

Blockchain System

1 Definitions

Definition 1 (Set). Let \mathbf{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 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 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 a single account. It is represented by 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 (n, b) , where n is a natural number and b is a boolean value.

Definition 5 (Contractor). A contractor manages a smart contract. It is represented by a tuple $\langle \mathbf{puh}, \mathbf{bal}, \mathbf{code}, \mathbf{storage} \rangle$, where \mathbf{puh} is the public key hash of the contract, \mathbf{bal} is its current balance, \mathbf{code} is its code, and $\mathbf{storage}$ is its current storage.

Definition 6 (Operation). Operations are defined by the following grammar:

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

Definition 7 (Query). Queries are defined by the following grammar:

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

Definition 8 (State of a node). The state of a node is a tuple $[\mathbf{C}, \mathbf{O}, \mathbf{S}]$ where \mathbf{C} is a set of accounts, \mathbf{O} a set of operations, and \mathbf{S} a set of contracts.

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

Definition 9 (Pending operation). A pending operation is 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 the operation was injected.

After some time, a pending operation may be included in the blockchain as an accepted operation.

Definition 10 (Accepted operation). An accepted operation is 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 was included in the blockchain.

Definition 11 (Blockchain). The state of a blockchain is a tuple $[\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}]$ where \mathbf{P} is a set of pending operations, \mathbf{A} is a set of accepted operations, \mathbf{K} is a set of managers, \mathbf{T} is a set of contractors, and \mathbf{t} is the current time of the blockchain.

Definition 12 (Blockchain system). A blockchain system is a pair $\langle \mathbf{M}, \mathbf{B} \rangle$ where

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

2 Rules

2.1 Transfers

Rule 1 [proposal]:

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

Rule 2 [injected]:

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

Rule 3 [rejected of counter]:

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

Rule 4 [rejected of balance]:

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

Rule 5 [rejected of public key]:

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

Rule 6 [included]:

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

Rule 7 [timeout]:

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

2.2 Smart Contracts

A. Originate

Rule 1 [proposal]:

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

Rule 2 [injected]:

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

Rule 3 [rejected of code]:

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

Rule 4 [rejected of counter]:

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

Rule 5 [rejected of balance]:

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

Rule 6 [included]:

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

Rule 7 [timeout]:

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

B. Transfer

Rule 1 [proposal]:

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

Rule 2 [injected]:

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

Rule 3 [rejected of counter]:

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

Rule 4 [rejected of balance]:

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

Rule 5 [rejected of public hash]:

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

Rule 6 [rejected of argument]:

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

Rule 7 [included]:

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

Rule 8 [timeout]:

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

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 revealed to the blockchain.
3. Function `checkBal(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 unlocked (i.e., its flag is False)
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
 - $\langle \text{puk}, \text{pkh}, \text{bal}, (\mathbf{n}, \text{True}) \rangle \Rightarrow \langle \text{puk}, \text{pkh}, \text{bal} - \mathbf{n} - \mathbf{m}, (\mathbf{n} + 1, \text{False}) \rangle$
 - $\langle \text{puk}', \text{pkh}', \text{bal}', \text{cou}' \rangle \Rightarrow \langle \text{puk}', \text{pkh}', \text{bal}' + \mathbf{n}, \text{cou}' \rangle$
6. Function `updateCou(K, puk, b')` updates the counter lock of the account `pkh` (True = locked, False = unlocked), where
 - $\langle \text{puk}, \text{pkh}, \text{bal}, (\mathbf{n}, \mathbf{b}) \rangle \Rightarrow \langle \text{puk}, \text{pkh}, \text{bal}, (\mathbf{n}, \mathbf{b}') \rangle$
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 updateSucc K puk puk' m n =
  match K with
  | 0 -> 0
  | < puk'', bal, cou > :: K' ->
    if (puk = puk'') then < puk'', bal - (n + m), T >
      :: updateSucc (K', puk, puk', n, m)
    else if (puk' = puk'') then < puk'', bal + n, cou > :: K'
      else < puk'', bal, cou >
        :: updateSucc (K', puk, puk', n, m)

```