

- If the current reward update ru is empty and s is greater than the sum of the first slot of its epoch and the duration RandomnessStabilisationWindow, then a new rewards update is calculated and the state is updated. (Note the errata in Section 17.3.)
- If the current reward update ru is not \diamond , i.e., a reward update has already been calculated but not yet applied, then the state is not updated.
- If the current reward update ru is empty and s is less than or equal to the sum of the first slot of its epoch and the duration to start rewards RandomnessStabilisationWindow, then the state is not updated.

$$\begin{array}{c}
 s > \text{firstSlot}(\text{epoch } s) + \text{RandomnessStabilisationWindow} \quad ru = \diamond \\
 \text{Create-Reward-Update} \frac{ru' := \text{createRUPd SlotsPerEpoch } b \text{ es MaxLovelaceSupply}}{b \text{ es} \vdash ru \xrightarrow[\text{RUPD}]{s} ru'}
 \end{array} \quad (34)$$

$$\begin{array}{c}
 \text{Reward-Update-Exists} \frac{ru \neq \diamond}{b \text{ es} \vdash ru \xrightarrow[\text{RUPD}]{s} ru}
 \end{array} \quad (35)$$

$$\begin{array}{c}
 ru = \diamond \\
 \text{Reward-Too-Early} \frac{s \leq \text{firstSlot}(\text{epoch } s) + \text{RandomnessStabilisationWindow}}{b \text{ es} \vdash ru \xrightarrow[\text{RUPD}]{s} ru}
 \end{array} \quad (36)$$

Figure 62: Reward Update rules

12.7 Chain Tick Transition

The Chain Tick Transition performs some chain level upkeep. The environment consists of a set of genesis keys, and the state is the epoch specific state necessary for the NEWEPOCH transition.

Part of the upkeep is updating the genesis key delegation mapping according to the future delegation mapping. For each genesis key, we adopt the most recent delegation in $fGenDelegs$ that is past the current slot, and any future genesis key delegations past the current slot is removed. The helper function `adoptGenesisDelegs` accomplishes the update.

The TICK transition rule is shown in Figure 64. The signal is a slot s .

Three transitions are done:

- The NEWEPOCH transition performs any state change needed if it is the first block of a new epoch.
- The RUPD creates the reward update if it is late enough in the epoch. **Note** that for every block header, either NEWEPOCH or RUPD will be the identity transition, and so, for instance, it does not matter if RUPD uses nes or nes' to obtain the needed state.

Chain Tick Transitions

$$\vdash \frac{}{\text{TICK}} \vdash \subseteq \mathbb{P} (\text{NewEpochState} \times \text{Slot} \times \text{NewEpochState})$$

helper function

$$\text{adoptGenesisDelegs} \in \text{EpochState} \rightarrow \text{Slot} \rightarrow \text{EpochState}$$

$$\text{adoptGenesisDelegs } es \text{ slot} = es'$$
where

$$(acnt, ss, (us, (ds, ps)), prevPp, pp) := es$$

$$(rewards, delegations, ptrs, fGenDelegs, genDelegs, i_{rwd}) := ds$$

$$curr := \{(s, gkh) \mapsto (vkh, vrf) \in fGenDelegs \mid s \leq slot\}$$

$$fGenDelegs' := fGenDelegs \setminus curr$$

$$genDelegs' := \left\{ gkh \mapsto (vkh, vrf) \mid \begin{array}{l} (s, gkh) \mapsto (vkh, vrf) \in curr \\ s = \max\{s' \mid (s', gkh) \in \text{dom } curr\} \end{array} \right\}$$

$$ds' := (rewards, delegations, ptrs, fGenDelegs', genDelegs \sqcup genDelegs', i_{rwd})$$

$$es' := (acnt, ss, (us, (ds', ps)), prevPp, pp)$$
Figure 63: Tick transition-system types

$$\begin{array}{c}
 \vdash nes \xrightarrow[\text{NEW EPOCH}]{\text{epoch slot}} nes' \\
 (_, b_{prev}, _, es, _, _) := nes \\
 \begin{array}{c} b_{prev} \\ es \end{array} \vdash ru' \xrightarrow[\text{RUPD}]{\text{slot}} ru'' \\
 (e'_\ell, b'_{prev}, b'_{cur}, es', ru', pd') := nes' \\
 es'' := \text{adoptGenesisDelegs } es' \text{ slot} \\
 nes'' := (e'_\ell, b'_{prev}, b'_{cur}, es'', ru'', pd') \\
 \text{Tick} \frac{}{} \vdash nes \xrightarrow[\text{TICK}]{\text{slot}} nes''
 \end{array} \tag{37}$$

Figure 64: Tick rules

Operational Certificate environments

$$\text{OCertEnv} = \left(\begin{array}{ll} \text{stools} \in \mathbb{P} \text{ KeyHash} & \text{stake pools} \\ \text{genDelegs} \in \mathbb{P} \text{ KeyHash} & \text{genesis key delegates} \end{array} \right)$$

Operational Certificate Transitions

$$_ \vdash _ \xrightarrow{\text{OCERT}} _ \subseteq \mathbb{P} (\text{OCertEnv} \times \text{KeyHash}_{\text{pool}} \mapsto \mathbb{N} \times \text{BHeader} \times \text{KeyHash}_{\text{pool}} \mapsto \mathbb{N})$$

Operational Certificate helper function

$$\begin{aligned} \text{currentIssueNo} \in \text{OCertEnv} &\rightarrow (\text{KeyHash}_{\text{pool}} \mapsto \mathbb{N}) \rightarrow \text{KeyHash}_{\text{pool}} \rightarrow \mathbb{N}^? \\ \text{currentIssueNo} (\text{stools}, \text{genDelegs}) \text{ cs } hk &= \begin{cases} hk \mapsto n \in \text{cs} & n \\ hk \in \text{stools} & 0 \\ hk \in \text{genDelegs} & 0 \\ \text{otherwise} & \diamond \end{cases} \end{aligned}$$

Figure 65: OCert transition-system types

12.8 Operational Certificate Transition

The Operational Certificate Transition environment consists of the genesis key delegation map *genDelegs* and the set of stake pools *stools*. Its state is the mapping of operation certificate issue numbers. Its signal is a block header.

The transition rule is shown in Figure 66. From the block header body *bhb* we first extract the following:

- The operational certificate, consisting of the hot key vk_{hot} , the certificate issue number n , the KES period start c_0 and the cold key signature.
- The cold key vk_{cold} .
- The slot s for the block.
- The number of KES periods that have elapsed since the start period on the certificate.

Using this we verify the preconditions of the operational certificate state transition which are the following:

- The KES period of the slot in the block header body must be greater than or equal to the start value c_0 listed in the operational certificate, and less than MaxKESEvo -many KES periods after c_0 . The value of MaxKESEvo is the agreed-upon lifetime of an operational certificate, see [SL-D1].
- hk exists as key in the mapping of certificate issues numbers to a KES period m and that period is less than or equal to n .
- The signature τ can be verified with the cold verification key vk_{cold} .
- The KES signature σ can be verified with the hot verification key vk_{hot} .

After this, the transition system updates the operational certificate state by updating the mapping of operational certificates where it overwrites the entry of the key hk with the KES period n .

The OCERT rule has six predicate failures:

$$\begin{array}{c}
(bhb, \sigma) := bh \quad (vk_{hot}, n, c_0, \tau) := bocert\ bhb \quad vk_{cold} := bvk_{cold}\ bhb \\
hk := hashKey\ vk_{cold} \quad s := bslot\ bhb \quad t := kesPeriod\ s - c_0 \\
\\
c_0 \leq kesPeriod\ s < c_0 + MaxKESEvo \\
currentIssueNo\ oce\ cs\ hk = m \quad m \leq n \\
\\
OCert \frac{\mathcal{V}_{vk_{cold}}[(vk_{hot}, n, c_0)]_\tau \quad \mathcal{V}_{vk_{hot}}^{KES}[bhb]_\sigma^t}{oce \vdash cs \xrightarrow[OCERT]{bh} cs \sqcup \{hk \mapsto n\}} \quad (38)
\end{array}$$

Figure 66: OCert rules

- If the KES period is less than the KES period start in the certificate, there is a *KESBeforeStart* failure.
- If the KES period is greater than or equal to the KES period end (start + MaxKESEvo) in the certificate, there is a *KESAAfterEnd* failure.
- If the period counter in the original key hash counter mapping is larger than the period number in the certificate, there is a *CounterTooSmall* failure.
- If the signature of the hot key, KES period number and period start is incorrect, there is an *InvalidSignature* failure.
- If the KES signature using the hot key of the block header body is incorrect, there is an *InvalideKesSignature* failure.
- If there is no entry in the key hash to counter mapping for the cold key, there is a *NoCounterForKeyHash* failure.

12.9 Verifiable Random Function

In this section we define a function `vrfChecks` which performs all the VRF related checks on a given block header body. In addition to the block header body, the function requires the epoch nonce, the stake distribution (aggregated by pool), and the active slots coefficient from the protocol parameters. The function checks:

- The validity of the proofs for the leader value and the new nonce.
- The verification key is associated with relative stake σ in the stake distribution.
- The bleader value of *bhb* indicates a possible leader for this slot. The function `checkLeaderVal` is defined in 16.

VRF helper function

$$\begin{aligned} \text{issuerIDfromBHBody} &\in \text{BHBody} \rightarrow \text{KeyHash}_{\text{pool}} \\ \text{issuerIDfromBHBody} &= \text{hashKey} \circ \text{bvkcold} \end{aligned}$$

$$\begin{aligned} \text{mkSeed} &\in \text{Seed} \rightarrow \text{Slot} \rightarrow \text{Seed} \rightarrow \text{Seed} \\ \text{mkSeed } \text{ucNonce } \text{slot } \eta_0 &= \text{ucNonce} \text{ XOR } (\text{slotToSeed } \text{slot} \star \eta_0) \end{aligned}$$

$$\text{vrfChecks} \in \text{Seed} \rightarrow \text{BHBody} \rightarrow \text{Bool}$$

$$\text{vrfChecks } \eta_0 \text{ bhb} =$$

$$\begin{aligned} &\text{verifyVrf}_{\text{Seed}} \text{ vrfK } (\text{mkSeed } \text{Seed}_\eta \text{ slot } \eta_0) (\text{bprf}_\eta \text{ bhb}, \text{bnonce } \text{bhb}) \\ \wedge &\text{verifyVrf}_{[0,1]} \text{ vrfK } (\text{mkSeed } \text{Seed}_\ell \text{ slot } \eta_0) (\text{bprf}_\ell \text{ bhb}, \text{bleader } \text{bhb}) \end{aligned}$$

where

$$\text{slot} := \text{bslot } \text{bhb}$$

$$\text{vrfK} := \text{bvkvrf } \text{bhb}$$

$$\text{praosVrfChecks} \in \text{Seed} \rightarrow \text{PoolDistr} \rightarrow (0, 1] \rightarrow \text{BHBody} \rightarrow \text{Bool}$$

$$\text{praosVrfChecks } \eta_0 \text{ pd } f \text{ bhb} =$$

$$\begin{aligned} &hk \mapsto (\sigma, \text{vrfHK}) \in \text{pd} \\ \wedge &\text{vrfHK} = \text{hashKey } \text{vrfK} \\ \wedge &\text{vrfChecks } \eta_0 \text{ bhb} \\ \wedge &\text{checkLeaderVal } (\text{bleader } \text{bhb}) \sigma f \end{aligned}$$

where

$$hk := \text{issuerIDfromBHBody } \text{bhb}$$

$$\text{vrfK} := \text{bvkvrf } \text{bhb}$$

$$\text{pbftVrfChecks} \in \text{KeyHash}_{\text{vrf}} \rightarrow \text{Seed} \rightarrow \text{BHBody} \rightarrow \text{Bool}$$

$$\text{pbftVrfChecks } \text{vrfHK } \eta_0 \text{ bhb} =$$

$$\begin{aligned} &\text{vrfHK} = \text{hashKey } (\text{bvkvrf } \text{bhb}) \\ \wedge &\text{vrfChecks } \eta_0 \text{ bhb} \end{aligned}$$