Multi-Signature

Purpose of the cyber.msig smart contract

The `cyber.msig` system smart contract is used to both sign and send multi-signature transactions to the blockchain, as well as to offer the transaction to another user for signing.

The `cyber.msig` smart contract includes such actions as propose, approve, unapprove, cancel, exec and invalidate.

Terminology used

Multi-signature transaction — a transaction for the performance of which it is necessary to obtain permission (signature) from one or more accounts (actors) whose names are indicated in a separate list during its formation. The transaction can include one or more operations. The transaction is sent to the blockchain only after obtaining the number of permissions (signatures) necessary for its execution.

Multi-signature transaction actor — the name of the account which you need to obtain permission from to perform a multi-signature transaction. An actor can be an individual user or a company account. Only users entrusted to it specified during its registration in the system with a certain permission level can give permission on behalf of the company.

Transaction Permission — the signature of the actor which has the form of a structure containing two fields — the name of the account and the required level of permission.

Permission level — significance of the signature (authorization) which is assigned to an account during registration. Authorization has the form of a structure containing fields that uniquely identify the signature belonging to a particular account, taking into account its weight and the key being used for signing.

```
1  struct authority {
2     uint32_t threshold = 0;
```

```
    std::vector<key_weight> keys;
    std::vector<permission_level_weight> accounts;
    std::vector<wait_weight> waits;
}
```

Depending on the permission level set in the transaction, the number of signatures required to complete the transaction may vary. For example, permission on behalf of a company representative permission with the required `threshold=3` is considered to be obtained if the signature is affixed either with one account with the same threshold value, or with two accounts with values of «2» and «1», or three with «1».

Actions

propose

The `propose` action is used to create a multi-signature transaction offer (proposed transaction), which requires permission from third-party accounts.

```
void multisig::propose(
    name proposer,
    name proposal_name,
    std::vector<permission_level> requested,
    transaction trx
)
```

Parameters:

- `proposer` — name of account, author of a multi-signature transaction.
- `proposal_name` — the unique name assigned to the multi-signature transaction when it is created. This parameter, in conjunction with the proposer parameter, uniquely identifies a multi-signature transaction.
- `requested` — the unique name assigned to the multi-signature transaction when it is created. This parameter, in conjunction with the proposer parameter, uniquely identifies a multi-signature transaction.
- `trx` — proposed transaction.

For calling the propose action the authorization of proposer user is obligatory.

For example:
The `alice`, `bob`, and `carol` accounts created a common multisig account, for which transactions on behalf of which 2 signatures from 3 are sufficient. To work with this account, `bob` and `alice` decided to use separate keys, adding them to the `msig` permission. Now, to send a transaction to the network, `alice` can offer it for signing, indicating all three participants (with the required permissions) in the requested list:

```
cyber.msig::propose(
   alice,
   sendtofund,
   [alice@msig, bob@msig, carol@active],
   {actions:[{account:token, name:transfer, authorization:[multisig@active]
)
```

> **Side note**
> After the multi-subscription transaction is completed, its name `proposal_name` can be reused for other transactions (by the `proposer` account). However, this `proposal_name` cannot be reused unless permission was not obtained to execute a multi-subscription transaction.

approve

The `approve` action is used to send permission from an account in the requested list to execute a multi-subscription transaction.

```
void multisig::approve(
   name proposer,
   name proposal_name,
   permission_level level,
   const eosio::binary_extension<eosio::checksum256>& proposal_hash
)
```

Parameters:

- `proposer` — name of account, author of a multi-signature transaction..

- `proposal_name` — The unique name assigned to the multi-signature transaction when it is created. This parameter, in conjunction with the proposer parameter, uniquely identifies a multi-signature transaction.
- `level` — permission by which the account gives its consent to the implementation of the transaction proposed for signing.
- `proposal_hash` — optional parameter, «hash amount» of the multi-signature transaction. The parameter is used to control the occurrence of changes in a multi-signature transaction since the call to propose. Level permission is not issued in the event of a «hash amounts» mismatch specified by proposal_hash and obtained for an incoming transaction.

Authorization of the account specified in the level parameter is required to call the approve action.

For example:
In order to approve the transaction proposed by `alice`, member accounts must approve with the permission specified in propose and sign the transaction with a signature that satisfies this permission. For greater security, the `proposal_hash` parameter can be used:

```
cyber.msig::approve(alice, sendtofund, alice@msig, hash)
cyber.msig::approve(alice, sendtofund, bob@msig)
cyber.msig::approve(alice, sendtofund, carol@active)
```

unapprove

The `unapprove` action is used by an account in the requested list to revoke permission previously granted by this account to execute a multi-signature transaction in case of a change in decision.

```
void multisig::unapprove(
   name proposer,
   name proposal_name,
   permission_level level
)
```

Parameters:

- `proposer` — name of account, author of a multi-signature transaction.

- `proposal_name` — the unique name assigned to the multi-signature transaction when it is created. This parameter, in conjunction with the proposer parameter, uniquely identifies a multi-signature transaction.
- `level` — revoked permission

To call `unapprove` action, authorization of the account specified in the level parameter is required.

For example:
To withdraw approval, `alice` should unapprove with the permission specified in propose and sign the transaction with a signature that satisfies this permission:

```
cyber.msig::unapprove(alice, sendtofund, alice@msig)
```

cancel

The `cancel` action is used to cancel a multi-signature transaction offer.

```
void multisig::cancel(
   name proposer,
   name proposal_name,
   name canceler
)
```

Parameters:

- `proposer` — name of account, author of a multi-signature transaction.
- `proposal_name` — the unique name assigned to the multi-signature transaction when it is created. This parameter, in conjunction with the proposer parameter, uniquely identifies a multi-signature transaction.
- `canceler` — the name of the account that cancels its execution.

A certain amount of time is allotted for a multi-signature transaction. If after this time the transaction does not complete, it can be canceled by any user. Only author is authorized to cancel a multi-subscription transaction before the time allotted for its execution expires, for

example, after making an adjustment to it. In this case, the canceler parameter will match the proposer parameter.

To call the cancel operation, the authorization of the account specified in the canceler parameter is required.

For example:
At the moment, the transaction has 2 necessary signatures, but `alice` has made a decision not to send it to the network (see examples above). She can cancel the transaction as the creator of the proposed transaction:

```
cyber.msig::cancel(alice, sendtofund, alice)
```

Any user can cancel an expired transaction, thereby restoring STORAGE himself:

```
cyber.msig::cancel(alice, sendtofund, anybody)
```

exec

The `exec` action is called to execute a multi-signature transaction.

```
void multisig::exec(
   name proposer,
   name proposal_name,
   name executer
)
```

Parameters:

- `proposer` — name of account, author of a multi-signature transaction.
- `proposal_name` — the unique name assigned to the multi-signature transaction when it is created. This parameter, in conjunction with the proposer parameter, uniquely identifies a multi-signature transaction.

- `executer` — the name of the account performing the multi-signature transaction. An executor account can be either an actor or any other user who provides payment for the resources used. A transaction is executed if it contains the required number of permissions.

To invoke the `exec` action, authorization of the account specified in the executer parameter is required.

For example:
If the transaction was not canceled in the previous example, then it has the required 2 of 3 permissions, and can be executed by calling the `exec` :

```
cyber.msig::exec(alice, sendtofund, bob)
```

invalidate

The `invalidate` action is used to revoke all permissions previously issued by the account for performing multi-signature transactions. The action applies to all proposed transactions that are at the voting stage.

```
void multisig::invalidate(name account)
```

Parameter:

- `account` — the name of the account whose previously issued permissions to perform multi-signature transactions must be invalidated.

Calling invalidate action will invalidate all permissions issued by an account named account for ongoing multi-signature transactions. Invalidate does not apply to permissions of already completed transactions, as well as to permissions that will be issued by this account after calling invalidate.

The proposed transaction can still be sent to the network if it contains the necessary number of remaining permissions, taking into account the canceled one.

To invoke an invalidate action, authorization of an account named account is required.

For example:

`carol` learned that her key was stolen. It's time to cancel all proposed transactions (and change the key):

```
cyber.msig::invalidate(carol)
```