# OpenZeppelin Contract Verification

Instructions for various development environments

Page partially based onOpenZeppelin Tutorial . Due to compiler constraints, contracts which include OpenZeppelin-related source contracts (e.g.,import@openzeppelin/contracts/access/Ownable.sol; ) can cause problems with verification. The sources must be provided explicitly during the contract verification process.

Below we describe verification information when including these contracts specific to different development environments.



Hardhat

Hardhat is a full-featured development environment for contract compilation, deployment, and verification. TheHardhat Etherscan plugin supports contract verification on Blockscout and includes in-built functionality to address verification with OpenZeppelin-based imports.

A separate tutorial about contract verification via Hardhat on Blockscouts available here .

Truffle

Truffle is a world-class development environment, testing framework, and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM). Thetruffle-plugin-verify supports contract verification on BlockScout.

You can find more details abouttruffle-plugin-verify in theirReadme file andtutorial .

There are two main differences to keep in mind when verifying contracts on Blockscout vs Etherscan:

1. Blockscout does not require ApiKey to verify smart-contracts. However, truffle-plugin-verify requires one to be provided.You can use any non-empty string
2. .
3. In search of constructor arguments, the plugin makes a request to the explorer's?module=
4. account
5. &action=
6. txlist
7. &address={
8. addressHash
9. } endpoint
10. . This request can fail if verification is taking too long to answer.\
11. If you encounter this problem, specify the constructor arguments explicitly via the-forceConstructorArgs string:
12. option (e.g.,truffle run verify OpenZeppelinSample@0xf1b1B44B70f4531217c8c4B6C7A9a52D2B052F81 --network sokol --forceConstructorArgs string:
13. ). This way, thetxlist
14. request is omitted, and the problem should be resolved.Note: you can always use just an empty string (as in the example) even if there are some arguments in reality. Blockscout obtains them automatically and ignores any provided value.
15.

Foundry (forge)

Foundry is a smart contract development toolchain. Foundry manages your dependencies, compiles your project, runs tests, deploys, and lets you interact with the chain from the command line and via Solidity scripts.

Forge is a command-line tool that ships with Foundry. Forge tests, builds, and deploys your smart contracts. Forge supports contract verification out of the box (https://book.getfoundry.sh/reference/forge/forge-verify-contract )

Tips:

1. Setting the—verifier=blockscout
2. flag allows you to not specify any Api key.
3. While specifying--verifier-url
4. flag omit the final slash (e.g.,--verifier-url=
5. https://blockscout.com/poa/sokol/api
6. ). Otherwise, you will encounter an "Failed to deserialize response

7. ” error.
8. You can specify most configuration options (e.g., evm version, disabling optimizations) via the usual Forge configuration (seehttps://github.com/foundry-rs/foundry/blob/master/config/README.md
9. ). Note: You might not need an API key to verify on Blockscout. You can use the following format for the --verifier-url flag: /api?
10.

Example:

Verify a contract with Blockscout right after deployment (make sure you add "/api?" to the end of the Blockscout homepage explorer URL):

```

Copy forgecreate--rpc-url--private-keydevTestnetPrivateKeysrc/Contract.sol:SimpleStorage--verify--verifierblockscout--verifier-url/api\?

```

Remix

While there are 2 plugins available to help with this process, Etherscan and Flattener. Currently only Flattener works with Blockscout. You can use this plugin to flatten contracts and submit for verification as flattened source code. This is the recommended option.

We also include the Manual Standard Input JSON Method below the flattener instructions. This method is not recommended, but included since it is also a viable method.

Flattener Plugin

1. ?
2. ?
3. ?
4. ?
5. ?
6. ?
7. ?
8.

Manual Standard Input Json with Remix

Standard input json is basically the raw file fed into the Solidity compiler on verification. There are some minor updates made regarding info the output compiler should return, but the main contract-related fields remain the same. This way, standard json allows you to specify the most subtle compiler settings, as well as to specify all source files.

It is quite cumbersome and tedious work, so it is preferable to use any of the above methods, however, we describe the process in more detail for those who want to use it.

For this tutorial, we will create standard json for the following contract:

```

Copy // SPDX-License-Identifier: GPL-3.0

pragmasolidity>=0.7.0<0.9.0;

import"@openzeppelin/contracts/access/Ownable.sol";

contractOpenZeppelinSampleisOwnable{ uint256publica;

constructor(uint256_a) { a=_a; }

functionset(uint256_a)public{ a=_a; } }

```

1) Start by creating a template which will be filled with sources later. For this purpose, we will use a metadata file generated by the compiler. You can find it in the “Compilation Details” tab (ensure that correct contract is chosen).

?

```
```

Copy { "compiler":{ "version":"0.8.7+commit.e28d00a7" }, "language":"Solidity", "output":{ "abi":[ { "inputs":[ { "internalType":"uint256", "name":"_a", "type":"uint256" } ], "stateMutability":"nonpayable", "type":"constructor" }, { "anonymous":false, "inputs":[ { "indexed":true, "internalType":"address", "name":"previousOwner", "type":"address" }, { "indexed":true, "internalType":"address", "name":"newOwner", "type":"address" } ], "name":"OwnershipTransferred", "type":"event" }, { "inputs":[], "name":"a", "outputs":[ { "internalType":"uint256", "name":"", "type":"uint256" } ], "stateMutability":"view", "type":"function" }, { "inputs":[], "name":"owner", "outputs":[ { "internalType":"address", "name":"", "type":"address" } ], "stateMutability":"view", "type":"function" }, { "inputs":[], "name":"renounceOwnership", "outputs":[], "stateMutability":"nonpayable", "type":"function" }, { "inputs":[ { "internalType":"uint256", "name":"_a", "type":"uint256" } ], "name":"set", "outputs":[], "stateMutability":"nonpayable", "type":"function" }, { "inputs":[ { "internalType":"address", "name":"newOwner", "type":"address" } ], "name":"transferOwnership", "outputs":[], "stateMutability":"nonpayable", "type":"function" } ], "devdoc":{ "kind":"dev", "methods":{ "owner()":{ "details":"Returns the address of the current owner." }, "renounceOwnership()":{ "details":"Leaves the contract without owner. It will not be possible to call onlyOwner functions anymore. Can only be called by the current owner. NOTE: Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner." }, "transferOwnership(address)":{ "details":"Transfers ownership of the contract to a new account (newOwner). Can only be called by the current owner." } }, "version":1 }, "userdoc": { "kind":"user", "methods":{}, "version":1 } }, "settings":{ "compilationTarget":{ "contracts/OpenZeppelinSample.sol":"OpenZeppelinSample" }, "evmVersion":"london", "libraries":{}, "metadata":{ "bytecodeHash":"ipfs" }, "optimizer":{ "enabled":false, "runs":200 }, "remappings":[] }, "sources":{ "@openzeppelin/contracts/access/Ownable.sol":{ "keccak256":"0xa94b34880e3c1b0b931662cb1c09e5dfa6662f31cba80e07c5ee71cd135c9673", "license":"MIT", "urls":[ "bzz-raw://40fb1b5102468f783961d0af743f91b9980cf66b50d1d12009f6bb1869cea4d2", "dweb:/ipfs/QmYqEbJML4jB1GHbzD4cUZDtJg5wVwNm3vDJq1GbyDus8y" ] }, "@openzeppelin/contracts/utils/Context.sol": { "keccak256":"0xe2e337e6dde9ef6b680e07338c493ebea1b5fd09b43424112868e9cc1706bca7", "license":"MIT", "urls":[ "bzz-raw://6df0ddf21ce9f58271bdfaa85cde98b200ef242a05a3f85c2bc10a8294800a92", "dweb:/ipfs/QmRK2Y5Yc6BK7tGKkgsgn3aJEQGi5aakeSPZvS65PV8Xp3" ] }, "contracts/OpenZeppelinSample.sol":{ "keccak256":"0xca4fea32c78a11ca048ac26aa8655073e7a6ae534459ac4d07afbb55c9e751e0", "license":"GPL-3.0", "urls":[ "bzz-raw://8c5e775b4b8c2c445d056d2ff6adc97d258fdf86071be30dcb3b13f09d143efb", "dweb:/ipfs/QmauZE2ySRS9Xm9JF7FsfzzGitEjHfSxnykyxvMRsSF7Lr" ] } }, "version":1 }sol

```

2) The only fields required for standard json input are “language”, “settings”, and “sources”. In addition, only content is needed for sources. Add it for each source and remove everything unnecessary.

```

Copy { "language":"Solidity", "settings":{ "compilationTarget":{ "contracts/OpenZeppelinSample.sol":"OpenZeppelinSample" }, "evmVersion":"london", "libraries":{}, "metadata":{ "bytecodeHash":"ipfs" }, "optimizer":{ "enabled":false, "runs":200 }, "remappings":[] }, "sources":{ "@openzeppelin/contracts/access/Ownable.sol":{ "content":"" }, "@openzeppelin/contracts/utils/Context.sol":{ "content":"" }, "contracts/OpenZeppelinSample.sol":{ "content":"" } } }

```

Note that the only fields to fill in the contents for each source file. Everything else (including compiler settings) is already filled via the metadata file.

3) First, add the content for the mainOpenZeppelinSample.sol file

```

Copy { "language":"Solidity", "settings":{ "compilationTarget":{ "contracts/OpenZeppelinSample.sol":"OpenZeppelinSample" }, "evmVersion":"london", "libraries":{}, "metadata":{ "bytecodeHash":"ipfs" }, "optimizer":{ "enabled":false, "runs":200 }, "remappings":[] }, "sources":{ "@openzeppelin/contracts/access/Ownable.sol":{ "content":"" }, "@openzeppelin/contracts/utils/Context.sol":{ "content":"" }, "contracts/OpenZeppelinSample.sol":{ "content":"// SPDX-License-Identifier: GPL-3.0\n\npragma solidity >=0.7.0 <0.9.0;\n\nimport \"@openzeppelin/contracts/access/Ownable.sol\";\n\ncontract OpenZeppelinSample is Ownable {\n uint256 public a;\n\n constructor(uint256 _a) {\n a = _a;\n }\n\n function set(uint256 _a) public {\n a = _a;\n }\n}" } } }

```

4) Next, add the sources for @openzeppelin/contracts/access/Ownable.sol contract that is being imported. They can be found inside the.deps/npm/ directories, which will appear after contract compilation.

?

Copy the source code with the path used in imports:

```

Copy { "language":"Solidity", "settings":{ "compilationTarget":{ "contracts/OpenZeppelinSample.sol":"OpenZeppelinSample" }, "evmVersion":"london", "libraries":{}, "metadata":{ "bytecodeHash":"ipfs" }, "optimizer":{ "enabled":false, "runs":200 },

"remappings":[] }, "sources":{ "@openzeppelin/contracts/access/Ownable.sol":{ "content":"// SPDX-License-Identifier: MIT\n// OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)\n\npragma solidity ^0.8.0;\n\nimport \"../utils/Context.sol\";\n\n/**\n * @dev Contract module which provides a basic access control mechanism, where\n * there is an account (an owner) that can be granted exclusive access to\n * specific functions.\n *\n * By default, the owner account will be the one that deploys the contract. This\n * can later be changed with {transferOwnership}.\n *\n * This module is used through inheritance. It will make available the modifier\n * `onlyOwner`, which can be applied to your functions to restrict their use to\n * the owner.\n */\nabstract contract Ownable is Context {\n address private _owner;\n\n event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);\n\n /**\n * @dev Initializes the contract setting the deployer as the initial owner.\n */\n constructor() {\n _transferOwnership(_msgSender());\n }\n\n /**\n * @dev Throws if called by any account other than the owner.\n */\n modifier onlyOwner() {\n _checkOwner();\n _;\n }\n\n /**\n * @dev Returns the address of the current owner.\n */\n function owner() public view virtual returns (address) {\n return _owner;\n }\n\n /**\n * @dev Throws if the sender is not the owner.\n */\n function _checkOwner() internal view virtual {\n require(owner() == _msgSender(), \"Ownable: caller is not the owner\");\n }\n\n /**\n * @dev Leaves the contract without owner. It will not be possible to call\n * `onlyOwner` functions anymore. Can only be called by the current owner.\n *\n * NOTE: Renouncing ownership will leave the contract without an owner,\n * thereby removing any functionality that is only available to the owner.\n */\n function renounceOwnership() public virtual onlyOwner {\n _transferOwnership(address(0));\n }\n\n /**\n * @dev Transfers ownership of the contract to a new account (`newOwner`).\n * Can only be called by the current owner.\n */\n function transferOwnership(address newOwner) public virtual onlyOwner {\n require(newOwner != address(0), \"Ownable: new owner is the zero address\");\n _transferOwnership(newOwner);\n }\n\n /**\n * @dev Transfers ownership of the contract to a new account (`newOwner`).\n * Internal function without access restriction.\n */\n function _transferOwnership(address newOwner) internal virtual {\n address oldOwner = _owner;\n _owner = newOwner;\n emit OwnershipTransferred(oldOwner, newOwner);\n }\n}" }, "@openzeppelin/contracts/utils/Context.sol":{ "content":"" }, "contracts/OpenZeppelinSample.sol":{ "content":"// SPDX-License-Identifier: GPL-3.0\n\npragma solidity >=0.7.0 <0.9.0;\n\nimport \"@openzeppelin/contracts/access/Ownable.sol\";\n\ncontract OpenZeppelinSample is Ownable {\n uint256 public a;\n\n constructor(uint256 _a) {\n a = _a;\n }\n\n function set(uint256 _a) public {\n a = _a;\n }\n}" } } }

```

5) The last source is a "../utils/Context.sol " imported from@openzeppelin/contracts/access/Ownable.sol . You can find it in the same.deps/npm directory.

?

Although import "../utils/Context.sol"; specifies a relative path, the absolute path is used in standard json, and can be found relative to @openzeppelin/contracts/access/Ownable.sol ```

Copy { "language":"Solidity", "settings":{ "compilationTarget":{ "contracts/OpenZeppelinSample.sol":"OpenZeppelinSample" }, "evmVersion":"london", "libraries":{}, "metadata":{ "bytecodeHash":"ipfs" }, "optimizer":{ "enabled":false, "runs":200 }, "remappings":[] }, "sources":{ "@openzeppelin/contracts/access/Ownable.sol":{ "content":"// SPDX-License-Identifier: MIT\n// OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)\n\npragma solidity ^0.8.0;\n\nimport \"../utils/Context.sol\";\n\n/**\n * @dev Contract module which provides a basic access control mechanism, where\n * there is an account (an owner) that can be granted exclusive access to\n * specific functions.\n *\n * By default, the owner account will be the one that deploys the contract. This\n * can later be changed with {transferOwnership}.\n *\n * This module is used through inheritance. It will make available the modifier\n * `onlyOwner`, which can be applied to your functions to restrict their use to\n * the owner.\n */\nabstract contract Ownable is Context {\n address private _owner;\n\n event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);\n\n /**\n * @dev Initializes the contract setting the deployer as the initial owner.\n */\n constructor() {\n _transferOwnership(_msgSender());\n }\n\n /**\n * @dev Throws if called by any account other than the owner.\n */\n modifier onlyOwner() {\n _checkOwner();\n _;\n }\n\n /**\n * @dev Returns the address of the current owner.\n */\n function owner() public view virtual returns (address) {\n return _owner;\n }\n\n /**\n * @dev Throws if the sender is not the owner.\n */\n function _checkOwner() internal view virtual {\n require(owner() == _msgSender(), \"Ownable: caller is not the owner\");\n }\n\n /**\n * @dev Leaves the contract without owner. It will not be possible to call\n * `onlyOwner` functions anymore. Can only be called by the current owner.\n *\n * NOTE: Renouncing ownership will leave the contract without an owner,\n * thereby removing any functionality that is only available to the owner.\n */\n function renounceOwnership() public virtual onlyOwner {\n _transferOwnership(address(0));\n }\n\n /**\n * @dev Transfers ownership of the contract to a new account (`newOwner`).\n * Can only be called by the current owner.\n */\n function transferOwnership(address newOwner) public virtual onlyOwner {\n require(newOwner != address(0), \"Ownable: new owner is the zero address\");\n _transferOwnership(newOwner);\n }\n\n /**\n * @dev Transfers ownership of the contract to a new account (`newOwner`).\n * Internal function without access restriction.\n */\n function _transferOwnership(address newOwner) internal virtual {\n address oldOwner = _owner;\n _owner = newOwner;\n emit OwnershipTransferred(oldOwner, newOwner);\n }\n}" }, "@openzeppelin/contracts/utils/Context.sol":{ "content":"// SPDX-License-Identifier: MIT\n// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)\n\npragma solidity ^0.8.0;\n\n/**\n * @dev Provides information about the current execution context, including the\n * sender of the transaction and its data. While these are generally available\n * via msg.sender and msg.data, they should not be accessed in such a direct\n * manner, since when dealing with meta-transactions the account sending and\n * paying for execution may not be the actual sender (as far as an application\n * is
```

concerned).\n \n * This contract is only required for intermediate, library-like contracts.\n/\nabstract contract Context {\n function _msgSender() internal view virtual returns (address) {\n return msg.sender;\n }\n\n function _msgData() internal view virtual returns (bytes calldata) {\n return msg.data;\n }\n}" }, "contracts/OpenZeppelinSample.sol":{ "content":"// SPDX-License-Identifier: GPL-3.0\n\npragma solidity >=0.7.0 <0.9.0;\n\nimport \"@openzeppelin/contracts/access/Ownable.sol\";\n\ncontract OpenZeppelinSample is Ownable {\n uint256 public a;\n\n constructor(uint256 _a) {\n a = _a;\n }\n\n function set(uint256 _a) public {\n a = _a;\n }\n}" } } }

```

6) Submit Standard JSON input for verification.

?

Last updated2 months ago
```