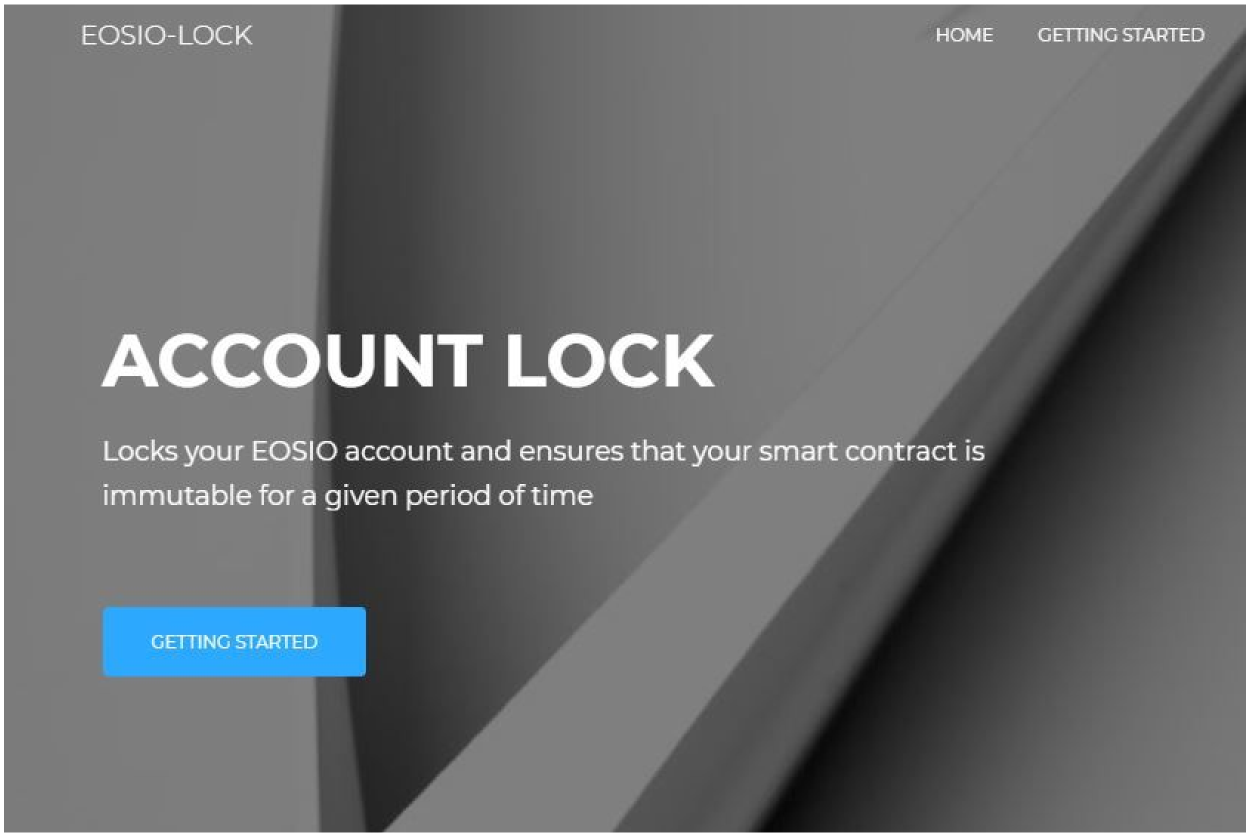


ACCOUNT LOCK

Locks your EOSIO account and ensures that your smart contract is immutable for a given period of time

ACCOUNT-LOCK 1

20.08.2019

bloxchanger@outlook.com

Website: http://EOSIO-lock.net
Eos mainnet account: accountlock1

Overview

One of the features of the blockchains based on the EOSIO framework is that the smart contract’s code can be easily updated and modified by the account owner. This possibility differentiates EOS based systems from others, like the Ethereum platform, where code is immutable once deployed on the blockchain.

While this feature is extremely useful from a developer’s perspective, as it allows programmers to fix bugs and update the code with new versions, it may result less convenient from the perspective of the smart contract’s users. In fact, the program owner could decide at any time to change the code and divert the users’ resources to his advantage.

Consider, for instance, the case of a distributed application (dapp) that during his normal operations needs to collect token deposits from its users. This would be the case, for example, for many gambling, financial or even internet-of-things related blockchain applications. Now, if the contract is not immutable, a significant amount of trust would be required from the users’ side in order to make a deposit on the dapp’s account where the developer can decide to change the code in any way that allows him to steal all the deposited funds.

The EOSIO framework already provides a solution to these issues. In particular, the account permission system allows to replace the account keys with an “eosio.null” permission which makes the account, and the corresponding smart contract, immutable. In this case, the code will never again become amendable and the account creator will never get back in control of the account and code. This solution will, in fact, reproduce the way smart contracts work in other platforms as for example the Ethereum blockchain.

However, for many applications, turning into a completely immutable and non recoverable account might be an unnecessary and too extreme solution. In particular, there are many cases which might require only temporary immutability and might still benefit from the possibility to update the code from time to time.

The EOSIO-lock service proposes a solution that exploits the possibilities offered by the extremely flexible Eos permission system together with the programming features of the platform in order to provide temporary immutability for eos accounts.

Temporary immutability might be very useful for all the cases where the users would make a deposit on a given smart contract’s account and then use or withdraw the deposited funds in a relatively short time. Think for instance to a gambling dapp. The developer might turn the account to temporary-immutable for consecutive periods of, say, one month. During each one-month period the users can play on the dapp being reassured that the developer cannot change the code to his advantage and no trust would be required on this side. At the same time, the developer would have the possibility to regain control of his account at the end of each one-month period in order to fix and update the code.

The first prototype developed by the EOSIO-lock service was deployed on the accountlock1 address on the Eos mainnet. It is a completely secure dapp that can lock a target account for a predefined time period and ensures that no-one has control of that account while locked.

How it works

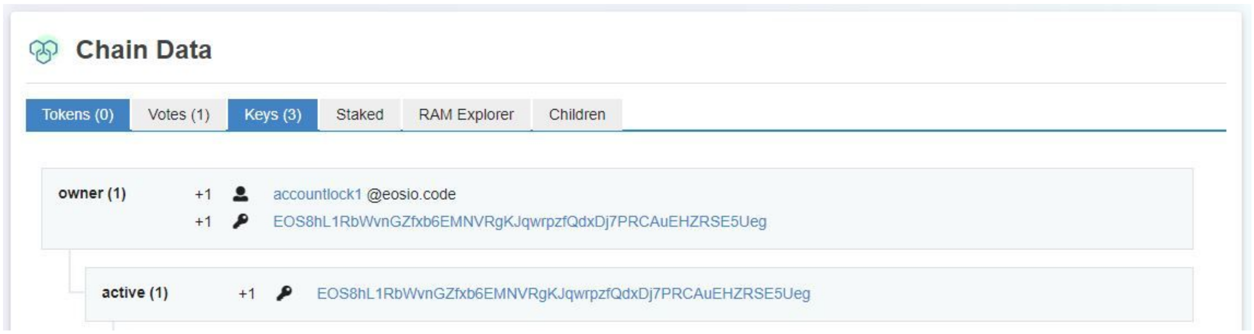
First of all, the original accountlock1 keys were removed and the permissions were set to “eosio.null” in order to turn the account into an immutable account. Therefore, the utility is deployed on an immutable account.

The smart contract deployed on the account has two main actions:

- *lock(target_contract, lock_time, public_key_string)*

This function can be used to lock a given target account/contract. The parameters to be provided include the name of the account that we want to lock, the lockup time (in seconds) and the public key (string) that should be assigned to the contract once it gets unlocked.

This action requires an authority from the target account ensuring that only the possessor/s of the account can activate the lockup procedure. In addition, in order to be able to correctly run the lockup process, the owner/s of the account must add the accountlock1@eosio.code authority to the owner permission before calling the lock action.



owner (1) +1 accountlock1 @eosio.code
+1 EOS8hL1RbWvnGZfxb6EMNVRgKJqwrpzfQdxDj7PRCAuEHZRSE5Ueg
active (1) +1 EOS8hL1RbWvnGZfxb6EMNVRgKJqwrpzfQdxDj7PRCAuEHZRSE5Ueg

Once the lock action is called, it will replace the keys assigned to the owner and active permissions with the accountlock1@eosio.code authority. In addition, it will add a record on a multi-index table stored on the accountlock1 address which will include the name of the locked account, the date and time after which it can be unlocked and the public key which will be assigned after the contract is unlocked.

At this point and until the expiry date of the lockup period, no-one will be able to operate the account except the accountlock1 contract’s code. Since the account-lock contract was turned into an immutable contract, the code cannot be changed and the only action that can be called for this contract would be the unlock action.

- *unlock(target_contract)*

The unlock action has only one parameter which specifies the name of the account to be unlocked. It will run successfully only once that the defined deadline has expired.

This action does not require any authorization and can be run by any account (please note that, of course, it cannot be called using the locked account itself as no-one has control of the required keys and authority while locked).

Once called, the action will restore the public key defined during the lockup as the owner and active permission key. At this point the account will thus be unlocked and the owner of the public key gets back in control of the account and contract's code.

In addition, the unlock action removes the previously saved record on the account-lock data table.

Please note that while an account is locked, the data table stored on the accountlock1 contract provides at the same time a certification of the lockup and information of the unlock time/date. Any dapp user can easily access these details using an eos blockchain explorer.

Getting Locked

The lock/unlock process described above can be activated with three simple steps:

1. Add the accountlock1@eosio.code permission to the owner authority of your account:

   ```
   cleos set account permission YOUR_CONTRACT owner '{"threshold":
   1,"keys": [{"key": "CURRENT_PUBLIC_KEY","weight": 1}],
   "accounts":
   [{"permission":{"actor":"accountlock1","permission":"eosio.code
   "},"weight":1}]}' -p YOUR_CONTRACT@owner
   ```

   In alternative to the above cleos command line you can use the eosio account management tools available online.

2. Call the lock action of the accountlock1 smart contract. Set the following parameters:

   target_contract: [YOUR_CONTRACT]

   lock_time: [insert the lock time period in seconds] public_key_string: [insert the public key that should be used to restore your account after the lock time has expired]

```
cleos push action accountlock1 lock '{"target_contract":
"YOUR_CONTRACT", "lock_time": 60, "public_key_string":
"YOUR_PUBLIC_KEY"}' -p YOUR_CONTRACT@active
```

Warning: make sure to insert correctly your public key string or you might permanently loose control of your account.

3. After the lock time has expired, call the unlock action of the accountlock1 contract to restore the owner authority:

   target_contract: [YOUR_CONTRACT]

```
cleos push action accountlock1 unlock '{"target_contract":
"YOUR_CONTRACT"}' -p ANY_ACCOUNT@active
```

   To call the unlock action you can use any eos account you control (please note that you will not be able to call the unlock action using YOUR_CONTRACT's permissions as these will be locked at this time); This action will unlock your account by setting the public_key_string provided at step 2 as the new owner authority.

Advanced settings

In some cases, an eosio.code authorization may be required in order to use external contract actions from within your smart contract's code. For instance, adding a YOUR_CONTRACT@eosio.code permission with active authority is required to allow your smart contract's actions to send tokens to other accounts.

PROBLEM

These eosio.code authorizations are often associated to the active permission and therefore will be removed when an account is locked with the accountlock1 utility (in fact, to lock the target account, the utility code replaces the active and owner permissions).

SOLUTON

The utility allows to maintain the required eosio.code authorizations when an account is locked if these authorizations are specified as custom permissions and linked to one or more external actions before locking the account.

The account-lock utility will take control of the active and owner permissions of the target account but will keep the custom permission settings unchanged to YOUR_CONTRACT@eosio.code.

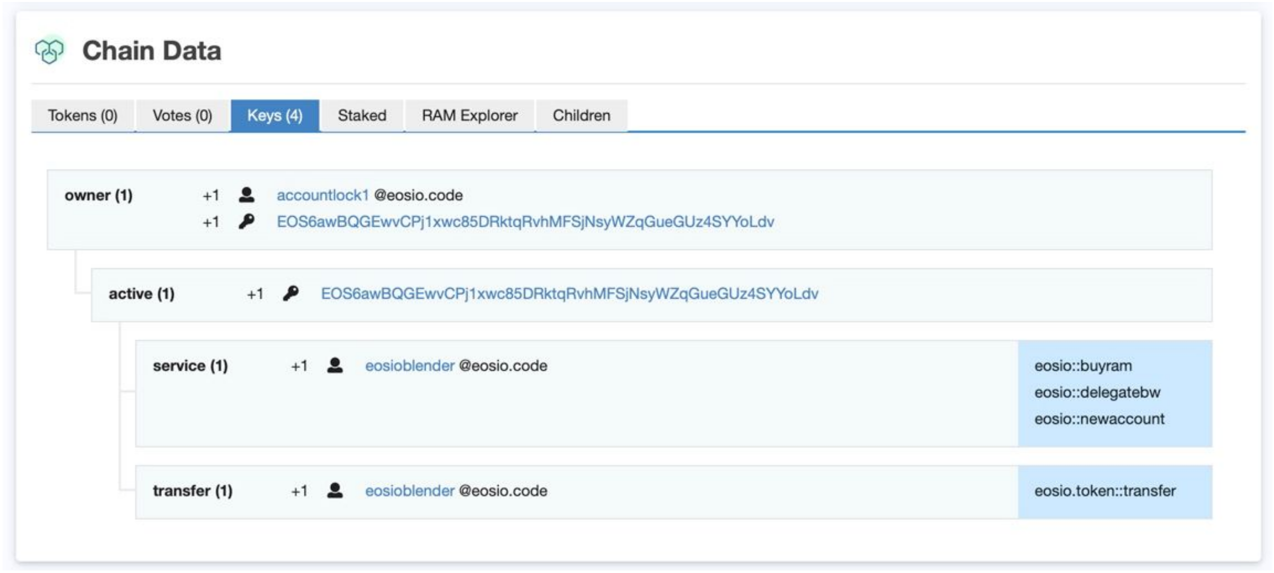
Therefore, defining custom permissions allows your code to access the required external actions even when locked by the accountlock1 utility.

Example: the eosioblender contract

The **eosioblender** contract requires an ***@eosio.code*** authorization to access the following external actions:

- eosio.token::transfer
- eosio::newaccount
- eosio::delegatebw
- eosio::buy-ram

In order to allow the smart contract's code to access these actions, before locking the account with the **accountlock1** utility, we create a "*transfer*" and a "*service*" custom permissions with authority ***eosioblender@eosio.code***. Next, we link the "*transfer*" permission to the *eosio.token::transfer* action and the "*service*" permission to the *eosio::newaccount*, *eosio::buyram* and *eosio::delegatebw* actions (see picture below).



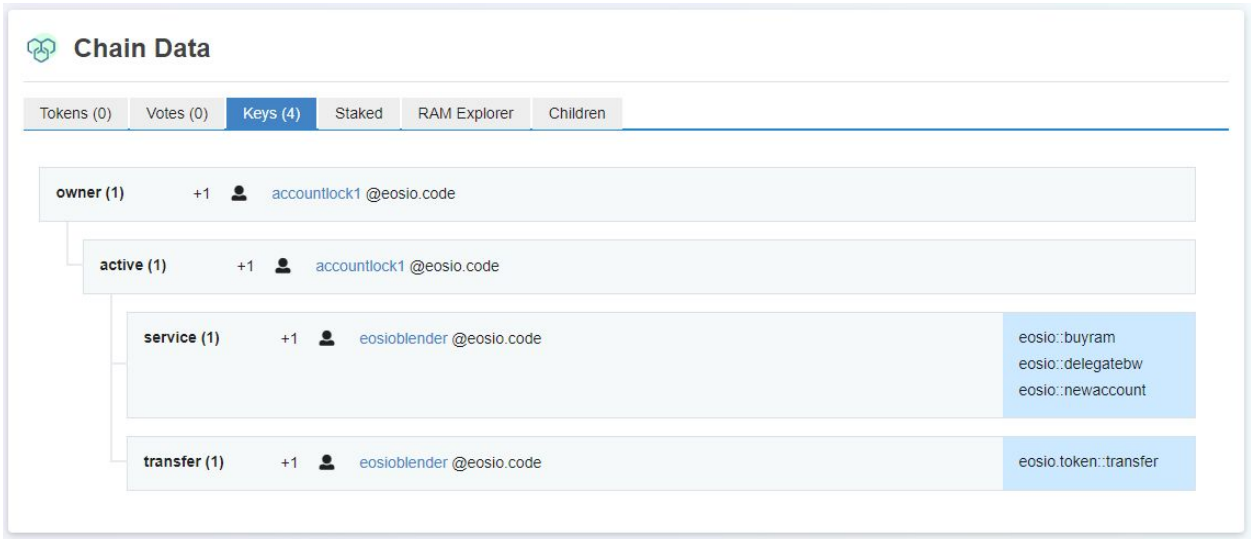
owner (1) +1 accountlock1 @eosio.code
+1 EOS6awBQGEwvCPj1xwc85DRktqRvhMFSjNsyWZqGueGUz4SYYoLdv

active (1) +1 EOS6awBQGEwvCPj1xwc85DRktqRvhMFSjNsyWZqGueGUz4SYYoLdv

service (1) +1 eosioblender @eosio.code — eosio::buyram, eosio::delegatebw, eosio::newaccount

transfer (1) +1 eosioblender @eosio.code — eosio.token::transfer

Now, the **accountlock1** tool can be used to lock the **eosioblender** contract while ensuring that the code has access to the required external actions.


Chain Data

Tokens (0) | Votes (0) | Keys (4) | Staked | RAM Explorer | Children

owner (1)	+1	accountlock1 @eosio.code	
active (1)	+1	accountlock1 @eosio.code	
service (1)	+1	eosioblender @eosio.code	eosio::buyram eosio::delegatebw eosio::newaccount
transfer (1)	+1	eosioblender @eosio.code	eosio.token::transfer

Please note that the custom permissions must be specified in the code containing the external calls.

For example, the "service" permission will be specified in the permission level when calling the *eosio::newaccount* action within the eosioblender's smart contract:

```
action(
        permission_level{ _self, "service"_n },
        "eosio"_n,
        "newaccount"_n,
        new_account
).send();
```

The "transfer" permission will be specified in the permission level when calling the *eosio.token::transfer* action in the eosioblender's smart contract:

```
action(
        permission_level{_self, "transfer"_n},
        "eosio.token"_n,
        "transfer"_n,
              std::make_tuple(_self, to, quantity, string("sent by
eosioblender"))
).send();
```

Possible future improvements and additional features

The accountlock1 contract is a first application that conditions the lock/unlock feature to an elapsing period of time. More advanced versions of the application may condition the lockup to different constraints and more articulated parameters. The utility may be implemented in such a way that a contract gets unlocked only when a given condition is met. For instance, it could be unlocked only when a given percentage of users express their vote in favor of a hypothetical unlock proposal. Otherwise, it could be structured in such a way that it gives control back to the owner/developer only when the deposits from the dapp users are all withdrawn or reduced to zero.

Other potential improvements involve the user interface of the application. Currently, the accountlock1 contract does not have a web interface and must be activated from the cleos command line or through one of the eos toolkits available on the web. Future improvements may involve the development of an online user interface that allows developers and users to directly activate the smart contract's actions and easily setup the required parameters.

A more advanced user interface could also be implemented in order to allow for an easier access to the list of locked accounts. The service could be aimed at providing an immediate and easily verifiable certification of the lockup status of a given account. The lockup certification could be even integrated directly into wallet applications so that the users may check, before signing a transaction, if a contract is locked and the lockup expiry date.