Native On-chain randomness

A step-by-step tutorial of how to use Secret Network's randomness API to generate a coin flip In this tutorial, you will learn how to access the randomness feature and use it with a smart contract that returns a truly random coin flip

For a detailed feature explainer head to thenetwork technical documentation

Import Secret VRF

In your Cargo.toml file, add secret-toolkit-storage 0.9.0:

```

Copy [dependencies] cosmwasm-std={ package="secret-cosmwasm-std", version="1.1.10"} cosmwasm-storage={ package="secret-cosmwasm-storage", version="1.1.10"} secret-toolkit-storage="0.9.0"

```

Make sure you're compiling with rust < 1.70. Newer versions are currently not compatible.

Tutorial - Coin Flip

What follows is a step-by-step tutorial of how to use Secret Network's randomness API to generate a coin flip (returning either 0 or 1) with true randomness. You can follow along and/or viewthe completed code in this repo.

Environment Configuration

LocalSecret is a tool that allows you to run a local Secret Network on your machine for testing and development purposes.

Here are the steps to use the randomness feature with LocalSecret:

1. Configure your developer environment
2. and be sure to install thelatest version of SecretCLI.
3. Clone the Secret Labs examples repo and then navigate to the vrf-randomness-tutorial folder:
4.

```

Copy git clone https://github.com/scrtlabs/examples.git

```

Contract.rs

To consume the random number, you need to import the necessary dependencies in yourcontract.rs file in order toaccess the random number from the env parameter.

In your contract, import the necessary dependencies (these are already imported in the cloned repo):

```

Copy usecosmwasm_std::{Binary,Env,MessageInfo,Response,Result};

```

In the contract's entry point (e.g., execute, instantiate, or query), you can access the random number from theenv parameter:

```

Copy

# [entry_point]

pubfnexecute( deps:DepsMut, env:Env, _info:MessageInfo, msg:ExecuteMsg, )->Result{ matchmsg { ExecuteMsg::Flip{}=>try_flip(deps, env), } }

```

The env and block_info structures are defined as:

```

Copy pubstructEnv{ pubblock:BlockInfo, pubcontract:ContractInfo, pubtransaction:Option, }

pubstructBlockInfo{ /// The height of a block is the number of blocks preceding it in the blockchain. pubheight:u64, pubtime:Timestamp, pubchain_id:String,

# [cfg(feature="random")]

# [serde(skip_serializing_if="Option::is_none")]

pubrandom:Option, }

```

Whererandom is 32 bytes and base64 encoded.

Accessing the Env struct

Below is a simple coin flip function that uses the randomness feature:

```

Copy pubfntry_flip(deps:DepsMut, env:Env)->Result { config(deps.storage).update(|mutstate|->Result<_,ContractError> { letcoin_flip=env.block.random.unwrap().0[0]%2; state.flip=coin_flip; Ok(state) })?;

deps.api.debug("flipped!"); Ok(Response::default()) }

```

try_flip() uses theconfig function to update the state of the smart contract by flipping a coin and storing the result in theflip field in thestate variable. Specifically,it generates a random number using the random field of the env.block object ,which is an optional value representing the most recent block's metadata, and takes the modulo 2 to obtain a value of either 0 or 1 . It then updates theflip field of thestate variable to this value.

Interacting with the Coin Flip Contract

Now, let's compile, upload, instantiate, and execute the contract to see it in action!

Compile

To compile your contract, in your terminal, make sure you have docker open, and then run:

```

Copy docker run--rm-v"(pwd)":/contract \ --mounttype=volume,source="(basename "(pwd)")_cache",target=/code/target \ --mounttype=volume,source=registry_cache,target=/usr/local/cargo/registry \ enigmampc/secret-contract-optimizer

```

This returns the optimized contract wasm file, iecontract.wasm.gz

Upload

To upload your contract to a containerized version of LocalSecret in docker, make sure you have docker installed and open, and then create a new tab in your terminal and run:

```

Copy docker run -it --rm -p 26657:26657 -p 26656:26656 -p 1317:1317 -p 5000:5000 --name localsecret ghcr.io/scrtlabs/localsecret

```

Congrats, you now have a new instance of LocalSecret running that can access the random number feature!

Next,create and fund a wallet so you can upload the contract to LocalSecret. Then run the following to upload:

```
```

Copy secretcli tx compute store contract.wasm.gz --gas 5000000 --from--chain-id secretdev-1

```

To confirm that the contract upload was successful:

```
Copy secretcli query compute list-code
```

## Instantiate

Now let's instantiate our contract with a starting flip of 1 (1 meaning Heads or Tales, up to you!)

```
Copy secretcli tx compute instantiate 1 '{"flip": 1}' --from--label flipContract
```

To confirm that the contract instantiation was successful:

```
Copy secretcli query compute list-contract-by-code 1
```

Execute

Now that we have acontract address (which is returned from thelist-contract-by-code query above), we an execute the coin flip with the randomness feature!

To flip the coin simply run:

```
Copy secretcli tx compute execute'{"flip": {}}' --from myWallet
```

And to query that it was successful, run:

```
Copy secretcli query compute query'{"get_flip": {}}'
```

You might have to execute the flip function a few times to see the queried flip change, since there is a 50% chance the flip will return the same number :D

Summary

Congrats! In this step-by-step tutorial on creating a coin flip contract, you learned how to compile, upload, instantiate, and execute a contract on LocalSecret using Secret Network's randomness API to generate random numbers

For documentation on Secret VRF in a contract on another IBC-connected chainclick here .
```