SecretVRF for IBC via proxy contracts

An example of how to use Secret VRF to fetch random numbers via IBC between Secret and Juno testnet

Cross-chain random numbers demo

This documentation serves as a demo on how to sendcross-chain random numbers from Secret Network testnet to Juno testnet via IBC . The demo repositorycan be cloned here .

The design system we'll be using consists ofone Secret contract and two Juno contracts :

- Secret Proxy Contract
- : A Secret contract that requests a random number and specifies the callback information for the response.
- Juno Proxy Contract
- : A Juno contract that sends an IBC message to fetch the random number from the Secret Network proxy contract.
- Juno Consumer Contract
- : A Juno contract that consumes the random numbers generated by the Secret proxy contract.
-

The Secret proxy contract will produce adifferent, non-predictable number for each request it receives. You can read more about random numbers on Secret Network in thetechnical specification of this feature. See here for a diagram of the system architecture:

Secret VRF IBC smart contract architecture

Environment Configuration

There are two steps to configuring your environment:

1. Upload and instantiate the three smart contracts
2. Configure an IBC relayer to properly relay packets between your chains of choice (for this demo we will use
3.).
4.

Upload and instantiate Secret Smart Contract

To begin, let's upload and instantiate the Secret Network proxy smart contract with Secret.jsClone the repository :

```

Copy gitclonehttps://github.com/scrtlabs/examples

```

thencd intosecret-ibc-rng-template/node:

```

Copy cdsecret-ibc-rng-template/node

```

Runnpm i to install the dependencies:

```

Copy npmi

```

Then, create a.env file and add your wallet mnemonic:

Compile the contract:

Open a new terminal window andcd intosecret-ibc-rng-template/proxy:

```

Copy cdsecret-ibc-rng-template/proxy

```

Run the Makefile compile script:

```

Copy makebuild-mainnet-reproducible

```

Then upload + instantiate the contract insecret-ibc-rng-template/node:

```

Copy nodeupload_instantiate_secret_proxy

```

Upload and instantiate Juno Smart Contracts

Now that we have our Secret proxy contract, let's upload and instantiate the two Juno smart contracts.

You can configure the consumer contracts for any IBC-compatible chain of your choosing. However, for this demo will be uploading and instantiating our contracts on Juno. Compile Juno Proxy Contract

Open a new terminal window andcd intosecret-ibc-rng-template/consumer-side-proxy:

```

Copy cdsecret-ibc-rng-template/consumer-side-proxy

```

Run the Makefile compile script:

```

Copy makebuild-mainnet-reproducible

```

Compile Juno Consumer Contract

Open a new terminal window andcd intosecret-ibc-rng-template/consumer:

```

Copy cdsecret-ibc-rng-template/consumer

```

Run the Makefile compile script:

```

Copy makebuild-mainnet-reproducible

```

Upload Juno Contracts

Next, upload the compiledwasm files to Juno testnet usingJuno Tools , and be sure to note the respectivecodeIds:

You should see thetransaction result returned upon successful upload.

Instantiate contracts

If using the CLI, update to junod v16.0. If using js/ts, updateCosmJS to 0.31 . To upload the contracts to Juno testnet, you need Juno testnet tokens in your wallet. Visit faucet.reece.sh/uni-6/ to receive testnet tokens Next, instantiate the Juno proxy contract by running the following in your terminal:

```

Copy junod tx wasm instantiate'{}' --label 'juno-proxy' --no-admin --from --gas 200000 -y --chain-id uni-6 --node https://rpc.uni.junonetwork.io:443 --gas-prices 0.025ujunox
```

```

Then, to query that the instantiation was successful and find the contract address, query the returnedtxHash with:

```

Copy junod q tx --node https://rpc.uni.junonetwork.io:443

```

You should see thecontract_address variable:

```

Copy juno1r9awn4hek5s8kuvfm46kh775csqvhuzs00j8c0djaul5729s48usht3j0s

```

Now, simply repeat the process for the Juno Consumer contract. The only difference is that the instantiation message is slightly modified because it needs to include the other Junocontract_address that you instantiated as a pointer:

```

Copy junod tx wasm instantiate '{"init": {"rand_provider": { "address": , "code_hash": ""}}}' --label 'juno-consumer' --no-admin --from --gas 300000 -y --chain-id uni-6 --node https://rpc.uni.junonetwork.io:443 --gas-prices 0.025ujunox

```

Query the returnedtxHash:

```

Copy junod q tx --node https://rpc.uni.junonetwork.io:443

```

You should see thecontract_address variable:

```

Copy juno1sznkzlleqlxw8lp8hy66l0zg0fe8khgkeyqlfmlvdkutgxjw69xq7jm32t

```

Configuring Hermes Relayer

Now that you have successfully uploaded and instantiated the three smart contracts, let's configure Hermes Relayer to relay packets between Secret test and Juno testnet.

First,install Hermes and Gaiad manager .

Then, configure Hermes by navigating to the folder.hermes and opening theconfig.toml file.

If you're on a Mac, you may need to pressCommand + Shift + Period to see hidden files, such as the.hermes folder. To relay packets between Secret Network testnet and Juno testnet, update theconfig.toml file with this configuration:

```

Copy [global] log_level='info'

[mode]

[mode.clients] enabled=true refresh=true misbehaviour=true

[mode.connections] enabled=true

[mode.channels] enabled=true

[mode.packets] enabled=true clear_interval=100 clear_on_start=true tx_confirmation=true

[telemetry] enabled=true host='127.0.0.1' port=3001

[[chains]] id='pulsar-3' type='CosmosSdk' rpc_addr='https://rpc.pulsar.scrttestnet.com:443'
grpc_addr='http://grpcbin.pulsar.scrttestnet.com:9099' event_source={
```
```

mode='push',url='wss://rpc.pulsar.scrttestnet.com:443/websocket',batch_delay='500ms'} rpc_timeout='10s' account_prefix='secret' key_name='wallet' store_prefix='ibc' default_gas=100000 max_gas=400000 gas_multiplier=1.1 max_msg_num=30 max_tx_size=180000 clock_drift='5s' max_block_time='30s' memo_prefix='' sequential_batch_tx=false trusting_period='16hrs'

[chains.trust_threshold] numerator='1' denominator='3'

[chains.gas_price] price=0.1 denom='uscrt'

[chains.packet_filter] policy='allow' list=[['wasm.*',''],['transfer','*']]

[[chains]] id='uni-6' rpc_addr='https://juno-testnet-rpc.polkachu.com:443' grpc_addr='http://juno-testnet-grpc.polkachu.com:12690' event_source={ mode='push',url='wss://juno-testnet-rpc.polkachu.com:443/websocket',batch_delay='500ms'} rpc_timeout='10s' account_prefix='juno' key_name='wallet' address_type={ derivation='cosmos'} store_prefix='ibc' default_gas=1800000 max_gas=9000000 gas_price={ price=0.026,denom='ujunox'} gas_multiplier=1.2 max_msg_num=30 max_tx_size=179999 clock_drift='15s' max_block_time='10s' trusting_period='448h' memo_prefix='relayed by CryptoCrew Validators' trust_threshold={ numerator='1',denominator='3'}

[chains.packet_filter] policy='allow' list=[['wasm.*',''],['transfer','*']]

```

Next, configure Gaiad Manager by navigating to the folder.gm , and then update the gm.toml file with theconfiguration seen here.

If you are using Hermes Relayer, make sure your paths are set up correctly in both theconfig.toml and gm.toml . See the image below for reference: gm.toml configuration paths Now let's relay packets!

Consuming the random number via IBC

Now it's time to execute our IBC smart contracts and relay packets between Juno testnet and Secret testnet. If you run into any issues at this step, refer to thehermes docs for guidance, and also ask questions in theSecret Network developer discord chat !

1. Start Gaiad Manager
2.

```

Copy gm start

```

1. Add your Secret and Juno testnet wallets to Hermes:
2.

cd intoexamples/secret-ibc-rng-template:

```

Copy cdexamples/secret-ibc-rng-template

```

and update thea.mnemonic file to include your testnet wallet address like so:

```

Copy your mnemonic wallet words to go here with no quotes

```

Then run the following to add your testnet wallet address to Hermes:

```

Copy hermeskeysadd--hd-path"m/44'/529'/0'/0/0"--mnemonic-filea.mnemonic--chainpulsar-3

hermeskeysadd--hd-path"m/44'/529'/0'/0/0"--mnemonic-filea.mnemonic--chainuni-6

```
```

If you run into errors at this step, see the official Hermes docs for adding keys here . 1. Create clients 2.

```

Copy hermescreateclient--host-chainpulsar-3--reference-chainuni-6 hermescreateclient--host-chainuni-6--reference-chainpulsar-3

```

1. Create connections
2.

```

Copy hermes create connection --a-chain uni-6 --a-client 07-tendermint-468 --b-client 07-tendermint-235

```

In place of07-tendermint-235 and 07-tendermint-468 , use the client IDs returned to you in your terminal. Upon success, you should see a message like so:

Now that a channel is established, let's create a channel identifier, which links the Juno proxy contract to the Secret proxy contract.Note that the ports listed below are the addresses of the Juno and Secret proxy contracts which we instantiated earlier.

1. Create channel identifier
2.

```

Copy hermes create channel --a-chain uni-6 --a-connection connection-612 --a-port wasm.juno1ecl4r6dhhlluz56jqm24t6ss7s9gr6d0pu2lumvpwnnk56gnw7gqpz8m6c --b-port wasm.secret1rmccmgwf6zf2kawrv7h5faq3tx883epz7ty6tj

```

After successfully creating a channel identifier, we can relay packets! Let's start Hermes and then execute the Juno consumer contract to send a random network from Secret Network to Juno

1. Start Hermes (open a new terminal window and then run the following)
2.

```

Copy hermes start

```

Hermes will scan the chain for all clients, connections and channels. This might take some time, which is normal. If you want to specify which channels it scans, update the hermes config file to include the following at the end of the chain configuration: ```

Copy [chains.packet_filter] policy = 'allow' list = [['transfer', 'channel-495'],]

```

After Hermes has started running, execute the Juno consumer contract to return a random number from Secret via IBC:

```

Copy junod tx wasm execute --fromjuno1z4n39vfckeuv6udx6f4e6h8d5mt3xucuwdjs6lk2ets60ejruseqnpt00k '{"do_something": {}}' --gas 300000 -y --chain-id uni-6 --node https://rpc.uni.junonetwork.io:443 --gas-prices 0.025ujunox

```

Then, query the smart contract to see if it returned the random number:

```

Copy junod query wasm contract-state smart'{"last_random": {}}' --chain-id uni-6 --node https://rpc.uni.junonetwork.io:443

```
```

Upon successful execution, a random number will be returned:

Conclusion

Congrats! You now have the tools to implement cross-chain random number generation via IBC. By following these steps, you can facilitate and execute smart contracts and relay packets between Juno and Secret, as well as any other IBC-compatible chain, expanding the possibilities for blockchain interoperability and fostering new avenues for decentralized application development.

Epilogue: IBC Fundamentals

If you are brand new to IBC, here is a quick crash course!

To connect two CosmWasm contracts over IBC you must establish an IBC channel between them. The IBC channel establishment process uses a four way handshake. Here is a summary of the steps:

1. OpenInit
2. Hello chain B, here is information that you can use to verify I am chain A. Do you have information I can use?
3. OpenTry
4. Hello chain A, I have verified that you are who you say you are. Here is my verification information.
5. OpenAck
6. Hello chain B. Thank you for that information I have verified you are who you say you are. I am now ready to talk.
7. OpenConfirm
8. Hello chain A. I am also now ready to talk.
9.

Once the handshake has been completed a channel will be established that the ibc messages may be sent over. In order to do a handshake and receive IBC messages your contract must implement the following entry points (which are implemented in our proxy contract):

1. ibc_channel_open
2.
    - Handles theOpenInit
3. andOpenTry
4. handshake steps.
5. ibc_channel_connect
6.
    - Handles theOpenAck
7. andOpenConfirm
8. handshake steps.
9. ibc_channel_close
10.
    - Handles the closing of an IBC channel by the counterparty.
11. ibc_packet_receive
12.
    - Handles receiving IBC packets from the counter-party.
13. ibc_packet_ack
14.
    - Handles ACK messages from the counter-party.
15. ibc_packet_timeout
16.
    - Handles packet timeouts.
17.

Here is a great repo to learn more about IBC fundamentals

Last updated1 month ago On this page *Cross-chain random numbers demo * Environment Configuration * Consuming the random number via IBC * Conclusion * Epilogue: IBC Fundamentals

Was this helpful? Edit on GitHub Export as PDF