Blockchain System

1 Definitions

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

Definition 2 (Account). *An account is expressed as a tuple* < **als**, **pak**, **puk**, **pkh** >*, where* **als** *is the alias of the account,* **pak** *is its private key,* **puk** *is its public key and* **pkh** *is its public key hash .*

Definition 3 (Manager). *An manager manages an account on the blockchain. It is expressed as a tuple* < **puk**, **pkh**, **bal**, **cou** >*, where* **puk** *is the public key of an account,* **pkh** *is its public key hash,* **bal** *is its balance and* **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 4 (Operation). *An operation is expressed as the following syntax:*
op *::=* *transfer* $\mathbf{n}$ *from* **pkh** *to* $\mathbf{pkh}'$ *fee* $\mathbf{m}$
| *reveal key* **pkh** *fee* $\mathbf{m}$
| *register key* **pkh** *as delegate fee* $\mathbf{m}$
| *set delegate for* **pkh** *to* $\mathbf{pkh}'$ *fee* $\mathbf{m}$
| *withdraw delegate from* **pkh** *fee* $\mathbf{m}$
| *originate contract* **id** *transferring* $\mathbf{n}$ *from* **pkh** *running* **sr** *init* $\mathbf{s}$
| *transfer* $\mathbf{n}$ *from* **pkh** *to* **id** *arg* $\mathbf{s}$

Definition 5 (Query). *An query is expressed as the following syntax:*
qry *::= get balance for* **pkh**
| *check reveal for* **als** */* **pkh**
| *get counter for* **pkh**
| *get status for* **oph**
| *get store for* **pkh**
| *waiting for* **oph** *to be included in* $\mathbf{m}$ *blocks*
| *get timestamp*
| *get contracts*
| *show contracts for* **als**/**pkh**
| *get public key for* **als**/**pkh**

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

Definition 6 (State of a node). *The state of a node is expressed as a pair* [$\mathbf{C}$, $\mathbf{O}$] .

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

Definition 7 (Pending operation). *A pending operation is expressed as a pair* $<$ **op**, **oph**, **t** $>$, *where* **op** *is an operation,* **oph** *is the operation hash and* **t** *is the time when it is injected.*

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

Definition 8 (Accepted operation). *An accepted operation is expressed as a tuple* $<$ **op**, **oph**, **t** $>$, *where* **h** *is the hash of the block head that includes the operation and and* **t** *is the time when it is included in the blockchain.*

Let **P** be a set of pending operations, **A** be a set of accepted operations, **K** be a set of managers and **t** is the current time of the blockchain.

Definition 9 (Blockchain). *The state of a blockchain is expressed as a tuple* [**P**, **A**, **K**, **t**].

Definition 10 (Blockchain system). *A blockchain system* $\mathbf{S} \triangleq \langle\ \mathbf{M}, \mathbf{B}\ \rangle$ *consists of*

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

2 Rules

Rule 1 [proposal]:

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

Rule 2 [injected]:

$$\frac{\text{checkBan(K, } pkh, m, n) \wedge \text{checkCou(K, } pkh) \wedge \text{checkPub(K, } pkh')}{\begin{array}{c}\langle[\text{C, (transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: \text{O}],\ [\text{P, A, K, t}]\ \rangle \rightarrow \langle\ [\text{C, O}],\\ [(< \text{transfer } n \text{ from } puk \text{ to } puk' \text{ fee } m, \text{ophMake(pkh, pkh', m, n), t} >)\\ :: \text{P, A, updateCou(K, pkh), t}]\ \rangle\end{array}} \quad (2)$$

Rule 3 [rejected of counter]:

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

Rule 4 [rejected of balance]:

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

Rule 5 [rejected of public key]:

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

Rule 2 [included]:

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

Rule 3 [timeout]:

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

3 Queries

Query 1 [query of counter]:

$$\frac{\text{getCounter(pkh, K)} = n}{\text{get counter for } pkh = n} \quad (8)$$

Query 2 [query of reveal]:

$$\frac{\text{getReveal(pkh, K)} = b}{\text{get reveal for } pkh = b} \quad (9)$$

4 Functions

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)
```