eth-merkle-bridge

Release 0.4.0

Feb 13, 2020

Contents:

1 Merkle bridge design. 3
- 1.1 Getting started . . . 3
  - 1.1.1 Download . . . 3
  - 1.1.2 Install . . . 3
- 1.2 Using the EthAergo CLI . . . 4
  - 1.2.1 CLI for the proposer/validator . . . 4
  - 1.2.2 CLI for asset transfers . . . 8
- 1.3 Proposer . . . 11
  - 1.3.1 Starting a Proposer . . . 11
  - 1.3.2 Updating bridge settings . . . 13
- 1.4 Validator . . . 13
  - 1.4.1 Starting a Validator . . . 14
  - 1.4.2 Updating bridge settings . . . 15
- 1.5 Deploying a new bridge . . . 15
  - 1.5.1 Process . . . 15
  - 1.5.2 Create a new config file . . . 16
  - 1.5.3 Deploy the bridge contracts . . . 17
  - 1.5.4 Transfer control of the bridge to the multisig oracle . . . 18
  - 1.5.5 Send native aer to the bridge contract . . . 19
- 1.6 Configuration file . . . 19
- 1.7 Unfreeze service . . . 21
  - 1.7.1 Starting the unfreeze grpc service . . . 22
  - 1.7.2 Starting the Envoy proxy . . . 22
- 1.8 `bridge_operator` . . . 23
- 1.9 `ethaergo_wallet` . . . 28
- 1.10 `ethaergo_cli` . . . 31
- 1.11 `unfreeze_service` . . . 34

2 Indices and tables 35

Python Module Index 37

Index 39

The EthAergo bridge is an efficient and descentralized way of connecting blockchains. The repository is the POC implementation of Aergo's bridge design between Aergo and Ethereum networks.

CHAPTER 1

Merkle bridge design.

https://medium.com/coinmonks/merkle-bridge-32bd0f06c308

In order to transfer an asset from one blockchain to another blockchain, it should be locked on it's origin chain and minted on the destination chain. At all times the minted assets should be pegged to the locked assets.

The EthAergo Merkle Bridge enables decentralized custody and efficient minting of assets.

At regular intervals, a proposer publishes the block state root of each chain on the other connected chain's oracle contract. The state root is recorded only if it has been signed by 2/3 of validators. Validators only sign the general block state root, and the proposer creates a Merkle proof, proving that the bridge contract storage state is included in the general block state. Users can then independently mint assets on the destination bridge contract by verifying a merkle proof of their locked assets with the anchored storage root.

The proposers do not need to watch and validate user transfers: the benefit of the merkle bridge design comes from the fact that validators simply make sure that the state roots they sign are correct. Since onchain signature verification is only done once per root anchor, it is possible use a large number of validators for best safety and sensorship resistance.

1.1 Getting started

1.1.1 Download

```
$ git clone git@github.com:aergoio/eth-merkle-bridge.git
```

1.1.2 Install

Install dependencies

```
$ cd eth-merkle-bridge
$ virtualenv -p python3 venv
$ source venv/bin/activate
$ pip install -r requirements.txt
```

Optional dev dependencies (lint, testing. . .)

```
$ pip install -r dev-dependencies.txt
```

Now you can start using the bridge tools to:

- create a configuration file with the cli
- deploy a new bridge
- start a proposer
- start a validator
- update bridge settings
- transfer assets through the bridge with the cli

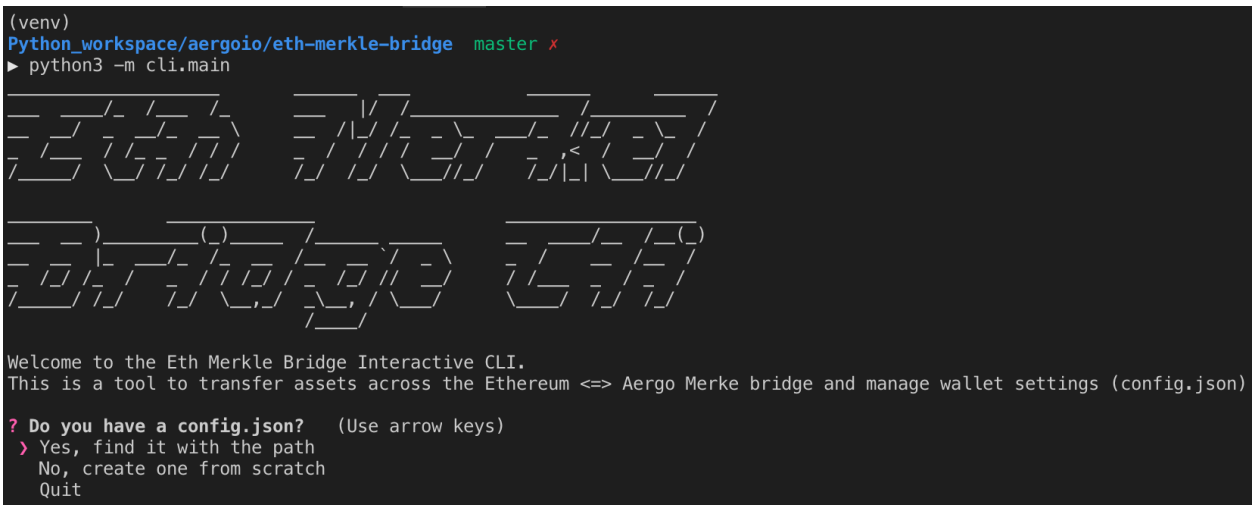
1.2 Using the EthAergo CLI

1.2.1 CLI for the proposer/validator

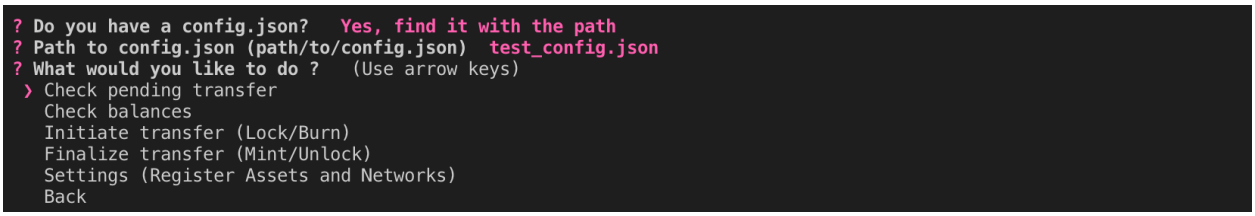
Start the cli:

```
$ python3 -m ethaergo_cli.main
```

The first step is to create a config file or load an existing one



Then the main menu appears with cli functionalty:



These are the settings available from the cli

```
? What would you like to do ?   Settings (Register Assets and Networks)
? What would you like to do ?   (Use arrow keys)
 ❯ Register new asset
   Register new network
   Register new bridge
   Register new set of validators
   Update anchoring periode
   Update finality
   Back
```

Creating a config file from scratch

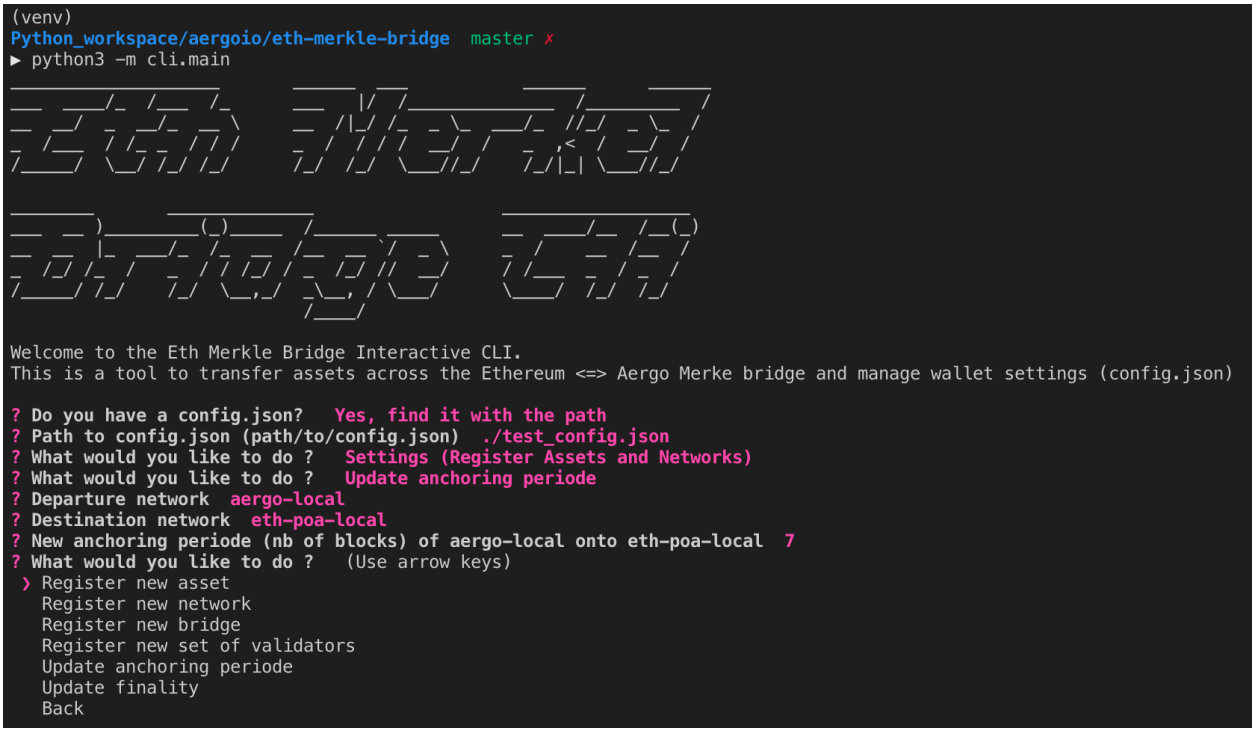
```
▶ python3 -m ethaergo_cli.main
 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|

 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|

Welcome to the Eth Merkle Bridge Interactive CLI.
This is a tool to transfer assets across the Ethereum <=> Aergo Merke bridge and manage wallet settings (config.json)

? Do you have a config.json?  No, create one from scratch
Let's register 2 networks, validators(optional) and a private key for interacting with each network.
? Network name  ropsten
? Network IP  ...
? Network type  ethereum
? Is this an Ethereum POA network ?  No
? Add redundant providers for validator data source security ?  Yes
? Provider ip  ...
? Add next provider ?  Yes
? Provider ip  ...
? Add next provider ?  No
? Network name  aergo-testnet
? Network IP  ...
? Network type  aergo
? Add redundant providers for validator data source security ?  Yes
? Provider ip  ...
? Add next provider ?  Yes
? Provider ip  ...
? Add next provider ?  No
? Would you like to register a bridge ?  Yes
Bridge between ropsten and aergo-testnet
? Bridge contract address on ropsten
? Oracle address on ropsten
? Anchoring periode of aergo-testnet on ropsten  3600
? Finality of aergo-testnet  3600
? Bridge contract address on aergo-testnet
? Oracle contract address on aergo-testnet
? Anchoring periode of ropsten on aergo-testnet  360
? Finality of ropsten  360
? Aergo native unfreeze fee  100000
? Path to Ethereum bridge abi text file  contracts/solidity/bridge_abi.txt
? Path to Ethereum bridge minted token abi text file  contracts/solidity/bridge_abi.txt
? Path to Ethereum oracle abi text file  contracts/solidity/oracle_abi.txt
? Would you like to register validators ? (not needed for bridge users)  Yes
WARNING : Validators must be registered in the correct order
? Aergo Address  AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ
? Ethereum Address  0x210467b3849a408c3a3bEE14b4627aa57F342134
? Validator ip  ..
? Add next validator ?  Yes
? Aergo Address  ...
? Ethereum Address  ...
? Validator ip  ...
? Add next validator ?  No
Register a private key for ropsten
? Give your key a short descriptive name  proposer
? Path to json keystore  keystore/UTC--2019-05-13T09-26-29.993150000Z--c19b695911414436 76a3ee56fbf1d3ea869d53d8
? Ethereum address matching private key  0xc19b695911414436 76a3EE56fbf1d3EA869d53D8
Register a private key for aergo-testnet
? Give your key a short descriptive name  proposer
? Encrypted exported key string  47sDAWjMFTP7r2JP2BJ29PJRfY13yUTtVvoLjAf8knhH4GryQrpMJoTqscDjed1YPHVZXY4sN
? Aergo address matching private key  AmPxVdu993eosN3UjnPDdN3wb7TNbHeiHDvn2dvZUcH8KXDK3RLU
? Path to save new config file  config.json
```

Registering a new bridge

```
(venv)
Python_workspace/aergoio/eth-merkle-bridge  master ✗
▶ python3 -m cli.main
 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|

 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|

Welcome to the Eth Merkle Bridge Interactive CLI.
This is a tool to transfer assets across the Ethereum <=> Aergo Merke bridge and manage wallet settings (config.json)

? Do you have a config.json?  Yes, find it with the path
? Path to config.json (path/to/config.json)  dummy.json
? What would you like to do ?  Settings (Register Assets and Networks)
? What would you like to do ?  Register new bridge
? Departure network  aergo-local
? Destination network  eth-poa-local
Bridge between aergo-local and eth-poa-local
? Bridge contract address on aergo-local  AmgQqVWX3JADRBEVkVCM4CyWdoeXuumeYGGJJxEeoAukRC26hxmw
? Anchoring periode of eth-poa-local on aergo-local  6
? Finality of eth-poa-local  4
? Bridge contract address on eth-poa-local  0x89eD1D1C145F6bF3A7e62d2B8eB0e1Bf15Cb2374
? Anchoring periode of aergo-local on eth-poa-local  7
? Finality of aergo-local  5
? Path to Ethereum bridge abi text file  ./contracts/solidity/bridge_abi.txt
? Path to Ethereum bridge minted token abi text file  ./contracts/solidity/minted_erc20_abi.txt
? What would you like to do ?  (Use arrow keys)
 ❯ Register new asset
   Register new network
   Register new bridge
   Register new set of validators
   Update anchoring periode
   Update finality
   Back
```

When creating the initial config file for a bridge that is not yet deployed, leave oracle and bridge addresses empty.

Updating bridge settings

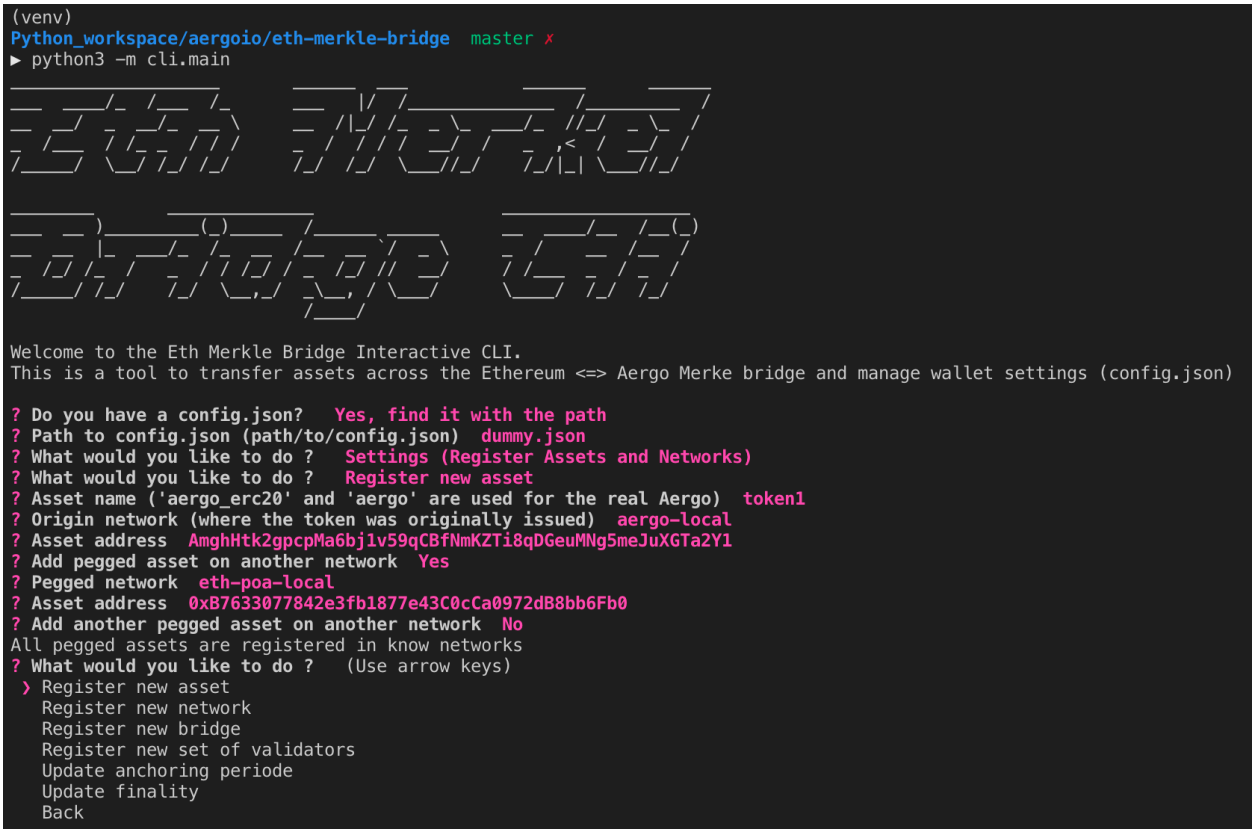


If the new anchoring periode reached validator consensus, it can then be automatically updated in the bridge contract by the proposer.

```
proposer.aergo: "Anchoring periode update requested: 7"
proposer.aergo: " tAnchorUpdate success"
```

1.2.2 CLI for asset transfers

Registering a new asset in config file



Transfering a registered asset

```
(venv)
Python_workspace/aergoio/eth-merkle-bridge  master ✗
▶ python3 -m cli.main
```

```
Welcome to the Eth Merkle Bridge Interactive CLI.
This is a tool to transfer assets across the Ethereum <=> Aergo Merke bridge and manage wallet settings (config.json)

? Do you have a config.json?   Yes, find it with the path
? Path to config.json (path/to/config.json)  ./test_config.json
? What would you like to do ?   Initiate transfer (Lock/Burn)
? Departure network  aergo-local
? Destination network  eth-poa-local
? Name of asset to transfer  token1
? Receiver of assets on other side of bridge  0xfec3c905bcd3d9a5471452e53f82106844cb1e76
? Amount of assets to transfer  22.5
? Choose account to sign transaction :   default
Lock transfer summary:
Departure chain: aergo-local (AmgQqVWX3JADRBEVkVCM4CyWdoeXuumeYGGJJxEeoAukRC26hxmw)
Destination chain: eth-poa-local (0x89eD1D1C145F6bF3A7e62d2B8eB0e1Bf15Cb2374)
Asset name: token1 (AmghHtk2gpcpMa6bj1v59qCBfNmKZTi8qDGeuMNg5meJuXGTa2Y1)
Receiver at destination: 0xfec3c905bcd3d9a5471452e53f82106844cb1e76
Amount: 22500000000000000000

? Confirm you want to execute tranfer tx  Yes, execute transfer

aergo-local -> eth-poa-local
Decrypt exported private key 'default'
Password:
💰 token1 balance on origin before transfer: 500000000.0
🔒 Lock success:  EEXtfFNwNERdzM6WUC5bf75wFEAZ9jKaq3mUPTFLewVf
💰 remaining token1 balance on origin after transfer: 499999977.5
Transaction Hash : EEXtfFNwNERdzM6WUC5bf75wFEAZ9jKaq3mUPTFLewVf
Block Height : 12701
```

```
? What would you like to do ?   Finalize transfer (Mint/Unlock)
? Choose a pending transfer  ['aergo-local', 'eth-poa-local', 'token1', '0xfec3c905bcd3d9a5471452e53f82106844cb1e76', 12701]
? Choose account to sign transaction :   default
Mint transfer summary:
Departure chain: aergo-local (AmgQqVWX3JADRBEVkVCM4CyWdoeXuumeYGGJJxEeoAukRC26hxmw)
Destination chain: eth-poa-local (0x89eD1D1C145F6bF3A7e62d2B8eB0e1Bf15Cb2374)
Asset name: token1 (AmghHtk2gpcpMa6bj1v59qCBfNmKZTi8qDGeuMNg5meJuXGTa2Y1)
Receiver at destination: 0xfec3c905bcd3d9a5471452e53f82106844cb1e76
Block height of lock/burn/freeze: 12701

? Confirm you want to execute tranfer tx  Yes, execute transfer

aergo-local -> eth-poa-local
Decrypt Ethereum keystore 'default'
Password:
💰 token1 balance on destination before transfer : 0.0
⚙ Built lock proof
⛏ Mint success:  0x4f75ed92da344c4e9f759d239c75e6149e959122790e8cbe76eb92b143c0a25d
💰 token1 balance on destination after transfer : 22.5
? What would you like to do ?   (Use arrow keys)
 ❯ Check pending transfer
   Check balances
   Initiate transfer (Lock/Burn)
   Finalize transfer (Mint/Unlock)
   Settings (Register Assets and Networks)
   Back
```

Check pending transfers

It is possible to check withdrawable balances of pending transfer between chains.

```
Welcome to the Eth Merkle Bridge Interactive CLI.
This is a tool to transfer assets across the Ethereum <=> Aergo Merke bridge and manage wallet settings (config.json)

? Do you have a config.json?   Yes, find it with the path
? Path to config.json (path/to/config.json)  test_config.json
? What would you like to do ?   Initiate transfer (Lock/Burn)
? Departure network  eth-poa-local
? Destination network  aergo-local
? Name of asset to transfer  aergo_erc20
? Receiver of assets on other side of bridge  AmNMFbiVsqy6vg4njsTjgy7bKPFHFYhLV4rzQyrENUS9AM1e3tw5
? Amount of assets to transfer  3.3
? Choose account to sign transaction :   default
Lock transfer summary:
Departure chain: eth-poa-local (0x89eD1D1C145F6bF3A7e62d2B8eB0e1Bf15Cb2374)
Destination chain: aergo-local (AmgQqVWX3JADRBEVkVCM4CyWdoeXuumeYGGJJxEeoAukRC26hxmw)
Asset name: aergo_erc20 (0xd898383A12CDE0eDF7642F7dD4D7006FdE5c433e)
Receiver at destination: AmNMFbiVsqy6vg4njsTjgy7bKPFHFYhLV4rzQyrENUS9AM1e3tw5
Amount: 3300000000000000000

? Confirm you want to execute tranfer tx  Yes, execute transfer

eth-poa-local -> aergo-local
Decrypt Ethereum keystore 'default'
Password:
💰 aergo_erc20 balance on destination before transfer : 500000000.0
⬆ Increase approval success:  0x96dd0049066aa1fcd6bb5d0f0b3ac03340db2d8ee56587e8dc1fe0ce020b1846
🔒 Lock success:  0xe68d9e9147b733aaa398799684cf4d196af4eae5f2cfc2fb0e39b28c39a0cb0b
💰 remaining aergo_erc20 balance on origin after transfer: 499999996.7
Transaction Hash : 0xe68d9e9147b733aaa398799684cf4d196af4eae5f2cfc2fb0e39b28c39a0cb0b
Block Height : 87

? What would you like to do ?   Check pending transfer
? Choose a pending transfer  ['eth-poa-local', 'aergo-local', 'aergo_erc20', 'AmNMFbiVsqy6vg4njsTjgy7bKPFHFYhLV4rzQyrENUS9AM1e3tw5', 87]
Withdrawable: 3.3  Pending: 0.0
? What would you like to do ?   Check pending transfer
? Choose a pending transfer  Custom transfer
? Departure network  eth-poa-local
? Destination network  aergo-local
? Name of asset to transfer  aergo_erc20
? Receiver of assets on other side of bridge  AmNMFbiVsqy6vg4njsTjgy7bKPFHFYhLV4rzQyrENUS9AM1e3tw5
Withdrawable: 3.3  Pending: 0.0
? What would you like to do ?   (Use arrow keys)
 ❯ Check pending transfer
   Check balances
   Initiate transfer (Lock/Burn)
   Finalize transfer (Mint/Unlock)
   Settings (Register Assets and Networks)
   Back
```

If a transfer was made with the cli, the transfer parameters are recorded but it is also possible to check the withdrawable balance of a custom transfer between any chain. 'Withdrawable' is the balance that can be immediatly withdrawn on the other side of the bridge. 'Pending' is the balance that was deposited in the bridge contract but the anchor has not happened on the other side of the bridge so it is not yet withdrawable.

Pending transfers are recorded as an array of [departure chain, destination chain, asset name, receiver, block height of lock/burn/freeze]. All pending transfer are store in ethaergo_cli/pending_transfers.json and deleted once finalized.

1.3 Proposer

A proposer connects to all validators and requests them to sign a new anchor with the GetEthAnchorSignature and GetAergoAnchorSignature rpc requests. To prevent downtime, anybody can become a proposer and request signatures to validators. It is the validator's responsibility to only sign correct anchors. The bridge contracts will not update the state root if the anchoring time is not reached (t_anchor).

1.3.1 Starting a Proposer

```
$ python3 -m ethaergo_bridge_operator.proposer.client --help
```

(continues on next page)

(continued from previous page)

```
    usage: client.py [-h] -c CONFIG_FILE_PATH -a AERGO -e ETH --eth_block_time
                ETH_BLOCK_TIME [--privkey_name PRIVKEY_NAME]
                [--privkey_pwd PRIVKEY_PWD] [--anchoring_on] [--auto_update]
                [--oracle_update] [--eth_gas_price ETH_GAS_PRICE]
                [--aergo_gas_price AERGO_GAS_PRICE] [--eco] [--eth_eco]

    Start a proposer on Ethereum and Aergo.

    optional arguments:
    -h, --help            show this help message and exit
    -c CONFIG_FILE_PATH, --config_file_path CONFIG_FILE_PATH
                          Path to config.json
    -a AERGO, --aergo AERGO
                          Name of Aergo network in config file
    -e ETH, --eth ETH     Name of Ethereum network in config file
    --eth_block_time ETH_BLOCK_TIME
                          Average Ethereum block time
    --privkey_name PRIVKEY_NAME
                          Name of account in config file to sign anchors
    --privkey_pwd PRIVKEY_PWD
                          Password to decrypt privkey_name(Eth and Aergo Keys
                          need the same password)
    --anchoring_on        Enable anchoring (can be diseabled when wanting to
                          only update settings)
    --auto_update         Update bridge contract when settings change in config
                          file
    --oracle_update       Update bridge contract when validators or oracle addr
                          change in config file
    --eth_gas_price ETH_GAS_PRICE
                          Gas price (gWei) to use in transactions
    --aergo_gas_price AERGO_GAS_PRICE
                          Gas price to use in transactions
    --eco                 In eco mode, anchoring will be skipped when
                          lock/burn/freeze events don't happen in the bridge
                          contracts
    --eth_eco             In eco mode, anchoring on Ethereum will be skipped
                          when lock/burn/freeze events don't happen in the
                          bridge contracts on Aergo

$ python3 -m ethaergo_bridge_operator.proposer.client -c './test_config.json' -a
↪'aergo-local' -e 'eth-poa-local' --eth_block_time 3 --privkey_name "proposer" --
↪anchoring_on

    proposer.eth: "Connect Aergo and Ethereum providers"
    proposer.eth: "aergo-local (t_final=5 ) -> eth-poa-local : t_anchor=7"
    proposer.eth: "Proposer Address: 0xc19b69591141443676a3EE56fbf1d3EA869d53D8"
    proposer.eth: "Connect to EthValidators"
    proposer.eth: "Validators: ['0x210467b3849a408c3a3bEE14b4627aa57F342134',
↪'0x210467b3849a408c3a3bEE14b4627aa57F342134',
↪'0x210467b3849a408c3a3bEE14b4627aa57F342134']"
    proposer.aergo: "Connect Aergo and Ethereum providers"
    proposer.aergo: "aergo-local <- eth-poa-local (t_final=4) : t_anchor=6"
    proposer.aergo: "Proposer Address:␣
↪AmPxVdu993eosN3UjnPDdN3wb7TNbHeiHDvn2dvZUcH8KXDK3RLU"
    proposer.aergo: "Connect to AergoValidators"
    proposer.aergo: "Validators: [
↪'AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ',
↪'AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ',
↪'AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ']"
```

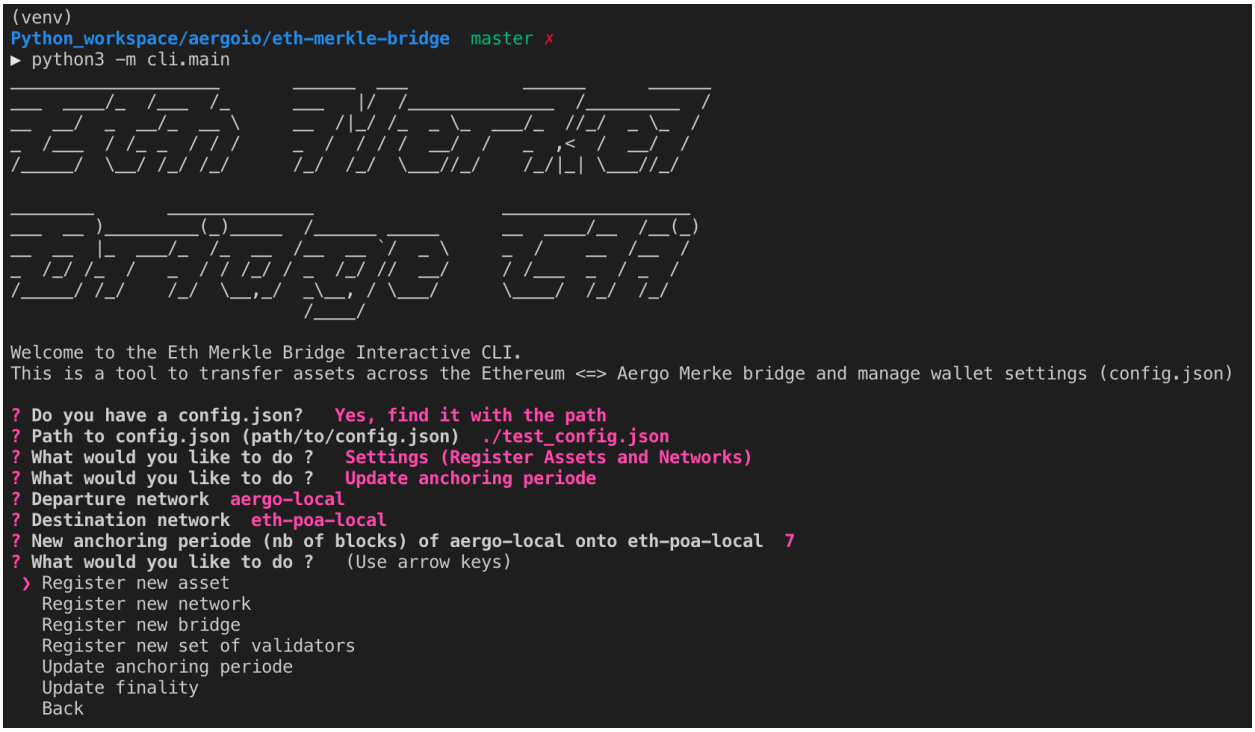
(continues on next page)

(continued from previous page)

```
    proposer.eth: "Start Eth proposer"
    proposer.aergo: "Current Eth -> Aergo  anchor: height: 0, root: 0xconstructor,␣
↪nonce: 0"
    proposer.aergo: " Gathering validator signatures for: root:␣
↪0xd97d33cb90c9e58befdba86467907ba68258b49f0f85a22781db7c4eda3033e4, height: 8262'"
    proposer.eth: "Current Aergo -> Eth  anchor: height: 0, root:␣
↪0x0000000000000000000000000000000000000000000000000000000000000000, nonce: 0"
    proposer.eth: " Gathering validator signatures for: root:␣
↪0x5d471941372b64d66361c29fca4e13c899819afe212cce87143794d80b510613, height: 8280'"
    proposer.eth: " Anchor success,  wait until next anchor time: 7s..."
    proposer.eth: " Gas used: 109287"
    proposer.aergo: " Anchor success,  wait until next anchor time: 6s..."
```

1.3.2 Updating bridge settings

Bridge settings are updated when the config file changes and the proposer is started with –auto_update The proposer will then try to gather signatures from validators to make the update on chain.



If the new anchoring periode reached validator consensus, it can then be automatically updated in the bridge contract by the proposer.

```
proposer.aergo: "Anchoring periode update requested: 7"
proposer.aergo: " tAnchorUpdate success"
```

1.4 Validator

A validator will sign any state root from any proposer via the GetAnchorSignature rpc request as long as it is valid. Therefore a validator must run a full node. Assets on the sidechain are secure as long as 2/3 of the validators validate

both chains and are honnest. Since signature verification only happens when anchoring (and not when transfering assets), the number of validators can be very high as the signature verification cost is necessary only once per anchor.

1.4.1 Starting a Validator

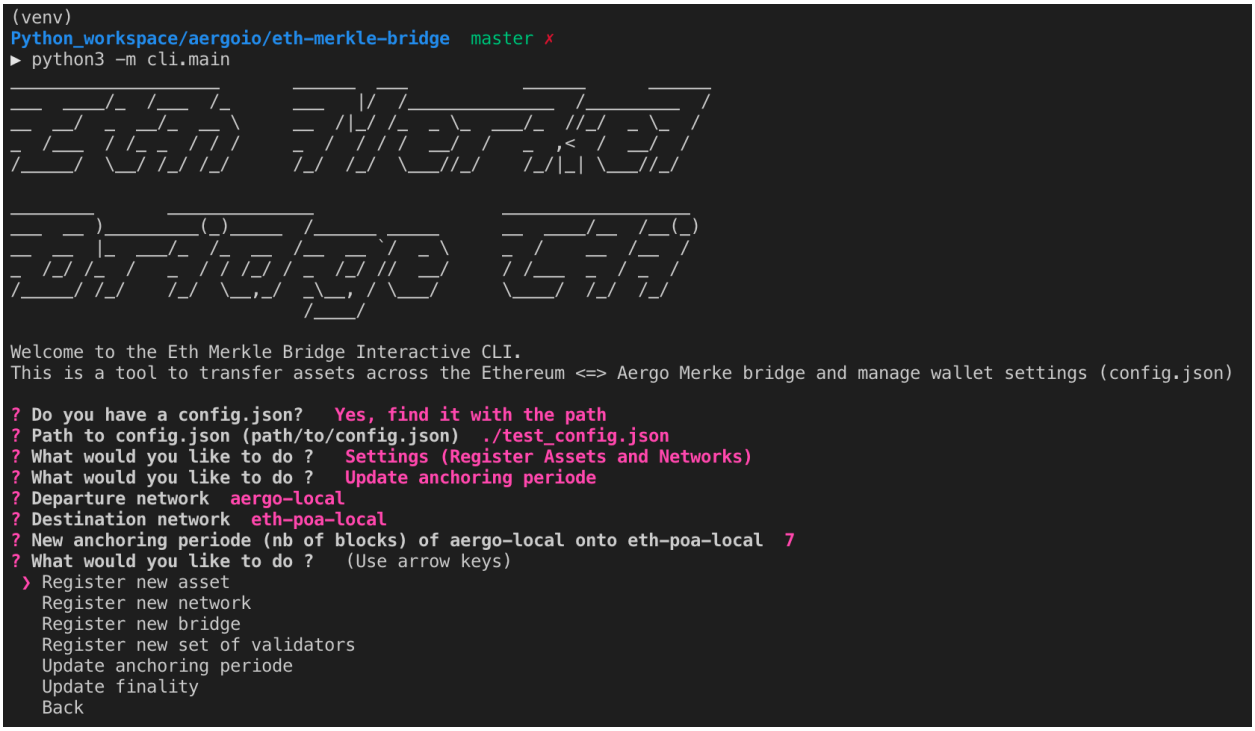
```
$ python3 -m ethaergo_bridge_operator.validator.server --help

    usage: server.py [-h] -c CONFIG_FILE_PATH -a AERGO -e ETH -i VALIDATOR_INDEX
                     [--privkey_name PRIVKEY_NAME] [--anchoring_on]
                     [--auto_update] [--oracle_update] [--local_test]

    Start a validator on Ethereum and Aergo.

    optional arguments:
    -h, --help            show this help message and exit
    -c CONFIG_FILE_PATH, --config_file_path CONFIG_FILE_PATH
                            Path to config.json
    -a AERGO, --aergo AERGO
                            Name of Aergo network in config file
    -e ETH, --eth ETH     Name of Ethereum network in config file
    -i VALIDATOR_INDEX, --validator_index VALIDATOR_INDEX
                            Index of the validator in the ordered list of
                            validators
    --privkey_name PRIVKEY_NAME
                            Name of account in config file to sign anchors
    --anchoring_on        Enable anchoring (can be diseabled when wanting to
                            only update settings)
    --auto_update         Update bridge contract when settings change in config
                            file
    --oracle_update       Update bridge contract when validators or oracle addr
                            change in config file
    --local_test          Start all validators locally for convenient testing

$ python3 -m ethaergo_bridge_operator.validator.server -c './test_config.json' -a
↪'aergo-local' -e 'eth-poa-local' --validator_index 1 --privkey_name "validator" --
↪auto_update

    "Connect Aergo and Ethereum"
    "Current Aergo validators : ['AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ
↪', 'AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ',
↪'AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ']"
    "Current Ethereum validators : ['0x210467b3849a408c3a3bEE14b4627aa57F342134',
↪'0x210467b3849a408c3a3bEE14b4627aa57F342134',
↪'0x210467b3849a408c3a3bEE14b4627aa57F342134']"
    "aergo-local <- eth-poa-local (t_final=4) : t_anchor=6"
    "aergo-local (t_final=5) -> eth-poa-local : t_anchor=7"
    "WARNING: This validator will vote for settings update in config.json"
    Decrypt Aergo and Ethereum accounts 'validator'
    Password:
    "Aergo validator Address: AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ"
    "Ethereum validator Address: 0x210467b3849a408c3a3bEE14b4627aa57F342134"
    "server 1 started"
    {"val_index": 1, "signed": true, "type": " anchor", "value": {"root":
↪"0xd97d33cb90c9e58befdba86467907ba68258b49f0f85a22781db7c4eda3033e4", "height":␣
↪8102}, "destination": "aergo-local", "nonce": 0}
    {"val_index": 1, "signed": true, "type": " anchor", "value": {"root":
↪"0x5d471941372b64d66361c29fca4e13c899819afe212cce87143794d80b510613", "height":␣
↪8119}, "destination": "eth-poa-local", "nonce": 0}
```

(continues on next page)

(continued from previous page)

1.4.2 Updating bridge settings

The information (validator set, anchoring periods, finality of blockchains) contained in the config file will be used by the validator to vote on changes if –auto_update is enabled. Be careful that the information in config file is correct as any proposer can request a signature of that information. If the proposer gathers 2/3 signatures for the same information them the bridge settings can be updated.



1.5 Deploying a new bridge

1.5.1 Process

1- Each Validator generates a private key and address to sign bridge messages (anchors, settings update. . .) and shares the address and validator ip with the bridge Proposer.

2- Proposer creates a config.json file draft. (See *Create a new config file* below).

3- Proposer deploys the eth-merkle-birdge.sol and eth-merkle-bridge.lua contracts (See *Deploy the bridge contracts* below).

4- Proposer deploys the oracle.sol and oracle.lua and transfers bridge controle to oracles (See *Transfer control of the bridge to the multisig oracle* below).

5- Proposer removes his private key registered in config.json, and shares config.json with Validators.

6- Each Validator adds his private key to his config.json.

7- The Validators start validating (see validator docs) with the correct validator index (see position of validator in config.json).

8- Proposer starts operating the bridge (see proposer docs).

1.5.2 Create a new config file

A config file can be created with the cli tool or manually.

```
▶ python3 -m ethaergo_cli.main
 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|

 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|

Welcome to the Eth Merkle Bridge Interactive CLI.
This is a tool to transfer assets across the Ethereum <=> Aergo Merke bridge and manage wallet settings (config.json)

? Do you have a config.json?  No, create one from scratch
Let's register 2 networks, validators(optional) and a private key for interacting with each network.
? Network name  ropsten
? Network IP  ...
? Network type  ethereum
? Is this an Ethereum POA network ?  No
? Add redundant providers for validator data source security ?  Yes
? Provider ip  ...
? Add next provider ?  Yes
? Provider ip  ...
? Add next provider ?  No
? Network name  aergo-testnet
? Network IP  ...
? Network type  aergo
? Add redundant providers for validator data source security ?  Yes
? Provider ip  ...
? Add next provider ?  Yes
? Provider ip  ...
? Add next provider ?  No
? Would you like to register a bridge ?  Yes
Bridge between ropsten and aergo-testnet
? Bridge contract address on ropsten
? Oracle address on ropsten
? Anchoring periode of aergo-testnet on ropsten  3600
? Finality of aergo-testnet  3600
? Bridge contract address on aergo-testnet
? Oracle contract address on aergo-testnet
? Anchoring periode of ropsten on aergo-testnet  360
? Finality of ropsten  360
? Aergo native unfreeze fee  100000
? Path to Ethereum bridge abi text file  contracts/solidity/bridge_abi.txt
? Path to Ethereum bridge minted token abi text file  contracts/solidity/bridge_abi.txt
? Path to Ethereum oracle abi text file  contracts/solidity/oracle_abi.txt
? Would you like to register validators ? (not needed for bridge users)  Yes
WARNING : Validators must be registered in the correct order
? Aergo Address  AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ
? Ethereum Address  0x210467b3849a408c3a3bEE14b4627aa57F342134
? Validator ip  ..
? Add next validator ?  Yes
? Aergo Address  ...
? Ethereum Address  ...
? Validator ip  ...
? Add next validator ?  No
Register a private key for ropsten
? Give your key a short descriptive name  proposer
? Path to json keystore  keystore/UTC--2019-05-13T09-26-29.993150000Z--c19b69591141443676a3ee56fbf1d3ea869d53d8
? Ethereum address matching private key  0xc19b69591141443676a3EE56fbf1d3EA869d53D8
Register a private key for aergo-testnet
? Give your key a short descriptive name  proposer
? Encrypted exported key string  47sDAWjMFTP7r2JP2BJ29PJRfY13yUTtVvoLjAf8knhH4GryQrpMJoTqscDjed1YPHVZXY4sN
? Aergo address matching private key  AmPxVdu993eosN3UjnPDdN3wb7TNbHeiHDvn2dvZUcH8KXDK3RLU
? Path to save new config file  config.json
```

1.5.3 Deploy the bridge contracts

The sender of the deployment tx will be the bridge owner. Ownership is then transfered to the multisig oracle.

```
$ python3 -m ethaergo_bridge_operator.bridge_deployer --help                  ␣
↪                                                                              ␣
↪                                                              18h17m
    usage: bridge_deployer.py [-h] -c CONFIG_FILE_PATH -a AERGO -e ETH
                            [--privkey_name PRIVKEY_NAME] [--local_test]

    Deploy bridge contracts between Ethereum and Aergo.

    optional arguments:
    -h, --help            show this help message and exit
    -c CONFIG_FILE_PATH, --config_file_path CONFIG_FILE_PATH
                            Path to config.json
    -a AERGO, --aergo AERGO
                            Name of Aergo network in config file
    -e ETH, --eth ETH     Name of Ethereum network in config file
    --privkey_name PRIVKEY_NAME
                            Name of account in config file to sign anchors
    --local_test          Start proposer with password for testing

$ python3 -m ethaergo_bridge_operator.bridge_deployer -c './test_config.json' -a
↪'aergo-local' -e eth-poa-local --privkey_name "proposer"

    DEPLOY MERKLE BRIDGE
    ------ DEPLOY BRIDGE BETWEEN Aergo & Ethereum -----------
    ------ Connect Hera and Web3 providers -----------
    ------ Set Sender Account -----------
    > Sender Address Aergo: AmPxVdu993eosN3UjnPDdN3wb7TNbHeiHDvn2dvZUcH8KXDK3RLU
    > Sender Address Ethereum: 0xc19b69591141443676a3EE56fbf1d3EA869d53D8
    ------ Deploy Aergo SC -----------
        > result[CrbmTHjGLEdgtKF7zUicgPwTZMYuueiX76mu87w7U2YE] : TX_OK
    ------ Deploy Ethereum SC -----------
    > Bridge Address Ethereum: 0xa146911F6779301D131139353960216D179693D6
    > Bridge Address Aergo: AmhJjVxa7Yp8CiXpTXVhoDXiDa66SD6rsejbimPFNxzvPNbLzEg5
    ------ Store bridge addresses in test_config.json  -----------
```

1.5.4 Transfer control of the bridge to the multisig oracle

The oracle_deployer script will deploy the oracle contract (with validators previously registered in config.json), and transfer ownership to the newly deployed contract.

```
$ python3 -m ethaergo_bridge_operator.oracle_deployer --help                  ␣
↪                                                                              ␣
↪                                                              18h17m

    DEPLOY ORACLE
    usage: oracle_deployer.py [-h] -c CONFIG_FILE_PATH -a AERGO -e ETH
                            [--privkey_name PRIVKEY_NAME] [--local_test]

    Deploy oracle contracts to controle the bridge between Ethereum and Aergo.

    optional arguments:
    -h, --help            show this help message and exit
    -c CONFIG_FILE_PATH, --config_file_path CONFIG_FILE_PATH
                            Path to config.json
    -a AERGO, --aergo AERGO
                            Name of Aergo network in config file
```

(continues on next page)

(continued from previous page)

```
    -e ETH, --eth ETH     Name of Ethereum network in config file
    --privkey_name PRIVKEY_NAME
                          Name of account in config file to sign anchors
    --local_test          Start proposer with password for testing

$ python3 -m ethaergo_bridge_operator.oracle_deployer -c './test_config.json' -a
↪'aergo-local' -e eth-poa-local --privkey_name "proposer"

    DEPLOY ORACLE
    aergo validators :  ['AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ',
↪'AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ',
↪'AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ']
    ethereum validators :  ['0x210467b3849a408c3a3bEE14b4627aa57F342134',
↪'0x210467b3849a408c3a3bEE14b4627aa57F342134',
↪'0x210467b3849a408c3a3bEE14b4627aa57F342134']
    ------ DEPLOY BRIDGE BETWEEN Aergo & Ethereum -----------
    ------ Connect AERGO -----------
    ------ Connect Web3 -----------
    ------ Set Sender Account -----------
    > Sender Address Aergo: AmPxVdu993eosN3UjnPDdN3wb7TNbHeiHDvn2dvZUcH8KXDK3RLU
    > Sender Address Ethereum: 0xc19b69591141443676a3EE56fbf1d3EA869d53D8
    ------ Deploy Aergo SC -----------
        > result[7bATQt58yd64cYY7h8YUSvQoU6NLFB6SXDUnRD1x39Mx] : TX_OK
    ------ Deploy Ethereum SC -----------
    > Oracle Address Ethereum: 0xF05692cE866f21b5E108781055AdEDde00E50872
    > Oracle Address Aergo: AmgwgSFDwtdxzdfa4kUxuYXMWkHN1MLZMVANBcm85rpsDSaAymFU
    ------ Store bridge addresses in test_config.json  -----------
    ------ Transfer bridge control to oracles -----------
```

1.5.5 Send native aer to the bridge contract

After deployment, the aer on the Aergo network should be sent (frozen) to the bridge contract so that it can be unfrozen when users send their erc20 from the ethereum network.

1.6 Configuration file

The config file is used by the bridge operators, the wallet and the cli to store information about node connections, validator connections, bridge parameters, assets and private keys.

It can be created and updated manually of with the help of the cli.

```
{
    "networks": { // list of registered networks
        "aergo-local": { // name of the network
            "bridges": { // list of bridges between 'aergo-local' and other␣
↪blockchains
                "eth-poa-local": { // name of bridged network
                    "addr": "AmhXrQ7KdNA4naBi2sTwHj13aBzVBohRhxy262nXsPbV2YbULXUR", //
↪ address of bridge contract
                    "oracle": "AmgQdbUqDuoX5krsmvSEHc9X3apBuXyJTQ4mimfWzejEsYScTo3f",␣
↪// address of oracle controlling 'addr' bridge contract
                    "t_anchor": 6, // anchoring periode in bridge contract
                    "t_final": 4 // finality of chain anchored on bridge contract
```

(continues on next page)

(continued from previous page)

```
                }
            },
            "ip": "localhost:7845", // node connection ip
            "providers": [
                "localhost:7845",
                "localhost:7845"
            ], // redundant providers for validators to query different data sources
            "tokens": { // list of tokens originating from 'aergo-local'
                "aergo": { // aer native asset
                    "addr": "aergo", // 'aergo' is the reserved name and address
                    "pegs": {}
                },
                "token1": { // asset name
                    "addr": "AmghHtk2gpcpMa6bj1v59qCBfNmKZTi8qDGeuMNg5meJuXGTa2Y1", //
↪ asset addresss
                    "pegs": { // list of networks where this asset has a peg
                        "eth-poa-local": "0xB7633077842e3fb1877e43C0cCa0972dB8bb6Fb0"␣
↪// address of pegged asset
                    }
                }
            },
            "type": "aergo" // type of network to differenciate between Aergo and␣
↪Ethereum
        },
        "eth-poa-local": {
            "bridges": {
                "aergo-local": {
                    "addr": "0xbC5385259C2Dfdd99996CFb9B6C2f92767FcB32b",
                    "bridge_abi": "contracts/solidity/bridge_abi.txt", // path to␣
↪bridge abi
                    "minted_abi": "contracts/solidity/minted_erc20_abi.txt", // path␣
↪to minted token abi
                    "oracle": "0x5b9fd5f3e14F0F886AD11aCc24Ff53823Bf9bdb5",
                    "oracle_abi": "contracts/solidity/oracle_abi.txt", // path to␣
↪oracle abi
                    "t_anchor": 7,
                    "t_final": 5
                }
            },
            "ip": "localhost:8545",
            "providers": [
                "http://localhost:8545",
                "http://localhost:8545"
            ], // redundant providers for validators to query different data sources
            "isPOA": true, // web3py needs middleware to connect to POA chains
            "tokens": {
                "aergo_erc20": { // reserved name of aergo erc20 issued at ico
                    "abi": "contracts/solidity/aergo_erc20_abi.txt",
                    "addr": "0xd898383A12CDE0eDF7642F7dD4D7006FdE5c433e",
                    "pegs": {
                        "aergo-local": "aergo"
                    }
                },
                "test_erc20": {
                    "abi": "contracts/solidity/aergo_erc20_abi.txt",
                    "addr": "0xeeEF65f288b39d1514A54852566415b973927142",
                    "pegs": {
```

(continues on next page)

(continued from previous page)

```
                        "aergo-local":
↪"AmhiUx2hZ9phVDMZoBShEWD2sCFXPJ5BZpagNC8WfssPuZg7wzZS"
                    }
                }
            },
            "type": "ethereum"
        }
    },
    "validators": [ // list of validators, only needed for bridge operator
        {
            "addr": "AmNLjcxUDmxeGZL7F8bqyaGt3zqog5HAoJmFBEZAx1RvfTKLSBsQ", //␣
↪validator address in Aergo bridge contract
            "eth-addr": "0x210467b3849a408c3a3bEE14b4627aa57F342134", // validator␣
↪address in Ethereum bridge contract
            "ip": "localhost:9841" // ip of validator API
        },
        {
            "addr": "AmNyNPEqeXPfdHeECMNhsH1QcnZsqCtDAudjgFyG5qpasN6tyLPE",
            "eth-addr": "0x7acb4a265bf759ec772510c2465fb7c8f4eaf54e",
            "ip": "localhost:9842"
        },
        {
            "addr": "AmPf349iHWd6kQGU45BxFzFCzEDu75Y3FqFPd4WBMteFq4mtDuZd",
            "eth-addr": "0xf1bf3497d98ead7f6a1bb9ee6dfbde9d448d7062",
            "ip": "localhost:9843"
        }
    ],
    "wallet": { // list of Aergo wallets
        "default": { // name of wallet
            "addr": "AmNPWDJMjU4g98Scm4AikW8JwQMGwWMztM7Qy8ggxNTkhgZMJHFp", //␣
↪address matching private key
            "keystore": "keystore/
↪AmNPWDJMjU4g98Scm4AikW8JwQMGwWMztM7Qy8ggxNTkhgZMJHFp__2020-01-20T04:13:06__keystore.
↪json" // path to json keystore
        }
    },
    "wallet-eth": { // list of Ethereum wallets
        "default": { // name of wallet
            "addr": "0xfec3c905bcd3d9a5471452e53f82106844cb1e76", // address matching␣
↪private key
            "keystore": "keystore/UTC--2019-05-13T09-23-35.377701000Z--
↪fec3c905bcd3d9a5471452e53f82106844cb1e76" // path to json keystore
        }
    }
}
```

1.7 Unfreeze service

The unfreeze service provides the service to users that want to transfer Aergo erc20 to Aergo Mainnet (Aergo Native) but don't already own Aergo Native to pay for the unfreeze transaction fee.

The bridge contract on Aergo check if the tx sender is the same as the Aergo Native receiver and if they are different, if will transfer _unfreezeFee to the tx sender and send the rest to the receiver.

The RequestUnfreeze service will check the receiver address is valid and that the amount to unfreeze is higher than

the _unfreezeFee.

In initial tests, the minimum unfreeze fee is about 7359000000000000 aer (0.007359 aergo) which we can expect to increase with larger Merkle proofs.

1.7.1 Starting the unfreeze grpc service

```
$ python3 -m unfreeze_service.server --help
    usage: server.py [-h] -ip IP_PORT -c CONFIG_FILE_PATH -a AERGO -e ETH
                --privkey_name PRIVKEY_NAME [--local_test]

    Aergo native unfreeze service

    optional arguments:
    -h, --help            show this help message and exit
    -ip IP_PORT, --ip_port IP_PORT
                            ip and port to run unfreeze service
    -c CONFIG_FILE_PATH, --config_file_path CONFIG_FILE_PATH
                            Path to config.json
    -a AERGO, --aergo AERGO
                            Name of Aergo network in config file
    -e ETH, --eth ETH     Name of Ethereum network in config file
    --privkey_name PRIVKEY_NAME
                            Name of account in config file to sign anchors
    --local_test          Start service for running tests


$ python3 -m unfreeze_service.server -ip 'localhost:7891' -c './test_config.json' -a
↪'aergo-local' -e 'eth-poa-local' --privkey_name "broadcaster"
    "Ethereum bridge contract: 0x89eD1D1C145F6bF3A7e62d2B8eB0e1Bf15Cb2374"
    "Aergo bridge contract: AmgQqVWX3JADRBEVkVCM4CyWdoeXuumeYGGJJxEeoAukRC26hxmw"
    "Aergo ERC20: 0xd898383A12CDE0eDF7642F7dD4D7006FdE5c433e"
    Decrypt exported private key 'broadcaster'
    Password:
    "Unfreezer Address: AmPiFGxLvETrs13QYrHUiYoFqAqqWv7TKYXG21zC8TJfJTDHc7HJ"
    "Unfreeze fee for broadcaster: 1000aer"
    "Unfreeze server started"
```

1.7.2 Starting the Envoy proxy

```
$ docker run --rm --name=proxy -p 8081:8080 -v $(pwd)/unfreeze_service//envoy/envoy.
↪yaml:/etc/envoy/envoy.yaml envoyproxy/envoy:latest
```

1.8 bridge_operator

class ethaergo_bridge_operator.proposer.client.**ProposerClient**(*config_file_path: str*, *aergo_net: str*, *eth_net: str*, *eth_block_time: int*, *aergo_gas_price: int*, *eth_gas_price: int*, *privkey_name: str = None*, *privkey_pwd: str = None*, *anchoring_on: bool = False*, *auto_update: bool = False*, *oracle_update: bool = False*, *bridge_anchoring: bool = True*, *root_path: str = './'*, *eco: bool = False*, *eth_eco: bool = False*)

The ProposerClient starts Aergo and Ethereum proposers

class ethaergo_bridge_operator.proposer.eth.client.**EthProposerClient**(*config_file_path: str*, *aergo_net: str*, *eth_net: str*, *privkey_name: str = None*, *privkey_pwd: str = None*, *anchoring_on: bool = False*, *auto_update: bool = False*, *oracle_update: bool = False*, *root_path: str = './'*, *eth_gas_price: int = None*, *bridge_anchoring: bool = True*, *eco: bool = False*)

The ethereum bridge proposer periodically (every t_anchor) broadcasts the finalized Aergo trie state root (after lib) onto the ethereum bridge contract after validation by the Validators. It first checks the last merged height and waits until now > lib + t_anchor is reached, then merges the current finalised block (lib). If bridge_anchoring is True(default), then the proposer will create a Merkle proof of the bridge storage root to anchor both roots in the same transaction. Start again after waiting t_anchor. EthProposerClient anchors an Aergo state root onto Ethereum.

Note on config_data:

- config_data is used to store current validators and their ip when the proposer starts. (change validators after the proposer has started)
- After starting, when users change the config.json, the proposer will attempt to gather signatures to reflect the changes.
- t_anchor value is always taken from the bridge contract
- validators are taken from the config_data because ip information is not stored on chain
- when a validator set update succeeds, self.config_data is updated
- if another proposer updates to a new set of validators and the proposer doesnt know about it, proposer

must be restarted with the new current validator set to create new connections to them.

buildBridgeAnchorArgs(*root: bytes*) → Tuple[bytes, List[bytes], bytes, int]
Build arguments to derive bridge storage root from the anchored state root with a merkle proof

monitor_settings()
Check if a modification of bridge settings is requested by seeing if the config file has been changed and try to update the bridge contract (gather 2/3 validators signatures).

monitor_settings_and_sleep(*sleeping_time*)
While sleeping, periodicaly check changes to the config file and update settings if necessary. If another proposer updated settings it doesnt matter, validators will just not give signatures.

run() → None
Gathers signatures from validators, verifies them, and if 2/3 majority is acquired, set the new anchored root in eth_bridge.

update_oracle(*oracle*)
Try to update the oracle registered in the bridge contract.

update_t_anchor(*t_anchor*)
Try to update the anchoring periode registered in the bridge contract.

update_t_final(*t_final*)
Try to update the anchoring periode registered in the bridge contract.

update_validators(*new_validators*)
Try to update the validator set with the one in the config file.

wait_next_anchor(*merged_height: int*) → int
Wait until t_anchor has passed after merged height. Return the next finalized block after t_anchor to be the next anchor

class ethaergo_bridge_operator.proposer.aergo.client.**AergoProposerClient**(*config_file_path: str*, *aergo_net: str*, *eth_net: str*, *eth_block_time: int*, *privkey_name: str = None*, *privkey_pwd: str = None*, *anchoring_on: bool = False*, *auto_update: bool = False*, *oracle_update: bool = False*, *aergo_gas_price: int = None*, *bridge_anchoring: bool = True*, *root_path: str = './'*, *eco: bool = False*)

The aergo bridge proposer periodically (every t_anchor) broadcasts the finalized trie state root (after lib) onto the aergo bridge contract after validation by the Validator servers. It first checks the last merged height and waits until now > lib + t_anchor is reached, then merges the current finalised block (lib). If bridge_anchoring is True(default), then the proposer will create a Merkle proof of the bridge storage root to anchor both roots in the same transaction. Start again after waiting t_anchor.

Note on config_data:

- config_data is used to store current validators and their ip when the proposer starts. (change validators after the proposer has started)
- After starting, when users change the config.json, the proposer will attempt to gather signatures to reflect the changes.
- t_anchor value is always taken from the bridge contract
- validators are taken from the config_data because ip information is not stored on chain
- when a validator set update succeeds, self.config_data is updated
- if another proposer updates to a new set of validators and the proposer doesnt know about it, proposer must be restarted with the new current validator set to create new connections to them.

buildBridgeAnchorArgs(*next_anchor_height*) → Tuple[List[str], List[str]]
Build arguments to derive bridge storage root from the anchored state root with a merkle proof

monitor_settings()
Check if a modification of bridge settings is requested by seeing if the config file has been changed and try to update the bridge contract (gather 2/3 validators signatures).

monitor_settings_and_sleep(*sleeping_time*)
While sleeping, periodicaly check changes to the config file and update settings if necessary. If another proposer updated settings it doesnt matter, validators will just not give signatures.

run() → None
Gathers signatures from validators, verifies them, and if 2/3 majority is acquired, set the new anchored root in aergo_bridge.

update_oracle(*oracle*)
Try to update the oracle periode registered in the bridge contract.

update_t_anchor(*t_anchor*)
Try to update the anchoring periode registered in the bridge contract.

update_t_final(*t_final*)
Try to update the anchoring periode registered in the bridge contract.

update_unfreeze_fee(*fee*)
Try to update the anchoring periode registered in the bridge contract.

update_validators(*new_validators*)
Try to update the validator set with the one in the config file.

wait_next_anchor(*merged_height: int*) → int
Wait until t_anchor has passed after merged height. Return the next finalized block after t_anchor to be the next anchor

ethaergo_bridge_operator.bridge_deployer.**deploy_bridge**(*config_path: str*, *lua_bytecode_path: str*, *sol_bytecode_path: str*, *eth_net: str*, *aergo_net: str*, *aergo_erc20: str = 'aergo_erc20'*, *privkey_name: str = None*, *privkey_pwd: str = None*) → None
Deploy brige contract on Aergo and Ethereum.

exception ethaergo_bridge_operator.proposer.exceptions.**ValidatorMajorityError**
Exception raised by proposers when they fail to gather 2/3 validator signatures to make an update.

1.9 ethaergo_wallet

class ethaergo_wallet.wallet.**EthAergoWallet**(*config_file_path: str*, *config_data: Dict[KT, VT] = None*, *root_path: str = './'*, *eth_gas_price: int = 10*, *aergo_gas_price: int = 0*)

EthAergoWallet transfers tokens on the Eth<->Aergo Bridge

aergo_to_eth_sidechain(*from_chain: str*, *to_chain: str*, *asset_name: str*, *amount: int*, *aergo_receiver: str*, *aergo_privkey_name: str = 'default'*, *aergo_privkey_pwd: str = None*, *eth_privkey_name: str = 'default'*, *eth_privkey_pwd: str = None*) → None

Transfer a native Aergo Standard Token or Aer to Ethereum

burn_to_aergo(*from_chain: str*, *to_chain: str*, *asset_name: str*, *amount: int*, *receiver: str*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → Tuple[int, str]

Initiate minted Standard token transfer back to aergo origin

burn_to_eth(*from_chain: str*, *to_chain: str*, *asset_name: str*, *amount: int*, *receiver: str*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → Tuple[int, str]

Initiate minted token transfer back to ethereum origin

eth_to_aergo_sidechain(*from_chain: str*, *to_chain: str*, *asset_name: str*, *amount: int*, *aergo_receiver: str*, *aergo_privkey_name: str = 'default'*, *aergo_privkey_pwd: str = None*, *eth_privkey_name: str = 'default'*, *eth_privkey_pwd: str = None*) → None

Transfer a native ERC20 or ether to Aergo

freeze(*from_chain: str*, *to_chain: str*, *amount: int*, *receiver: str*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → Tuple[int, str]

Initiate Aer transfer back to Ethereum AergoERC20 sidechain

get_aergo(*network_name: str*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*, *skip_state: bool = False*) → aergo.herapy.aergo.Aergo

Return aergo provider with account loaded from keystore

get_balance_aergo(*asset_name: str*, *network_name: str*, *asset_origin_chain: str = None*, *account_name: str = 'default'*, *account_addr: str = None*) → Tuple[int, str]

Get account name balance of asset_name on network_name, and specify asset_origin_chain for a pegged asset query,

get_balance_eth(*asset_name: str*, *network_name: str*, *asset_origin_chain: str = None*, *account_name: str = 'default'*, *account_addr: str = None*) → Tuple[int, str]

Get account name balance of asset_name on network_name, and specify asset_origin_chain for a pegged asset query,

get_signer(*w3: web3.main.Web3*, *privkey_name: str*, *privkey_pwd: str = None*)

Get the web3 signer object from the ethereum private key.

load_bridge_abi(*from_chain: str*, *to_chain: str*) → str

Load Ethereum bridge contract abi from file location in config.

load_erc20_abi(*origin_chain: str*, *asset_name: str*) → str

Load erc20 contract abi from file location in config.

load_keystore(*privkey_name: str*) → str

Load encrypted private key from Ethereum json keystore.

load_minted_erc20_abi(*from_chain: str*, *to_chain: str*) → str

Load Ethereum bridge contract minted token abi from file location in config.

lock_to_aergo(*from_chain: str*, *to_chain: str*, *asset_name: str*, *amount: int*, *receiver: str*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → Tuple[int, str]
Initiate ERC20 token or Ether transfer to Aergo sidechain

lock_to_eth(*from_chain: str*, *to_chain: str*, *asset_name: str*, *amount: int*, *receiver: str*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → Tuple[int, str]
Initiate Aergo Standard Token transfer to Ethereum sidechain

mint_to_aergo(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str = None*, *lock_height: int = 0*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → str
Finalize ERC20 token or Ether transfer to Aergo sidechain

mint_to_eth(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str = None*, *lock_height: int = 0*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → Tuple[str, str]
Finalize Aergo Standard Token transfer to Ethereum sidechain NOTE anybody can mint so sender is not necessary. The amount to mint is the difference between total deposit and already minted amount. Bridge tempo is taken from config_data

mintable_to_aergo(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str*) → Tuple[int, int]
Check mintable balance on Aergo.

mintable_to_eth(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str*) → Tuple[int, int]
Check mintable balance on Ethereum.

unfreezable(*from_chain: str*, *to_chain: str*, *receiver: str*) → Tuple[int, int]
Check unfreezable balance on Aergo.

unfreeze(*from_chain: str*, *to_chain: str*, *receiver: str = None*, *lock_height: int = 0*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → str
Finalize ERC20Aergo transfer to Aergo Mainnet by unfreezing (aers are already minted and freezed in the bridge contract)

unlock_to_aergo(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str*, *burn_height: int = 0*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → str
Finalize Aergo Standard token transfer back to Aergo Origin

unlock_to_eth(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str = None*, *burn_height: int = 0*, *privkey_name: str = 'default'*, *privkey_pwd: str = None*) → str
Finalize ERC20 or Eth transfer back to Ethereum origin

unlockable_to_aergo(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str*) → Tuple[int, int]
Check unlockable balance on Aergo.

unlockable_to_eth(*from_chain: str*, *to_chain: str*, *asset_name: str*, *receiver: str*) → Tuple[int, int]
Check unlockable balance on Ethereum.

ethaergo_wallet.aergo_to_eth.**build_burn_proof**(*aergo_from: aergo.herapy.aergo.Aergo*, *w3: web3.main.Web3*, *receiver: str*, *bridge_from: str*, *bridge_to: str*, *bridge_to_abi: str*, *burn_height: int*, *token_origin: str*)
Check the last anchored root includes the burn and build a burn proof for that root

ethaergo_wallet.aergo_to_eth.**build_lock_proof**(*aergo_from: aergo.herapy.aergo.Aergo*, *w3: web3.main.Web3*, *receiver: str*, *bridge_from: str*, *bridge_to: str*, *bridge_to_abi: str*, *lock_height: int*, *token_origin: str*)
Check the last anchored root includes the lock and build a lock proof for that root

ethaergo_wallet.aergo_to_eth.**burn**(*aergo_from: aergo.herapy.aergo.Aergo*, *bridge_from: str*, *receiver: str*, *value: int*, *token_pegged: str*, *gas_limit: int*, *gas_price: int*) → Tuple[int, str, aergo.herapy.obj.transaction.Transaction]

Burn a minted token on a sidechain.

ethaergo_wallet.aergo_to_eth.**freeze**(*aergo_from: aergo.herapy.aergo.Aergo*, *bridge_from: str*, *receiver: str*, *value: int*, *gas_limit: int*, *gas_price: int*) → Tuple[int, str, aergo.herapy.obj.transaction.Transaction]

Freeze aergo native

ethaergo_wallet.aergo_to_eth.**lock**(*aergo_from: aergo.herapy.aergo.Aergo*, *bridge_from: str*, *receiver: str*, *value: int*, *asset: str*, *gas_limit: int*, *gas_price: int*) → Tuple[int, str, aergo.herapy.obj.transaction.Transaction]

Lock can be called to lock aer or tokens. it supports delegated transfers when tx broadcaster is not the same as the token owner

ethaergo_wallet.aergo_to_eth.**mint**(*w3: web3.main.Web3*, *signer_acct*, *receiver: str*, *lock_proof: aergo.herapy.obj.sc_state.SCState*, *token_origin: str*, *bridge_to: str*, *bridge_to_abi: str*, *gas_limit: int*, *gas_price: int*) → Tuple[str, str, web3.datastructures.AttributeDict]

Mint the receiver's deposit balance on aergo_to.

ethaergo_wallet.aergo_to_eth.**unlock**(*w3: web3.main.Web3*, *signer_acct*, *receiver: str*, *burn_proof: aergo.herapy.obj.sc_state.SCState*, *token_origin: str*, *bridge_to: str*, *bridge_to_abi: str*, *gas_limit: int*, *gas_price: int*) → Tuple[str, web3.datastructures.AttributeDict]

Unlock the receiver's burnt balance on aergo_to.

ethaergo_wallet.eth_to_aergo.**build_burn_proof**(*w3: web3.main.Web3*, *aergo_to: aergo.herapy.aergo.Aergo*, *receiver: str*, *bridge_from: str*, *bridge_to: str*, *burn_height: int*, *token_origin: str*)

Check the last anchored root includes the lock and build a lock proof for that root

ethaergo_wallet.eth_to_aergo.**build_lock_proof**(*w3: web3.main.Web3*, *aergo_to: aergo.herapy.aergo.Aergo*, *receiver: str*, *bridge_from: str*, *bridge_to: str*, *lock_height: int*, *token_origin: str*)

Check the last anchored root includes the lock and build a lock proof for that root

ethaergo_wallet.eth_to_aergo.**burn**(*w3: web3.main.Web3*, *signer_acct*, *receiver: str*, *amount: int*, *bridge_from: str*, *bridge_from_abi: str*, *token_pegged: str*, *gas_limit: int*, *gas_price: int*) → Tuple[int, str, web3.datastructures.AttributeDict]

Burn a token that was minted on ethereum.

ethaergo_wallet.eth_to_aergo.**lock**(*w3: web3.main.Web3*, *signer_acct*, *receiver: str*, *amount: int*, *bridge_from: str*, *bridge_from_abi: str*, *erc20_address: str*, *gas_limit: int*, *gas_price: int*, *next_nonce: int = None*) → Tuple[int, str, web3.datastructures.AttributeDict]

Lock an Ethereum ERC20 token.

ethaergo_wallet.eth_to_aergo.**mint**(*aergo_to: aergo.herapy.aergo.Aergo*, *receiver: str*, *lock_proof: web3.datastructures.AttributeDict*, *token_origin: str*, *bridge_to: str*, *gas_limit: int*, *gas_price: int*) → Tuple[str, str, aergo.herapy.obj.transaction.Transaction]

Unlock the receiver's deposit balance on aergo_to.

ethaergo_wallet.eth_to_aergo.**unfreeze**(*aergo_to: aergo.herapy.aergo.Aergo*, *receiver: str*, *lock_proof: web3.datastructures.AttributeDict*, *bridge_to: str*, *gas_limit: int*, *gas_price: int*) → Tuple[str, aergo.herapy.obj.transaction.Transaction]

Unlock the receiver's deposit balance on aergo_to.

ethaergo_wallet.eth_to_aergo.**unlock**(*aergo_to: aergo.herapy.aergo.Aergo*, *receiver: str*, *burn_proof: web3.datastructures.AttributeDict*, *token_origin: str*, *bridge_to: str*, *gas_limit: int*, *gas_price: int*) → Tuple[str, aergo.herapy.obj.transaction.Transaction]

Unlock the receiver's deposit balance on aergo_to.

ethaergo_wallet.eth_utils.contract_deployer.**deploy_contract**(*bytecode: str*, *abi: str*, *w3: web3.main.Web3*, *gas_limit: int*, *gas_price: int*, *privkey: bytes*, **args*)

Deploy a new contract to ethereum.

ethaergo_wallet.eth_utils.erc20.**increase_approval**(*spender: str*, *asset_addr: str*, *amount: int*, *w3: web3.main.Web3*, *erc20_abi: str*, *signer_acct*, *gas_limit: int*, *gas_price: int*) → Tuple[int, str]

Increase approval increases the amount of tokens that spender can withdraw. For older tokens without the increaseApproval function in the abi, approval should be set to 0 and then to amount. Newer tokens with increaseAllowance should also be suported

1.10 ethaergo_cli

class ethaergo_cli.main.**EthMerkleBridgeCli**(*root_path: str = './'*)

CLI tool for interacting with the EthAergoWallet.

First choose an existing config file or create one from scratch. Once a config file is chosen, the CLI provides an interface to the EthAergoWallet and has the following features: - edit config file settings - transfer assets between networks - check status of transfers - check balances for each asset on each network

check_balances()

Iterate every registered wallet, network and asset and query balances.

check_withdrawable_balance()

Check the status of cross chain transfers.

create_config()

Create a new configuration file from scratch.

This tool registers 2 networks, bridge contracts, paths to bridge abis, a private key for each network and bridge validators

edit_settings()
Menu for editing the config file of the currently loaded wallet

finalize_transfer()
Finalize a token transfer between 2 chains.

finalize_transfer_arguments(*prompt_last_deposit=True*)
Prompt the arguments needed to finalize a transfer.

The arguments can be taken from the pending transfers or inputed manually by users.

Returns: List of transfer arguments

get_registered_assets(*from_chain*, *to_chain*)
Get the list of registered assets on each network.

get_registered_networks()
Get the list of networks registered in the wallet config.

initiate_transfer()
Initiate a new transfer of tokens between 2 networks.

load_config()
Load the configuration file from path and create a wallet object.

menu()
Menu for interacting with network.

Users can change settings, query balances, check pending transfers, execute cross chain transactions

prompt_bridge_networks()
Prompt user to choose 2 networks between registered networks.

prompt_commun_transfer_params()
Prompt the common parameters necessary for all transfers.

Returns: List of transfer parameters : from_chain, to_chain, from_assets, to_assets, asset_name, receiver

prompt_signing_key(*wallet_name*)
Prompt user to select a private key.

Note: Keys are displayed by name and should have been registered in wallet config.

prompt_transfer_networks()
Prompt user to choose 2 networks between registered bridged networks.

register_asset()
Register a new asset and it's pegs on other networks in the wallet's config.

register_bridge()
Register bridge contracts between 2 already defined networks.

register_key()
Register new key in wallet's config.

register_network()
Register a new network in the wallet's config.

register_new_validators()
Register new validators in the wallet's config.

start()
Entry point of cli : load a wallet configuration file of create a new one

store_pending_transfers()
Record pending transfers in json file so they can be finalized later.

ethaergo_cli.utils.**format_amount**(*num: str*)
Format a float string to an integer with 18 decimals.

Example: '2.3' -> 2300000000000000000

ethaergo_cli.utils.**promptYN**(*q*, *y*, *n*)
Prompt user to procede with a transfer of not.

ethaergo_cli.utils.**prompt_aergo_keystore**()
Prompt user to input a new aergo private key.

Returns:

- name of the key
- address of the key
- path to keystore

ethaergo_cli.utils.**prompt_amount**()
Prompt a number of tokens to transfer.

ethaergo_cli.utils.**prompt_bridge_abi_paths**()
Prompt user to input paths to text files containing abis.

ethaergo_cli.utils.**prompt_deposit_height**()
Prompt the block number of deposit.

ethaergo_cli.utils.**prompt_eth_keystore**()
Prompt use to input a new ethereum private key.

Returns:

- name of the key
- address of the key
- path to the json key file

ethaergo_cli.utils.**prompt_file_path**(*message*)
Prompt user to input a path to a file and check it exists.

ethaergo_cli.utils.**prompt_gas_price**()
Prompt aergo and eth gas price

ethaergo_cli.utils.**prompt_new_asset**(*networks*)
Prompt user to input a new asset by providing the following: - asset name - origin network (where it was first issued) - address on origin network - other networks where the asset exists as a peg - address of pegs

ethaergo_cli.utils.**prompt_new_bridge**(*net1*, *net2*)
Prompt user to input bridge contracts and tempo.

For each contract on each bridged network, provide: - bridge contract address - anchoring periode - finality of the anchored chain

ethaergo_cli.utils.**prompt_new_network**()
Prompt user to input a new network's information: - Name - IP/url - Network type (aergo/eth) - is POA (only needed for ethereum)

ethaergo_cli.utils.**prompt_new_validators**()
Prompt user to input validators

Note: The list of validators must have the same order as defined in the bridge contracts

Returns: List of ordered validators

ethaergo_cli.utils.**prompt_number**(*message*, *formator=<class 'int'>*)
Prompt a number.

ethaergo_cli.utils.**prompt_providers**()
Prompt user to input provider ip. If not registered, the validator will use the single 'ip' field in config.json

1.11 unfreeze_service

class unfreeze_service.server.**UnfreezeService**(*ip_port: str*, *config_file_path: str*, *aergo_net: str*, *eth_net: str*, *privkey_name: str*, *privkey_pwd: str = None*, *root_path: str = './'*)
Unfreezes freezed native aergo for users

RequestUnfreeze(*account_ref*, *context*)
Create and broadcast unfreeze transactions if conditions are met: - the receiver is a valid aergo address - the unfreezable amount covers the unfreeze fee

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

ethaergo_bridge_operator.bridge_deployer, 27
ethaergo_bridge_operator.proposer.aergo.client, 25
ethaergo_bridge_operator.proposer.client, 23
ethaergo_bridge_operator.proposer.eth.client, 23
ethaergo_bridge_operator.proposer.exceptions, 27
ethaergo_bridge_operator.validator.server, 23
ethaergo_cli.main, 31
ethaergo_cli.utils, 33
ethaergo_wallet.aergo_to_eth, 29
ethaergo_wallet.eth_to_aergo, 30
ethaergo_wallet.eth_utils.contract_deployer, 31
ethaergo_wallet.eth_utils.erc20, 31
ethaergo_wallet.eth_utils.merkle_proof, 31
ethaergo_wallet.wallet, 28
ethaergo_wallet.wallet_config, 28

u

unfreeze_service.server, 34

Index

A

aergo_to_eth_sidechain() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
AergoProposerClient (*class in ethaergo_bridge_operator.proposer.aergo.client*), 25

B

build_burn_proof() (*in module ethaergo_wallet.aergo_to_eth*), 29
build_burn_proof() (*in module ethaergo_wallet.eth_to_aergo*), 30
build_lock_proof() (*in module ethaergo_wallet.aergo_to_eth*), 29
build_lock_proof() (*in module ethaergo_wallet.eth_to_aergo*), 30
buildBridgeAnchorArgs() (*ethaergo_bridge_operator.proposer.aergo.client.AergoProposerClient method*), 27
buildBridgeAnchorArgs() (*ethaergo_bridge_operator.proposer.eth.client.EthProposerClient method*), 25
burn() (*in module ethaergo_wallet.aergo_to_eth*), 29
burn() (*in module ethaergo_wallet.eth_to_aergo*), 30
burn_to_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
burn_to_eth() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28

C

check_balances() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 31
check_withdrawable_balance() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 31
create_config() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 31

D

deploy_bridge() (*in module ethaergo_bridge_operator.bridge_deployer*), 27
deploy_contract() (*in module ethaergo_wallet.eth_utils.contract_deployer*), 31

E

edit_settings() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
eth_to_aergo_sidechain() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
ethaergo_bridge_operator.bridge_deployer (*module*), 27
ethaergo_bridge_operator.proposer.aergo.client (*module*), 25
ethaergo_bridge_operator.proposer.client (*module*), 23
ethaergo_bridge_operator.proposer.eth.client (*module*), 23
ethaergo_bridge_operator.proposer.exceptions (*module*), 27
ethaergo_bridge_operator.validator.server (*module*), 23
ethaergo_cli.main (*module*), 31
ethaergo_cli.utils (*module*), 33
ethaergo_wallet.aergo_to_eth (*module*), 29
ethaergo_wallet.eth_to_aergo (*module*), 30
ethaergo_wallet.eth_utils.contract_deployer (*module*), 31
ethaergo_wallet.eth_utils.erc20 (*module*), 31
ethaergo_wallet.eth_utils.merkle_proof (*module*), 31
ethaergo_wallet.wallet (*module*), 28
ethaergo_wallet.wallet_config (*module*), 28
EthAergoWallet (*class in ethaergo_wallet.wallet*),

28
EthMerkleBridgeCli (*class in ethaergo_cli.main*), 31
EthProposerClient (*class in ethaergo_bridge_operator.proposer.eth.client*), 23

F

finalize_transfer() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
finalize_transfer_arguments() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
format_amount() (*in module ethaergo_cli.utils*), 33
freeze() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
freeze() (*in module ethaergo_wallet.aergo_to_eth*), 30

G

get_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
get_balance_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
get_balance_eth() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
get_registered_assets() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
get_registered_networks() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
get_signer() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28

I

increase_approval() (*in module ethaergo_wallet.eth_utils.erc20*), 31
initiate_transfer() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32

L

load_bridge_abi() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
load_config() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
load_erc20_abi() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
load_keystore() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
load_minted_erc20_abi() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
lock() (*in module ethaergo_wallet.aergo_to_eth*), 30
lock() (*in module ethaergo_wallet.eth_to_aergo*), 30
lock_to_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 28
lock_to_eth() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29

M

menu() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
mint() (*in module ethaergo_wallet.aergo_to_eth*), 30
mint() (*in module ethaergo_wallet.eth_to_aergo*), 30
mint_to_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
mint_to_eth() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
mintable_to_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
mintable_to_eth() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
monitor_settings() (*ethaergo_bridge_operator.proposer.aergo.client.AergoProposerC method*), 27
monitor_settings() (*ethaergo_bridge_operator.proposer.eth.client.EthProposerClient method*), 25
monitor_settings_and_sleep() (*ethaergo_bridge_operator.proposer.aergo.client.AergoProposerC method*), 27
monitor_settings_and_sleep() (*ethaergo_bridge_operator.proposer.eth.client.EthProposerClient method*), 25

P

prompt_aergo_keystore() (*in module ethaergo_cli.utils*), 33
prompt_amount() (*in module ethaergo_cli.utils*), 33
prompt_bridge_abi_paths() (*in module ethaergo_cli.utils*), 33
prompt_bridge_networks() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
prompt_commun_transfer_params() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
prompt_deposit_height() (*in module ethaergo_cli.utils*), 33

prompt_eth_keystore() (*in module ethaergo_cli.utils*), 33
prompt_file_path() (*in module ethaergo_cli.utils*), 33
prompt_gas_price() (*in module ethaergo_cli.utils*), 33
prompt_new_asset() (*in module ethaergo_cli.utils*), 33
prompt_new_bridge() (*in module ethaergo_cli.utils*), 33
prompt_new_network() (*in module ethaergo_cli.utils*), 33
prompt_new_validators() (*in module ethaergo_cli.utils*), 33
prompt_number() (*in module ethaergo_cli.utils*), 34
prompt_providers() (*in module ethaergo_cli.utils*), 34
prompt_signing_key() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
prompt_transfer_networks() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
promptYN() (*in module ethaergo_cli.utils*), 33
ProposerClient (*class in ethaergo_bridge_operator.proposer.client*), 23

R

register_asset() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
register_bridge() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
register_key() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
register_network() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
register_new_validators() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
RequestUnfreeze() (*unfreeze_service.server.UnfreezeService method*), 34
run() (*ethaergo_bridge_operator.proposer.aergo.client.AergoProposerClient method*), 27
run() (*ethaergo_bridge_operator.proposer.eth.client.EthProposerClient method*), 25

S

start() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32
store_pending_transfers() (*ethaergo_cli.main.EthMerkleBridgeCli method*), 32

U

unfreezable() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
unfreeze() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
unfreeze() (*in module ethaergo_wallet.eth_to_aergo*), 31
unfreeze_service.server (*module*), 34
UnfreezeService (*class in unfreeze_service.server*), 34
unlock() (*in module ethaergo_wallet.aergo_to_eth*), 30
unlock() (*in module ethaergo_wallet.eth_to_aergo*), 31
unlock_to_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
unlock_to_eth() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
unlockable_to_aergo() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
unlockable_to_eth() (*ethaergo_wallet.wallet.EthAergoWallet method*), 29
update_oracle() (*ethaergo_bridge_operator.proposer.aergo.client.Ae method*), 27
update_oracle() (*ethaergo_bridge_operator.proposer.eth.client.EthPr method*), 25
update_t_anchor() (*ethaergo_bridge_operator.proposer.aergo.client.AergoProposer method*), 27
update_t_anchor() (*ethaergo_bridge_operator.proposer.eth.client.EthProposerClient method*), 25
update_t_final() (*ethaergo_bridge_operator.proposer.aergo.client.A method*), 27
update_t_final() (*ethaergo_bridge_operator.proposer.eth.client.EthP method*), 25
update_unfreeze_fee() (*ethaergo_bridge_operator.proposer.aergo.client.AergoProposer method*), 27
update_validators() (*ethaergo_bridge_operator.proposer.aergo.client.AergoProposer method*), 27
update_validators() (*ethaergo_bridge_operator.proposer.eth.client.EthProposerClient method*), 25

V

ValidatorMajorityError, 27

W

wait_next_anchor()
(*ethaergo_bridge_operator.proposer.aergo.client.AergoProposerClient method*), 27
wait_next_anchor()
(*ethaergo_bridge_operator.proposer.eth.client.EthProposerClient method*), 25