending on the serialisation format for addresses, it might be possible to not use a separate index at all: if $\mathcal{H}(vk)$ is a prefix of the serialised address, we can directly do a prefix query in the database.

[13] This is similar to the account discovery in BIP44.

3.13.2 Taller Trees

This scheme can be generalised for trees of larger depth. The current wallet in Cardano has a fixed depth of 2. Each address in this wallet has an index $(i, j) \in \mathbb{N} \times \mathbb{N}$. In order to generalise the above wallet restoration procedure for this wallet, we will require that there is no gap in the i, and a maximal gap $\bar{j}$ in j.

Identifying the maximal value $i_{\max}$ is straightforward: look at lists of indices

$$[(i, j) | j \in I_0]$$

for increasing values of i, until there is no address found on the chain for a specific value of i. Once $i_{\max}$ is found, we can iterate the method for trees of depth 1 over all $i \in [0, i_{\max}]$.

Further generalisations to arbitrary depths are straightforward, provided that

- all the leaves are at the same depth
- at each depth, we can require a certain maximal gap

3.13.2.1 Retrieving Delegation Information

After the wallet software has determined the set of addresses that belong to it via the payment keys, it needs to set its delegation preference. In order to do so, it compares the stake address references of its addresses.

If the wallet consists of base and/or addresses using the same stake address the wallet should look whether there is a stake address registration and delegation certificate for this key. If there are, and the delegation certificate points to an active stake pool, the wallet should set its delegation preference to use pointer addresses to the same stake address, and inform the user of this choice. Otherwise – if the stake address is unregistered, or there is either no delegation certificate or one that does not point to an active pool – it should inform the user that the stake is currently undelegated, and that they should consider delegating to receive rewards and add to the stability of the system.

If the wallet consists of addresses with different stake addresses the wallet should repeat the process above for all the stake addresses, present the list of stake pools that are delegated to by the wallet, and ask the user to pick one for future addresses, as well as provide an option to re-delegate all funds to that pool.

After setting the delegation preferences of the newly restored wallet, the wallet software should encourage the user to visit the delegation centre to make sure that this choice is still competitive.

3.13.3 Maximal Address Gap

As explained above, the wallet recovery process depends on a defined constant for the maximal address gap. A value of $i > 0$ allows a wallet owner to create several addresses at once which do not have to be processed in order. The wallet software needs to be aware of this constant so that it will not create undiscoverable addresses and so that it can warn the owner when it reaches the limit.

By default, the maximal address gap will be $i = 20$. Wallets can allow using a custom value (which should be convenient for exchanges or merchants), but when they do, the custom value will have to be known during wallet restoration.

4 Delegation Scenarios

4.1 Stake Pool Registration

Publicly announcing a stake pool for other people to delegate to requires two steps: posting a stake pool registration certificate to the blockchain, and providing *pool metadata*, additional information about the pool. The certificate contains all the information that is relevant for the execution of the protocol (public key hashes, cost, margin, and pledge) as well as the content hash of the metadata, while the metadata will be displayed to end users by their wallet. For specifics about the metadata, see Section 4.2. If no metadata is provided, the stake pool is considered private, and will not be displayed in wallets.

A stake pool operator can change the costs and margin of the pool by replacing the registration certificate of the pool with a new one. This allows operators to react, for example, to a change in its operational costs or the exchange rate of ada.

The rewards that a stake pool gets depend on a pledge of funds that the stake pool owner(s) provide. This adds a cost to creating a competitive stake pool, and protects against Sybil attacks on the stake pool level (Section 2.2.1). In order to differentiate between delegated and pledged stake, the stake pool operator will include a list of stake addresses, the owner stake addresses, in the certificate. Stake delegated from any of the owner stake addresses will be counted towards the stake pledged by the owner(s). Note that this still requires delegation certificates to be posted[14]. Using a *list* of owner stake addresses allows for stake pool operators to use multiple accounts/wallets for delegating the stake they pledged. It also allows a group of people combining their stake to form a competitive pool, without losing any control over their funds (see also the discussion in Section 3.3.5).

A stake pool operator will indicate, in the stake pool registration certificate, the amount of stake that the owners pledge to the pool, when registering a pool. It is important that the amount pledged is registered in the certificate: otherwise, an adversarial stake pool operator could circumvent the Sybil protection of the pledge mechanism, by pledging stake to a pool until it attracted stake, and then simply pledging the stake to the next pool. The pledge will be enforced during the reward calculation: pools where the owners do not meet the pledge in a given epoch will earn no rewards for that epoch. Note that this affects *all* pool rewards, both for the operator and for pool members.

Note that it will still be possible for a stake pool operator to decrease the amount of stake that they pledge to the pool, but this will require them to post a new certificate, which will notify the stakeholders that delegated to the pool (if it reduces the desirability of the pool), possibly triggering a re-delegation (see Section 4.3).

Remark: Due to the nature of our incentives mechanism (see Section 5), very large stakeholders are incentivised to split their stake and create several pools. For a future version of Cardano, we may facilitate this by allowing such stakeholders to set up all their pools with a single certificate. For the present version, however, these pools will have to be created manually. This seems justified, given that there is only a handful of such very large stakeholders and seeing as such a feature would complicate engineering.

4.2 Stake Pool Metadata

The stake pool registration certificate (see Section 3.3.4) optionally contains a content hash and a URL (up to 64 bytes), pointing to a JSON object with the following content:

A Ticker of 3-5 characters, for a compact display of stake pools in a wallet.

[14] We also contemplated *automatically* counting the stake controlled by any owner stake address towards the pledge, but that complicates the design, since we had to forbid any of those addresses from posting valid delegation certificates to prevent double delegation. Imposing a special treatment of those addresses would also be a violation of the design goal from Section 2.5.1.

A Name of up to 50 characters.

A Short Textual Description of up to 255 characters.

A URL to a homepage with additional information about the pool.

All characters in the metadata will be UTF8 encoded, as per the JSON specification. The metadata is restricted to have a total size of no more than 512 bytes, including all JSON encoding overheads.

The stake pool operators are responsible for serving this data at the URL provided in the stake pool registration certificate. However, wallets should not retrieve the data for each stake pool at those individual URLs. Not only would that be inefficient, it would also allow malicious actors to slow down all wallets by intentionally delaying the response of their server. Instead, metadata will be cached on *metadata proxy servers*.

Those proxy servers will query the metadata URLs in the stake pool registration certificates, and cache the metadata. The wallet will then retrieve the metadata for pools it needs to display from one of the proxy servers, instead of having to send a request to each of the pool's metadata URLs.

Those servers are simple, and in particular, they require relatively little trust: because of the content hash, someone running a proxy server can not display forged metadata. The worst thing they can do is filter the list of stake pools.

In order to avoid those proxy servers to become a point of centralisation of the system, it is encouraged that third parties (stake pools and other members of the community) should also run metadata proxy servers. Wallets should be configurable to query a number of those proxy servers.

Another function that the metadata proxy servers provide is filtering malicious entries: it is possible to embed a variety of malicious content in the metadata, and in particular via the link to the stake pool's homepage. Should it become known that a particular pool hosts dangerous or illegal content[15], maintainers of a metadata proxy can filter that entry and not provide it to wallets. This is an advantage over writing the metadata directly to the chain, in which case there would be no way to protect wallet users from visiting malicious sites directly from their wallet.

4.3 Display of Stake Pools in the Wallet

The wallet software will maintain a set of all the active stake pools. For each, it will perform a lookup of the metadata (which is indexed by the metadata hash) to display to the user.

In order for stakeholders to be able to delegate their stake to a pool, the wallet will provide a listing of stake pools, in a section of the UI called the *delegation centre*. In order to make it easy for users to do a rational choice when delegating, this listing will be ordered by the rewards that the user should expect from delegating to each pool. In particular, we use the non-myopic pool member rewards, Equation (3) in Section 5.6.1. Since this ordering depends not only on the costs and margin set by the stake pool operator, but also on the performance of the pool and on the amount of stake that it already has accumulated, this will promote pools that are reliable, have not yet reached saturation, and have a low cost and margin. In other words, the users selfish interest to pick a stake pool that is promising large rewards is aligned with the goal of placing the system in the hands of a number of reliable stake pool operators, and of avoiding centralisation. The influence of the stake pool operator's pledge on the rewards provides protection against a Sybil attack on the stake pool level (Section 2.2.1).

When calculating the expected rewards, the wallet will use the best data available:

- The cost, margin, and pledged stake will be taken from the most recent stake pool registration certificate of the pool.

[15] for example phishing or Trojan software, with the purpose of infecting the computer the wallet is running on

- The performance of the pool will be estimated using historical data, as described in [BC20].
- The stake of the pool, and the amount of stake that the owners of the pool contribute (in order to check whether their pledge is honoured) is taken from the current stake distribution (calculated from the current UTxO set and delegation relation on demand when the wallet performs the ordering).
- The member stake t in Equation (3) is taken to be the stake that the user is about to delegate.

  For listings outside of the wallet, for informational purposes of which pools are generally desirable, we can instead divide off the factor t in Equation (3)[16] and get the reward per stake delegated to a pool, assuming that the delegated stake is small enough to not push the pool over the saturation threshold.

When the wallet is running and the user has delegated to a stake pool, the wallet should monitor the non-myopic rewards regularly. Should the stake pool become less favourable (by missing blocks, or even becoming inactive, or by changing its cost/margin), the wallet should notify the user, and ask them to consider changing their delegation.

We had considered adding some jittering to the ordered list of stake pools, in order to prevent a situation where a slight difference in the expected rewards would lead to stakeholders all delegating to the same, slightly more preferable, pool. We decided against this, since

- Our incentive structure will have stake pools saturating anyway.
- Randomising the order of display makes it more difficult for stake pool operators to behave rationally when setting their cost and margin.

4.4 Basic Delegation

Delegating stake requires posting two certificates to the chain: a stake address registration, and a delegation certificate. Posting those certificates requires funds, so a user setting up their first wallet will need a bootstrapping mechanism. This mechanism relies on the possibility of base addresses using a stake address before posting the registration certificate for that key.

4.4.0.1 Bootstrapping a New Wallet

A user about to receive their first ada (whether through redemption, from a trade on an exchange, or some other source), will set up a new wallet, and create a value address to receive those funds. This address will refer to a stake address (by value) that is generated by the wallet, but not yet registered on the chain.

After receiving the initial funds, the user can then delegate, by posting a stake address registration certificate, as well as a delegation certificate for this stake address. Once the stake address is registered, newly created value addresses can refer to it by pointer instead.

As mentioned in Section 3.2.1, there is a slight possibility that the stake address registration certificate can be lost due to a fork. In that case, the pointer addresses would no longer point to a valid certificate. Such addresses are considered valid addresses for the purpose of moving funds, but ignored when determining the stake distribution (just like an enterprise address). The wallet software should detect usage of such broken pointer addresses, and ask and assist the user to create a new stake address registration, and to move the funds to value addresses referring to this new stake address. Wallets can try to avoid this situation, by either allowing a number of blocks between transactions t_1 registering a stake address and t_2 moving funds to a pointer address for this stake address, or by using an output of t_1 as an input to t_2.

[16] Effectively taking the constant term in the Taylor expansion in t

4.4.0.2 Additional Accounts

The user might want to create an additional account in their wallet later on, using a different stake address, to prevent linkability of all their addresses. In principle, they could use the funds that are already in their wallet to post the stake address registration certificate for the new account, and only have pointer addresses in the new account. However, this provides a strong hint for observers of the chain that the two accounts belong to the same person, so it is recommended to also bootstrap additional accounts in the manner described above.

4.4.0.3 Re-Delegating

Re-delegating the funds belonging to one stake address of the wallet requires posting a single transaction, containing a delegation certificate. This will only incur the usual transaction fees. In particular, the deposit paid for the first delegation certificate (which is thus overridden) will be good for the new certificate. Consequently, re-delegation does not carry a heavy cost, as required by Section 2.1.4.

4.5 Delegation of Cold Wallets

Cold wallets are to be used for long-term storage of larger funds, so it is important that we encourage owners of cold wallets to participate in the PoS through the delegation mechanism. This will require a second, non-cold, wallet, to post the initial certificates, as well as any delegation certificates for re-delegation. There are two scenarios to be considered:

The User Does Have a Non-Empty Wallet Already Suppose a user owns a wallet with some funds, and wants to move most of those to a cold wallet, such as a paper wallet. They will use Daedalus to create this cold wallet. Daedalus can offer to post the stake address registration certificate for the stake address of the cold wallet upon creation of the wallet, and to store that stake address with the non-cold wallet, so that the user will be able to sign and post delegation certificates for the cold wallet. In this case, all addresses in the cold wallet can be pointer addresses.

The User Does Not Control Any Funds When Creating the Cold Wallet In this case, the user will use Daedalus to create a cold wallet, which will use a base address. Daedalus will provide the stake address, including the signing key(s), to the user, so that they can post a registration certificate, and delegation certificates, whenever they have funds in a non-cold wallet.

4.6 Self Delegation

Stakeholders should not be forced to delegate their stake to a pool. Instead, they should have the option of running their own node, using their own stake.

Technically, such stakeholders will create a *private pool*, which is just a stake pool with margin $m = 1$, and without providing metadata. Such pools will pay all rewards to the pool operator (which is not a special rule, but just the effect of having a margin of 1), and they will not be shown in the stake pool directory in Daedalus (although even if they were, they would always be listed at the very bottom, since they would not promise any rewards to their members).

We had looked at other options that would not require individual stakeholders to register a pool, but they either complicated the design, or made it possible for free riders to contribute stake and get a share of the rewards by using suitably chosen addresses. The mechanism of private pools adds no additional complexity to the delegation system (the only added work is to suppress their listing in Daedalus). Optionally, the front-end could even set up (and retire) a private pool at the press of a button, but this is not a must-have feature for the initial release.

5 Design of Incentives

5.1 Overview of Incentives

On a high level, the goal of the incentives mechanism is to incentivise stakeholders to follow the protocol and thereby to guarantee the secure and efficient operation of Cardano.

More specifically, we want a majority of stake holders to delegate to a number k of *stake pools* (where k is a parameter of the system – see Section 5.2). The *pool operators* of those stake pools are supposed to

- fulfill their Ouroboros protocol participation responsibilities, such as being online during slots for which they are a slot leader and then creating a block containing as many transactions as possible.
- provide additional network infrastructure.

Other stakeholders can then *delegate* their stake to a registered pool. Stakeholders are also free to either run their own private pools, or not take part in the protocol at all. In the latter case, their stake is ignored, and they will not receive any rewards.

Incentives are provided in the form of *social pressure* (by making pool operator performance and adherence to the protocol public), but mostly by *monetary incentives* in the form of ada.

A design goal of the mechanism is to align monetary incentives as perfectly as possible with protocol adherence: If every stakeholder follows their own financial interests, the system should settle into a desirable state. If possible, there should never be a conflict of interest between maximizing rewards and "doing the right thing".

Rewards will be paid for each epoch and will be drawn from the following sources:

- monetary expansion
- transaction fees
- decayed deposits.

All rewards will be collected in a (virtual) pot, and then shared amongst stake pools depending on their contribution to the operation of the system. The main factor will be the relative stake that a pool controls. However, there will be several refinements to this general principle:

- Rewards for a stake pool will be capped when the pool gets too large (otherwise, the system would converge towards a state with all stake being delegated to one giant stake pool).
- Rewards will decrease if a pool operator does not create the blocks they are supposed to create.
- Pool operators will be compensated for their trouble and risk by
  - reimbursing their costs and
  - giving them a *margin* before distributing the remaining pool rewards proportionally amongst pool operator and pool members. (Pool operators publicly declare their cost and margin, which they can freely choose.)
- Pool rewards will slightly increase with the stake their owner(s)[17] pledge to delegate to the pool. There is no minimal stake required to create a pool – anybody can do this. However,

[17] For a discussion of stake pool owners vs stake pool operators see Section 3.3.5.

pools where the owners contribute more stake will get slightly higher rewards. (This will discourage pool owners from splitting their stake to operate several pools. It will also help preventing Sybil attacks, where an attacker with low stake tries to gain control over a majority of stake by creating a lot of pools with low costs.)

Our game theoretic analysis has shown that if stakeholders try to maximize their rewards in a "short-sighted" (*myopic*) way (pool members joining the pool with the highest rewards *at this moment*, pool operators raising their margins to get higher rewards *at this moment*), chaotic behaviour will ensue.

Therefore, we will calculate *non-myopic* rewards. Wallets will display pools ranked by those non-myopic rewards, thus guiding stakeholders to behave in a way that will benefit them in the long run. Our analysis shows that if everybody follows this advice, the system will stabilize in a *Nash Equilibrium*, meaning that no stakeholder will have incentive to act differently.

Rewards to both the pool operators and the pool members will be calculated and distributed by the system some time after the end of an epoch. No manual intervention (transfer of funds) will be necessary. In particular, a pool operator cannot simply refuse to share rewards with the stake pool members.

5.2 Parameters

There will be a couple of parameters whose values have to be set in advance:

- The desired number of pools $k \in \mathbb{N}_+$.
- The influence $a_0 \in [0, \infty)$ the stake pledged by the owners should have on the desirability of the pool. Small values of a_0 indicate little influence.
- The *expansion rate* $\rho \in [0, 1]$, determining the fraction of still available ada that will be created per epoch.
- The fraction $\tau \in [0, 1]$ of rewards going to the treasury.

We will discuss in Section 5.10 how one could approach choosing reasonable values for these.

5.3 Reminder: Stake Pool Registration

Recall from Section 4.1 that stakeholders who wish to operate a stake pool have to *register* their pool on the blockchain. From the point of view of reward calculation (see Section 5.4), the following information has to be included in the registration:

- The *costs* of operating the pool (in ada/epoch).
- The pool operator *margin* (in $[0, 1]$), indicating the additional share that the pool operator will take from the pool's rewards (after the costs have been deducted) before splitting rewards amongst members (see Section 5.5.4).
- Proof of *ada pledged to the pool*, as a list of stake addresses that belong to the owner(s) of the pool.

There will be no lower bound on the amount of ada that has to be pledged, but we will see in Section 5.5.3 that pool rewards will increase with this amount. This is necessary to prevent people with low stake from registering many pools, gaining control over a lot of stake and attacking the system (see requirement 2.2.1 and Section 4.1).

5.4 Epoch Rewards

There will be three sources of rewards for an epoch: *transaction fees*, non-refundable parts of *deposits*, and *monetary expansion*.

5.4.1 Transaction Fees

All transaction fees from all transactions from all blocks created during the epoch will be added to the rewards pot of that epoch.

5.4.2 Deposits

As explained in Section 3.10.2, a part of the deposits for certificates and UTxO entries are non-refundable, and contribute to the rewards pot instead.

5.4.3 Monetary Expansion

Every epoch, the total amount of ada in circulation T will be increased by adding ada to the rewards pot. To ensure the amount of ada never exceeds a finite, specified limit T_∞, the increase will reduce exponentially. Also, the increase will depend on the number of blocks that have been produced during the epoch. Specifically, each epoch, the contribution from monetary expansion to the rewards pot is given by

$$\min(\eta, 1)\rho\,(T_\infty - T)\ ,$$

where

η is the ratio between the number of blocks that have been produced during the epoch, and the expectation value[18] of blocks that should have been produced during the epoch under ideal conditions (i.e., no forks, no missed blocks).

For Ouroboros Classic [KRDO17], the number of expected blocks will be the number of slots in an epoch, while for Ouroboros Praos, we will use the number of slots per epoch times the active slots coefficient f (see Equation (1) in [DGKR17]).

ρ is the monetary expansion parameter.

T_∞ is the maximal amount of ada to ever be in circulation (i.e., $45 \cdot 10^9$ ada).

T is the amount of ada in circulation at the beginning of the epoch for which we want to calculate the rewards pot.

The dependence on η incentivises cooperative behaviour, and in particular discourages the pools sabotaging each others blocks. Note that η can exceed 1 when there are more blocks produced in an epoch than would be expected on average. But the product $\min(\eta, 1)\rho$ is always bounded by 1, which is necessary to ensure we never exceed T_∞ ada in circulation.

Since T_∞ is finite, rewards from monetary expansion will decrease over time. This has to be compensated by

- rising transaction fees when more and more people use the system and
- higher exchange rates from ada to USD when the system becomes more valuable.

Note that the fees that have been collected during the Byron era, where all nodes have been provided by IOHK, Emurgo, and the Cardano Foundation, have not been paid out. Those fees have reduced the amount of ada currently in circulation, but they did not change T_∞. Effectively, those fees will be distributed amongst future stakeholders via monetary expansion.

[18] Note that in Ouroboros Classic, this is just the number of slots in an epoch, but for instance in Praos, where we do not always have one leader per slot, we only have a notion of how many blocks we expect per epoch *on average*.

5.4.4 Treasury

A fraction τ of the rewards pot for each epoch will go to the *treasury*.

Note that we do not have a full treasury system yet, and implementing it requires further research. Nevertheless, we start collecting funds for the treasury already, in a pot called the T pot. Once the treasury is implemented, funds in the T pot will be made available for decentralised development of the system.

5.5 Reward Splitting

In this section we describe how the total rewards R from one epoch are split amongst stakeholders.

These calculations proceed in two steps: First, rewards are split amongst *pools*. Next, each pool splits its share of R amongst its operator and its members.

5.5.1 Relative Stake: Active vs Total

As explained in Section 3.4.3, we distinguish the amount of *active stake* and the *total stake*. Stake is only considered active when it is correctly delegated to a non-retired pool. Whenever we look at fractions of stake (in the leader election, or when distributing rewards according to stake), we need to specify whether we normalise to the active or to the total stake.

Leader Schedule For the purpose of determining the leader schedule, we use the construction in Section 3.4.3, which yields the stake relative to the amount of *active* stake.

The benefit of doing this is that the chain growth will be independent of how many stakeholders do delegate their stake.

Rewards Distribution When we distribute rewards depending on how much stake any party provides, we will always use the stake relative to the *total* stake.

That way, we ensure that the share of the rewards that any one party gets does not change when the amount of active stake changes. Not only is this more predictable for individual players, it also prevents creating an incentive to block new delegation certificates. If we were using the stake relative to the active stake, then any player who already had active stake would lose rewards when other players started delegating their stake, which could lead to a collective censorship of transactions with delegation certificates.

Performance Estimations The rewards of a pool will depend on how well they perform, i.e., on how many blocks they produce, and on how many blocks we would expect a pool with their stake to produce (Section 5.5.2).

Since leader election depends on the fraction of *active* stake that a pool controls, we have to normalise to active stake when estimating performance as well.

5.5.2 Stake, Performance, and Block Production

The incentives scheme developed in [BKKS18] distributes rewards solely on the basis of the size of pools in terms of the stake that they control (and the stake pledged by the owner(s)). This does eliminate any incentive for a pool to try to sabotage another pool (be it through ignoring their blocks, or even by mounting DoS attacks on their nodes). However, it does not incentivise pool operators to ensure that their pool performs well (i.e., produces most of the blocks it is eligible to create). This invites free-riders, pools that rely on other pools to maintain the system, and collect rewards without doing any work.

For Cardano, we will stay close to the scheme of [BKKS18], but we will also take into account the *performance* of a pool. Unfortunately, tracking the performance of a pool is not trivial in protocols without a public leader schedule, like Ouroboros Praos.

In order to get a handle on the performance of a pool, let us consider the fraction β of all blocks within a given epoch that the pool created.

There are multiple factors that influence β: the stake of the pool, how well it performs, and some randomness due to the leader election and the overall performance of the other pools and the network. We can write[19]

$$\beta = \sigma_a p r_e \,,$$

where

σ_a is the relative stake σ of the pool (the fraction of the *active* stake that the pool controls),

p is the performance p of the pool, i.e. the fraction

$$p = \frac{n}{\max(N, 1)}$$

of the number n of blocks it successfully added to the chain and the number N of slots it was elected as a leader, and

r_e is a factor that captures the relation between the relative stake σ_a of the pool, the number N of slots it is elected as a leader, and the total number $\overline{N}$ of blocks that were added to the chain during the epoch. To be precise, we have

$$r_e = \frac{N}{\sigma_a \max(1, \overline{N})} \,.$$

The factor r_e captures random influences on β: the randomness in the leader election that influences N, and the randomness both from the leader election and the bunch of random influences on $\overline{N}$ (leader election, forks, performance of the network and other pools).

If we insert the definitions of p and r_e into β, we find that

$$\beta = \sigma_a \frac{n}{\max(N, 1)} \frac{N}{\sigma_a \max(1, \overline{N})} = \frac{n}{\max(1, \overline{N})} \,,$$

which is indeed the fraction of blocks produced by the pool[20].

In Ouroboros Praos, we can observe σ_a, n, and $\overline{N}$, but we do not have a direct handle on any of N, r_e, or p. From the observables, we can extract

$$\beta = \frac{n}{\max(1, \overline{N})} \,. \tag{1}$$

While the true performance p is not accessible, we can define and measure the *apparent performance*

$$\overline{p} := p r_e = \frac{\beta}{\sigma_a} \,.$$

We will use the apparent performance $\overline{p}$ as a proxy for the performance when determining the rewards for pools.

[19]We assume a linear relation between stake and the number of blocks that a pool is eligible to create. Strictly speaking, this is not the case for Ouroboros Praos. However, the error when linearising the leader election function is small, particularly for the range of parameters we are considering for Cardano.

[20]When cancelling N and $\max(N, 1)$, note that for $N = 0$, also $n = 0$, and the equation trivially holds.

5.5.3 Pool Rewards

For a given epoch, the *optimal* rewards for a pool are

$$f(s,\sigma) := \frac{R}{1+a_0} \cdot \left(\sigma' + s' \cdot a_0 \cdot \frac{\sigma' - s' \frac{z_0 - \sigma'}{z_0}}{z_0} \right) . \tag{2}$$

Here

- R are the total available rewards for the epoch (in ada).
- $a_0 \in [0, \infty)$ is a parameter determining owner-stake influence on pool rewards.
- $z_0 := 1/k$ is the size of a saturated pool.
- $\sigma' := \min(\sigma, z_0)$, where σ is the relative stake of the pool (note that this is relative to the *total stake*, not the active stake).
- $s' := \min(s, z_0)$, where s is the relative stake of the pool owner(s) (the amount of ada pledged during pool registration, see Section 4.1).

As mentioned in Section 4.1, the rewards for a pool where the owner(s) fail to honour their pledge of stake will receive zero rewards, and so will have $f = 0$.

Note that σ includes the stake s pledged by the pool owner(s). For example, let us assume that the total existing supply of ada is $T = 31,000,000,000$, and consider a pool whose owners pledged ada 15,500,000 and who attracted another ada 15,500,000 from their pool members. Then

$$\begin{aligned} s &= \frac{15,500,000}{31,000,000,000} &&= 0.0005 \text{ and} \\ \sigma &= \frac{15,500,000 + 15,500,000}{31,000,000,000} &&= 0.001. \end{aligned}$$

The *actual* rewards take the apparent performance into account, and are given by [21]

$$\hat{f}(s,\sigma,\overline{p}) := \overline{p} f(s,\sigma) .$$

Note that

- Even when a pool's true performance is 1, its actual rewards might be less than its optimal rewards, because of the randomness in r_e. Likewise, sometimes a pool's actual rewards will be *higher* than their optimal rewards, if they are lucky in the leader election process.

  These effects will balance each other, so that a well performing pool will get their optimal rewards *on average over multiple epochs*.

- We will always have

  $$\sum_{\text{pools}} \hat{f}(s,\sigma,\overline{p}) \le R ,$$

  so that all actual rewards can be paid from the rewards pot[22].

  The difference $R - \sum_{\text{pools}} \hat{f}(s,\sigma,\overline{p})$ will be sent back to the reserves.

[21] Note that since $\overline{p}\sigma' = \beta\frac{\sigma'}{\sigma}$, this nearly amounts to replacing σ' by $\beta\frac{\sigma'}{\sigma}$ in Equation (2), i.e. to rewarding pools based on the number of blocks that they produced, rescaled by a punishing factor σ'/σ for pools that grow beyond z_0.

[22] To prove this, use that $s' \le \sigma' \le z_0$, and $\sum_{\text{pools}} \beta = 1$.

5.5.4 Reward Splitting inside a pool

After the rewards for a pool have been determined according to Section 5.5.3, those rewards are then split amongst the *pool operator* and the *pool members*.

Consider

- $\hat{f}$, the *pool rewards*,
- c, the pool *costs* (in ada),
- $m \in [0,1]$, the *margin*,
- $\sigma \in [0,1]$, the relative stake of the pool.

Note that the values c and m for registered pools are available from the pool registration, see Section 4.1.

5.5.4.1 Pool Operator Reward

The *pool operator reward* r_{operator} (in ada) is calculated as follows (where $s \in [0,1]$ is the stake delegated to the pool by its owner(s)):

$$r_{\text{operator}}(\hat{f},c,m,s,\sigma) := \begin{cases} \hat{f} & \text{if } \hat{f} \leq c, \\ c + (\hat{f} - c) \cdot \left(m + (1-m) \cdot \frac{s}{\sigma}\right) & \text{otherwise.} \end{cases}$$

5.5.4.2 Pool Member Reward

The *pool member reward* r_{member} (in ada) is calculated as follows (where $t \in [0,1]$ is the stake of the pool member):

$$r_{\text{member}}(\hat{f},c,m,t,\sigma) := \begin{cases} 0 & \text{if } \hat{f} \leq c, \\ (\hat{f} - c) \cdot (1-m) \cdot \frac{t}{\sigma} & \text{otherwise.} \end{cases}$$

5.6 Non-Myopic Utility

It would be short-sighted ("myopic") for stakeholders to directly use the reward splitting formulas from Section 5.5, and base their delegation choice on those. They should instead take the long-term ("non-myopic") view. To this end, the system will calculate and display the "non-myopic" rewards that pool operators and pool members can expect, thus supporting stakeholders in their decision whether to create a pool and to which pool to delegate their stake.

The idea is to first rank all pools by "desirability", to then assume that the k most desirable pools will eventually be saturated, whereas all other pools will lose all their members, then to finally base all reward calculations on these assumptions.

5.6.1 Pool Desirability and Ranking

First we define the *desirability* of a pool with pledged owner stake s, costs c and margin m. Simply put, this number indicates how "desirable" or "attractive" this pool is to (potential) members.

If the pool is *saturated*, the pool rewards are

$$\tilde{f}(s,\overline{p}) := \hat{f}(s,z_0,\overline{p}) = \frac{\overline{p}R}{1+a_0} \cdot (z_0 + \min(s,z_0) \cdot a_0).$$

The *desirability* is then defined as

$$d(c,m,s,\overline{p}) := \begin{cases} 0 & \text{if } \tilde{f}(s,\overline{p}) \le c, \\ \big(\tilde{f}(s,\overline{p}) - c\big) \cdot (1-m) & \text{otherwise.} \end{cases}$$

To determine a pool's *rank*, we order pools by decreasing desirability. The most desirable pool gets rank 1, the second most desirable pool gets rank 2 and so on.

We predict that pools with rank $\le k$ will eventually be saturated, whereas pools with rank $> k$ will lose all members and only consist of the owner(s).

5.6.2 Non-Myopic Pool Stake

Consider a pool with pledged owner stake s, total stake σ and rank r. Consider also a potential delegator with stake t. We define the *non-myopic stake* σ_{nm} as

$$\sigma_{\text{nm}}(s,\sigma,t,r) := \begin{cases} \max(\sigma + t, z_0) & \text{if } r \le k, \\ s+t & \text{otherwise.} \end{cases}$$

5.6.3 Non-Myopic Pool Operator Rewards

The non-myopic pool operator rewards of a pool with costs c, margin m, pledged owner stake s, stake σ, rank r, and apparent performance $\overline{p}$ are

$$r_{\text{operator,nm}}(c,m,s,\sigma,r,\overline{p}) := r_{\text{operator}}\Big(\hat{f}\big(s,\sigma_{\text{nm}}(s,\sigma,0,r),\overline{p}\big),c,m,s,\sigma_{\text{nm}}(s,\sigma,0,r)\Big).$$

5.6.4 Non-Myopic Pool Member Rewards

The non-myopic pool member rewards of a pool with costs c, margin m, pledged owner stake s, stake σ, rank r, and apparent performance $\overline{p}$, for a member contributing member stake t, are

$$r_{\text{member,nm}}(c,m,s,\sigma,t,r,\overline{p}) := r_{\text{member}}\Big(\hat{f}\big(s,\sigma_{\text{nm}}(s,\sigma,t,r),\overline{p}\big),c,m,t,\sigma_{\text{nm}}(s,\sigma,t,r)\Big). \tag{3}$$

5.6.5 Replacing Apparent Performance

Using the apparent performance of a pool *within the last epoch* is not suitable for determining the long-term expected rewards for delegating to a pool. Rather, one should use the estimate the performance using the historical data. This avoids preferring or discarding a pool just because it performed exceptionally well or bad in one particular epoch. If the ratio of the number of stake pools and the number of expected blocks per epoch is large, this becomes even more important, since the apparent performance in a single epoch would be bound to fluctuate quite a bit. Our method of estimating stake pool performance is explained in [BC20].

5.7 Utility

When deciding whether to operate a stake pool or participating in an existing pool as a member, stakeholders should not look at the plain rewards. Instead, they should look at the *utility*, the difference between rewards and costs. The simulations in [BKKS18] use the utility as a basis for rational decisions of the players.

For the purpose of this document, it is sufficient to consider the rewards: for pool members, they are identical (since they do not have to consider running costs), and we do not plan the wallet to assist pool operators in setting up a stake pool or defining the cost and margin parameters (at least not for the initial release).

5.8 Claiming Rewards

All information necessary to calculate each stakeholder's rewards for each epoch are contained in the blockchain, so there is in principle no need to record any extra information related to the Incentives mechanism.

However, there is the challenge to avoid "bloat" caused by thousands of "micro payments" from rewards after each epoch.

This proved to be quite a challenge. In the end, we have converged to the mechanism described in Section 3.9.1. Alternative approaches that we considered are described in Appendix A (therein, Appendix A.3.3 is the option that we have picked).

5.9 Information in Daedalus

Our game theoretic analysis assumes that every stakeholder has all relevant information available at any time.

This means that pool *costs*, *margins*, average apparent performance, and pool owners *stakes*, as well as the (non-myopic) utilities derived from these figures, have to be easily accessible, so that stakeholders can quickly react to changes and always choose the strategy that maximizes their own rewards.

Daedalus will make this information readily available, as detailed in Section 4.3.

5.10 Deciding on Good Values for the Parameters

We need to decide on reasonable values for the parameters k, a_0, ρ and τ (see Section 5.2).

5.10.1 k

The desired number of pools k depends on the level of decentralization we want on the one hand and network efficiency and overall costs of the Cardano system on the other hand. A value of $k = 100$ or $k = 1000$ seems to be reasonable.

5.10.2 a_0

As explained above, parameter a_0 determines the influence that the stake pledged by the pool owner(s) has on pool rewards.

Our game theoretic analysis predicts that the k pools with the highest *potential*, the highest value of

$$P(\lambda, c) := \left[z_0 + a_0 \cdot \min\left(\lambda, \frac{1}{k}\right)\right] \cdot \frac{R}{1 + a_0} - c$$

(where λ is the stake pledged by the pool owner(s) and c are the pool costs) will create the saturated pools.

Let us consider an attacker with stake $S < \frac{1}{2}$, who wants to gain control over a majority of stake. This means they have to lead $\frac{k}{2}$ pools, committing $\lambda = \frac{2S}{k}$ stake to each.

In order for their $\frac{k}{2}$ pools to be successful, each of these needs to have higher potential than the honest stakeholder with the $\frac{k}{2}$-highest potential has. If that honest player has committed stake $\tilde{\lambda} \leq \frac{1}{k}$ and has costs $\tilde{c}$ and if our malicious attacker is willing to lie and claim lower "dumping" costs $c = r \cdot \tilde{c}$ (for $r \in [0, 1)$), this means

$$\begin{aligned}
P\left(\frac{2S}{k}, c\right) > P(\tilde{\lambda}, \tilde{c}) &\iff \left(z_0 + a_0 \cdot \frac{2S}{k}\right) \cdot \frac{R}{1 + a_0} - c > \left(z_0 + a_0 \cdot \tilde{\lambda}\right) \cdot \frac{R}{1 + a_0} - \tilde{c} \\
&\iff a_0 \cdot \frac{2S}{k} \cdot \frac{R}{1 + a_0} - c > a_0 \cdot \tilde{\lambda} \cdot \frac{R}{1 + a_0} - \tilde{c} \\
&\iff a_0 \cdot \left(\frac{2S}{k} - \tilde{\lambda}\right) \cdot \frac{R}{1 + a_0} > c - \tilde{c} = -(1 - r) \cdot \tilde{c} \\
&\overset{a_0 > 0}{\iff} \frac{2S}{k} - \tilde{\lambda} > -\frac{\tilde{c} \cdot (1 - r) \cdot (1 + a_0)}{R \cdot a_0} = -\frac{\tilde{c}}{R} \cdot (1 - r) \cdot \left(1 + \frac{1}{a_0}\right) \\
&\iff S > \frac{k}{2} \cdot \left[\tilde{\lambda} - \frac{\tilde{c}}{R} \cdot (1 - r) \cdot \left(1 + \frac{1}{a_0}\right)\right]
\end{aligned}$$

In the following tables, we can see how the choice of a_0 influences the minimal stake S needed for a successful attack for various values of $\tilde{\lambda}$, $\tilde{c}$ and r:

$\tilde{\lambda} = 0.01, \tilde{c} = 0.001, r = 0.9$

a_0	S
0.010	0.0000
0.020	0.2450
0.030	0.3283
0.040	0.3700
0.050	0.3950
0.060	0.4117
0.070	0.4236
0.080	0.4325
0.090	0.4394
0.100	0.4450

$\tilde{\lambda} = 0.01, \tilde{c} = 0.005, r = 0.9$

a_0	S
0.050	0.0000
0.100	0.2250
0.150	0.3083
0.200	0.3500
0.250	0.3750
0.300	0.3917
0.350	0.4036
0.400	0.4125
0.450	0.4194
0.500	0.4250

$\tilde{\lambda} = 0.01, \tilde{c} = 0.01, r = 0.9$

a_0	S
0.050	0.0000
0.100	0.0000
0.150	0.1167
0.200	0.2000
0.250	0.2500
0.300	0.2833
0.350	0.3071
0.400	0.3250
0.450	0.3389
0.500	0.3500

$\tilde{\lambda} = 0.005, \tilde{c} = 0.001, r = 0.9$

a_0	S
0.010	0.0000
0.020	0.0000
0.030	0.0783
0.040	0.1200
0.050	0.1450
0.060	0.1617
0.070	0.1736
0.080	0.1825
0.090	0.1894
0.100	0.1950

$\tilde{\lambda} = 0.005, \tilde{c} = 0.005, r = 0.9$

a_0	S
0.050	0.0000
0.100	0.0000
0.150	0.0583
0.200	0.1000
0.250	0.1250
0.300	0.1417
0.350	0.1536
0.400	0.1625
0.450	0.1694
0.500	0.1750

$\tilde{\lambda} = 0.005, \tilde{c} = 0.01, r = 0.9$

a_0	S
0.100	0.0000
0.200	0.0000
0.300	0.0333
0.400	0.0750
0.500	0.1000
0.600	0.1167
0.700	0.1286
0.800	0.1375
0.900	0.1444
1.000	0.1500

$\tilde{\lambda} = 0.001, \tilde{c} = 0.001, r = 0.9$

a_0	S
0.100	0.0000
0.200	0.0200
0.300	0.0283
0.400	0.0325
0.500	0.0350
0.600	0.0367
0.700	0.0379
0.800	0.0388
0.900	0.0394
1.000	0.0400

$\tilde{\lambda} = 0.001, \tilde{c} = 0.005, r = 0.9$

a_0	S
0.500	0.0000
1.000	0.0000
1.500	0.0083
2.000	0.0125
2.500	0.0150
3.000	0.0167
3.500	0.0179
4.000	0.0188
4.500	0.0194
5.000	0.0200

$\tilde{\lambda} = 0.001, \tilde{c} = 0.01, r = 0.9$

a_0	S
0.100	0.0000
0.200	0.0000
0.300	0.0000
0.400	0.0000
0.500	0.0000
0.600	0.0000
0.700	0.0000
0.800	0.0000
0.900	0.0000
1.000	0.0000

$\tilde{\lambda} = 0.01, \tilde{c} = 0.001, r = 0.5$

a_0	S
0.050	0.0000
0.100	0.2250
0.150	0.3083
0.200	0.3500
0.250	0.3750
0.300	0.3917
0.350	0.4036
0.400	0.4125
0.450	0.4194
0.500	0.4250

$\tilde{\lambda} = 0.01, \tilde{c} = 0.002, r = 0.5$

a_0	S
0.050	0.0000
0.100	0.0000
0.150	0.1167
0.200	0.2000
0.250	0.2500
0.300	0.2833
0.350	0.3071
0.400	0.3250
0.450	0.3389
0.500	0.3500

$\tilde{\lambda} = 0.01, \tilde{c} = 0.003, r = 0.5$

a_0	S
0.100	0.0000
0.200	0.0500
0.300	0.1750
0.400	0.2375
0.500	0.2750
0.600	0.3000
0.700	0.3179
0.800	0.3313
0.900	0.3417
1.000	0.3500

$\tilde{\lambda} = 0.005, \tilde{c} = 0.001, r = 0.5$

a_0	S
0.050	0.0000
0.100	0.0000
0.150	0.0583
0.200	0.1000
0.250	0.1250
0.300	0.1417
0.350	0.1536
0.400	0.1625
0.450	0.1694
0.500	0.1750

$\tilde{\lambda} = 0.005, \tilde{c} = 0.002, r = 0.5$

a_0	S
0.100	0.0000
0.200	0.0000
0.300	0.0333
0.400	0.0750
0.500	0.1000
0.600	0.1167
0.700	0.1286
0.800	0.1375
0.900	0.1444
1.000	0.1500

$\tilde{\lambda} = 0.005, \tilde{c} = 0.003, r = 0.5$

a_0	S
0.200	0.0000
0.400	0.0000
0.600	0.0500
0.800	0.0812
1.000	0.1000
1.200	0.1125
1.400	0.1214
1.600	0.1281
1.800	0.1333
2.000	0.1375

$\tilde{\lambda} = 0.001, \tilde{c} = 0.001, r = 0.5$

a_0	S
0.500	0.0000
1.000	0.0000
1.500	0.0083
2.000	0.0125
2.500	0.0150
3.000	0.0167
3.500	0.0179
4.000	0.0188
4.500	0.0194
5.000	0.0200

$\tilde{\lambda} = 0.001, \tilde{c} = 0.002, r = 0.5$

a_0	S
5.000	0.0000
10.000	0.0000
15.000	0.0000
20.000	0.0000
25.000	0.0000
30.000	0.0000
35.000	0.0000
40.000	0.0000
45.000	0.0000
50.000	0.0000

$\tilde{\lambda} = 0.001, \tilde{c} = 0.003, r = 0.5$

a_0	S
5.000	0.0000
10.000	0.0000
15.000	0.0000
20.000	0.0000
25.000	0.0000
30.000	0.0000
35.000	0.0000
40.000	0.0000
45.000	0.0000
50.000	0.0000

See Figure 4 for the effect of various choices for a_0 on pool rewards (for $k = 10$).

5.10.3 ρ

In order to determine the inflation rate per epoch ρ, we need five more pieces of information:

- The expected *exchange rate* e from ada to USD (in USD/ADA).
- The average *costs* c (in USD) to run a pool for one year.
- The average *transaction fees* F (in ada) paid during one epoch.
- The expected ratio r of *rewards* per year per staked ada.
- The expected value of η, the ratio of actually produced blocks versus expected produced blocks (see 5.4.3).

The available rewards for one epoch (assuming an equilibrium state with k pools and noticing that there are $\frac{365}{5} = 73$ epochs per year) will be

$$(1-\tau) \cdot \big(F + \min(\eta, 1) \cdot \rho \cdot (T\infty - T)\big) - \frac{k \cdot c}{73 \cdot e}.$$

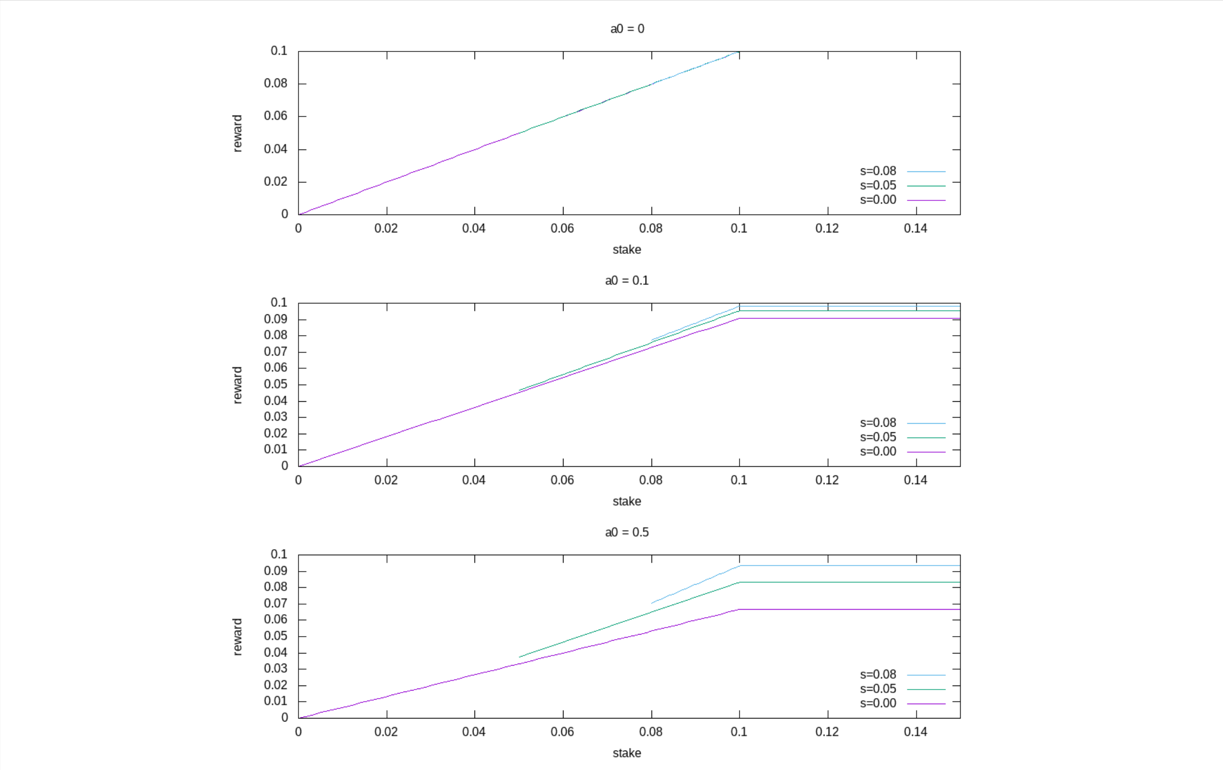

Figure 4: Effect of different choices for a_0

On the other hand, *expected* rewards per epoch are

$$T \cdot \left(\sqrt[73]{1+r} - 1 \right).$$

Equating the two, we get

$$\rho = \frac{T \cdot \left(\sqrt[73]{1+r} - 1 \right) - (1-\tau) \cdot F + \frac{k \cdot c}{73 \cdot e}}{(1-\tau) \cdot \min(\eta, 1) \cdot (T_\infty - T)}.$$

For example, using

- $k = 100$,
- $T = 31,000,000,000\,\mathrm{ada}$,
- $T_\infty = 45,000,000,000\,\mathrm{ada}$,
- $e = 0.5\,\mathrm{USD/ada}$,
- $c = 1,000\,\mathrm{USD}$,
- $F = 2,000\,\mathrm{ada}$,
- $r = 0.05$,
- $\tau = 0.2$ and
- $\eta = 0.9$

we would get

$$\rho = \frac{31,000,000,000 \cdot \left(\sqrt[73]{1+0.05} - 1 \right) - 0.8 \cdot 2000 + \frac{100 \cdot 1000}{73 \cdot 0.5}}{0.8 \cdot 0.9 \cdot (45,000,000,000 - 31,000,000,000)} \approx 0.0021.$$

This would correspond to reducing the remaining amount of available ada by $1.0021^{73} - 1 \approx 0.17 = 17\%$ per year (which sounds pretty high…).

5.10.4 τ

Setting τ is a policy decision; we will probably use $\tau = 0.2$, i.e. 20% of available epoch rewards will be sent to the treasury.

6 Satisfying the Requirements

In the following, we describe how the requirements listed in Section 2 are satisfied by the design in this document.

Section 2.1.1 Proof of Eligibility The leader election process takes delegation into account (Section 3.6), so the leader schedule will contain the key hash of the pool that is expected to sign the block. The operational key certificate will be included in the block header.

Section 2.1.2 Visibility of Delegation on the Blockchain Delegation via delegation certificates is visible on the blockchain. Operational key certificates are only used for hot/cold key management within a stake pool. Thus, they are not relevant for the rewards sharing process.

Section 2.1.3 Restricting Chain Delegation Chain delegation is properly restricted, as described in Section 3.4.4.

Section 2.1.4 Cheap Re-Delegation Re-delegation can be performed cheaply by issuing a new delegation certificate.

Section 2.1.5 Neutral Addresses The design includes enterprise addresses (Section 3.2.2), which are disregarded by the PoS protocol.

Section 2.1.7 Multi-Sig Delegation By allowing script credentials not only for payment, but also for stake credentials, and implementing a language for expressing multi-sig conditions, we provide a mechanism for multi-sig delegation (see Section 3.2).

Section 2.2.1 Sybil Attack Protection at Stake Pool Level Stake pool owners are expected to pledge an amount of stake to their pools that has an influence on the rewards for their stake pool, and consequently on the position of the stake pool in the listing displayed to stakeholders (Section 4.1, Section 4.3, Section 5.3).

Since this pledge cannot be shared between multiple pools, creating n viable stake pools will require funds linear in n.

Section 2.2.2 Address Nonmalleability Protection against the malleability attack, by the wallet, is described in Section 3.2.6.

Section 2.2.3 Public Payment Keys Should not be Disclosed Prematurely The introduction of a dedicated stake address (Section 3.2) avoids the need to use the payment key for delegation purposes.

Section 2.2.4 Mitigate Key Exposure Stake pool operators are required to use operational key certificates for hot/cold key management, as described in Section 3.3.7.

Section 2.2.5 Handle Inactive Stake Pools Stake pools can be retired via a retirement certificate (Section 3.3.4. If a stake pool ceases to operate without being properly retired, its members will be incentivised to re-delegate: their rewards will start to diminish, and their wallet will notify them that the pool they have delegated to is not producing blocks anymore Section 4.3.

In addition to this, Appendix C describes an optional mechanism to detect and ignore inactive pools that still have stake.

Section 2.2.6 Avoid Hard Transition As described in Section 3.8, we will have a smooth transition from Byron to Shelley, with the core nodes gradually transferring the right and obligation to sign blocks to stake pools.

Section 2.2.7 Change Delegation Without Payment Key Delegation of cold wallets is described in Section 4.5, and does not require having the payment key of the cold wallet online.

Section 2.4.1 Master Recovery Key Wallet recovery is described in Section 3.13, and does not require any information in addition to the master key.

Section 2.4.2 Address Recognition Wallets will recognise addresses belonging to it by looking at the payment key hash part of the address, as described in Section 3.2.6.

Section 2.4.3 Wallet should be Runnable on Independent Devices With the caveats listed in that requirement, nothing in this document requires wallets running on different devices to share state.

Section 2.4.4 Maintain Privacy Having an efficient delegation mechanism – and in particular a mechanism where delegation is rewarded – requires a slight compromise on the level of pseudonymity, since addresses using the same stake address will be linkable. However, users can decide to use a number of different accounts, with separate stake addresses, if they are willing to pay the fees for using multiple stake addresses. This will give them a level of pseudonymity that is not worse than that in the Ethereum network.

They can also choose to use a distinct stake address per value address, or value addresses with no stake address at all, which gives pseudonymity that is no worse than in Bitcoin.

Section 2.4.5 Short Addresses The goal of having reasonably short addresses has guided the design of delegation, and we do not see an obvious way of making them even shorter, while still satisfying the rest of the requirements.

A Assessment of Rewards Sharing Mechanisms

This appendix gives an overview over the different mechanisms for rewards sharing that we took into consideration. While this is not needed for implementing the delegation system, the information is still useful enough to be included in this document. Choosing a mechanism for rewards sharing involves a number of non-trivial trade-offs, and future systems might want to pick one that we discarded for Cardano.

A.1 General Considerations

1. We use HD Wallets to provide some level of anonymity to stakeholders. We would not like to abandon this anonymity for the ability to share rewards.

   - To preserve this level of anonymity HD wallet users will need to associate separate stake addresses with each HD wallet generated address.

2. We wish to avoid arbitrary growth in the UTxO (or any other globally replicated record, e.g. contents of epoch boundary blocks).

   - This is potentially at odds with the rewarding of all stakeholders at all epochs

3. We want to avoid creating dust (entries in the UTxO that are so small that including them in a transaction is not economical, since their balance is close to or even less than the increase in fees resulting from including another input).

- The systemic issue is that dust is likely to have an unbounded lifetime in the UTxO
- Transaction fee structure could be modified to remove the transaction cost constraint. The requirement on action by the receiver still remains.

4. The network has a finite capacity to process transactions. We should avoid using a significant fraction of this capacity for sharing rewards. In particular, we want to avoid causing unreasonable spikes in the transaction rate. Those could either bring the system down on their own, or act as an invitation to a timed DoS attack.

5. The stake pool operator should not be required to take an action to initiate sharing rewards with members.

6. Verifying that a reward is legitimate will require a node to access some information (like the leader schedule of the epoch in which the reward was earned, as well as the delegation pattern at the time the leader election for that epoch took place). The time and space complexity for this should be constant in the size of the blockchain and/or the UTxO of non-reward entries.

Unless we want to give up on anonymity (1.), each address has to separately receive rewards. Together with 2., 3., and 4., this severely restricts any approach that distributes rewards using ordinary transactions.

A.1.1 Hierarchy of desirability of reward distribution

- Reward stakeholders on the basis of their holding at an epoch boundary
  - Stakeholders are not explicitly represented - there can be a proxy
  - One representation of stake delegation (direct to stake pool) which has the property of anonymity-via-aggregation. This, combined with the desire to not require stake pools to do the distribution a UTxO centric reward distribution mechanism.
- Reward stakeholders that maintain a UTxO/stake over the total epoch length.
  - This may be seen a "regressive" property in that it would not reward those stakeholders who engage in high-velocity value movements (e.g make use of the HD wallet).
  - This is a property of certain solutions.

A.1.2 Summary of key points of when rewards are calculated

- Point in Time
  - Just considers addresses at an epoch boundary
- Duration in Time
  - Set of stakeholder address and pool arrangement is fixed at an epoch boundary (say epoch $N-1$ to epoch N)
  - Rewards are calculated at the transition from epoch N to epoch $N+1$
  - Only stakeholder addresses that have non-zero associated value at the epoch N to $N+1$ boundary (i.e have value at both the epoch $N-1$ to N and the epoch N to $N+1$ boundaries) will be eligible to receive rewards
    * Noting that this could interact badly with HD wallet users

A.2 Approaches that are Ruled Out

A.2.1 Manual Sharing

In this approach, only stake pool operators are rewarded directly by the system, and it is their responsibility to share rewards with members of the pool.

This approach has been ruled out, since it:

1. requires additional trust in stake pool operators to do this correctly (5.)
2. requires at least stake pool operators to group the addresses of each member, to keep the volume of transactions somewhat reasonable (1., 2., 3., and 4.)
3. The rewards for members that did not contribute much stake are likely to be dust (3.)

A.2.2 Automatically Issue Transactions Each Epoch

In this approach, the system automatically distributes rewards at the end of an epoch, by sending transactions with outputs to every address that delegated to a stake pool that produced at least one block during that epoch.

This approach has been ruled out, since it:

1. Leads to a super-linear growth of the UTxO, creating an output per address per epoch (2.)
2. Is likely to create lots of dust for small stakeholders (3.)
3. Will lead to a huge burst of transactions, proportional to the number of addresses with non-zero balance in the system (4.). This could be lessened somewhat by sending the transactions over the course of the following epoch, but it would still use up a large fraction of the system's ability to process transactions (4.)

A.2.2.1 Complexity

- Creates one "UTxO" per non-zero address at the boundary/duration - this would create (today) ˜650k transactions per epoch

A.2.3 Let Members Collect Rewards

An alternative is to let every stake pool member be responsible for collecting their own rewards. This approach has the virtue that members could wait several epochs until they had accumulated enough rewards to warrant a transaction. The overall rate of transactions for sharing rewards would be reduced, the transactions would not come in bursts, and the problem of creating dust could be avoided.

However, this approach has been ruled out, since it:

1. Requires nodes to cache or quickly retrieve the whole history of leader schedules, as well as the delegation configurations at the time of each leader selection (6.)

A.3 Feasible Approaches

A.3.1 Automatic UTxO Updates

This unique approach circumvents the problems of transaction rates, dust entries, and UTxO growth, at the expense of introducing an implicit modification of the UTxO set.

After an epoch, each UTxO entry that delegated to a stake pool will have its balance updated to reflect the rewards that it earned. Since the update can be derived from information that every node has (leader schedule and delegation pattern at the last election), it can be carried out by each node individually.

Sadly, this approach does come with its own drawbacks:

1. It is not yet clear how a lightweight wallet would determine the correct UTxO set.
2. It introduces an implicit update of each UTxO entry, a huge moving part that makes it much harder to reason about the system.
3. Transactions that are formed before an update, but included after it, will have a larger total input than the issuer anticipated.
4. (Public Perception) This may be perceived as subverting the notion of immutability of the blockchain (at least in its UTxO model)

A.3.2 Lotteries per Stake Pool

A variation of "Automatically Issue Transactions Each Epoch", this approach avoids dust and creating a huge number of transactions by performing one lottery per stake pool. A number of winning addresses is determined, and the rewards are distributed amongst those addresses. The probability of any address winning the lottery is proportional to the stake that that address contributed to the pool. Benefits of this approach are:

1. The number of transactions will be proportional to the number of stake pools that signed at least one block, which is nicely bounded by the number of slots in an epoch.
2. The chances of creating dust entries is fairly low, since each winning address will receive a sizeable fraction of the pools rewards.
3. There is no need to group addresses per stake pool member.
4. Possibly – this would have to be investigated by legal – this could make ada less like a security.

The remaining drawbacks are:

1. It will still create a burst of transactions. This could be prevented by staggering the transactions that share rewards
2. An individual stake pool member will on average receive the same rewards as with any of the other approaches, but it will be much less predictable. This might be problematic from a Public Perception perspective.
3. (Public Perception) although (in the limit) this is the same outcome as sharing, apparently most humans don't see things that way[23] – they would prefer known outputs (even if smaller) to unknown ones. An additional indicator of human response might be to look at a similar mechanism (random rewards for depositing a fixed stake) has run since 1956. Premium Bonds[24] – computer nerds / crypto nuts should note who helped create the original ERNIE). The public might like the gambling aspect, businesses might not!

[23] See Prospect Theory (https://en.wikipedia.org/wiki/Prospect_theory)

[24] https://en.wikipedia.org/wiki/Premium_Bond

A.3.3 Reward accounts per stake address

This is in some sense a variation of the "Automatic UTxO updates", but trying to address its shortcomings.

Introoduce the notion of a reward account, tied to a stake address. Reward accounts have special rules:

- Account style accumulation, not UTxO style
- Paid into only by reward payout mechanism, never by normal Txs.
- Withdrawn from by normal Txs, using a witness for the stake address.

At the end of an epoch once the pool rewards are known, identify all the stake addresses that contribute to a pool and the rewards per stake address. The system implicitly issues a transaction/state-change to pay out rewards to the reward account of each stake address. These rewards accumulate if they are not withdrawn.

Value held in a reward account contributes to stake that is delegated to a stake pool and hence itself attracts rewards. This reduces the incentive to withdraw early and means the stake corresponding to the reward is not effectively offline.

Withdrawal of rewards is done similarly to the withdrawal transaction from the Chimeric Ledgers paper. This uses a witness for the stake address. Note that we also require at least one UTxO input to the transaction for replay protection (see Section 3.3.2, Section 3.9.1).

This aggregation of rewards – account style – is the key to resolving the UTxO storage asymptotic complexity problem. It is the same fundamental approach as the "Automatic UTxO updates" approach, but putting the aggregation off to a separate class of addresses, so that normal addresses remain in a pure UTxO style.

The asymptotic storage complexity of the ledger state (i.e. UTxO size) is linear in the number of stake addresses, but is unrelated to the number of epochs that have passed. This is in contrast to approaches that create UTxO entries for rewards on every epoch.

An important constraint for this approach is that it relies on stake addresses belonging to stakeholders. This means every stakeholder's value address must be associated with some stake address belonging to the stakeholder. This means it is not possible to use addresses that point directly to a stake pool and still be able to have a corresponding reward address, since there is not stake address to use for that reward address. There are alternatives to using addresses that point directly to pools[25], but these either reduce privacy or increase fees. One alternative that reduces privacy is for all addresses in a wallet to share the same stake address. This reduces privacy since all addresses in the wallet can be tied together by using the same stake address. Another alternative is to use a separate stake address for every value address. This means using one delegation certificate per (value) address. This increases the fees for creating addresses in a wallet following this policy, and for changing delegation choices. In principle there's a sliding scale between the two previous options, using a number of stake addresses, more than one but fewer than the number of value addresses.

- stake in reward accounts is ordinary stake, and hence is counted in delegation to stake pools.
- There is a potential interaction with UTxO deposit/refund approach. It may be that (because the refund is smaller than the reward) that negative values need to be stored. Though this may be able to done by some registration cost.

Advantages:

[25] During an early stage of the design, we had anticipated pointer addresses that would refer directly to a pool registration certificate.

- doesn't "mutate" the UTxO. This reduces conceptual and implementation complexity. It does not disrupt wallets and other things based on the UTxO model.

Disadvantages:

- introduces limited degree of account style chimeric ledgers. This adds a degree of conceptual and implementation complexity.
- Cannot use pointer addresses directly to stake pools. Increases fee and complexity cost of maintaining wallet privacy.
- When people use multiple stake addresses to retain some privacy, it gets somewhat less efficient at limiting the size of the required state. But it will still prevent the exponential growth of the UTxO.

B Deposits

B.1 Motivation

One fundamental raison-d'être for transaction fees in Cardano (or any other cryptocurrency for that matter) is to compensate node operators for their costs: Processing a transaction incurs costs, and the person doing the processing should be reimbursed accordingly.

In reality however, there are more than just one-time processing costs. In particular, there are long term *storage* costs whenever a transaction forces a node to dedicate local storage for the stake associated with the transaction.

The prototypical example for this are *UTxO-entries*: Each additional such entry takes up storage on each node running the protocol. There are other examples as well, including *stake pool registrations* and *delegation certificates*.

We plan to address this issue by requiring a *deposit* to be paid for each resource that will incur storage costs.

This deposit must be (partially) *refundable*, so that the holder of the resource has an incentive to release the resource when it is no longer needed. So for example, somebody with a lot of "dust" in their wallet would have an incentive to remove that dust, thus reclaiming some of the deposit paid for UTxO-entries.

On the other hand, refunds should also *decrease over time*, so that there is an incentive to release a resource sooner rather than later.

B.2 Mechanism

We propose to introduce the following configurable parameters:

1. A deposit amount (in ada) $d_R \in (0, \infty)$ for each type of resource R. The value of d_R for a resource type R should roughly reflect the cost to "rent the resource forever".
2. A factor $d_{\min} \in (0, 1)$, which determines the minimal proportion of d_R that will be refunded on resource release. Higher value of $d_{\min}$ mean higher guaranteed refunds.
3. A decay constant $\lambda \in (0, \infty)$ determining how refunds decrease over time. Higher values of λ correspond to faster decrease of refunds over time.

Given these parameters, on acquiring a resource of type R, one would have to pay an amount of d_R ada. When the resource is released after $t \geq 1$ slots, the holder of the resource is refunded

$$r_R(t) = d_R \cdot \left(d_{\min} + (1 - d_{\min}) \cdot e^{-\lambda t}\right) \in (d_{\min} \cdot d_R, d_R),$$

whereas the difference $d_R - r_R(t)$ is added incrementally to the reward pools of the epochs between registering and releasing the resource.

Note that it easily follows from well-known properties of the exponential function that

$$d_r > r_R(t) \xrightarrow{t\to\infty} d_{\min} \cdot d_R,$$

as desired.

As a fictional example, consider parameter values $d_{\min} = 0.25$ and $\lambda = 0.0001$ and a resource of type R with $d_R = 2$. A user acquiring such a resource will initially include a deposit of $d_R = 2$ ada in the transaction creating that resource. This deposit will be held in escrow until the resource gets released. If the user releases the resource after 10,000 slots, a refund of $r_R(10,000) = 1.0518$ ada will be added to the available input of the associated transaction. The difference $d_R - r_R(10,000) = 2 - 1.0518 = 0.9482$ ada will be added to the rewards pool of that epoch.

If our fictional user held onto the resource for 40,000 slots instead, their refund would only be $r_R(40,000) = 0.5275$ ada, and 1.4725 ada would be added to the epoch rewards. In this example, refunds will never drop below $d_{\min} \cdot d_R = 0.5$ ada.

C Design Option: Stale Stake

This section sketches an optional mechanism for tracking *stale stake*, i.e., stake that is no longer being actively controlled. Stale stake can limit the chain growth (since elected leaders might fail to show up and sign blocks), and decrease the amount of honest stake, making the system easier to attack. The mechanism described below is aimed at mitigating the first effect.

In the current design, stale stake is much less likely to become a problem than in earlier iterations (since we automatically discard stake that is not delegated to a valid stake pool), so we propose to not implement this design option, at least not in the initial Shelley release. We keep it in this document for further reference.

In the current design, the only circumstance where an actor becoming inactive would limit the chain growth is when a stake pool operator ceases to operate their pool, without retiring it. Furthermore, since a failure to produce blocks will reduce the rewards for stake pool members, such a pool would lose members and become irrelevant. Thus, an abandoned pool will be an impediment to chain growth only if there are stakeholders delegating to that pool who also become inactive and do not re-delegate.

In order to further mitigate this potential problem, the system could monitor the apparent performance of all pools over time, and prune pools that fulfill both of the following conditions:

- The apparent performance of the pool has consistently been zero for a certain number of epochs (i.e., the block did not produce any blocks after a certain moment in time).
- The pool has enough stake that it should have been elected as slot leader within those epochs several times.

The number of epochs and size in stake should be such that we can rule out the hypothesis that the pool is still active, but was just not elected as leader during those epochs, with statistical significance[26].

We do not anticipate that abandoned pools should become a problem anytime soon. By monitoring the chain growth, we could detect whenever a significant fraction of pools accumulates in abandoned pools, and implement abandoned pool detection when necessary.

[26]Setting those numbers would require some research if we were to implement this feature.

D FAQ

D.1 Why will stake pools accept new stake pool registration certificates?

It may seem counterintuitive that any of the existing stake pools would accept a stake pool registration certificate for a new pool, for fear of losing some of their future rewards to the increased competition. After all, with a naive approach to rewarding pools, a new pool would potentially reduce the rewards of every existing pool. Existing systems like Bitcoin tend to becoming more centralised because they use naive incentives.

One thing to realise is that Cardano uses a sophisticated incentives scheme [BKKS18], summarised in Section 5, where the system tends to a fixed number k of saturated stake pools, and no pool can increase their own rewards by trying to reduce the number of active pools below k. So there is no general incentive, that would cause every stake pool to try to censor the registration of a new pool.

The operator of a pool that is near the bottom of the list of competitive pools might fear to be replaced by a new pool, and it would not be unreasonable for that operator to try to prevent new pools from registering. But since it only takes a *single pool* to include the certificate[27],there is no hope to achieve this, and the rational behaviour is to just play by the rules and include the certificate.

The situation where each pool accepts submitted certificates is a *Nash Equilibrium*, where no player can benefit from deviating from this behaviour. Such configurations are stable, since getting to a different state requires either collusion between a large number of players, or players acting irrationally against their own interests.

However, there is a subtlety here: the state where *no* pool accepts new certificates might also be a Nash Equilibrium: in this case, pools may refrain from entering a new certificate for fear to lose rewards due to increased competition, which will either kick them out of the k best pools or lower their margins.

Let us call the former equilibrium, where certificates are accepted, NE1, and the latter, where they are rejected, NE2. Should we be worried that the system will end up in NE2? There are three arguments why this is unlikely to happen:

- When the system is initially decentralised, a majority of blocks will be created by the federation that ran the Byron network, and those players will behave honestly. So the system will start in NE1.
- Stake pool *members* benefit from competition, and while censorship of certificates is not observable from the final chain itself, the community can be expected to identify pools that try to block the competition, by looking at the certificates that are being broadcast, the produced blocks, and temporary forks. Once this becomes known, members will leave such a pool. So there is a high risk involved for stake pools.
- Last but not least, the project is run by a community, and it is not unreasonable to expect members to be at least somewhat cooperative.

D.2 Won't stake pools reject delegation certificates that delegate away from them?

That would only work if a majority of stake pools colluded to censor such a certificate. But all pools are incentivised to include the certificate, via fees. So this censorship would only happen

[27] To be precise, this also requires that a majority of players is going to accept a block that contains a certificate. But dropping a block because it contains a certificate is much worse than just not including the certificate in a block: it creates a fork and thereby attacks the integrity of the system, and a pool doing that risks losing their own block when the fork is resolved. Also, a pool that repeatedly creates a fork after a block that contains a stake pool registration certificate would sooner or later be detected and blamed by the community

if a majority of pools decided to partake in malicious behaviour and attack the system, against their direct incentives.

E Transaction Metadata

Adding metadata to transactions is a useful new feature in Cardano Shelley. It is not related to delegation or decentralisation.

E.1 Motivation and design goals

The purpose is to enable a range of new applications by allowing arbitrary structured data to be included onto the chain, and to make effective use of that data. The term 'metadata' is perhaps a misnomer since it is simply about placing application specific data on the chain; it is only metadata from the point of view of a transaction since it is carried along with transactions and not involved in validation.

A design goal is to add very little complexity to the on-chain part of the system but to get (or allow for) as much functionality as possible, in combination with other features or components. This helps keep implementation complexity lower. Importantly it keeps the size of the trusted base low, by having the complex functionality to use the metadata outside of the trusted base.

A design principle that we preserve is that the historical data on the chain is not needed to validate the next block or transaction. All data needed for later validation must be explicitly tracked in the ledger state. This means the old part of the chain does not need to be preserved locally at all, or at least not in random access storage. This avoids a problem that Ethereum ran into with disk I/O becoming a performance bottleneck. This is why the design does not include metadata into the ledger state, and does not make it accessible to later scripts.

E.2 Detail

The transaction can contain metadata. The metadata hash is part of the body of the transaction so is covered by all transaction signatures. This allows for integrity checking and authentication of the metadata.

The metadata value is kept outside of the transaction body, much like the transaction witnesses. This follows the 'segmented witness' design idea that allows witnesses to be discarded when the data is known to have been checked. We go one step further and keep the metadata outside the body separately from the witnesses too, in principle allowing an implementation to store or discard the metadata or the witnesses independently of each other.

The structure of the metadata is a mapping from keys to values. The keys are unsigned integers limited in size up to 64 bits. The values are simple structured terms, consisting of integers, text strings, byte strings, lists and maps.

There is no limit on the number of key-value pairs, except that imposed by the overall transaction size limit. There is also no limit on individual structured values, but there is a limit on the size of text strings and byte strings within the structured values.

A key aspect of the design is that metadata included in transactions is not available for later retrieval from within the ledger validation rules, including scripts. The metadata is not entered into the ledger state, and general historical chain data is not otherwise available to the ledger validation rules.

The changes to the ledger validation rules are thus very limited: only the metadata syntax, metadata size limits and the effect of the metadata on the transaction size calculation and thus the transaction fees. No data is added to the ledger state. The metadata resides only on the chain.

There are no special fees for metadata. The metadata simply contributes to the size of the transaction and fees are based on the transaction size. This choice is justified by the fact that the cost to operators is only the one-time processing cost and any long term storage of the blockchain. There is no long term random access state.

The metadata within a transaction will be made available to validation scripts, including Plutus Core scripts. Note again that this is only the immediate transaction being validated. No metadata from predecessor transactions is available.

E.3 Explanation and use

The purpose of the metadata being a key value mapping is to make it straightforward to combine metadata for multiple purposes into the same transaction. Think of the metadata key as being a schema identifier, that says what the metadata value is. There is however no on-chain schema enforcement. The interpretation of the data is entirely up to the applications that consume it.

It may make sense to establish a public registry of known metadata keys and corresponding schemas.

The metadata value is required to be structured data rather than a single unstructured blob. The available structure is like a simplified version of JSON. This makes the data easier to inspect and manipulate, particularly by scripts, such as Plutus scripts, in future evolutions of the system. The metadata values do not include floating point numbers because on-chain script languages cannot support such types.

The size of strings in the structured value is limited to mitigate the problem of unpleasant or illegal content being posted to the blockchain. It does not prevent this problem entirely, but it means that it is not as simple as posting large binary blobs.

Of course posting data to the chain is only half the story. It must also be possible to use it effectively. Part of the design is that the data is not kept in random-access storage for use by on-chain scripts, so that validating the chain does not require random access to old parts of the chain or large databases. So the design calls for metadata use to be managed off-chain using an indexing service.

An indexing service, much like an explorer, enables the collection, authentication and query of the metadata that is posted on the chain. It is clear that an agent can follow the chain and write all transaction metadata into a relational database for later query. This is the design that the backends for many blockchain explorers use. This solves the collection and query parts of the problem, but not the authentication part.

Using HD wallet schemes however, the authenticity of the metadata can be ensured. Depending on the HD scheme – using public or non-public key derivation – the metadata can be publicly verifiable, or only privately verifiable.

For example, a simple scheme to track the issuance of physical items could involve the original owner posting metadata within transactions that spend from a designated wallet. An indexing server that knows the HD wallet structure (and either public or private keys depending on the HD scheme) can track the wallet and index all the metadata in transactions from that wallet (or wallet sub-account).

Such schemes have a great deal of flexibility since there is a lot of flexibility in HD wallet schemes. With public HD derivation, the indexing server does not need any signing keys, just an appropriate verification key of a sub-tree in the HD wallet space. If that verification key is revealed then anyone can reliably run the indexing service, and anyone can verify that the metadata is authentic. If the verification key is not revealed then only the owner can run the indexing service, and be used to implement some lookup or verification service, or it can reveal the authenticity of a particular address without revealing all addresses.

It is even possible in principle to use multi-signature wallets, or wallets involving scripts. There just needs to be some wallet scheme that the indexing service can use to reliably track and authenticate the transactions using the wallet.

Obviously, to take advantage of these possibilities requires suitable wallet and indexing components. These are however independent components and their complexity does not impact the complexity of the on-chain rules, so does not add to the size of the trusted base of the overall system.

E.4 Binary schema

The binary schema is very simple. The notation is CBOR CDDL (much like BNF).

```
metadata = { * metadata_key => metadata_value }

metadata_key = uint

metadata_value =
    int
  / bytes .size 64
  / text  .size 64
  / [ * metadata_value ]
  / { * metadata_value => metadata_value }
```

References

[ACG14] M. Araoz, R. X. Charles, and M. A. Garcia. Structure for deterministic p2sh multisignature wallets, April 2014. URL `https://github.com/bitcoin/bips/blob/master/bip-0045.mediawiki`. BIP-45.

[BC20] A. Byaly and J. Corduan. Stake pool ranking in cardano, 2020. URL `https://github.com/input-output-hk/cardano-ledger/releases/latest/download/pool-ranking.pdf`.

[BKKS18] L. Bruenjes, A. Kiayias, E. Koutsoupias, and A.-P. Stouka. Reward sharing schemes for stake pools. Computer Science and Game Theory (cs.GT) arXiv:1807.11218, 2018.

[CG19] J. Corduan and M. Güdemann. A formal specification of a multi-signature scheme using scripts, 2019.

[DGKR17] B. M. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. *IACR Cryptology ePrint Archive*, 2017:573, 2017.

[KRDO17] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology – CRYPTO 2017*, volume 10401 of *Security and Cryptology*. Springer International Publishing, 2017. doi:10.1007/978-3-319-63688-7.

[MMM01] T. Malkin, D. Micciancio, and S. Miner. Composition and efficiency tradeoffs for forward-secure digital signatures. Cryptology ePrint Archive, Report 2001/034, 2001. `https://eprint.iacr.org/2001/034`.

[Wui12] P. Wuille. Hierarchical deterministic wallets, February 2012. URL `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`. BIP-32.