Verifying Smart Contracts

Smart Contract verification is an important process to ensure you are running the correct contract. You can verify any smart contract on chain.

Let's go over real case. There is a live contract on chain that you want to determine code name and version.

Inspect Code

Our case contract address isjuno1unclk8rny4s8he4v2j826rattnc7qhmhwlv3wm9qlc2gamhad0usxl7jnd on juno uni testing network.

CW2 Spec defines contract information to be stored on state. We can query the contract information using command:

junod query wasm contract-state raw juno1unclk8rny4s8he4v2j826rattnc7qhmhwlv3wm9qlc2gamhad0usxl7jnd 636F6E74726163745F696E666F --node RPC --output json | jq -r .data | base64 -d | jq { "contract" :

"crates.io:cw20-base" , "version" :

"0.10.3" } What is 636F6E74726163745F696E666F?

ContractInfo is stored under "contract_info" key which translates to "636F6E74726163745F696E666F" in hex format as documentedhere .

Now we found out this code is a cw20-base and the version is0.10.3 .

We need hash, for that we need code ID. Here is how you get that:

junod query wasm contract juno1unclk8rny4s8he4v2j826rattnc7qhmhwlv3wm9qlc2gamhad0usxl7jnd --node RPC --output json | jq { "address" :

"juno1unclk8rny4s8he4v2j826rattnc7qhmhwlv3wm9qlc2gamhad0usxl7jnd" , "contract_info" :

{ "code_id" :

"122" , "creator" :

"juno1d3axtckm7f777vlu5v8dy8dsd6fefhhnmsrrps" , "admin" :

"" , "label" :

"Hidden" , "created" : null, "ibc_port_id" :

"" , "extension" : null } } Nice one Detective Gadget, you found code_id, creator address, admin and label too.

We need the actual code now:

junod query wasm code 122 122_code.wasm --node RPC Downloading wasm code to 122_code.wasm Now the hash:

sha256sum 122_code.wasm 46bd624fff7f11967aac6ddaecf29201d1897be5216335ccddb659be5b524c52 122_code.wasm
We found it46bd624fff7f11967aac6ddaecf29201d1897be5216335ccddb659be5b524c52 !

Find the Original Code

You can find hashes it on source repo if provider published them. cw-plus smart contracts hashes are published alongside smart contract code atcw-plus .

You can download hashes fromchecksums.txt

Here is a sample one:

fe34cfff1cbc24594740164abb953f87735afcdecbe8cf79a70310e36fc13aa0 cw1155_base.wasm
de49426e9deed6acf23d5e930a81241697b77b18131c9aea5c3ca800c028459e cw1_subkeys.wasm
c424b66e7f289cef69e1408ec18732e034b0604e4b22bfcca7546cc9d57875e3 cw1_whitelist.wasm
e462d44a086a936c681f3b3389d50b8404ce2152c8f0fb32b257064576210c03 cw1_whitelist_ng.wasm
0b2e5d5dc895f8f49f833b076a919774bb5b0d25bf72819e9a1cbdf70f9bf79b cw20_atomic_swap.wasm
6c1fa5872e1db821ee207b5043d679ad1f57c40032d2fd01834cd04d0f3dbafb cw20_base.wasm
f00759aa9a221efeb58b61a1a1d4cc4281cdce39d71ac4d8d78d234f03b3b0eb cw20_bonding.wasm
b6041789cc227472c801763c3fab57a81005fb0c30cf986185aba5e0b429d2e6 cw20_escrow.wasm
91b35168d761de9b0372668dd8fa8491f2c8faedf95da602647f4bade7cb9f57 cw20_ics20.wasm

d408a2195df29379b14c11277f785b5d3f57b71886b0f72e0c90b4e84c2baa4a cw20_merkle_airdrop.wasm
934ba53242e158910a2528eb6c6b82deb95fe866bbc32a8c9afa7b97cfcb9af4 cw20_staking.wasm
ac1f2327f3c80f897110f0fca0369c7022586e109f856016aef91f3cd1f417c1 cw3_fixed_multisig.wasm
785340c9eff28e0faeb77df8cca0fafee6b93a1fa033d41bda4074cd97600ec1 cw3_flex_multisig.wasm
87b3ad1dee979afc70e5c0f19e8510d9dcc8372c8ef49fc1da76725cad706975 cw4_group.wasm
4651e90405917897f48d929198278f238ec182ac018c414ee22f2a007a052c1e cw4_stake.wasm

Compile Yourself

If you tried to find0.10.3 code hash on GitHub, you would have failed to find it because it is a minor release, so no hash provided.

Let's generate it ourselves.

The reason we userust-optimizer is not just for generating performant and small code size, also for deterministic code output so that it will be comparable. Hashes in the example above are all output of rust-optimizer.

docker run --rm -v " (pwd) " :/code \ --mount type = volume,source = "(basename " (pwd) ") _cache",target = /code/target \ --mount type = volume,source = registry_cache,target = /usr/local/cargo/registry \ cosmwasm/workspace-optimizer:0.12.4 On Windows, you can use the following command instead

docker run --rm -v {pwd}:/code --mount type=volume,source="("(Split-Path -Path pwd -Leaf)")_cache",target=/code/target --mount type=volume,source=registry_cache,target=/usr/local/cargo/registry ` cosmwasm/rust-optimizer:0.12.11 Hashes will be generated at./artifacts/checksums.txt .

danger There is anrust-optimizer-arm64 docker builder image that gives faster output on Apple M1 chip. Arm images are released to ease development and testing on Mac M1 machines. The native Arm version produces different wasm artifacts than the Intel version. For release / production use, only contracts built with the Intel optimizers must be used. You can find the value and compare it to the value we got.

cat ./artifacts/checksums.txt |

grep cw20_base.wasm 46bd624fff7f11967aac6ddaecf29201d1897be5216335ccddb659be5b524c52 cw20_base.wasm diff

< (echo

"46bd624fff7f11967aac6ddaecf29201d1897be5216335ccddb659be5b524c52"

)

< (echo

"46bd624fff7f11967aac6ddaecf29201d1897be5216335ccddb659be5b524c52") Hashes do match, contract is verified.

Last Note

Documentation above should give a clear idea about verification. This logic can be easily implemented on different languages, CosmJS in particular. Previous Develop dApp Next Migration * Inspect Code * Find the Original Code * * Compile Yourself * Last Note