THE RADICLE REGISTRY

VERSION 0.1

Abstract. What follows is a semi-formal description of the semantics of the Radicle Registry.

Contents

1. Transactions 2
2. Accounts 2
2.1. Transferring value 2
3. Orgs 3
3.1. Registering 3
3.2. Unregistering 3
3.3. Registering members 4
3.4. Unregistering members 4
3.5. The Contract 4
3.6. The Fund 5
4. Projects 5
4.1. Registering 5
4.2. Unregistering 6
4.3. Creating a checkpoint 6
4.4. Setting a checkpoint 7
5. User Identity 7
5.1. Registering 7
5.2. Unregistering 8
5.3. Associating an external key 8
5.4. Revoking an external key 8

by `Monadic`.

1. Transactions

All transactions on the registry take the form $\mathsf{transaction}(arg_1, \ldots, arg_n)_\sigma$, where $arg_1, \ldots arg_n$ are the *inputs* and σ is the EdDSA signature of the author of the transaction. Transactions always have an *author* and an *origin* (formally α), which is the author's account.

Transactions can be uniquely identified by their *hash*. The set of all *known* transactions is $\mathcal{T}$ (the "ledger"), and the set of all known transaction hashes is $\mathcal{T}_{\mathsf{hash}}$.

2. Accounts

An account A is a tuple:

$$A = \langle A_{\mathsf{id}}, A_{\mathsf{nonce}}, A_{\mathsf{bal}} \rangle$$

DEFINITION

- A_{id} is the unique account identifier obtained by hashing the account owner's public key,
- A_{nonce} is a number which starts at 0 and is incremented every time a transaction originates from this account.
- A_{bal} is the account's balance in the smallest denomination, and $A_{\mathsf{bal}} \in \mathbb{N}_{\geq 0}$.

The set of all accounts is $\mathcal{A}$. Accounts are never created or destroyed, rather, if they have never been used to transact, they have an initial state of:

$$A = \langle A_{\mathsf{id}}, 0, 0 \rangle$$

Hence, for all valid account ids, there exists an account with that id. In other words, $\forall a \in A_{\mathsf{id}} (A \in \mathcal{A})$.

Note that accounts can *never be removed*, since that would violate the invariant that nonces are only ever incremented, and removing an account is equivalent to setting A_{nonce} and A_{bal} to 0.

2.1. Transferring value.

The act of transferring coins between two accounts:

$$\mathsf{transfer}(A_{\mathsf{id}}, v)_\sigma$$

which will transfer value from the transaction origin α to account A.

INPUTS

- A_{id} is the account id of the *receiver* of the transfer,
- v is the value or 'balance' to transfer from the origin to the receiver, in the smallest denomination.

VALIDATION

- The transfer balance is positive, or $v \geq 1$,
- The origin's balance minus any transaction fee is $\geq v$.

OUTPUTS

- v is debited from the origin and credited to A.

3. Orgs

An org is a logical grouping of people and projects with common governance and funds. An org O is a tuple:

$$O = \langle O_{\mathsf{id}}, O_{\mathsf{account}}, O_{\mathsf{members}}, O_{\mathsf{projs}}, O_{\mathsf{contract}} \rangle$$

DEFINITION

- O_{id} is the globally unique org identifier,
- O_{account} is the org account or *fund*,
- O_{members} is the set of registered org members,
- O_{projs} is the set of registered projects under this org,
- O_{contract} is the org contract, which governs permissions around the org, as well as its fund. It can be described as a function:

  $$f : \mathcal{T} \to \{\top, \bot\}$$

  where $\mathcal{T}$ is any transaction t operating on an org, and $\top$ signifies t is *authorized* to execute by the contract, while $\bot$ means it is *unauthorized*. Note that a transaction can be verified and included in the transaction ledger $\mathcal{T}$ yet still be unauthorized to run by the contract

3.1. Registering.

$$\mathsf{register\text{-}org}(O_{\mathsf{id}}, O_{\mathsf{contract}})_\sigma$$

INPUTS

- O_{id} is the unique identifier being registered,
- O_{contract} is the initial org contract that includes the initial permission set around the org.

VALIDATION

- O_{id} must be unique, i.e. not currently in use, between 1 and 32 bytes long, and valid UTF-8.
- $\alpha_{\mathsf{bal}} \geq \mathcal{D}_{\mathsf{register\text{-}org}}$.

OUTPUTS

- $O \in \mathcal{O}$, where $O = \langle O_{\mathsf{id}}, O_{\mathsf{account}}, \{\alpha_{\mathsf{id}}\}, \varnothing, O_{\mathsf{contract}} \rangle$
- $O_{\mathsf{account}} \in \mathcal{A}$,
- $\alpha_{\mathsf{bal'}} = \alpha_{\mathsf{bal}} - \mathcal{D}_{\mathsf{register\text{-}org}}$.

3.2. Unregistering.

$$\mathsf{unregister\text{-}org}(O_{\mathsf{id}})_\sigma$$

INPUTS

- O_{id} is the identifier of the org being unregistered,

VALIDATION

- $O \in \mathcal{O}$,
- $O_{\mathsf{members}} = \{\alpha_{\mathsf{id}}\}$, the transaction origin must be the only member,
- $O_{\mathsf{projs}} = \varnothing$, there must be no projects under the org,

OUTPUTS

- $O \notin \mathcal{O}$,

– $\alpha_{\mathsf{bal}'} = \alpha_{\mathsf{bal}} + \mathcal{D}_{\mathsf{register\text{-}org}} + O_{\mathsf{account}_{\mathsf{bal}}}$.

3.3. Registering members.

$$\mathsf{register\text{-}member}(O_{\mathsf{id}}, A_{\mathsf{id}})_{\sigma}$$

INPUTS

– A_{id} is the account id being registered as a member,
– O_{id} is the id of the org under which to register A,

VALIDATION

– $O \in \mathcal{O}$,
– A_{id} must not already be registered under O, or $A_{\mathsf{id}} \notin O_{\mathsf{members}}$
– The transaction author is authorized to execute register-member,
– $\alpha_{bal} \geq \mathcal{D}_{\mathsf{register\text{-}member}}$.

OUTPUTS

– $A_{\mathsf{id}} \in O_{\mathsf{members}}$,
– $\alpha_{bal'} = \alpha_{bal} - \mathcal{D}_{\mathsf{register\text{-}member}}$.

3.4. Unregistering members.

$$\mathsf{unregister\text{-}member}(O_{\mathsf{id}}, A_{\mathsf{id}})_{\sigma}$$

INPUTS

– O_{id} is the id of the org under which the member is registered,
– A_{id} is the account id of the member being unregistered,

VALIDATION

– $O \in \mathcal{O}$,
– $A_{\mathsf{id}} \in O_{\mathsf{members}}$,
– The transaction author is authorized to execute unregister-member.

OUTPUTS

– $A_{\mathsf{id}} \notin O_{\mathsf{members}}$,
– $\alpha_{bal'} = \alpha_{bal} + \mathcal{D}_{\mathsf{register\text{-}member}}$.

3.5. **The Contract.** Every org O in the registry has a contract denoted O_{contract}. The way this contract is invoked is through transactions that act on O. For example, the fund transaction (§3.6) which transfers value out of a org is always validated by the org contract before it is authorized to execute.

A contract is made of a set of *rules* that each handle a specific action relating to the org. In the fund example, the fund *rule* would be invoked to determine the outcome of the transaction.

Setting the org's contract is done with:

$$\mathsf{set\text{-}contract}(O_{\mathsf{id}}, c)_{\sigma}$$

INPUTS

– O_{id} is the id of the org,
– c is the new contract.

VALIDATION

- $O \in \mathcal{O}$,
- The transaction author is authorized to execute $\mathsf{set\text{-}contract}$,

OUTPUTS

- $O_{\mathsf{contract}} = c$

3.6. **The Fund.** Each org has an associated account O_{account} called the *fund*. To use that account to fund maintenance of projects, the fund transaction is used:

$$\mathsf{fund}(O_{\mathsf{id}}, A_{\mathsf{id}}, v)_\sigma$$

INPUTS

- O_{id} is the id of the org from which the transfer should be initiated,
- A_{id} is the id of the account that should receive the transfer,
- v is the value to transfer.

VALIDATION

- $O_{\mathsf{account}_{\mathsf{bal}}} - \mathcal{D}_{\mathsf{register\text{-}org}} \geq v$,
- The transaction author is authorized to execute fund,

OUTPUTS

- $A_{\mathsf{bal}'} = A_{\mathsf{bal}} + v$
- $O_{\mathsf{account}_{\mathsf{bal}'}} = O_{\mathsf{account}_{\mathsf{bal}}} - v$

4. Projects

A project P is a tuple:

$$P = \langle P_{\mathsf{id}}, P_{\mathsf{org}}, P_{\mathsf{k}}, P_{\mathsf{proof}}, P_{\mathsf{meta}} \rangle$$

DEFINITION

- P_{id} is the unique project identifier within the context of an P_{org},
- P_{org} is the org under which this project lives,
- P_{k} is the current project *checkpoint* (See §4.3),
- P_{proof} is the proof that was supplied during proposal, verifying the owner's authority over the project,
- P_{meta} is opaque metadata to associate with P. For example, the RADICLE *project id*. Note that once defined, the metadata is immutable.

Projects are registered with the $\mathsf{register\text{-}project}$ transaction and unregistered with the $\mathsf{unregsiter\text{-}project}$ transaction. Projects always exist within the context of an org.

4.1. Registering.

$$\mathsf{register\text{-}project}(P_{\mathsf{org}}, P_{\mathsf{id}}, P_{\mathsf{k}}, P_{\mathsf{meta}})_\sigma$$

INPUTS

- P_{id} is the unique project id being requested,
- P_{org} is the id of the org under which to register P,
- P_{k} is the id of the initial *checkpoint* associated with this project, formally k_0. This checkpoint must always remain in the project ancestry,
- P_{meta} is associated project metadata.

VALIDATION

- P_{org} identifies an existing org O,
- P_{id} is unique under O, between 1 and 32 bytes long, and valid UTF-8.
- P_{k} represents an existing checkpoint,
- P_{meta} is $\leq$ 128 bytes long,
- The transaction author is authorized to execute register-project,
- $\alpha_{bal} \geq \mathcal{D}_{\mathsf{register\text{-}project}}$.

OUTPUTS

- $P \in O_{\mathsf{projs}}$, where $O_{\mathsf{id}} = P_{\mathsf{org}}$,
- $\alpha_{bal'} = \alpha_{bal} - \mathcal{D}_{\mathsf{register\text{-}project}}$.

4.2. **Unregistering.**

$$\mathsf{unregister\text{-}project}(P_{\mathsf{org}}, P_{\mathsf{id}})_\sigma$$

INPUTS

- P_{id} is the project id being unregistered,
- P_{org} is the id of the org under which P lives,

VALIDATION

- P_{org} identifies an existing org O,
- $P \in O_{\mathsf{projs}}$,
- The transaction author is authorized to execute unregister-project.

OUTPUTS

- $P \notin O_{\mathsf{projs}}$,
- $\alpha_{bal'} = \alpha_{bal} + \mathcal{D}_{\mathsf{register\text{-}project}}$.

4.3. **Creating a checkpoint.** The act of anchoring a project's state in the registry:

$$\mathsf{checkpoint}(K_{\mathsf{parent}}, K_{\mathsf{hash}})_\sigma$$

Checkpoints within the scope of a single project form a chain going from the latest, or "current" checkpoint k_{n-1} to the first and original checkpoint k_0. Checkpoints are identified by their transaction hash, so $k \in T_{\mathsf{hash}}$.

From the perspective of k_0, we can talk of a checkpoint *tree*, since due to their nature, they are able to represent branching. Hence, the original checkpoint k_0 is also called the *root* checkpoint.

INPUTS

- K_{parent} is the *id* of the previous or 'parent' checkpoint,
- K_{hash} is the new hash of the project state,

VALIDATION

- K_{parent} refers to an existing checkpoint in the registry, or is $\varnothing$.
- K_{hash} is a valid hash that hasn't been used in a parent checkpoint.

4.4. **Setting a checkpoint.** The act of updating the project to point to a new checkpoint:

$$\mathsf{set\text{-}checkpoint}(P_{\mathsf{org}}, P_{\mathsf{id}}, k')_\sigma$$

which updates $P_{\mathsf{checkpoint}}$ from k to k'.

INPUTS

- P_{org} is the id of the org O under which the project lives,
- P_{id} is the id of the project being updated,
- k' is the id of the checkpoint the project should be associated to.

VALIDATION

- $P_{\mathsf{id}} \in O_{\mathsf{projs}}$, or P lives under O,
- k' is a checkpoint which has the original project checkpoint k_0 in its ancestry,
- The transaction author is authorized to execute set-checkpoint.

OUTPUTS

- $P_{\mathsf{k}} = k'$

Note that the semantics of this transaction allows for projects to revert to a previous checkpoint, or to adopt a "fork", as long as the new checkpoint shares part of its ancestry with the previous checkpoint.

5. User Identity

Identity in the registry serves as a way for users to consolidate the various keys and external identities they use under a short, human-readable name.

A user U is a logical grouping of *identities*, or user identifiers under a single, unique identifier, U_{id}. The set of all users is $\mathfrak{U}$.

$$U = \langle U_{\mathsf{id}}, U_{\mathsf{account}}, U_{\mathsf{keys}} \rangle$$

DEFINITION

- U_{id} is the globally unique human-readable identifier of the user,
- U_{account} is the account id which owns this user identity,
- U_{keys} is the set of off-registry public keys associated with this identity.

5.1. **Registering.** We register a new user identity and thus user

$$\mathsf{register\text{-}identity}(U_{\mathsf{id}})_\sigma$$

INPUTS

- U_{id} is the gobally unique user identifier being registered,

VALIDATION

- U_{id} must be unique, i.e. not currently in use, between 1 and 32 bytes long, and valid UTF-8.
- $\alpha_{\mathsf{bal}} \geq \mathfrak{D}_{\mathsf{register\text{-}identity}}$.

OUTPUTS

- $U \in \mathfrak{U}$, where $U = \langle U_{\mathsf{id}}, \alpha_{\mathsf{id}}, \varnothing \rangle$
- $\alpha_{\mathsf{bal'}} = \alpha_{\mathsf{bal}} - \mathfrak{D}_{\mathsf{register\text{-}identity}}$

5.2. **Unregistering.**

$$\mathsf{unregister\text{-}identity}(U_{\mathsf{id}})_\sigma$$

INPUTS

- U_{id} is the identity being unregistered,

VALIDATION

- $U \in \mathcal{U}$,
- α must be the owner of this identity, in other words $U_{\mathsf{account}} \equiv \alpha_{\mathsf{id}}$,

OUTPUTS

- $U \notin \mathcal{U}$,
- $\alpha_{\mathsf{bal}'} = \alpha_{\mathsf{bal}} + \mathcal{D}_{\mathsf{register\text{-}identity}}$.

5.3. **Associating an external key.** The act of associating an external public key to a registered user identity:

$$\mathsf{associate\text{-}key}(U_{\mathsf{id}}, k, \pi)_\sigma$$

INPUTS

- U_{id} is the identity under which to associate the key,
- k is the public portion of the key pair $\langle k, S_k \rangle$ that is to be associated,
- π is a proof or signature verifying that the transaction author owns k, defined as:
  $$\pi = \mathsf{encrypt}(\mathsf{hash}(U_{\mathsf{id}}), S_k)$$
  where S_k is the secret key from which k was derived.

VALIDATION

- $k \notin U_{\mathsf{keys}}$,
- k is a valid 32 byte Ed25519 key,
- $\alpha_{\mathsf{id}} \equiv U_{\mathsf{account}}$,
- $\mathsf{decrypt}(\pi, k) \equiv \mathsf{hash}(U_{\mathsf{id}})$

OUTPUTS

- $k \in U_{\mathsf{keys}}$

5.4. **Revoking an external key.** When a public key associated with the associate-key transaction is lost or no longer used, the following transaction will 'revoke' the association:

$$\mathsf{revoke\text{-}key}(U_{\mathsf{id}}, k)_\sigma$$

INPUTS

- U_{id} is the identity under which the key is currently associated,
- k is the key being revoked,

VALIDATION

- $k \in U_{\mathsf{keys}}$,
- $\alpha_{\mathsf{id}} \equiv U_{\mathsf{account}}$,

OUTPUTS

- $k \notin U_{\mathsf{keys}}$