Using Ubikom CLI

Table of contents generated with markdown-toc

Prerequisites

You must have Go installed. If you don't, follow the instructions here.

In order to use blockchain-related functions, you tell ubikom-cli how to connect to an Ethereum node. One way to do it is to create an Infura account, and specify the project id as INFURA_PROJECT_ID environment variable. You can also:

- Pass your Infura project ID to any blockchain-related command as "–infura-project-id" flag;
- Pass the node URL as "–node-url" flag.

Install CLI

The easiest way to install Ubikom CLI is by using go get command:

```
$ go get github.com/regnull/ubikom/cmd/ubikom-cli
```

Or, if you've already cloned ubikom repo, do:

```
$ cd ubikom-directory/cmd/ubikom-cli
$ go install
```

The binary will appear under your $GOROOT/bin directory, by default it will be $HOME/go/bin. Either go to that directory or add it to path. Run ubikom-cli to see the list of available commands:

```
$ ubikom-cli allows you to run local and remote Ubikom commands

Usage:
  ubikom-cli [flags]
  ubikom-cli [command]

Available Commands:
  bc          Blockchain-related commands
  completion  Generate the autocompletion script for the specified shell
  create      Create various things
  disable     Disable something
  get         Get various things
  help        Help about any command
  lookup      Look stuff up
  receive     Receive stuff
  register    Register various things
  send        Send stuff

Flags:
  -h, --help   help for ubikom-cli

Use "ubikom-cli [command] --help" for more information about a command.
```

Output Format

When you request data using ubikom-cli, the output format will be either a string or JSON:

- If the data includes a single value (for example, your account balance), it's printed out as a string;
- If the data includes multiple values (for example, the result of blockchain interaction), it's printed out as JSON.

Create a Private Key

Before you can do pretty much anything, you need to create a private key. You can do it by using "create key" command:

```
$ ubikom-cli create key --out=secret.key
Passphrase (enter for none):
Confirm passphrase (enter for none):
15:43:48 WRN saving private key without passphrase
15:43:48 INF private key saved location=secret.key
```

The key will be saved as secret.key file in the current directory.

It is recommended that you use a passphrase when you create a key. If you don't, anyone with an access to this file will be able to use it right away. Which is not to say that using a passphrase would give you perfect protection - ubikom-cli is a practical tool, not an industrial-strength wallet. Never use it to protect your valuable crypto accounts.

Create Key Using Password

It is possible to construct a private key given two pieces of data: password and salt. This makes it possible, for example, to transmit a private key over as a "user name" and "password", where salt plays the role of "user name". Email clients use it to send the email key over to Ubikom proxy using POP3 or SMTP protocol.

```
$ ubikom-cli create key --from-password=supersecretpassword123 \
  --salt=123456 --out=secret1.key
Passphrase (enter for none):
Confirm passphrase (enter for none):
14:37:01 WRN saving private key without passphrase
14:37:01 INF private key saved location=secret1.key
```

If you later use this command with –salt flag and specify the same salt, you will end up with the same key.

Let's make sure this is the case. Generate another key using the same password and salt:

```
$ ubikom-cli create key --from-password=supersecretpassword123 \
  --salt=123456 --out=secret2.key
```

Now we can compute SHA256 hash of both files and compare the hashes:

```
$ sha256sum secret1.key
97a9a2...fa3fa5  secret1.key

$ sha256sum secret2.key
97a9a2...fa3fa5  secret2.key
```

Create Key from Mnemonic

Another way to create a private key is from mnemonic. A popular way to store a private key is to use a list of 24 words, as specified in BIP 39.

To re-create private key from mnemonic, use –from-mnemonic flag:

```
$ ubikom-cli create key --from-mnemonic --out=secret.key
```

You will need to enter 24 words one by one, so that the key can be created.

Getting Key Information

Now that you have your key, you can get various details about it.

Get Key Address

To get the Ethereum-style key address, use "get address" command:

```
$ ubikom-cli get address --key=secret.key
0x27A5f262Be45D99068C157c5A10430ddA252B1f6

# Same thing - returns Ethereum address.
$ ubikom-cli get ethereum-address --key=secret.key
0x27A5f262Be45D99068C157c5A10430ddA252B1f6
```

If your key was saved with a passphrase, you will be prompted for one.

To get Bitcoin-style address, use "get bitcoin-address" command:

```
$ ubikom-cli get bitcoin-address --key secret.key
166UnLK76wCm4qwTAkU9SF4WLBJ8kmgEN7
```

Get Public Key

To get the public key associated with this private key, use "get public-key" command:

```
$ ubikom-cli get public-key --key=secret.key
0x036e7...23b994
```

Public key here is 33 byte-long compressed public key.

Get Key Mnemonic

Key mnemonic allows you to restore the key (using "create key –from-mnemonic" command) if you don't have the key file. Use this one with caution! You want to do it on a secure machine (ideally air-gapped). Write the mnemonic down and keep it in a safe place.

```
$ ubikom-cli get mnemonic --key=secret.key
1:  glass
2:  vessel
3:  transfer
4:  broken
5:  ....
```

Getting Account Balance, Funding

Now that we have our key registered, we can query our balance:

```
$ ADDRESS=$(ubikom-cli get ethereum-address --key secret.key)
$ ubikom-cli bc get balance $ADDRESS --network=sepolia
15:11:26 WRN using Sepolia testnet
0
```

In the first line, we got our Ethereum address and assigned it to the ADDRESS variable. In the second line, we queried the balance of this address, which, unsurprisingly, is zero. Notice that we are using Sepolia testnet (–mode=test argument). Without it, we would go to the Ethereum mainnet. The warning message was printed out to alert us to the fact that we are working on the testnet.

Let's get some funds into our test account. To do that, find a Sepolia faucet and request funds using our Ethereum address, which, again, you can obtain using this command:

```
$ ubikom-cli get ethereum-address --key secret.key
```

At the moment when this was written, Sepolia faucets were somewhat unreliable, but if you try a few of them you can find one that works. Google for 'free Sepolia faucet'. As always, be carefull by not giving your private information to anyone.

I had some luck using this one: https://sepolia-faucet.com/, which mines Ether in your browser for a few minutes and then sends the reward to you. Use at your own risk.

After you get some funds, your balance will change:

```
$ ubikom-cli bc get balance $ADDRESS --network=sepolia
15:29:33 WRN using Sepolia testnet
32375000000000000
```

Notice that your balance is in wei, the smallest unit in Ethereum ecosystem. To convert it into Ether, you may use a tool like this one: https://eth-converter.com/

Registering Names, Updating Configuration

Registering Name

Once we have some funds in our account, we can register a name. After you register a name, you own it - you can transfer the ownership to anyone else (if you so choose), or sell it in the future. Here, we will register a name on Sepolia testnet - the Ethereum mainnet works the same way, but it might not be a great idea to use ubikom-cli on the mainnet - our keys are for testing only, we don't have industrial strength protection that is offered by widely used software and hardware wallets.

Before we register a name, we should create an encryption key. This is optional - if you don't specify an encryption key, you won't be able to send and receive secure messages. You can always change the encryption key later (but you will have to pay the gas fees).

The encryption key is just another key. Let's create it:

```
$ ubikom-cli create key --out=encrypt.key
```

Now we can register a new name:

```
$ ubikom-cli bc register name test666 --key=secret.key \
  --enc-key=encrypt.key --network=sepolia
... some logs omitted...
17:03:20 WRN using Sepolia testnet
17:03:20 DBG got nonce nonce=1
17:03:20 DBG got gas price gas-price=500000000007
17:03:20 DBG got chain ID chain-id=11155111
17:03:20 INF tx sent tx=0x387ca...488f920
{
  "root": "0x",
  "status": "0x1",
  "cumulativeGasUsed": "0x227a8",
  "logsBloom": "0x000000...00000",
  "logs": [
    {
      "address": "0xcc8650c9cd8d99b62375c22f270a803e7abf0de9",
      "topics": [
        "0x1c6eac...37c17f"
      ],
      "data": "0x00000...000000",
      "blockNumber": "0x26f669",
      "transactionHash": "0x8375b51...f0bdfd24",
      "transactionIndex": "0x1",
      "blockHash": "0xabc049...51bca3",
      "logIndex": "0x0",
      "removed": false
    }
  ],
  "transactionHash": "0x8375b5...dfd24",
  "contractAddress": "0x0000000000000000000000000000000000000000",
  "gasUsed": "0x1d5a0",
  "blockHash": "0xabc04...1bca3",
  "blockNumber": "0x26f669",
  "transactionIndex": "0x1"
}
```

If you try to register the same name again, you will get an error:

```
$ ubikom-cli bc register name test666 --key=secret.key \
  --enc-key=encrypt.key --network=sepolia
... some logs omitted...
17:05:08 WRN using Sepolia testnet
```

```
17:05:09 DBG got nonce nonce=2
17:05:09 DBG got gas price gas-price=500000000007
17:05:09 DBG got chain ID chain-id=11155111
17:05:09 INF tx sent tx=0x203a08...74f4f2
{
  "root": "0x",
  "status": "0x0",
  "cumulativeGasUsed": "0x3cb27",
  "logsBloom": "0x00000000...000000000",
  "logs": [],
  "transactionHash": "0x203a0890...74f4f2",
  "contractAddress": "0x0000000000000000000000000000000000000000",
  "gasUsed": "0x65e2",
  "blockHash": "0x9accd...09ae9e",
  "blockNumber": "0x26f671",
  "transactionIndex": "0x4"
}
17:05:25 ERR transaction failed
17:05:25 FTL failed to register name error="transaction failed"
```

Let's verify that the name was successfully registered:

```
$ ubikom-cli bc lookup name test666 --network=sepolia
... some logs omitted...
17:10:21 WRN using Sepolia testnet
{
  "PublicKey": "0x0367714...710c3",
  "Owner": "0x27a5f262be45d99068c157c5a10430dda252b1f6",
  "Price": 0
}
```

Here's what we see: * The public key is our encryption public key (you can get it by running "ubikom-cli get public-key –key=encrypt.key"); * The owner is us (this is our Ethereum address); * The price of the name is zero, which means it's not for sale.

Registering Messaging Endpoint

Before you can start receiving messages, you must register a messaging endpoint. This is the endpoint other users will use to send messages to you. Each name may have a number of configuration entries associated with it, each entry is a key/value pair (both are strings). For messaging, the name of the configuration entry must be "dms-endpoint". Ubikom provides a public server located at alpha.ubikom.cc:8826, which can be used as a messaging endpoint. It's fine to use another address (as long as it's running a server implementing the messaging protocol). You can run your own server, if you want.

To register a messaging endpoint, run the following command:

```
$ ubikom-cli bc update config test666 --config-name=dms-endpoint \
   --config-value=alpha.ubikom.cc:8826 \
   --network=sepolia --key=secret.key --gas-price=3000000
... some logs omitted...
18:49:14 WRN using Sepolia testnet
18:49:14 DBG got nonce nonce=3
18:49:14 DBG got gas price gas-price=3000000
18:49:14 DBG got chain ID chain-id=11155111
18:49:14 INF tx sent tx=0x2e508...bd41652
{
  "root": "0x",
  "status": "0x1",
  "cumulativeGasUsed": "0x2e260",
  "logsBloom": "0x0000...0000000",
  "logs": [
    {
      "address": "0xcc8650c9cd8d99b62375c22f270a803e7abf0de9",
      "topics": [
        "0xcde50e...828"
      ],
      "data": "0x000000...00000",
      "blockNumber": "0x26f859",
      "transactionHash": "0x2e5089d...1652",
      "transactionIndex": "0x1",
      "blockHash": "0x00f4ed...b9af",
      "logIndex": "0x0",
      "removed": false
    }
  ],
  "transactionHash": "0x2e5089...1652",
  "contractAddress": "0x0000000000000000000000000000000000000000",
  "gasUsed": "0xd115",
  "blockHash": "0x00f4e...efb9af",
  "blockNumber": "0x26f859",
  "transactionIndex": "0x1"
}
```

Here, we used gas-price flag to specify the gas price. Sometimes, the suggested gas price in Sepolia is too high (probably an issue with the new test network), and you might need to specify gas-price explicitly. If you don't, the suggested gas price value will be used.

To verify that the config value was updated, run:

```
ubikom-cli bc lookup config test666 --config-name=dms-endpoint \
  --network=sepolia
... some logs omitted...
```

```
18:55:41 WRN using Sepolia testnet
alpha.ubikom.cc:8826
```

Sending and Receiving Encrypted Messages

There are two components involved in sending and receiving encrypted messages:

- The identity registry (deployed on the blockchain) keeps track of the names, public keys, and messaging endpoints;
- The dump server stores and returns encrypted messages.

Here, we will use a local instance of the dump server.

Starting Dump Server

Let's start a local instance of the dump server. If you cloned the Ubikom repo, the dump server resides under cmd/ubikom-dump:

```
# From the repo directory.
$ cd cmd/ubikom-dump
# The server binary will be placed under $GOROOT/bin.
$ go install
$ ubikom-dump --data-dir=some_data_directory --lookup-server-url="" \
  --network=sepolia --log-level=debug
```

–data-dir specifies the directory where dump server will write the data.

–lookup-server-url=“” disables the legacy identity registry (this option will go away after migration to Ethereum-based identity registry is complete).

–network=sepolia instructs dump server to use sepolia test network.

The server will start on the default port, 8826. Now we can proceed in a different terminal window.

Creating Keys and Registering Names

Let's register a new set of keys and names. We will use the previously created (and funded) secret key (file secret.key, see above). Use your own names instead of “alice111” and “bob111”.

```
# Create a key and associate it with name "alice111".
$ ubikom-cli create key --out=alice.key
...

$ ubikom-cli bc register name alice111 --key=secret.key \
  --enc-key=alice.key --network=sepolia
...

# Register the messaging endpoint for alice111.
```

```
$ ubikom-cli bc update config alice111 \
  --config-name=dms-endpoint --config-value=localhost:8826 \
  --network=sepolia --key=secret.key
...

# Create a key and associate it with name "bob111".
$ ubikom-cli create key --out=bob.key
...

$ ubikom-cli bc register name bob111 --key=secret.key \
  --enc-key=bob.key --network=sepolia
...

# Register the messaging endpoint for bob111.
$ ubikom-cli bc update config bob111 \
  --config-name=dms-endpoint --config-value=localhost:8826 \
  --network=sepolia --key=secret.key
...
```

Here's what happened:

- We have created a new key and registered it as an encryption key for name "alice111".
- We have created a new key and registered it as an encryption key for name "bob111".
- For both names, we've updated the configuration so that the messages are delivered to localhost:8826
- Notice that the names are owned by the address that was used to register them, which is the address associated with secret.key file. It has the ultimate authority to change the registration data later on.

Sending Messages

Finally, we are ready to send an encrypted message:

```
$ ubikom-cli send message --sender=alice111 --receiver=bob111 \
  --key=alice.key --network=sepolia
14:03:03 WRN using Sepolia testnet
14:03:03 WRN using Sepolia testnet
Enter your message, dot on an empty line to finish:
Attack at dawn
.
14:03:13 DBG got receiver's public key
14:03:13 DBG got receiver's address address=localhost:8826
14:03:14 DBG sent message successfully
```

Let's go over the steps:

- We are sending a message (as alice111) to bob111, using our encryption key located in alice.key file.
- First, ubikom-cli must figure out where to send the message. It queries the identity registry (on Sepolia test network), and finds localhost:8826 as the destination.
- The message is sent to localhost:8826, where our dump server is running.
- Dump server verifies that the message originates from someone who has control over the private key associated with "alice111" name. The verification succeeds, and it accepts the message, and writes it to the local storage. The message is encrypted, only the original sender or the recipient can read it.
- The message sits in the local storage until bob111 shows up to receive it.

Receiving Messages

Now, we can receive the message (as bob111):

```
 ubikom-cli receive message --key=bob.key --network=sepolia \
   --dump-service-url=localhost:8826
14:13:00 WRN using Sepolia testnet
Attack at dawn
```

Receiving the message included those steps:

- We generated a proof that we have control over the key associated with the name "bob111" and sent it to dump server.
- Dump server, satisfied with the proof, sent us the encrypted message.
- We verified that the message came from the stated sender "alice111", by retrieving their public key from the identity registry and verifying the signature.
- The message was decrypted by using the key derived from alice111 public key and bob111 private key.