Secret Tokens (SNIP-20)

Learn how to create a SNIP-20 token on Secret Network

Introduction

In this tutorial, we are going to create our own SNIP-20 token on Secret Network using Secret Labs' SNIP-20 reference implementation contract, and we will learn how to upload, instantiate, execute, and query our SNIP-20 contract using Secret.js. Let's dive in!

Source Code

You can clone the source codehere , which we will reference throughout the course of this documentation.

Prerequisites

Usethe following guide to set up your developer environment.

Build and Deploy Contract

Now that you've cloned the SNIP-20 reference implementation repo above, let's compile the contract. In your terminal runmake compile-optimized .

In Rust, a Makefile can be used to automate tasks such as building the project, running tests, or even generating documentation.Make compile-optimized is running the following optimizer command, which you can view in the Makefile:

Optimizer command

```
Copy RUSTFLAGS='-C link-arg=-s' cargo build --release --target wasm32-unknown-unknown
```

Configuring Secret.js

1. In your root project folder, create a new folder callednode.
2. In yournode
3. folder, create a new javascript file calledindex.js
4. .
5. Runnpm init -y
6. to create a package.json file.
7. Add"type" : "module"
8. to your package.json file.
9. Install secret.js:npm i secretjs
10.

Uploading the SNIP-20 Contract

In your index.js file, paste the following (be sure to replace the wallet seed phrase with your wallet seed phrase):

```
Copy import{ Wallet,SecretNetworkClient,EncryptionUtilsImpl,fromUtf8,MsgExecuteContractResponse }from"secretjs";
import*asfsfrom"fs";

constwallet=newWallet( "your walltet seed phrase to go here" );

consttxEncryptionSeed=EncryptionUtilsImpl.GenerateNewSeed();

constcontract_wasm=fs.readFileSync("../contract.wasm.gz");

constcodeId=1072; constcontractCodeHash="26af567eadde095c909ca6ecf58806235877e5b7ec9bfe30f1057e005f548b17";
constcontractAddress="secret1xez6pv463a0elalnj0z53w60fz6tgclv368dw0";

constsecretjs=newSecretNetworkClient({ chainId:"pulsar-3", url:"https://api.pulsar.scrttestnet.com", wallet:wallet,
walletAddress:wallet.address, txEncryptionSeed:txEncryptionSeed });

letupload_contract=async()=>{ lettx=awaitsecretjs.tx.compute.storeCode( { sender:wallet.address,
wasm_byte_code:contract_wasm, source:"", builder:"", }, { gasLimit:4_000_000, } );
```

constcodeId=Number(tx.arrayLog.find((log)=>log.type==="message"&&log.key==="code_id") .value);

console.log("codeId: ",codeId); // contract hash, useful for contract composition constcontractCodeHash=(awaitsecretjs.query.compute.codeHashByCodeId({code_id:codeId})).code_hash; console.log(Contract hash:{contractCodeHash}); }

upload_contract();

```

Runnode index.js in your terminal to execute theupload_contract() function. Upon successful execution, a codeId and contract hash will be returned:

```

Copy codeId: 1070 Contract hash: 26af567eadde095c909ca6ecf58806235877e5b7ec9bfe30f1057e005f548b17

```

Instantiating the SNIP-20 Contract

In your index.js file, paste the following:

```

Copy letinstantiate_contract=async()=>{ constinitMsg={ name:"Zebra", symbol:"ZBRA", decimals:6, prng_seed:Buffer.from("Something really random").toString("base64"), admin:wallet.address, initial_balances:[{ address:wallet.address, amount:"1000000000", },], };

lettx=awaitsecretjs.tx.compute.instantiateContract({ code_id:codeId, sender:wallet.address, code_hash:contractCodeHash, init_msg:initMsg, label:" Snip-20 Example"+Math.ceil(Math.random()*10000), }, { gasLimit:400_000, });

//Find the contract_address in the logs constcontractAddress=tx.arrayLog.find((log)=>log.type==="message"&&log.key==="contract_address").value;

console.log(contractAddress); };

instantiate_contract();

```

TheinitMsg object in ourindex.js file is referencing the instantiation message defined inmsg.rs at line 20 . Notice that we chose to omit the optionalconfig variable. If we includeconfig , there is a variety of additional contract functionality that we could program, such as burn, mint, admin privileges, etcas seen here .

Now we are going to instantiate some ZBRA coin. If you want to create your own coin name, update thename, symbol, andamount fields respectively. Be sure to comment outupload_contract() and now runnode index.js to callinstantiate_contract() . Upon successful execution, a contract address will be returned:

```

Copy secret1xez6pv463a0elalnj0z53w60fz6tgclv368dw0

```

Query the Token Info

To check that the instantiation of our SNIP-20 ZEBRA token was successful, let's query the smart contract's token info:

```

Copy letquery_token_info=async()=>{ consttokenInfoQuery=awaitsecretjs.query.compute.queryContract({ contract_address:contractAddress, query:{ token_info:{}, }, code_hash:contractCodeHash, });

console.log(tokenInfoQuery); }; query_token_info();

```

The following is returned upon successful query:

```

Copy token_info: { name: 'Zebra', symbol: 'ZBRA', decimals: 6, total_supply: null } }

```

The reasontotal supply isnull is because we chose to maketotal supply hidden in our instantiation message. If you want it to be public, then in theInitConfig variable setpublic_total_supply to true.

SNIP-20 Contract Messages

Now that we have successfully instantiated our SNIP-20 contract, let's send anexecution message to better understand the contract's functionality.

Start by adding the token to your Keplr wallet. Click on Keplr, select the hamburger icon, select "Add Token", and then paste in your token's contract address. If you need to fund your wallet to execute the transaction, you can do so using thepulsar-3 faucet here . You should now see your token in your Keplr wallet!

keplr wallet with ZBRA token Let'stransfer some tokens to another wallet address. The transfer message is defined in msg.rs as follows:

```

Copy Transfer { recipient:String, amount:Uint128, memo:Option, decoys:Option>, entropy:Option, padding:Option, }

```

Now let's execute the transfer message with secret.js. Be sure to update therecipient wallet address with your own wallet before executing the code below. For testing purposes, I am using two Keplr wallet connected to the Secret Network testnet in order to move funds back and forth:

```

Copy lettransfer_snip20=async(receiver_wallet)=>{ letexecuteMsg={ transfer:{ owner:wallet.address, amount:"10000000", recipient:receiver_wallet, }, };

lettx=awaitsecretjs.tx.compute.executeContract({ sender:wallet.address, contract_address:contractAddress, code_hash:contractCodeHash, msg:executeMsg, }, { gasLimit:100_000, }); console.log(tx); };

transfer_snip20("secret1f9zykwvwc6jyhv6dtsjwx03e92j08nyffwuwcu");

```

Congrats! You just successfully transferred your own SNIP-20 token on Secret Network!

Last updated6 months ago On this page *Introduction * Source Code * Prerequisites * Build and Deploy Contract * Configuring Secret.js * Uploading the SNIP-20 Contract * Instantiating the SNIP-20 Contract * Query the Token Info * SNIP-20 Contract Messages

Was this helpful? Edit on GitHub Export as PDF
```