MYKEY Specification

v1.2

Jul 14, 2020

1 Specifications

1.1 Introduction

MYKEY is a smart contract wallet based on KEYID protocol and deployed on multiple blockchains, providing users with one-stop blockchain services, including asset storing, exchanging and interacting with DApps, etc. MYKEY is designed with main characteristics like separated permissions, recoverable account and meta-transaction etc. MYKEY accounts execute all operations via one or more authorized logic modules, which verify user identity by signatures. Besides, logic modules can be upgraded with delay.

1.2 Key Permissions

Every MYKEY account has an Admin Key and a set of Operation Keys. Admin Key (index 0) is the highest authority, managing all keys. While Operation Key (index above 0) manages a specific function.

Various action permissions are owned by separated keys, as listed below.

Index	Key Name	Function
0	Admin	Highest authority, no specific operational permission
1	Asset	Asset transfering
2	Adding	Add additional operation keys
3	Reserved	All other unspecified permissions
4	Assist	Propose or approve a proposal as emergency contact

The private Admin Key (known as "Recovery Phrase" in mobile APP) can be exported from mobile device and stored offline, while the private Operation Keys are stored encrypted in mobile device.

1.3 Account Freezing

In emergency situations, a user can freeze his account immediately (by freezing all the operation keys), in which case, all operation key permissions (Asset, Adding, Reserved, Assist and Modify) are disabled until the user unfreezes his account or changes all operation keys.

1.4 Emergency Contacts

MYKEY account can add other MYKEY accounts as his emergency contact to protect account security. Every account can have at most 6 emergency contacts.

Adding someone as emergency contact requires his consent (emergency contact's signature) and will take effect after 21 days. However, users can choose to add initial emergency contacts when creating account, in which way it takes effect without delay. In MYKEY APP, the initial emergency contact is MYKEY Lab.

A user can also remove an emergency contact. Also, it will take effect in 21 days.

1.5 Account Management Proposal

A multi-sig proposal can be proposed by a user's emergency contact or both the user and emergency contact together, then waiting for other emergency contacts to approve, if needed.

The number of signatures needed to execute a proposal is 60% of the total number of emergency contacts rounded up.

Total Number of Emergency Contacts	1	2	3	4	5	6
Minimum Number of Signatures Required	1	2	2	3	3	4

Once the multi-sig threshold is met, anyone can execute the proposal.

1.6 Account Recovery

All keys in MYKEY account are alterable. Therefore, a user's account can be recovered in emergency situations like private key losing or leaking. Different cases are listed in the tables below.

Please note "client" represents the user, and "backup" represents the user's emergency contact.

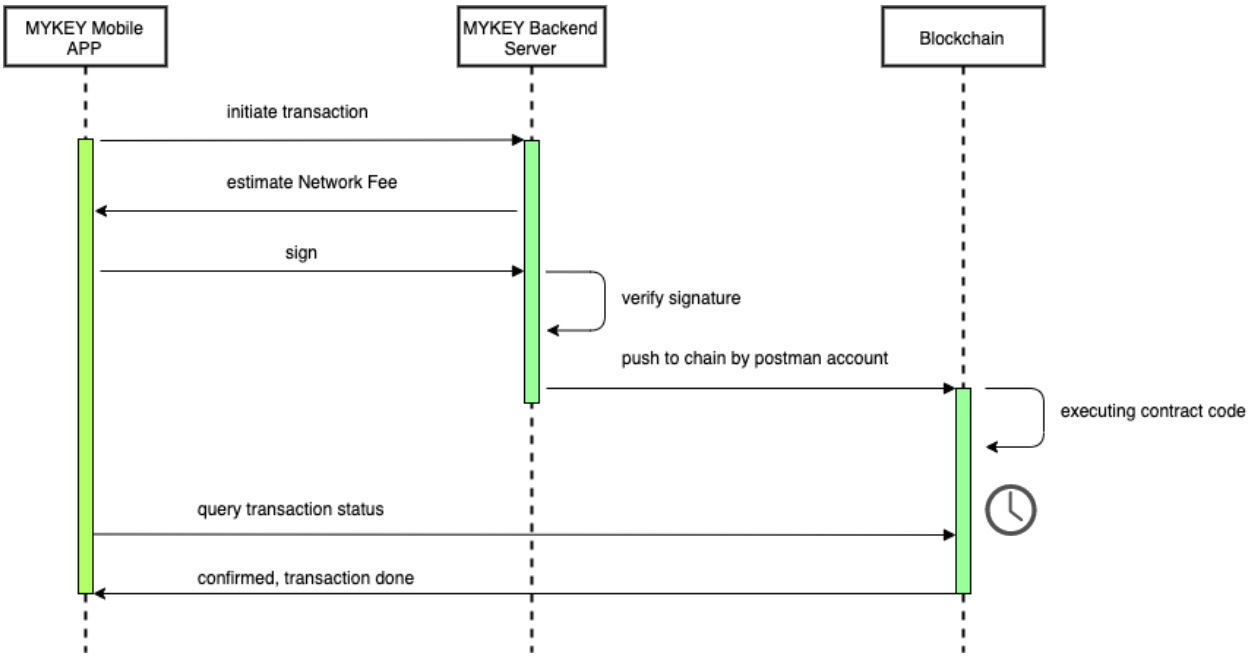
"Normal Way" is initiated by client himself, with no emergency contact involved, thus it's not an Account Management Proposal.

Change Admin Key	Initiated By	Delay Time	Use Case
Normal Way	only client	21 days	-
Emergency Proposal	only backup	30 days after executing proposal	Recovery Phrase lost
Expedited Proposal	both client and backup	Immediate after executing proposal	Recovery Phrase leaked

Change or Unfreeze Operation Keys	Initiated By	Delay Time	Use Case
Normal Way	only client	7 days	-
Expedited Proposal	both client and backup	Immediate after executing proposal	Mobile device lost (freeze first, then change)

1.7 Meta-Transactions

MYKEY users don't need to pay transaction fees themselves. This is achieved by enabling MYKEY accounts to sign a message showing action intent and allowing a relayer (MYKEY "postman" account by default) to execute the transaction and pay the fee on their behalf. And MYKEY only charges *Network Fee*.



1.8 Network Fee

Network Fee is used to pay for executing operations on blockchain.

Transaction fee and resource model are different on multiple blockchains, e.g. gas price and gas limit on Ethereum, RAM/CPU/NET on EOS. For easy understanding, all these complex concepts are wrapped up and simplified into *Network Fee*.

Network Fee is a prepaid credit, which is available in MYKEY APP for users to purchase.

1.9 Upgradability of Logic Modules

Logic modules can be upgraded to add new features or fix potential bugs. Every update will take effect after a pending time. Currently the pending time is 4 days, and it will prolong in the future. Also, altering the pending time requires a time delay depending on the pending time.

1.10 Multi-Chain Consistency

MYKEY user data including Admin Key, Operation Keys and Emergency contacts are the same on multiple blockchains. Therefore, a user only needs to keep one Recovery Phrase and store one set of Operation Keys in mobile devices. Emergency contacts can also assist users on multiple chains.

Multi-Chain Consistency is guaranteed by MYKEY backend service.

2 Implementation

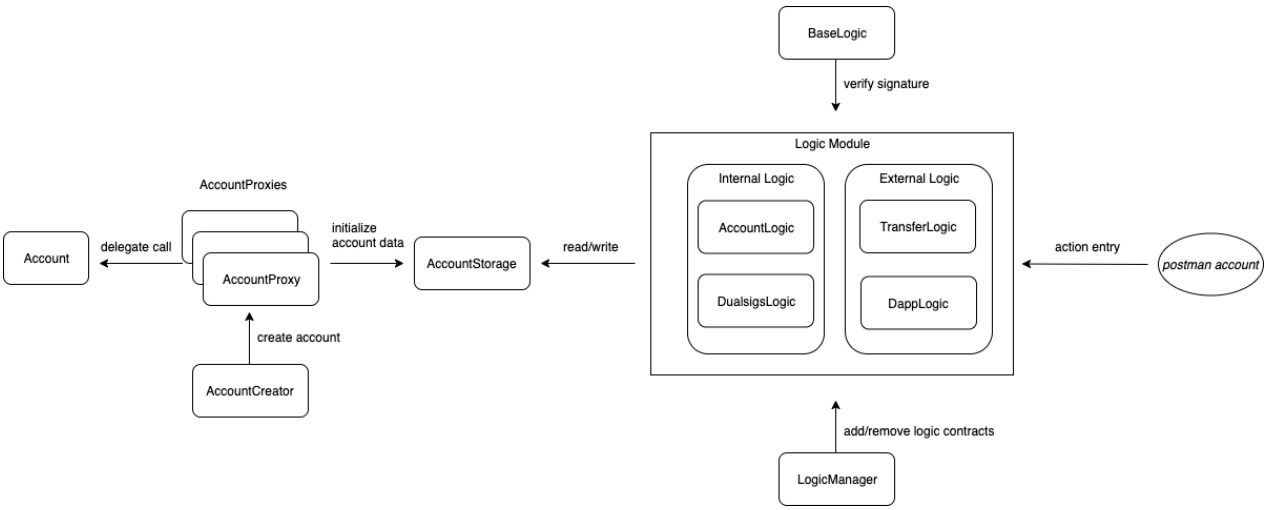
Currently, MYKEY has been implemented and deployed on two blockchains: Ethereum and EOS. More chains may be supported in the future according to user needs. Due to different characteristics of multiple blockchains, some implementation details may differ.

2.1 Implementation on Ethereum

2.1.1 Smart Contracts Architecture

MYKEY smart contracts can be divided by function into 4 main modules: **Account Module**, **Account Storage Module**, **Logic Management Module** and **Logic Module**.

Module	Contracts
Account Module	*Account.sol*, *AccountProxy.sol*, *AccountCreator.sol*
Account Storage Module	*AccountStorage.sol*
Logic Management Module	*LogicManager.sol*
Logic Module	*AccountLogic.sol*, *DualsigsLogic.sol*, *ProposalLogic.sol*, *TransferLogic.sol*, *DappLogic.sol*, *CommonStaticLogic.sol*



Note: recently, we simplified *AccountLogic*, *DualsigsLogic* and *TransferLogic*, extracted some logics into two new logic modules: *ProposalLogic* and *CommonStaticLogic*.

2.1.2 Account Creation

AccountCreator is a factory contract to create proxy accounts using CREATE or CREATE2 and initialize them with initial keys and emergency contacts. Proxy accounts don't have specific implementation, but delegate all invocations to account template contract, executing specific operations. This proxy mechanism is aiming to saves gas when creating accounts.

2.1.3 Data Storage

AccountStorage stores data of every MYKEY account, including a set of public keys, emergency contacts, delayed actions and multi-sig proposals. Only invocations sent from authorized Logic Module are allowed to add, modify or delete data in *AccountStorage*.

Mapping	Data	Mapping Structure
`keyData`	Public keys	account address => index => public key struct (including key and status)
`backupData`	Emergency contacts	account address => index => emergency contact struct (including address, effective time and expiry time)
`delayData`	Delayed actions	account address => action ID => delay action struct (including action hash and due time)
`proposalData`	Multi-sig proposals	client address => proposer address => action ID => proposal struct (including action hash and array of backups who have approved the proposal)

2.1.3.1 Public Keys

The public keys (in Ethereum address format) are stored in `keyData`. And default status is 0, frozen status is 1.

2.1.3.2 Emergency Contacts

Currently only MYKEY accounts can be emergency contacts. Therefore, the emergency contact addresses strored in `backupData` should be MYKEY accounts. And the effective time and expiry time are timestamps as seconds since unix epoch.

2.1.3.3 Delayed Actions

In `delayData`, action ID is 4 bytes long, e.g. `bytes4(keccak256("changeAdminKey(address,address)"))`.

Delayed action is stored as action hash and due time. Action hash is a hash of delayed action and its arguments, e.g. `keccak256(abi.encodePacked('changeAdminKey', _account, _pkNew))`. Due time is a future timestamp. Anyone can trigger the delayed action as long as it's due, no signature verification is needed.

2.1.3.4 Multi-sig Proposal

In multi-sig proposals, "client" is the user, and "backup" is the user's emergency contact. And "proposer" represents the one who proposed the proposal, either client or one of his backups.

There are 2 kinds of proposals:

1. `proposeByBoth`: proposed together by client and one of client's backups, two signatures required. In this case, proposer is client.
2. `proposeAsBackup`: proposed solely by one of client's backups. In this case, proposer is backup.

Once a proposal is proposed, other backups can approve it. Backups who have approved the proposal will be stored in an array. As long as the threshold is met, anyone can execute the proposal without signature verification.

In `proposalData`, action hash is a hash of proposed action and its arguments, e.g. `keccak256(abi.encodePacked('changeAdminKeyWithoutDelay', account, pkNew))`

2.1.4 Logics

Except *ProposalLogic* and *CommonStaticLogic*, each logic module has an entry method (`function enter()`), in which signature verification is performed before calling specific actions.

Logic	Function	Actions called from `enter`
AccountLogic	Logic of account management	`changeAdminKey`, `addOperationKey`, `changeAllOperationKeys`, `freeze`, `unfreeze`, `removeBackup`, `cancelDelay`, `cancelAddBackup`, `cancelRemoveBackup`, `proposeAsBackup`, `approveProposal`, `cancelProposal`
DualsigsLogic	Logic of actions requiring two signatures (client's and backup's)	`addBackup`, `proposeByBoth`
TransferLogic	Logic of asset transfer	`transferEth`, `transferErc20`, `transferApprovedErc20`, `transferNft`, `transferApprovedNft`
DappLogic	Logic of interacting with external contracts	`callContract`, `callMultiContract`

Logic	Function	Actions
ProposalLogic	Logic of proposal executing	`executeProposal`, `changeAdminKeyByBackup`, `triggerChangeAdminKeyByBackup`, `changeAdminKeyWithoutDelay`, `changeAllOperationKeysWithoutDelay`, `unfreezeWithoutDelay`
CommonStaticLogic	Logic of static call enabling	`isValidSignature`, `onERC721Received`

Details of the actions requiring signature verification are listed below:

Action	Description	Signing Key	Delay Time
`changeAdminKey`	change admin key	admin key	21 days

changeAllOperationKeys	change all opration keys	admin key	7 days
addOperationKey	add operation key	adding key	-
freeze	freeze all operation keys	admin key	-
unfreeze	unfreeze all operation keys	admin key	7 days
removeBackup	remove backup	admin key	21 days
cancelDelay	cancel delayed action	admin key	-
cancelAddBackup	cancel adding backup	admin key	-
cancelRemoveBackup	cancel removing backup	admin key	-
approveProposal	approve a proposal	assist key	-
proposeAsBackup	backup proposes a proposal	assist key	-
cancelProposal	cancel proposal	admin key	-
addBackup	add backup	client's admin key and backup's assist key	21 days
proposeByBoth	propose a proposal by client and one of his backups	client's admin key and backup's assist key	-
transferEth	transfer ETH	asset key	-
transferErc20	transfer ERC20 token	asset key	-
transferApprovedErc20	transfer approved	asset key	-

	ERC20 token		
`transferNft`	transfer ERC721 NFT	asset key	-
`transferApprovedNft`	transfer approved ERC721 NFT	asset key	-
`callContract`	call external contract	reserved key	-
`callMultiContract`	call multiple external contracts	reserved key	-

2.1.5 Signing and Verifying

According to EIP191, the raw message for signing is spliced as this: `0x19` + logic contract address + action data (including method ID and arguments) + nonce.

The first 4 bytes of action data is method ID, followed by arguments. And the first argument must be the account address of signer.

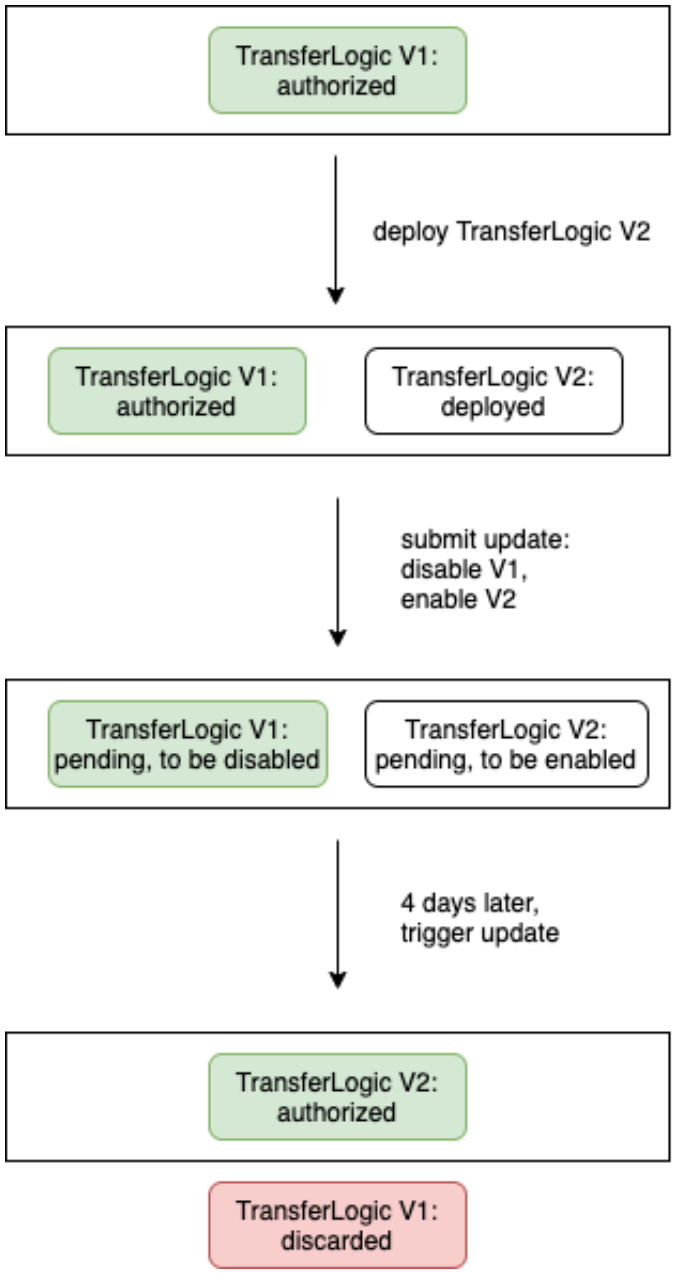
Every logic contract has a `keyNonce` mapping (public key => nonce). Nonce is current timestamp (in microsecond), passed in as argument when calling `function enter()`, in order to prevent replay attack.

When verifying signature, it is required that nonce is greater than the old nonce stored in `keyNonce` and not greater than current timestamp plus 24 hours (prevent using an infinite timestamp). If signature verification gets passed, the nonce will be stored in `keyNonce` to replace the old one.

2.1.6 Logic Upgrade

LogicManager maintains the storage of a list of authorized logics. The *LogicManager* owner (a multi-sig wallet) can disable an existing logic or enable a new one, but there is a time delay. The update can be triggered as long as the pending time is due.

The procedure of logic upgrade is shown below:

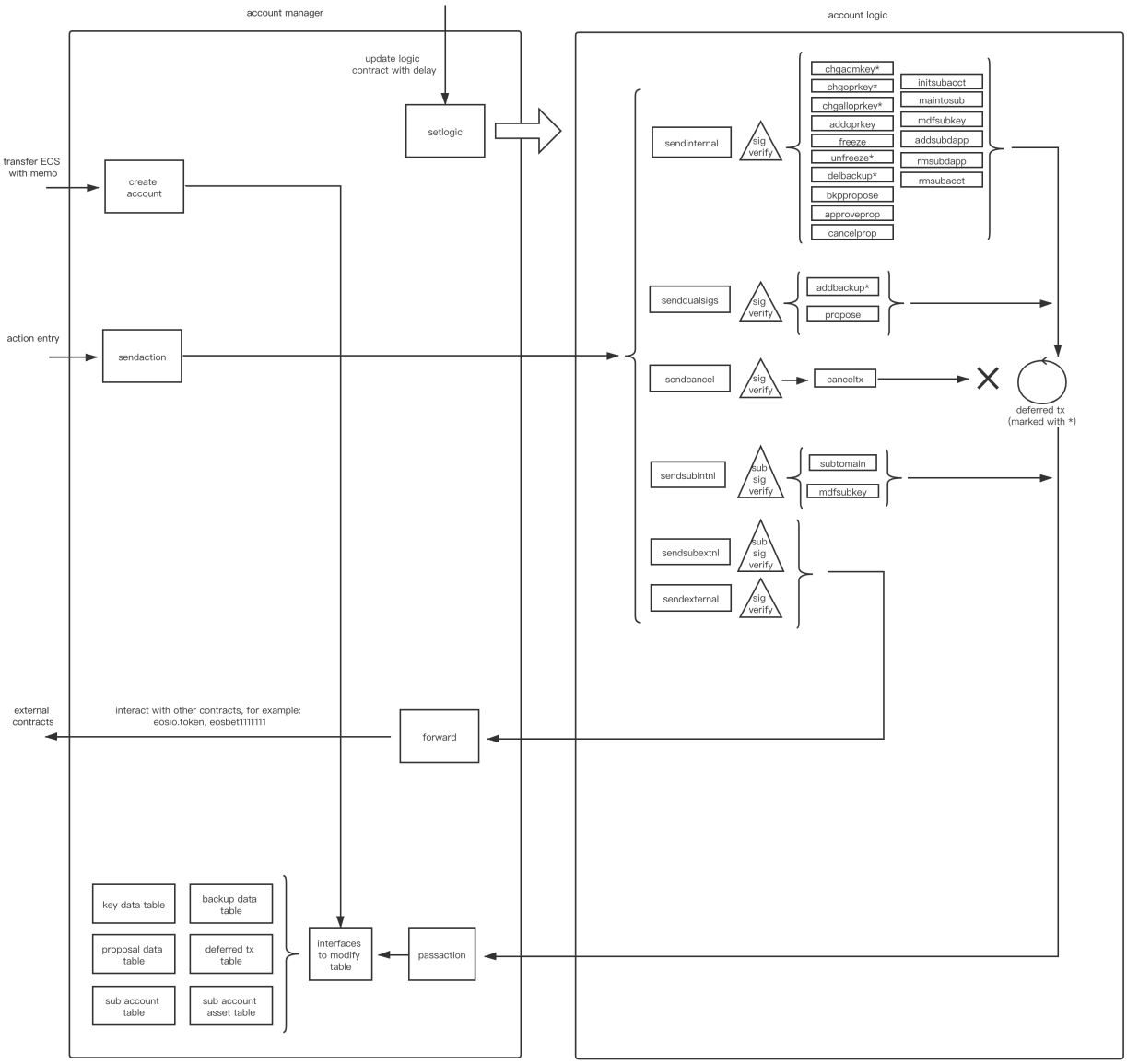


2.2 Implementation on EOS

2.2.1 Smart Contracts Architecture

MYKEY EOS contract is composed of two parts: *AccountManager* and *AccountLogic*.

AccountManager mainly manages account creation and data storage. Whereas specific logics are implemented in *AccountLogic*.



2.2.2 Account Creation

Anyone can create a MYKEY account by transferring some EOS token (about 0.3 EOS) to *AccountManager* with a memo containing messages in specified format: start with "create:", then new account name, emergency contact (optional) and 4 public keys (1 admin key and 3 operation keys) separated by "-". e.g. `create:newaccount11-defaultbkp11-EOS6JJM6qxxx-EOS6JJM6qxxx-EOS6JJM6qxxx-EOS6JJM6qxxx`

2.2.3 Data Storage

There are mainly 6 data tables in *AccountManager*:

Table Name	Description	Scope	Primary Key
`keydata`	Stores keys, admin key at index 0, followed by operation keys. Every key has a nonce, self-incremented after signature verification	account name	Index
`backupdata`	Stores emergency contacts, from index 0 to 5, at most 6 backups	account name	Index
`propdata`	Stores proposals, use `prop_id` as secondary index	account name	Self-incremented
`defdata`	Stores deferred tx (for manually triggering)	account name	Action ID (high 32 bits of action name + low 32 bits of index)

2.2.4 Signing and Verifying

The raw message for signing contains signer account name, action name, data bytes of action and nonce of the signing key, separated by ":", e.g. `mykeyuserbbb:transfer:00a6823403ea305xxx:1`.

If signature verification is passed, the nonce of signing key will be incremented by 1.

2.2.5 Deferred Transaction

Before sending a deferred transaction, the deferred ID, due time and transaction data will be stored in `defdata` table. In most cases, a deferred transaction will be executed automatically and the stored deferred data will be cleared. If a deferred transaction is not executed when time is due, anyone can manually trigger it by calling `kickdeferred`.

2.2.6 Logics

There are 6 action entries in *AccountLogic*. Signature verification is performed in these entry functions.

Entry Function	Description	Actions called from this entry
`sendinternal`	calling internal actions	`chgadmkey`, `chgoprkey`, `chgalloprkey`, `addoprkey`, `freeze`, `unfreeze`, `delbackup`, `approveprop`, `bkppropose`, `cancelprop`
`sendexternal`	calling actions in external contracts	e.g. calling token contract `eosio.token`
`sendcancel`	calling cancelling actions	`canceltx`
`senddualsigs`	calling actions requiring two signatures	`addbackup`, `propose`

Details of these actions are listed below:

Action	Description	Signing Key	Delay Time
`chgadmkey`	change admin key	admin key	21 days
`chgoprkey`	change operation key	admin key	7 days
`chgalloprkey`	change all opration keys	admin key	7 days
`addoprkey`	add operation key	adding key	-
`freeze`	freeze all operation keys	admin key	-
`unfreeze`	unfreeze all operation keys	admin key	7 days
`delbackup`	delete backup	admin key	21 days
`approveprop`	approve a proposal	assist key	-
`bkppropose`	backup proposes a proposal	assist key	-
`cancelprop`	cancel proposal	admin key	-
`addbackup`	add backup	client's admin key and backup's assist key	21 days
`propose`	propose a proposal by client and backup	client's admin key and backup's assist key	-
`canceltx`	cancel deferred transaction	admin key	-

2.2.7 Logic Upgrade

AccountLogic contract is upgradable. Its account name is stored in `logic` table of *AccountManager*. A multi-sig controlled account (`logicupdater`) is able to upgrade *AccountLogic* by changing the stored logic contract name. New logic contract will take effect in 4 days.