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* $\langle \boldsymbol{als}, \boldsymbol{pak}, \boldsymbol{puk}, \boldsymbol{pkh} \rangle$*, where* $\boldsymbol{als}$ *is the alias of the account,* $\boldsymbol{pak}$ *is its private key,* $\boldsymbol{puk}$ *is its public key and* $\boldsymbol{pkh}$ *is its public key hash .*

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

Definition 4 (Manager). *A manager manages a single account. It is represented by a tuple* $\langle \boldsymbol{puk}, \boldsymbol{pkh}, \boldsymbol{bal}, \boldsymbol{cou} \rangle$*, where* $\boldsymbol{puk}$ *is the public key of an account,* $\boldsymbol{pkh}$ *is its public key hash,* $\boldsymbol{bal}$ *is its balance and* $\boldsymbol{cou}$ *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* $\langle \boldsymbol{puh}, \boldsymbol{bal}, \boldsymbol{code}, \boldsymbol{storage} \rangle$*, where* $\boldsymbol{puh}$ *is the public key hash of the contract,* $\boldsymbol{bal}$ *is its current balance,* $\boldsymbol{code}$ *is its code, and* $\boldsymbol{storage}$ *is its current storage.*

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

$$
\begin{aligned}
\boldsymbol{op} ::= & \textit{ transfer } \boldsymbol{n} \textit{ from } \boldsymbol{pkh} \textit{ to } \boldsymbol{pkh}' \textit{ arg } \boldsymbol{s} \textit{ fee } \boldsymbol{m} \\
& \mid \textit{originate contract } \boldsymbol{id} \textit{ transferring } \boldsymbol{n} \textit{ from } \boldsymbol{pkh} \textit{ running } \boldsymbol{code} \textit{ init } \boldsymbol{s} \textit{ fee } \boldsymbol{m}
\end{aligned}
$$

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

$$
\begin{aligned}
\boldsymbol{qry} ::= & \textit{ get balance for } \boldsymbol{pkh}/\boldsymbol{puh} \\
& \mid \textit{get status for } \boldsymbol{oph} \\
& \mid \textit{get contract storage } \boldsymbol{puh} \\
& \mid \textit{get code for } \boldsymbol{puh} \\
& \mid \textit{get public key for } \boldsymbol{pkh} \\
& \mid \textit{get counter for } \boldsymbol{pkh}
\end{aligned}
$$

Definition 8 (State of a node). *The state of a node is a tuple* $\boldsymbol{N} = [\boldsymbol{C}, \boldsymbol{O}, \boldsymbol{S}]$ *where* $\boldsymbol{C}$ *is a set of accounts,* $\boldsymbol{O}$ *a set of operations, and* $\boldsymbol{S}$ *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* $\langle \boldsymbol{op}, \boldsymbol{oph}, \boldsymbol{t} \rangle$*, where* $\boldsymbol{op}$ *is an operation,* $\boldsymbol{oph}$ *is the operation hash, and* $\boldsymbol{t}$ *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* $\langle \boldsymbol{op}, \boldsymbol{oph}, \boldsymbol{t} \rangle$*, where* $\boldsymbol{op}$ *is an operation,* $\boldsymbol{oph}$ *is the operation hash, and* $\boldsymbol{t}$ *is the time when it was included in the blockchain.*

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

Definition 12 (Blockchain system). *A blockchain system is a pair* $\boldsymbol{N} \| \boldsymbol{B}$ *where*

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

2 Rules

2.1 Transitions on Nodes

Each node has (nondeterministic) rules to propose an operation.

NODE-TRANSFER

$$\frac{\text{checkAcc}(\mathbf{puk}, \mathbf{C})}{[\mathbf{C}, \mathbf{O}, \mathbf{S}] \longrightarrow_N [\mathbf{C}, (\text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk'} \text{ arg } \mathbf{()} \text{ fee } \mathbf{m}) :: \mathbf{O}, \mathbf{S}]}$$

NODE-ORIGINATE

$$\frac{\text{checkAcc}(\mathbf{puk}, \mathbf{C}) \qquad \text{checkId}(\mathbf{id}, \mathbf{S})}{[\mathbf{C}, \mathbf{O}, \mathbf{S}] \longrightarrow_N [\mathbf{C}, (\text{originate contract } \mathbf{id} \text{ transferring } \mathbf{n} \text{ from } \mathbf{puk} \text{ running } \mathbf{code} \text{ init } \mathbf{s} \text{ fee } \mathbf{m}) :: \mathbf{O}, \mathbf{S}]}$$

NODE-SYSTEM

$$\frac{\mathbf{N} \longrightarrow_N \mathbf{N'}}{\mathbf{N} \| \mathbf{B} \longrightarrow \mathbf{N'} \| \mathbf{B}}$$

2.2 Transfers

Rule 2 [injected]:

$$\frac{\begin{array}{c}\text{checkBal}(\mathbf{K}, \mathbf{puk}, \mathbf{n}, \mathbf{m}) \qquad \text{checkCou}(\mathbf{K}, \mathbf{puk}) \qquad \text{checkPub}(\mathbf{K}, \mathbf{puk}') \\ \mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m} \\ \mathbf{oph} = \text{genOpHash}(\mathbf{puk}, \mathbf{puk}', \mathbf{n}, \mathbf{m})\end{array}}{\begin{array}{c}[\mathbf{C}, \mathbf{op} :: \mathbf{O}, \mathbf{S}] \| [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \longrightarrow \\ [\mathbf{C}, \mathbf{O}, \mathbf{S}] \| [\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{K}, \mathbf{puk}, \mathbf{True}), \mathbf{T}, \mathbf{t}]\end{array}}$$

Rule 3 [rejected of counter]:

$$\frac{\neg\ \text{checkCou}(\mathbf{K},\ pkh)}{\begin{array}{l}\langle [\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}] \rangle \rightarrow \\ \langle [\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}] \rangle\end{array}} \tag{1}$$

Rule 4 [rejected of balance]:

$$\frac{\neg\ \text{checkBal}(\mathbf{K},\ pkh, m, n)}{\begin{array}{l}\langle [\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } pkh' \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}] \rangle \rightarrow \\ \langle [\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}] \rangle\end{array}} \tag{2}$$

Rule 5 [rejected of public key]:

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

Rule 6 [included]:

BLOCK-ACCEPT

$$\frac{\mathbf{t}' - \mathbf{t} < 60}{[\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}'] \rightarrow [\mathbf{P}, \langle \mathbf{op}, \mathbf{oph}, \mathbf{t}' \rangle :: \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}' + 1]}$$

Rule 7 [timeout]:

BLOCK-TIMEOUT

$$\frac{\mathbf{op} = \text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puk}' \text{ arg } () \text{ fee } \mathbf{m} \qquad \mathbf{t}' - \mathbf{t} \geq 60}{[\langle \mathbf{op}, \mathbf{oph}, \mathbf{t} \rangle :: \mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}'] \rightarrow [\mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{K}, \mathbf{puk}, \mathbf{False}), \mathbf{T}, \mathbf{t}']}$$

2.3 Smart Contracts

A. Originate

Rule 2 [injected]:

BLOCK-ORIGINATE

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

Rule 3 [rejected of code]:

$$\frac{\neg\ \text{checkPrg}(\mathit{code}, s)}{\langle[\mathbf{C},\ (\text{originate contract } \mathit{id} \text{ transferring } n \text{ from } \mathit{pkh} \text{ running } \mathit{code} \text{ init } s) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle} \quad (4)$$

Rule 4 [rejected of counter]:

$$\frac{\neg\ \text{checkCou}(\mathbf{K},\ \mathit{pkh})}{\langle[\mathbf{C},\ (\text{originate contract } \mathit{id} \text{ transferring } n \text{ from } \mathit{pkh} \text{ running } \mathit{code} \text{ init } s)) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \langle\ [\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle} \quad (5)$$

Rule 5 [rejected of balance]:

$$\frac{\neg\ \text{checkBal}(\mathbf{K},\ \mathit{pkh}, n, m)}{\langle[\mathbf{C},\ (\text{originate contract } \mathit{id} \text{ transferring } n \text{ from } \mathit{pkh} \text{ running } \mathit{code} \text{ init } s) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle \rightarrow \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle} \quad (6)$$

Rule 6 [included]:

$$\frac{}{\langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [< (\text{originate contract } \mathit{id} \text{ transferring } n \text{ from } \mathit{pkh} \text{ running } \mathit{code} \text{ init } s),\ \mathbf{t}> :: \mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t'}\ ]\rangle \rightarrow \langle[\mathbf{C},\ \mathbf{O}, (< \mathit{id}, \text{generateHash}(\mathit{id}, \mathit{code}, s,\ \mathbf{t'}),\ \mathit{code} > :: \mathbf{S})],\ [\mathbf{P},\ <\ (\text{originate contract } \mathit{id} \text{ transferring } n \text{ from } \mathit{pkh} \text{ running } \mathit{code} \text{ init } s),\ \mathbf{t'} > :: \mathbf{A},\ \text{updateSucc}(\mathbf{K}, \mathit{puk}, n, m),\ (< \text{generateHash}(\mathit{id}, \mathit{code}, s,\ \mathbf{t'}),\ n, \mathit{code}, \text{getStorage}(\mathit{code}, s)\ >) :: \mathbf{T}),\ \mathbf{t'} + 1]\rangle} \quad (7)$$

Rule 7 [timeout]:

$$\frac{\mathbf{t'} - \mathbf{t} \geq 60}{[< (\text{originate contract } id \text{ transferring } n \text{ from } pkh \text{ running } code \text{ init } s) > :: \mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t'}\ ] \rightarrow [\mathbf{P}, \mathbf{A}, \text{updateCou}(\mathbf{K}, puk, \text{False}), \mathbf{T}, \mathbf{t'}]} \quad (8)$$

B. **Transfer**

Rule 1 [proposal]:

$$\frac{\text{checkAcc}(pkh, \mathbf{C})}{\langle [\mathbf{C}, \mathbf{O}, \mathbf{S}], [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \rangle \rightarrow \langle [\mathbf{C}, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O}, \mathbf{S}], [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \rangle} \quad (9)$$

Rule 2 [injected]:

$$\frac{\text{checkBal}(\mathbf{K}, pkh, n, m) \wedge \text{checkCou}(\mathbf{K}, pkh) \wedge \text{checkPuh}(\mathbf{T}, puh) \wedge \text{checkArg}(\mathbf{T}, puh, s)}{\begin{array}{l}\langle [\mathbf{C}, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O}, \mathbf{S}], [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \rangle \\ \rightarrow \langle [\mathbf{C}, \mathbf{O}, \mathbf{S}], [(< (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m), \\ \text{generateOph}(pkh, puh, s, n, m, \mathbf{t}), \mathbf{t}>) :: \mathbf{P}, \mathbf{A}, \text{updateCou}(\mathbf{K}, pkh, \text{True}), \\ \mathbf{T}, \mathbf{t}] \rangle\end{array}} \quad (10)$$

Rule 3 [rejected of counter]:

$$\frac{\neg\ \text{checkCou}(\mathbf{K}, pkh)}{\langle [\mathbf{C}, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O}, \mathbf{S}], [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \rangle \rightarrow \langle [\mathbf{C}, \mathbf{O}, \mathbf{S}], [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \rangle} \quad (11)$$

Rule 4 [rejected of balance]:

$$\frac{\neg\ \text{checkBal}(\mathbf{K}, pkh, n, m)}{\langle [\mathbf{C}, (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O}, \mathbf{S}], [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \rangle \rightarrow \langle [\mathbf{C}, \mathbf{O}, \mathbf{S}], [\mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}] \rangle} \quad (12)$$

Rule 5 [rejected of public hash]:

$$\frac{\neg\ \text{checkPuh}(\mathbf{T},\ puh)}{\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle \to \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle} \quad (13)$$

Rule 6 [rejected of argument]:

$$\frac{\neg\ \text{checkArg}(\mathbf{T},\ puh, s)}{\langle[\mathbf{C},\ (\text{transfer } n \text{ from } pkh \text{ to } puh \text{ arg } s \text{ fee } m) :: \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle \to \langle[\mathbf{C},\ \mathbf{O},\ \mathbf{S}],\ [\mathbf{P},\ \mathbf{A},\ \mathbf{K},\ \mathbf{T},\ \mathbf{t}]\rangle} \quad (14)$$

Rule 7 [included]:

$$\frac{}{[\langle\text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{s} \text{ fee } \mathbf{m}, \mathbf{oph}, \mathbf{t}\rangle :: \mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}'] \to [\mathbf{P}, \langle\text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{s} \text{ fee } \mathbf{m}, \mathbf{oph}, \mathbf{t}'\rangle :: \mathbf{A},\ \text{updateSucc}(\mathbf{K},\ \mathbf{puk},\ ",\ \mathbf{n},\ \mathbf{m}),\ \text{updateConstr}(\mathbf{T}, \mathbf{puh}, \mathbf{n}, s),\ \mathbf{t}' + 1]} \quad (15)$$

Rule 8 [timeout]:

$$\frac{\mathbf{t}' - \mathbf{t} \geq 60}{[\langle\text{transfer } \mathbf{n} \text{ from } \mathbf{puk} \text{ to } \mathbf{puh} \text{ arg } \mathbf{s} \text{ fee } \mathbf{m}, \mathbf{t}\rangle :: \mathbf{P}, \mathbf{A}, \mathbf{K}, \mathbf{T}, \mathbf{t}'] \to [\mathbf{P}, \mathbf{A}, \text{updCount}(\mathbf{K}, \mathbf{puk}, \mathbf{False}), \mathbf{T}, \mathbf{t}']} \quad (16)$$

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 `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 *phk* 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 *phk* and the balance of the account *phk*′, where
   - `< puk, pkh, bal, (n, True) > =>`
     `< puk, pkh, bal - n - m, (n + 1, False) >`
   - `< puk', pkh', bal', cou' > => < puk', pkh', bal' + n, cou' >`
6. Function `updateCou(K, puk, b')` updates the counter lock of the account *phk* (True = lockeed, False = unlocked), where
   - `< puk, pkh, bal, (n, b) > => < puk, pkh, bal, (n, b') >`

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 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)
```