

Blockchain System

1 Definitions

Definition 1 (Set). Let \mathbf{Elt} be the set of (concrete) elements. Let \emptyset be an empty set and $e \in \mathbf{Elt}$. A set of elements is expressed as the following syntax:

$s ::= \emptyset \mid e \mid s :: s$

Definition 2 (Account). An account is expressed as a tuple $\langle \mathbf{als}, \mathbf{pak}, \mathbf{puk}, \mathbf{pkh} \rangle$, where \mathbf{als} is the alias of the account, \mathbf{pak} is its private key, \mathbf{puk} is its public key and \mathbf{pkh} is its public key hash .

Definition 3 (Contract). A contract is expressed as a tuple $\langle \mathbf{als}, \mathbf{puh}, \mathbf{code} \rangle$, where \mathbf{als} is the alias of the contract, \mathbf{puh} is its public hash and \mathbf{code} is the code of the contract.

Definition 4 (Manager). A manager manages accounts on the blockchain. It is expressed as a tuple $\langle \mathbf{puk}, \mathbf{pkh}, \mathbf{bal}, \mathbf{cou} \rangle$, where \mathbf{puk} is the public key of an account, \mathbf{pkh} is its public key hash, \mathbf{bal} is its balance and \mathbf{cou} is its counter whose form is a pair (\mathbf{n}, \mathbf{b}) , where \mathbf{n} is a nature number and \mathbf{b} is a boolean value.

Definition 5 (Contractor). A contractor manages smart contracts on the blockchain. It is expressed as a tuple $\langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle$, where \mathbf{puh} is the public hash of the contract, \mathbf{bal} is its balance, \mathbf{code} is its code and $\mathbf{storage}$ is its storage.

Definition 6 (Operation). An operation is expressed as the following syntax:

$\mathbf{op} ::= \text{transfer } \mathbf{n} \text{ from } \mathbf{pkh} \text{ to } \mathbf{pkh}' \text{ fee } \mathbf{m}$
| *originate contract* \mathbf{id} *transferring* \mathbf{n} *from* \mathbf{pkh} *running* \mathbf{code} *init* \mathbf{s} *fee* \mathbf{m}
| *transfer* \mathbf{n} *from* \mathbf{pkh} *to* \mathbf{puh} *arg* \mathbf{s} *fee* \mathbf{m}

Definition 7 (Query). A query is expressed as the following syntax:

$\mathbf{qry} ::= \text{get balance for } \mathbf{pkh} / \mathbf{puh}$
| *get status for* \mathbf{oph}
| *get contract storage* \mathbf{puh}
| *get code for* \mathbf{puh}
| *get public key for* \mathbf{pkh}
| *get counter for* \mathbf{pkh}

Let \mathbf{C} be the set of accounts, \mathbf{O} be a set of operations, and \mathbf{S} be the set of contracts.

Definition 8 (State of a node). The state of a node is expressed as a tuple $[\mathbf{C}, \mathbf{O}, \mathbf{S}]$.

When an operation is injected in a node, it enters in a pending pool (and called a pending operation).

Definition 9 (Pending operation). *A pending operation is expressed as a pair $\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle$, where \mathbf{op} is an operation, \mathbf{oph} is the operation hash and \mathbf{t} is the time when it is injected.*

After sometime, a pending operation could be included in the blockchain as a accepted operation.

Definition 10 (Accepted operation). *An accepted operation is expressed as a tuple $\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle$, where \mathbf{op} is an operation, \mathbf{oph} is the operation hash and \mathbf{t} is the time when it is included in the blockchain.*

Let \mathbf{P} be a set of pending operations, \mathbf{A} be a set of accepted operations, \mathbf{K} be a set of managers, \mathbf{T} be a set of contractors and \mathbf{t} is the current time of the blockchain.

Definition 11 (Blockchain). *The state of a blockchain is expressed as a tuple $[\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}]$.*

Definition 12 (Blockchain system). *A blockchain system $\mathbf{S} \triangleq \langle \mathbf{M}, \mathbf{B} \rangle$ consists of*

1. $\mathbf{M} \equiv [C, O, S]$ is the state of a node, and
2. $\mathbf{B} \equiv [P, A, K, T, t]$ is the state of a blockchain such as $\forall c \in C \implies \exists k \in K, k.pkh = c.pkh$ and $\forall s \in S \implies \exists p \in T, s.puh = p.puh$.

2 Rules

2.1 Transfers

Rule 1 [proposal]:

$$\frac{\text{checkAcc}(pkh, C)}{\langle [C, O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle} \quad (1)$$

Rule 2 [injected]:

$$\frac{\text{checkBan}(K, pkh, n, m) \wedge \text{checkCou}(K, pkh) \wedge \text{checkPub}(K, pkh')}{\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [(\text{transfer } n \text{ from } puk \text{ to } puk' \text{ fee } m, \text{generateOph}(pkh, pkh', n, m, t), t) :: P, A, \text{updateCou}(K, pkh, \text{True}), T, t] \rangle} \quad (2)$$

Rule 3 [rejected of counter]:

$$\frac{\neg \text{checkCou}(K, pkh)}{\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle} \quad (3)$$

Rule 4 [rejected of balance]:

$$\frac{\neg \text{checkBan}(K, pkh, m, n)}{\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle} \quad (4)$$

Rule 5 [rejected of public key]:

$$\frac{\neg \text{checkPub}(K, pkh')}{\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle} \quad (5)$$

Rule 6 [included]:

$$\frac{[< \text{transfer } n \text{ from } puk \text{ to } puk' \text{ fee } m, \text{ oph}, t > :: P, A, K, T, t'] \rightarrow [P, < \text{transfer } n \text{ from } puk \text{ to } puk' \text{ fee } m, \text{ oph}, t' > :: A, \text{updateSucc}(K, puk, puk', n, m), T, t' + 1]}{\quad} \quad (6)$$

Rule 7 [timeout]:

$$\frac{t' - t \geq 60}{[< \text{transfer } n \text{ from } puk \text{ to } puk' \text{ fee } m, \text{ oph}, t > :: P, A, K, T, t'] \rightarrow [P, A, \text{updateCou}(K, puk, \text{False}), T, t']} \quad (7)$$

2.2 Smart Contracts

A. Originate

Rule 1 [proposal]:

$$\frac{\text{checkAcc}(pkh, C) \wedge \text{checkId}(id, S)}{\langle [C, O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) :: O, S], [P, A, K, T, t] \rangle} \quad (8)$$

Rule 2 [injected]:

$$\frac{\text{checkBan}(K, pkh, n, m) \wedge \text{checkCou}(K, pkh) \wedge \text{checkPrg}(code, s)}{\langle [C, (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [(\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) :: P, A, \text{updateCou}(K, pkh, \text{True}), T, t] \rangle } \quad (9)$$

Rule 3 [rejected of code]:

$$\frac{\neg \text{checkPrg}(code, s)}{\langle [C, (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle } \quad (10)$$

Rule 4 [rejected of counter]:

$$\frac{\neg \text{checkCou}(K, pkh)}{\langle [C, (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle } \quad (11)$$

Rule 5 [rejected of balance]:

$$\frac{\neg \text{checkBan}(K, pkh, n, m)}{\langle [C, (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle } \quad (12)$$

Rule 6 [included]:

$$\frac{}{\langle [C, O, S], [< (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s), t > :: P, A, K, T, t'] \rangle \rightarrow \langle [C, O, (< id, \text{generateHash}(id, code, s, t'), code > :: S)], [P, (< (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s), t' > :: A, \text{updateSucc}(K, puk, n, m), (< \text{generateHash}(id, code, s, t'), n, code, \text{getStorage}(code, s) >) :: T), t' + 1] \rangle } \quad (13)$$

Rule 7 [timeout]:

$$\frac{t' - t \geq 60}{[< (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) > :: P, A, K, T, t'] \rightarrow [P, A, \text{updateCou}(K, puk, \text{False}), T, t']} \quad (14)$$

B. Transfer

Rule 1 [proposal]:

$$\frac{\text{checkAcc}(pkh, C)}{\langle [C, O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle} \quad (15)$$

Rule 2 [injected]:

$$\frac{\text{checkBan}(K, pkh, n, m) \wedge \text{checkCou}(K, pkh) \wedge \text{checkPuh}(T, puh) \wedge \text{checkArg}(T, puh, s)}{\begin{aligned} &\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \\ &\rightarrow \langle [C, O, S], [(< (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m), \\ &\quad \text{generateOph}(pkh, puh, s, n, m, t), t >) :: P, A, \text{updateCou}(K, pkh, \text{True}), \\ &\quad T, t] \rangle \end{aligned}} \quad (16)$$

Rule 3 [rejected of counter]:

$$\frac{\neg \text{checkCou}(K, pkh)}{\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle} \quad (17)$$

Rule 4 [rejected of balance]:

$$\frac{\neg \text{checkBan}(K, pkh, n, m)}{\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle} \quad (18)$$

Rule 5 [rejected of public hash]:

$$\frac{\neg \text{checkPuh}(T, puh)}{\langle [C, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle} \quad (19)$$

Rule 6 [rejected of argument]:

$$\frac{\neg \text{checkArg}(T, \text{puh}, s)}{\langle [C, (\text{transfer } n \text{ from } \text{pkh} \text{ to } \text{puh} \text{ arg } s \text{ fee } m) :: O, S], [P, A, K, T, t] \rangle \rightarrow \langle [C, O, S], [P, A, K, T, t] \rangle} \quad (20)$$

Rule 7 [included]:

$$\frac{[< \text{transfer } n \text{ from } \text{puk} \text{ to } \text{puh} \text{ fee } m, \text{oph}, t > :: P, A, K, T, t'] \rightarrow [P, < (\text{transfer } n \text{ from } \text{pkh} \text{ to } \text{puh} \text{ arg } s \text{ fee } m), \text{oph}, t' > :: A, \text{updateSucc}(K, \text{puk}, \text{'}, n, m), \text{updateConstr}(T, \text{puh}, n, s), t' + 1]}{\quad} \quad (21)$$

Rule 8 [timeout]:

$$\frac{t' - t \geq 60}{[\text{transfer } n \text{ from } \text{puk} \text{ to } \text{puh} \text{ fee } m, t > :: P, A, K, T, t'] \rightarrow [P, A, \text{updatecou}(K, \text{puk}, \text{False}), T, t']} \quad (22)$$

3 Functions

1. Function `checkAcc(pkh, C)` checks whether an account `pkh` exists in `C`
2. Function `checkPub(K, pkh)` checks whether the public key of the public key hash `pkh` is reveled to the blockchain.
3. Function `checkBan(K, pkh, n, m)` checks whether the balance of the account `pkh` is greater or equal to $m + n$
4. Function `checkCou(K, pkh)` checks whether the current counter of an account `pkh` is used
5. Function `updateSucc(K, pkh, pkh', n, m)` updates the balance and the counter of the account `pkh` and the balance of the account `pkh'`, where
 - $\langle \text{puk}, \text{pkh}, \text{bal}, (n, \text{True}) \rangle \Rightarrow$
 - $\langle \text{puk}, \text{pkh}, \text{bal} - n - m, (n + 1, \text{False}) \rangle$
 - $\langle \text{puk}', \text{pkh}', \text{bal}', \text{cou}' \rangle \Rightarrow \langle \text{puk}', \text{pkh}', \text{bal}' + n, \text{cou}' \rangle$
6. Function `updateCou(K, puk, b')` updates the counter of the account `pkh`, where
 - $\langle \text{puk}, \text{pkh}, \text{bal}, (n, b) \rangle \Rightarrow \langle \text{puk}, \text{pkh}, \text{bal}, (n, b') \rangle$
7. Function `checkId(id, S)` checks whether a contract `id` does not already exist in `S`
8. Function `checkPrg(code, s)` checks whether the code `code` are well type and `s` is well type input
9. Function `generateOph(pkh, pkh, n, m, t)` generates a operation hash
10. Function `generateHash(S, id, puh, code, t)` generates the public hash of a contract
11. Function `getStorage(code, s)` gets the storage for the code `code` and the input `s`

4 Some implementations

Function `checkAcc(puh, C)` checks whether an account exists and `checkPuk(puh, K)` checks the revelation of its public key to the blockchain.

```
let rec checkAcc puh C =
  match C with
  | 0 -> false
  | < als, pak, puk, pkh' > :: C' ->
    if (puh = puh') then true
    else checkAcc (puh, C')

let rec checkPuk puh K =
  match C with
  | 0 -> false
  | < als, pak, puk, pkh' > :: K' ->
    if (puh = puh') and (puk /= nil) then true
    else 5checkPuk (puh, K')
```

The following functions interact with **K**.

```
let rec checkBal K puk n m =
  match K with
  | 0 -> true
  | < puk', bal, cou > :: K' ->
    if (puk = puk') and (n + m) <= bal then true
    else checkBal (K', puk, n, m)

let rec checkPub K puk =
  match K with
  | 0 -> false
  | < puk', bal, cou > :: K' ->
    if (puk = puk') then true
    else checkExi (K', puk)

let rec checkCou K puk =
  match K with
  | 0 -> false
  | < puk', bal, cou > :: K' ->
    if (puk = puk') and (cou = T) then true
    else checkCou (K', puk)

let rec updateCou K puk =
  match K with
  | 0 -> 0
  | < puk', bal, cou > :: K' ->
    if (puk = puk') then < puk', bal, F > :: K'
    else < puk', bal, cou > :: updateCou (K', puk)
```

```

let rec updateSuc K puk puk' m n =
  match K with
  | 0 -> 0
  | < puk'', bal, cou > :: K' ->
    if (puk = puk'') then < puk'', bal - (n + m), T >
      :: updateSuc (K', puk, puk', n, m)
    else if (puk' = puk'') then < puk'', bal + n, cou > :: K'
      else < puk'', bal, cou >
        :: updateSuc (K', puk, puk', n, m)

```