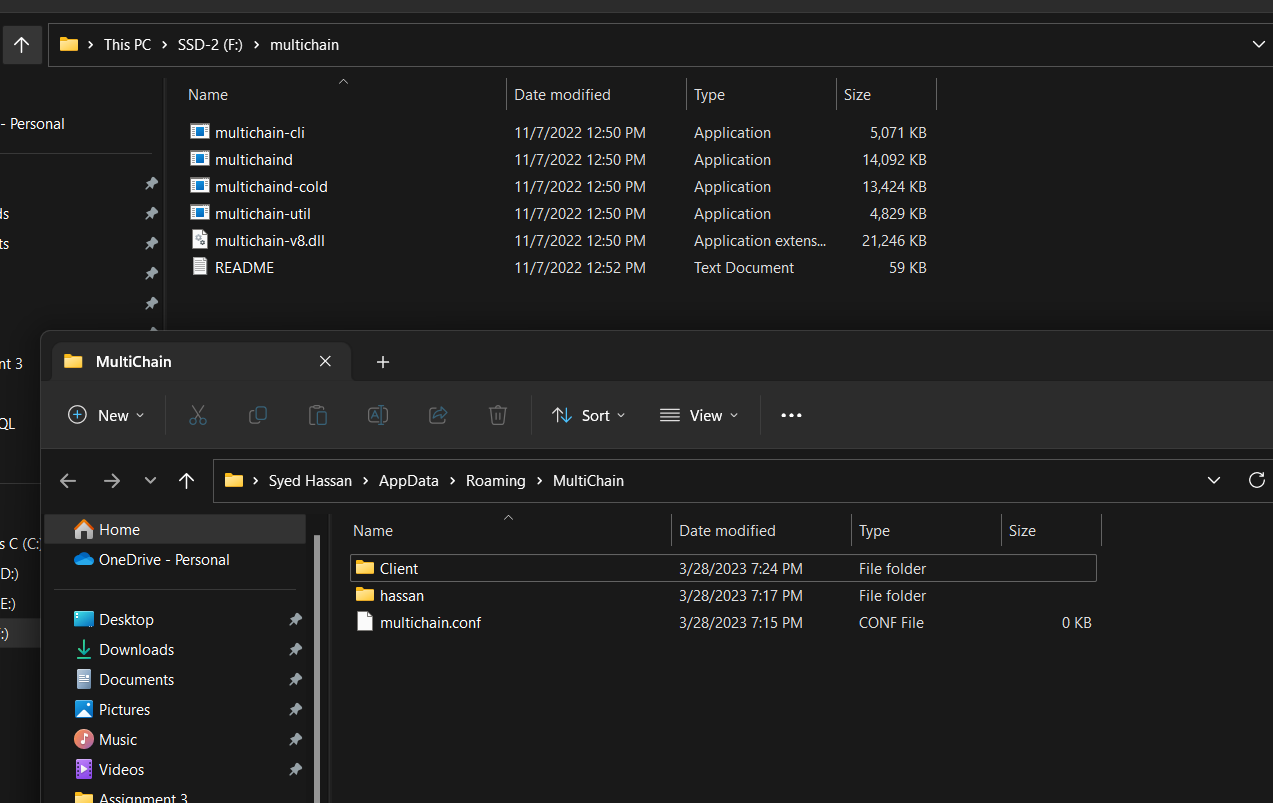
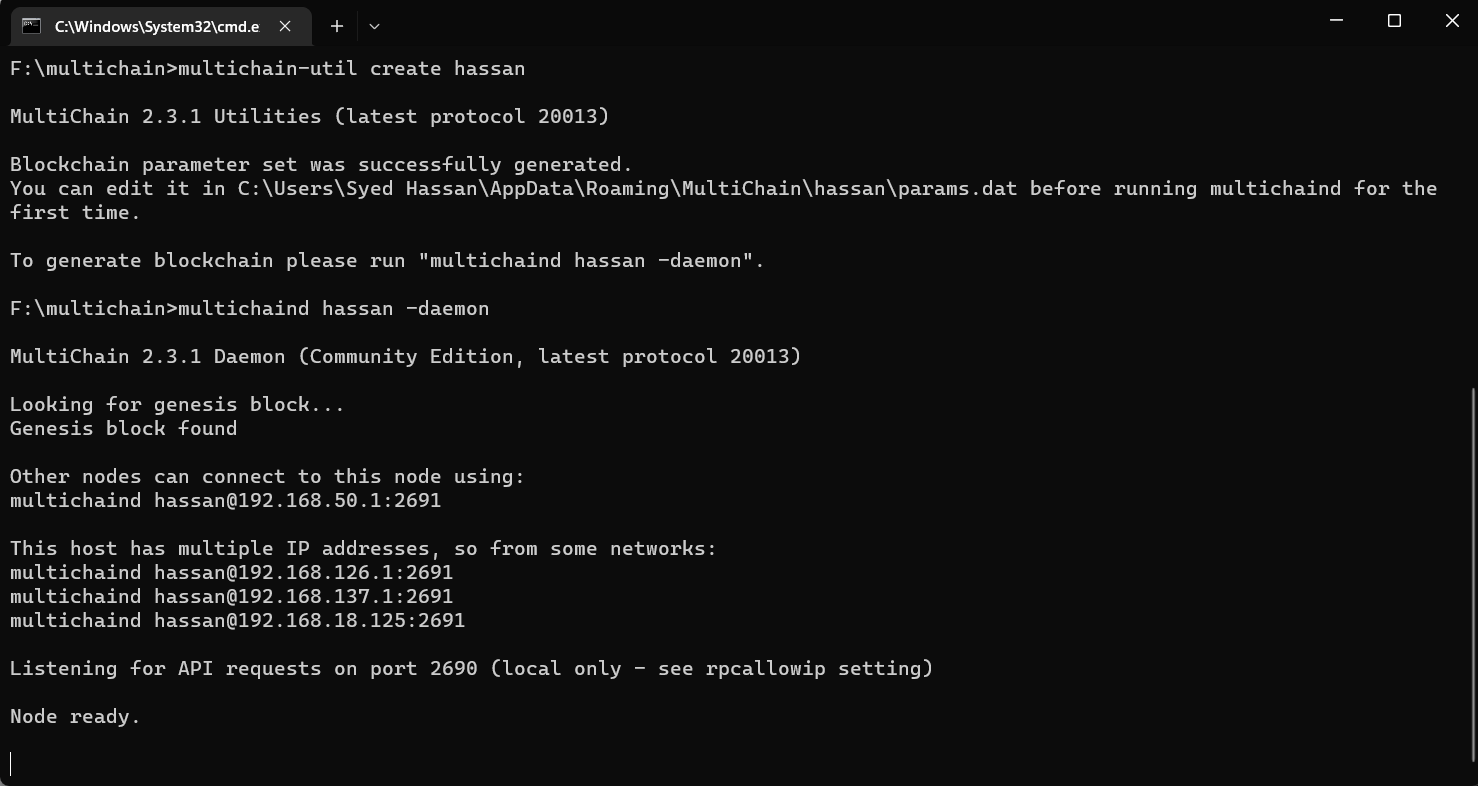
**Multichain Wallet Lab - Multisignature**

**K20-1052**

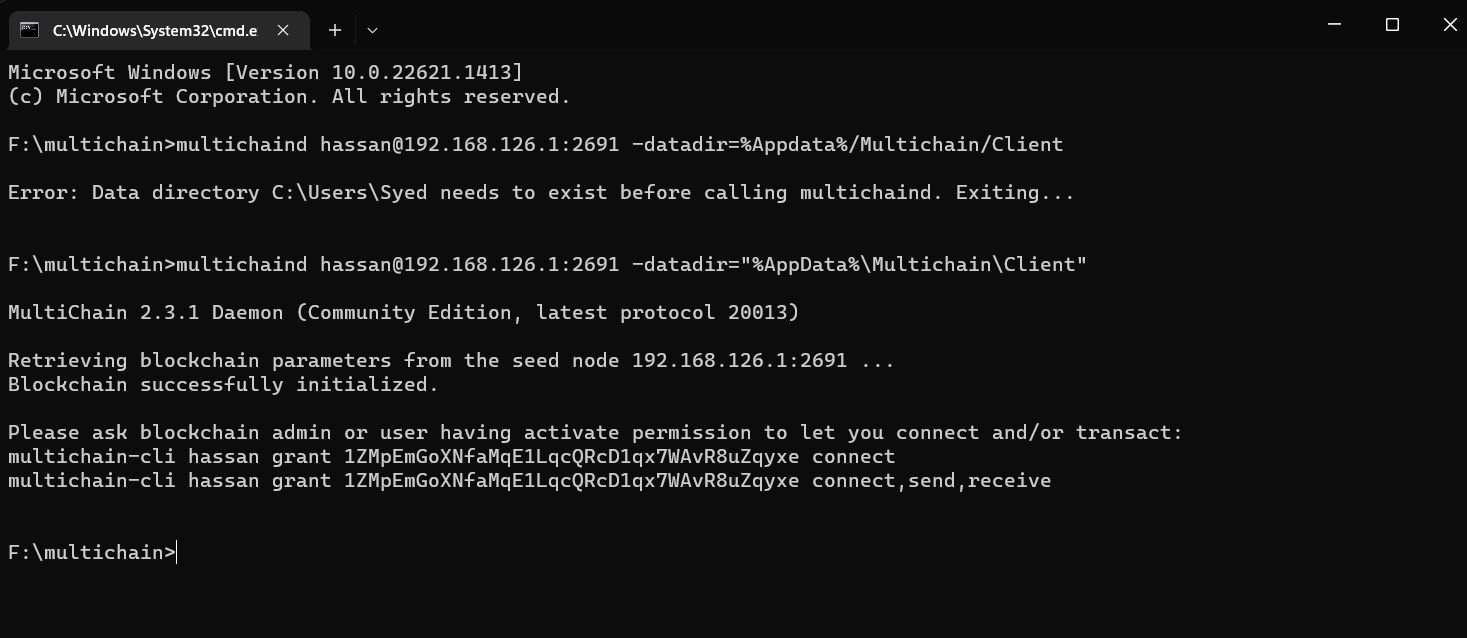
**BSE-6B**

**S.M.HASSAN ALI**





**multichaind [hassan@192.168.50.1:2691](mailto:hassan@192.168.50.1:2691)**



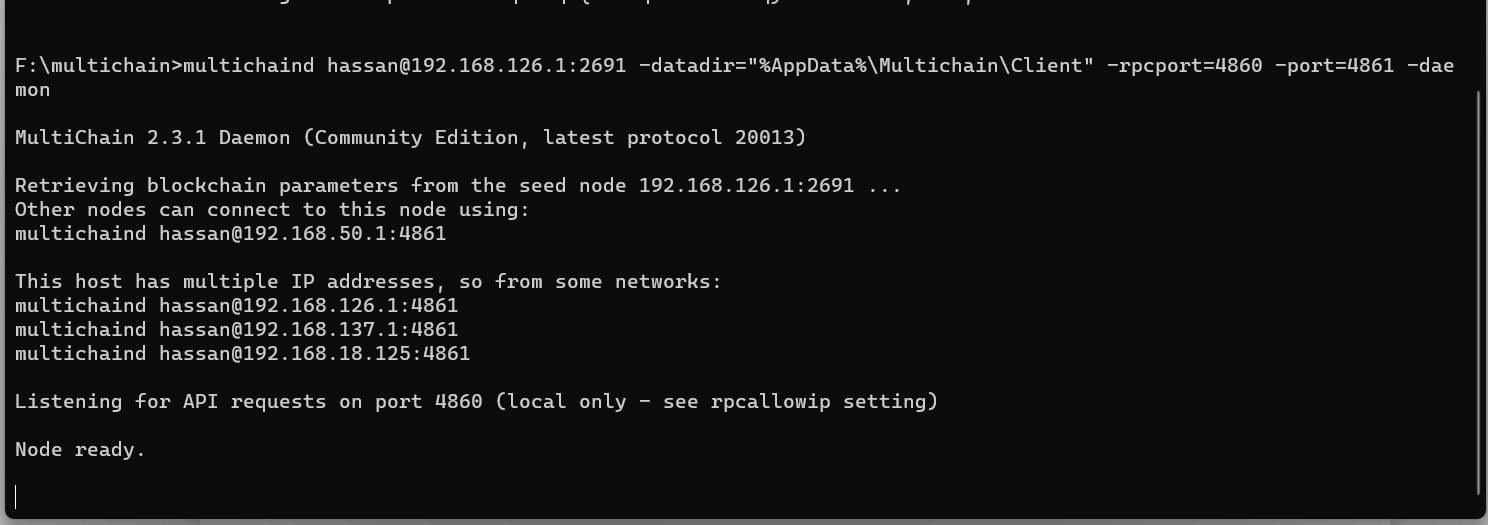
**Please ask blockchain admin or user having activate permission to let you connect and/or transact:**

**multichain-cli hassan grant 1ZMpEmGoXNfaMqE1LqcQRcD1qx7WAvR8uZqyxe connect**

**multichain-cli hassan grant 1ZMpEmGoXNfaMqE1LqcQRcD1qx7WAvR8uZqyxe connect,send,receive**



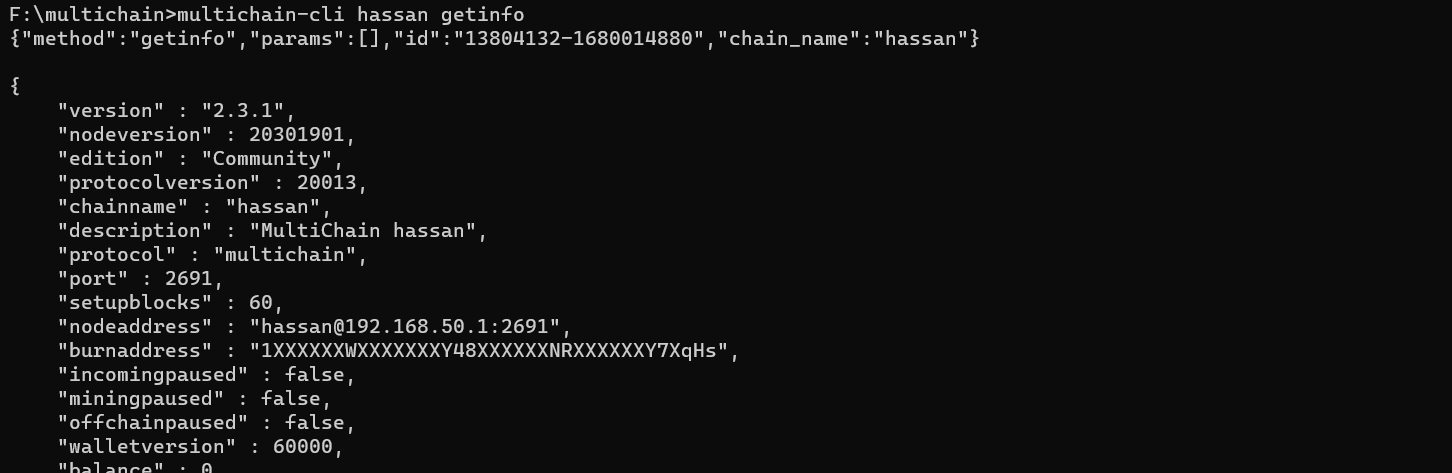
**1cf679aa3cd3b5655f28288ce0763f2352ff3aad942e8f3eae9b5b32f53d6560**



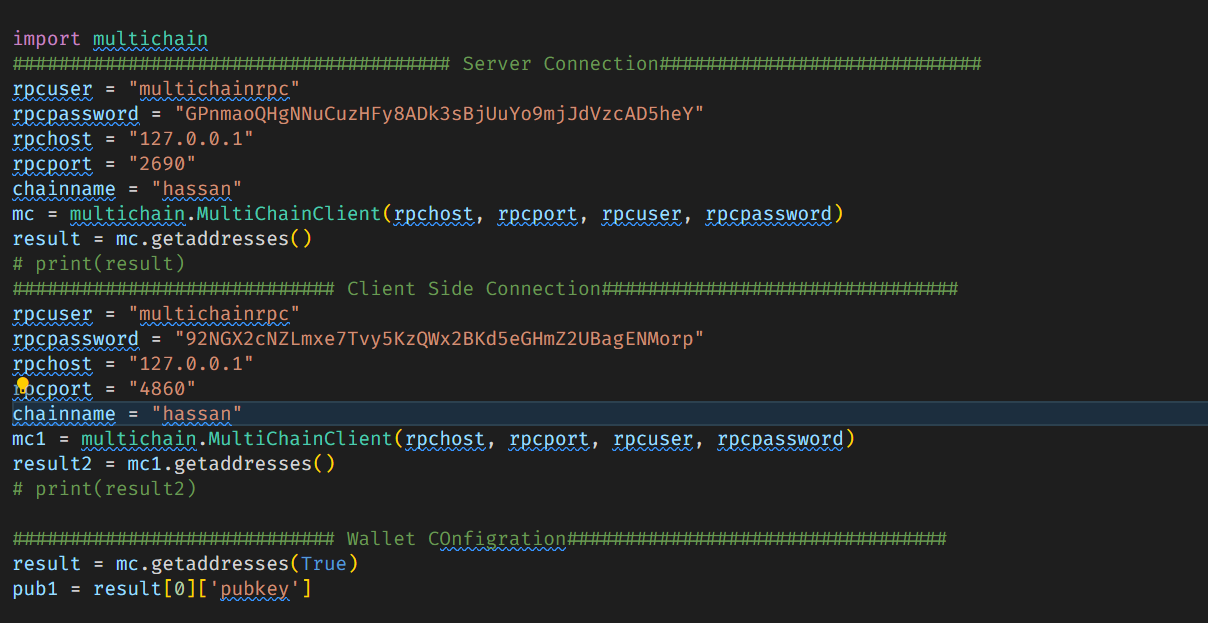
-rpcport=4860 –port=4861 –daemon

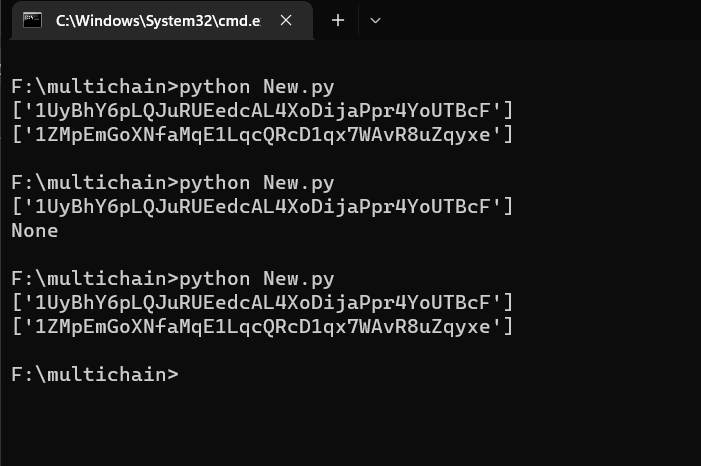
The highlighted command is provide a TCP connection to different port for the client node

1. **Run the command getinfo inorder to get the chain information of client node and server node**



**Now Connect the client and Server with Python Code**





**Transaction Code**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Mar  6 17:45:30 2023

@author: Administrator

"""

import multichain

###################################### Server Connection############################

rpcuser = "multichainrpc"

rpcpassword = "GPnmaoQHgNNuCuzHFy8ADk3sBjUuYo9mjJdVzcAD5heY"

rpchost = "127.0.0.1"

rpcport = "2690"

chainname = "hassan"

mc = multichain.MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result = mc.getaddresses()

print(result)

############################ Client Side Connection###############################

rpcuser = "multichainrpc"

rpcpassword = "92NGX2cNZLmxe7Tvy5KzQWx2BKd5eGHmZ2UBagENMorp"

rpchost = "127.0.0.1"

rpcport = "4860"

chainname = "hassan"

mc1 = multichain.MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result2 = mc1.getaddresses()

print(result2)

############################ Wallet COnfigration#################################

result = mc.getaddresses(True)

pub1 = result[0]['pubkey']

# r=mc1.getnewaddress()

result2 = mc1.getaddresses(True)

pub2 = result2[0]['pubkey']

########################### Create Walletinformation Create on ServerSide###########################

address = mc.addmultisigaddress(2, [pub1, pub2])

address1 = mc1.addmultisigaddress(2, [pub1, pub2])

txid1 = mc.grant(address, 'send,receive')  # global permission

txid12 = mc1.grant(address, 'send,receive')  # global permission

txid2 = mc.issue(address, {'name': 'MUltichainwallet'}, 1000, 0.01)

result32 = mc.getaddressbalances(address, 0)

print("Blance", result32)

########################### This particular address send and recevieammount on Serverside###

result = mc.getaddresses(True)

txid = mc.grant(result[0]['address'], 'send,receive')

print("Address Two ", result[0]['address'])

result69 = mc.createrawsendfrom(

    address, {result[0]['address']: {"MUltichainwallet": 150}}, [], 'sign')

Signature2 = result69['hex']

result123 = mc1.signrawtransaction(Signature2)

print(result123)

txid = mc.sendrawtransaction(result123['hex'])

print(txid)



#### Creating the multisignature address

On the first server, run the following command:

getaddresses true

Choose any address with "ismine" : true, which means that this node’s wallet contains the private key for the address.

Copy and paste the pubkey shown: ****

Now run the same getaddresses true command on the second server, again choosing an address with "ismine" : true.

Copy and paste the pubkey shown: ****

Now run the following command on either server to create the 2-of-2 multisig address and add it to the node’s wallet:

addmultisigaddress 2 '["0...", "0..."]'

The response contains the multisignature address. Copy and paste it here: ****

The response should be empty. Now run the same command on the other server, to add the address to the wallet and start tracking its balance:

addmultisigaddress 2 '["0...", "0..."]'

#### Issuing an asset to the multisig address

For most blockchain actions, a multisig address requires its own permissions, independent of the permissions of the individual regular addresses that were combined to create it ([more details here](https://www.multichain.com/developers/permissions-management/)). Let’s grant these permissions on the first server:

grant 4... receive,send

The txid of the grant transaction should be displayed in the response. Now let’s issue a new asset directly to the multisig address:

issue 4... asset9 10000 0.01

And now let’s check the multisig address has received the funds successfully:

getaddressbalances 4... 0

A balance of 10000 units of asset9 should be displayed.

#### Spending funds from the multisig address

Still on the first server, let’s create a new regular address to receive some funds from the multisig:

getnewaddress

Copy and paste the new address here: ****

Now let’s grant this address receive permissions, so it can be sent some funds:

grant 1... receive

Now we begin the process of building the transaction which sends funds from the multisig address to this new address. Because this is a 2-of-2 multisig, the process will require a signature from both servers. Let’s begin on the first server:

createrawsendfrom 4... '{"1...":{"asset9":500}}' '[]' sign

The response should contain a complete field with value false, along with a large hexadecimal blob in the hex field. This hexadecimal blob is the raw transaction, which has been partially signed, and should be copied to the clipboard.

Now switch to the second server and run the following, pasting the raw transaction from the clipboard where shown:

signrawtransaction [paste-hex-blob]

The response should contain a complete field with value true, along with an even larger hexadecimal blob in the hex field. This means that the transaction has enough signatures to be valid, and is ready for broadcasting to the blockchain. Copy the new hexadecimal blob, and run:

sendrawtransaction [paste-bigger-hex-blob]

The response should contain the 64-character hexadecimal txid of the sent transaction. Now let’s check that the 500 units of asset9 have been successfully transferred. On either server:

getaddressbalances 4... 0

And on the first server only, check the new address’s balance (including unconfirmed transactions):

getaddressbalances 1... 0

**CODE:**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Mar  6 17:45:30 2023

@author: Administrator

"""

import multichain

###################################### Server Connection############################

rpcuser = "multichainrpc"

rpcpassword = "GPnmaoQHgNNuCuzHFy8ADk3sBjUuYo9mjJdVzcAD5heY"

rpchost = "127.0.0.1"

rpcport = "2690"

chainname = "hassan"

mc = multichain.MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result = mc.getaddresses()

print(result)

############################ Client Side Connection###############################

rpcuser = "multichainrpc"

rpcpassword = "92NGX2cNZLmxe7Tvy5KzQWx2BKd5eGHmZ2UBagENMorp"

rpchost = "127.0.0.1"

rpcport = "4860"

chainname = "hassan"

mc1 = multichain.MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result2 = mc1.getaddresses()

print(result2)

############################ Wallet COnfigration#################################

"""

getaddresses true

Choose any address with "ismine" : true, which means that this node’s

wallet contains the private key for the address.

"""

result = mc.getaddresses(True)

pub1 = result[0]['pubkey']

"""

Now run the same getaddresses true command on the second server,

 again choosing an address with "ismine" : true.

"""

# r=mc1.getnewaddress()

result2 = mc1.getaddresses(True)

pub2 = result2[0]['pubkey']

# print("",pub1,pub2)

########################### Create Walletinformation Create on ServerSide###########################

"""

addmultisigaddress 2 '["0...", "0..."]'

"""

address = mc.addmultisigaddress(2, [pub1, pub2])

print(address)

"""

The response should be empty. Now run the same command on the other server,

to add the address to the wallet and start tracking its balance:

addmultisigaddress 2 '["0...", "0..."]'

"""

address1 = mc1.addmultisigaddress(2, [pub1, pub2])

print(address1)

"""

Issuing an asset to the multisig address

For most blockchain actions, a multisig address requires its own

permissions, independent of the permissions of the individual regular

addresses that were combined to create it (more details here).

Let’s grant these permissions on the first server:

grant 4... receive,send

The txid of the grant transaction should be displayed in the response. Now let’s issue a new asset directly to the multisig address:

"""

txid1 = mc.grant(address, 'receive,send')  # global permission

# txid12 = mc1.grant(address, 'send,receive') # global permission

"""

The txid of the grant transaction should be displayed in the response.

Now let’s issue a new asset directly to the multisig address:

issue 4... asset9 10000 0.01

And now let’s check the multisig address has received the

funds successfully:

getaddressbalances 4... 0

A balance of 10000 units of asset9 should be displayed.

"""

txid2 = mc.issue(address, {'name': 'Walletx'}, 1000, 0.01)

result32 = mc.getaddressbalances(address, 0)

print("Blance", result32)

########################### This particular address send and recevieammount on Serverside###

""""

Spending funds from the multisig address

Still on the first server, let’s create a new regular address to receive

some funds from the multisig:

getnewaddress

Copy and paste the new address here:

1...

Now let’s grant this address receive permissions, so it can be sent some funds:

grant 1... receive

"""

txid = mc.grant(result[1]['address'], 'send,receive')  # global permission

print("Address Two ", result[1]['address'])

print(address)

print(result[1]['address'])

"""

Now we begin the process of building the transaction which sends

funds from the multisig address to this new address.

Because this is a 2-of-2 multisig, the process will require a

signature from both servers. Let’s begin on the first server:

createrawsendfrom 4... '{"1...":{"asset9":500}}' '[]' sign

"""

result69 = mc.createrawsendfrom(

    address, {result[1]['address']: {"Walletx": 90}}, [], 'sign')

# print(result69)

Signature2 = result69['hex']

print(Signature2)

"""

The response should contain a complete field with value false, along with a large hexadecimal blob in the hex field. This hexadecimal blob is the raw transaction,

which has been partially signed, and should be copied to the clipboard.

Now switch to the second server and run the following,

pasting the raw transaction from the clipboard where shown:

"""

"""

signrawtransaction [paste-hex-blob]

The response should contain a complete field with value true,

along with an even larger hexadecimal blob in the hex field.

This means that the transaction has enough signatures to be valid,

and is ready for broadcasting to the blockchain.

"""

result123 = mc1.signrawtransaction(result69['hex'])

print(result123)

"""

Copy the new hexadecimal blob, and run:

sendrawtransaction [paste-bigger-hex-blob]

"""

txid = mc.sendrawtransaction(result123['hex'])

print(txid)

"""

getaddressbalances 4... 0

And on the first server only, check the new address’s balance

(including unconfirmed transactions):

getaddressbalances 1... 0

"""

result32 = mc.getaddressbalances(address, 0)

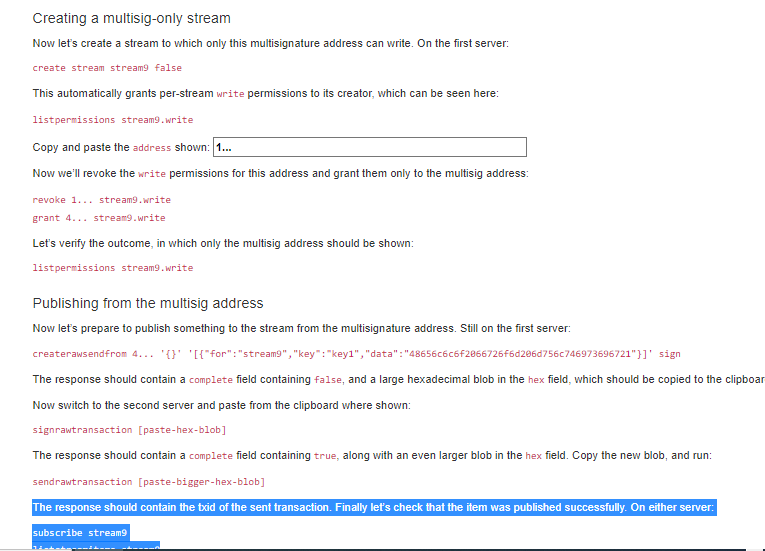
print("Blance", result32)

result32 = mc1.getaddressbalances(address, 0)

print("Blance", result32)



**For Creating Multisignature Stream Follow the Following steps from the Experiment**



**Code:**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Mar  6 17:45:30 2023

@author: Administrator

"""

import multichain

###################################### Server Connection############################

rpcuser = "multichainrpc"

rpcpassword = "GPnmaoQHgNNuCuzHFy8ADk3sBjUuYo9mjJdVzcAD5heY"

rpchost = "127.0.0.1"

rpcport = "2690"

chainname = "hassan"

mc = multichain.MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result = mc.getaddresses()

print(result)

############################ Client Side Connection###############################

rpcuser = "multichainrpc"

rpcpassword = "92NGX2cNZLmxe7Tvy5KzQWx2BKd5eGHmZ2UBagENMorp"

rpchost = "127.0.0.1"

rpcport = "4860"

chainname = "hassan"

mc1 = multichain.MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result2 = mc1.getaddresses()

print(result2)

############################ Wallet COnfigration#################################

"""

getaddresses true

Choose any address with "ismine" : true, which means that this node’s

wallet contains the private key for the address.

"""

result = mc.getaddresses(True)

pub1 = result[0]['pubkey']

"""

Now run the same getaddresses true command on the second server,

 again choosing an address with "ismine" : true.

"""

# r=mc1.getnewaddress()

result2 = mc1.getaddresses(True)

pub2 = result2[0]['pubkey']

# print("",pub1,pub2)

########################### Create Walletinformation Create on ServerSide###########################

"""

addmultisigaddress 2 '["0...", "0..."]'

"""

address = mc.addmultisigaddress(2, [pub1, pub2])

print(address)

"""

The response should be empty. Now run the same command on the other server,

to add the address to the wallet and start tracking its balance:

addmultisigaddress 2 '["0...", "0..."]'

"""

address1 = mc1.addmultisigaddress(2, [pub1, pub2])

print(address1)

"""

Creating a multisig-only stream

Now let’s create a stream to which only this multisignature address

can write. On the first server:

create stream stream9 false

This automatically grants per-stream write permissions to its creator,

which can be seen here:

listpermissions stream9.write

"""

txid = mc.create('stream', 'hassan', False)  # open to all to write

re1 = mc.listpermissions('hassan.write')  # all permissions

# print(result[0]['address'])

"""

Now we’ll revoke the write permissions for this address and

grant them only to the multisig address:

revoke 1... stream9.write

grant 4... stream9.write

"""

# txid = mc.revoke(result[0]['address'],'shahbaz1.write')

# print(txid)

txid = mc.grant(address, 'hassan.write')  # per-entity permission

print(txid)

"""

Publishing from the multisig address

Now let’s prepare to publish something to the stream from the

multisignature address. Still on the first server:

createrawsendfrom 4... '{}' '[{"for":"stream9","key":"key1","data":"48656c6c6f2066726f6d206d756c746973696721"}]' sign

The response should contain a complete field containing

false, and a large hexadecimal blob in the hex field,

which should be copied to the clipboard.

"""

re4 = mc.createrawsendfrom(address, {result[1]['address']: 0}, [

                           {'for': 'hassan', 'key': 'key1', 'data': 'a1b2c3d4'}], 'sign')

print(re4)

"""

Now switch to the second server and paste from the clipboard where shown:

signrawtransaction [paste-hex-blob]

"""

Signature2 = re4['hex']

print(Signature2)

result123 = mc1.signrawtransaction(re4['hex'])

print(result123)

"""

The response should contain a complete field containing true, along with an

even larger blob in the hex field. Copy the new blob, and run:

"""

"""

sendrawtransaction [paste-bigger-hex-blob]

"""

txid = mc.sendrawtransaction(result123['hex'])

print(txid)

"""

The response should contain the txid of the sent transaction.

Finally let’s check that the item was published successfully.

On either server:

subscribe stream9

liststreamitems stream9

"""

Server = mc.subscribe('hassan')

Client = mc1.subscribe('hassan')

result = mc.liststreamitems('hassan')

print(result)

"""

The response should contain the txid of the sent transaction.

Finally let’s check that the item was published successfully.

 On either server:

subscribe stream9

liststreamitems stream9

You should see the item listed with the multisig address shown

in the publishers field, as well as the key and data entered above.

"""

