Verify Contracts

Verify Contracts

Once verified, a smart contract or token contract's source code becomes publicly available and verifiable, creating transparency and trust.

There are several ways to verify a contract, programmatically or manually on the UI.

Verify on the UI

1. Go the the verify contract page (Other -> Verify Contract)
2. Enter in the contract address you received during deployment. The dropdown will show you several available verification options. Select the one you would like to use and continue.
3.
    1. Solidity (Flattended source code)
4.
    1. Solidity (Standard JSON Input)
5. 3.
6.

Solidity (Flattened Source Code)

1. Contract Address: The 0x address supplied on contract creation (added above)
2. Is Yul Contract: Select if the contract is coded in Yul for efficiency.
3. Include Nightly Builds: Select if you want to show nightly builds.
4. Compiler: derived from the first line in the contract pragma solidity X.X.X. Use the corresponding compiler version rather than the nightly build.
5. EVM Version: Select the correct EVM version if known, otherwise use default.
6. EVM Version: Select the correct EVM version if known, otherwise use default.
7. Enter the Solidity Contract Code: You may need to flatten your solidity code if it utilizes a library or inherits dependencies from another contract. We recommend hardhat or the POA solidity flattener. To flatten your contract using contract, run:
8.

```

Copy yarnhardhatflatten.\contracts\.sol>flattened.sol

```

1. Add Contract Libraries: Enter the name and 0x address for any required libraries called in the .sol file. You can add multiple contracts with the "+" button.
2. Click the Verify and Publish button.
3. If all goes well, you will see a checkmark next to Code in the code tab, and an additional tab called Read Contract. The contract name will now appear in BlockScout with any transactions related to your contract.
4.

Solidity (Standard JSON Input)

1. Include nightly builds. You can choose Yes or No depending on your compiler.
2. Compiler. Choose the compiler version used to compile your smart contract. If you selected yes for nightly builds, use the compiler version rather than the build.
3. Standard Input JSON. Upload your Standard Input JSON file. File should follows solidity format and all the sources must be in Literal Content format, not a URL.
4.

Click the Verify & publish button and wait for the response.

Verify Programmatically

You can use this repo to verify your contracts programmatically along with deploying them.

Repo Link: https://github.com/gelatodigital/verify-simple-proxy

In this repo, use thedeploy/ directory for deploying our contracts using the hardhat-deploy plugin. The deployment script deploys contracts in thecontracts/ directory.

Prior to deploying the contracts, make sure to addcustomChain to the hardhat.config.js file. This is necessary to verify the

contracts on the custom chain.

```

Copy customChains:[ { network:"raspberry", chainId:123420111, urls:{ apiURL:"https://blockscout.op-celestia-testnet.gelato.digital/api", browserURL:"https://blockscout.op-celestia-testnet.gelato.digital", }, }, ], },

```

and add etherscan API key to the hardhat.config.js file.

```

Copy etherscan:{ apiKey:{ raspberry:"XXXX", }, }

```

Previous Deploy using Hardhat Next Services Last updated1 day ago On this page *Verify Contracts * Verify on the UI * Verify Programmatically