TON Macro Assembler Multi-Signature Wallet

HOWTO

1. **Using keys-mswallet.fif** collect 'nkey' (0 < nkey <= 100) public keys from all multi-signature wallet members. The public key array **<filename-base>.pubk** created by the first member need to be send to the second one and so on. The last member has the full **<filename-base>.pubk** array with all 'nkey' public keys collected. Each member get a unique key index in the key array.
2. Someone (it can be the 3rd party person) build a multi-signature wallet creating query using **new-mswallet.fif** with the following parameters: **<workchain-id> <filename-base> <nkey> <nsig>** The public key array will be loaded from **<filename-base>.pubk** that must contain **<nkey>** number of public keys collected on the step 1.
3. Transfer some Grams to wallet address.
4. Send the wallet creating query to TON blockchain to load multi-signature wallet to blockchain (Lite Client sendfile).
5. Using **mswallet.fif** create message query. Any wallet member can initiate the process and sign the query. The message query 'boc' file can be sent to other members to collect other signatures or send to TON blockchain otherwise. The **mswallet.fif** will check if the message parameters including destination address, sequence number, amount, message expiry time and message body (if any) for each member corresponds to the initial query.
6. Multi-wallet signature smart contract will receive an inbound external messages related to the particular internal message, check the signature and if number of signatures is equal to **<nsig>** send internal message. If number of signatures is less than **<nsig>** the smart contract will store the message in the message array to send the internal message later if new query with other signatures arrived.
7. Using the multi signature smart contact get methods any one can receive the sequence number for the next message, the public keys array and stored message array. The message array can be processed later by the **msg-mswallet.fif** utility to select messages signed or not signed by particular member.

HOW IT WORKS

The macro assembler multi signature wallet works with 3 data structures : KeyArray, SigArray and MsgArray (KeyArray.fif, SigArray.fif, and MsgArray.fif files).

key[]

Key Array is regular tree structure that must be constructed before the wallet smart contract creating query and it's the part of smart contract initial state. The goal of the KeyArray is to optimise public key fetching time in the main cycle of smart contact where the message signatures are checked for each public key. The signature arrived in an inbound message as SigArray where for each signature there is a 2 byte key index. Using the key index 'KEY@' macros calculate the unique path for the given public key in the 'boc' tree KeyArray structure. For total number of keys <= 100 each path will be not more than 4 integers that give us to 4 cell uploads for extracting public key. Compare to the linear array where all cells connected with the only one reference, the key fetching for index near the linear array length can demand 8 times more cell

fetching than in the KeyArray case. Please not that KeyArray is mutable only before smart contract is created and we don't need to rebuild it in the multi signature wallet smart contract.

sig[]

SigArray is linear array or more precisely stack structure that is created during the message signing procedure when all multi signature wallet members are signing inertial message with own private keys. For each private key the public key can be calculated and the key index found in KeyArray stored for each member (**<filename-base>.pubk**). The member copy of KeyArray could be uncompleted but must contain at least the member public key. The full copy of KeyArray can be requested from the multi signature smart contract (keyarray get-method). As soon as the member public key index was calculated with hash value of the message the new entry for SigArray can be added ('sig&') to the head of the array. During signature checking cycle the smart contract will fetch the first element of the array ('SIG@') and check signature based on the key index stored with given signature.

msg[]

Message array is linear array that store messages in smart contract. Along with internal message the message array stores the message hash (256 bits), message expiry time (unix time 32 bits) and signature mask ('nkey' bits). When a new inbound message arrive that has sequence number less than current sequence number stored in the smart contract the smart contract try to find the message using the message hash in the stored message array ('MSG?'). If message is found the signature checking procedure will be initiated, and if not then the smart contract will throw the exception. The signature checking macros ('SIGCHECK') use the stored signature mask to check signature only for key that was not checked before to save processing time. The new signature mask will be calculated at the end of the signature checking process and if number of bits in the final mask is more than 'nsig' (k) parameter of the smart contract than internal message will be send to the destination address. If not, the new signature mask will be stored in the updated message array. We add new / updated message always on the head of MsgArray. The message array will store the nearest expire time for messages in the array to optimise the garbage collection procedure. If the stored nearest expire time (net) is less of the current time the garbage collection procedure will be initiated and the new 'net' value will be calculated. If the internal message is sent then the MsgArray must be updated and the sent message must be deleted. In this case we don't update the stored 'net' value but the smart contract will also check the number of messages stored and if the number of messages in the array id more then fixed value (10 fo default) the garbage collection will be initiated and new 'net' value will be calculated.

MSWALLET smart contract

The MSWALLET smart contract (MSWallet.fif) support the following methods :

Selector	Name	Change state	Description
85143	get-seqno	N	Get stored sequence number for the next query
85144	get-msgarray	N	Get stored message array
85145	get-keyarray	N	Get stored key array
-1	main	Y	Main multi signature wallet processing cycle

Smart Contract Test

The multi signature wallet smart contact can be tested using mswallet-test.fif. Just copy content of the file and paste it in the FIFT interpreter. The test code includes an external message preparation, initial contract state and stub code for TVM initial state before the smart contract will be processed.

PACKAGE CONTENT

Utilities: keys-mswallet.fif, new-mswallet.fif, mswallet.fif , msg-mswallet.fif
Libraries: KeyArray.fif, SigArray.fif, MsgArray.fif, MSWallet.fif
Test: mswallet-test.fif

DICTIONARY

Marcos	File	Description
key[]	KeyArray.fif	create empty array of keys with 'keyn' max number of elements
key#@	KeyArray.fif	@ max key number in the array and store it in 'pubkn' constant
key[]>	KeyArray.fif	print out the array
key#	KeyArray.fif	return number of key in the array
key,	KeyArray.fif	insert key in array in given index
key&	KeyArray.fif	append key to the array
key@	KeyArray.fif	@ key from array for given index
key@+	KeyArray.fif	@ key from array for given index and keep array in the stack
key?	KeyArray.fif	find the key in the array

Marcos	File	Description
key?+	KeyArray.fif	find the key in the array and keep array in the stack
B>u	KeyArray.fif	BYTE {0..256} to uint256 converter
KEY@	KeyArray.fif	TVM: key@
sig[]	SigArray.fif	create empty array of signatures with 'keyn' max number of keys and signum - max signature to store
sig&	SigArray.fif	add new signature to the array
sig[]>	SigArray.fif	print out the array
sig#	SigArray.fif	return array length
sigMSK	SigArray.fif	return signatures mask of the array
sigMSK#	SigArray.fif	return number of signatures in the given mask
i>m	SigArray.fif	key index to key mask
sig@	SigArray.fif	fetch the first signature from the array and return the rest of array
SIG[]	SigArray.fif	TVM: sig[]
SIG#	SigArray.fif	TVM: sig[]
SIGMSK	SigArray.fif	TVM: sigMSK
SIGMSK#	SigArray.fif	TVM: sigMSK#

Marcos	File	Description
I>M	SigArray.fif	TVM: i>m
SIG@	SigArray.fif	TVM: sig@
SIGCHECK	SigArray.fif	TVM: check signatures (sigarray) for given keyarray, hash and mask
msg[]	MsgArray.fif	create empty array of message for given 'keynumber' and 'signum'
msg&	MsgArray.fif	add new message to the array
msg?	MsgArray.fif	find message for given hash and return message envelop (emsg)
msg@	MsgArray.fif	@ message from the message envelop
msgX	MsgArray.fif	delete message for given hash from array
msgXT	MsgArray.fif	clean array based on unix time input parameter
msgU	MsgArray.fif	update signature mask for message envelop for given hash
msg[]>	MsgArray.fif	printout the array
msgSEL	MsgArray.fif	select messages from the array for given key index and selector = { true | false } and return selected messages array
MSG[]	MsgArray.fif	TVM: msg[]

Marcos	File	Description
MSG&	MsgArray.fif	TVM: msg&
MSG?	MsgArray.fif	TVM: msg?
MSG@	MsgArray.fif	TVM: msg@
MSGX	MsgArray.fif	TVM: msgX
MSGXT	MsgArray.fif	TVM: msgXT
MSGU	MsgArray.fif	TVM: msgU
MSWALLET	MSWallet.fif	TVM: multi signature wallet smart contract

https://github.com/sevriugin/mswallet/blob/master/README.md

12 Ottobre 2019
Foresto Sparso (BG)
Italia