Validator Keys

Validator keys are added in several sequential steps. These steps are similar for each time new keys are added.

- Validator Keys
- Generating signing keys
- Withdrawal Credentials
- Using staking-deposit-cli
- Validating the keys
- Submitting the keys
- Using the batch key submitter UI
- Importing the keys to a Lighthouse validator client
- Checking the keys of all Lido Node Operators
- Lido CLI
- Lido Node Operator Dashboard
- Results
- You don't see invalid keys
- You spot invalid keys
- Increasing the Staking Limits with an Easy Track motion

Generating signing keys

Upon inclusion into the protocol, a Node Operator should generate and submit a set of BLS12-381 public keys that will be used by the protocol for making ether deposits to the EthereumDepositContract . Along with the keys, a Node Operator submits a set of the corresponding signaturesas defined in the spec. TheDepositMessage used for generating the signature must be the following:

- pubkey
- must be derived from the private key used for signing the message;
- amount
- must equal to 32 ether;
- withdrawal_credentials
- must equal to the protocol credentials set by the DAO.

Withdrawal Credentials

Make sure to obtain correct withdrawal address by finding it inside the active withdrawal credentials by calling the contract viaStakingRouter.getWithdrawalCredentials() .

For example withdrawal credentials0x010000000000000000000000b9d7934878b5fb9610b3fe8a5e441e8fad7e293f mean that the withdrawal address is0xb9d7934878b5fb9610b3fe8a5e441e8fad7e293f . For Mainnet, always verify the address is correct using anexplorer - you will see that it was deployed from the Lido deployer.

Using staking-deposit-cli

Use the latest release ofstaking-deposit-cli .

Example command usage:

./deposit new-mnemonic --folder . --num_validators 123 --mnemonic_language english --chain mainnet --eth1_withdrawal_address 0x123 Here,chain is one of the available chain names (run the command with the--help flag to see the possible values:./deposit new-mnemonic --help) andeth1_withdrawal_address is the withdrawal address from the protocol documentation.

As a result of running this, thevalidator_keys directory will be created in the current working directory. It will contain a deposit data file nameddeposit-data-*.json and a number of private key stores namedkeystore*-json , the latter encrypted with the password you were asked for when running the command.

If you chose to use the UI for submitting the keys, you'll need to pass the JSON data found in the deposit data file to the protocol (see the next section). If you wish, you can remove any other fields exceptpubkey andsignature from the array items.

Never share the generated mnemonic and your private keys with anyone, including the protocol members and DAO holders.

Validating the keys

Please, make sure to check the keys validity before submitting them on-chain.

Lido submitter has validation functionality built-in, keys will be checked before submitting.

If you will be submitting keys manually via Lido contract, you can use Lido CLI. It's a Python package which you can install with pip:

pip install lido-cli lido-cli --rpc http://1.2.3.4:8545 validate_file_keys --file keys.json You would need an RPC endpoint - a local node / RPC provider (eg Alchemy/Infura).

Submitting the keys

Please note, that the withdrawal address should be added to the Lido Node Operators Registry before it can submit the signing keys. Adding an address to the Node Operators Registry happens via DAO voting. When providing withdrawal address to be added to the Node Operators Registry, keep in mind the following:

- it is the address that will receive rewards;
- it is the address you will be using for submitting keys to Lido;
- you should be able to access it at any time in case of emergency;
- you can use multi-sig for it if you wish to;

- you will not be able to replace it by another address/multi-sig later. After generating the keys, a Node Operator submits them to the protocol. To do this, they send a transaction from the Node Operator's withdrawal address to theNodeOperatorsRegistry contract instance, calling [addSigningKeys function] and with the following arguments:

- uint256 _nodeOperatorId the zero-based sequence number of the operator in the list;

- uint256 _keysCount the number of keys being submitted;
- bytes _publicKeys the concatenated keys;
- bytes _signatures the concatenated signatures. The address of theNodeOperatorsRegistry contract instance can be obtained by calling thegetOperators() function on theLido contract instance. The ABI of theNodeOperatorsRegistry contract can be found on the corresponding contract page on Etherscan or in***-abi.zip of the latest release on thelido-dao releases github page .

Operator id for a given reward address can be obtained by successively callingNodeOperatorsRegistry.getNodeOperator with the increasing_id argument until you get the operator with the matchingrewardAddress .

Etherscan pages for the Görli/Prater contracts:

- Lido
- NodeOperatorsRegistry

Etherscan pages for the Mainnet contracts:

- Lido
- NodeOperatorsRegistry

Using the batch key submitter UI

Lido provides UIs for key submission:Mainnet web interface for submitting the keys and aTestnet web interface for submitting the keys .

If you've used thestaking-deposit-cli , you can paste the content of the generateddeposit-data-*.json file as-is.

Else, prepare a JSON data of the following structure and paste it to the textarea that will appear in the center of the screen:

[{ "pubkey": "PUBLIC_KEY_1", "withdrawal_credentials": "WITHDRAWAL_CREDENTIALS_1", "amount": 32000000000, "signature": "SIGNATURE_1", "fork_version": "FORK_VERSION_1", "eth2_network_name": "ETH2_NETWORK_NAME_1", "deposit_message_root": "DEPOSIT_MESSAGE_ROOT_1", "deposit_data_root": "DEPOSIT_DATA_ROOT_1" }, { "pubkey": "PUBLIC_KEY_2", "withdrawal_credentials": "WITHDRAWAL_CREDENTIALS_2", "amount": 32000000000, "signature": "SIGNATURE_2", "fork_version": "FORK_VERSION_2", "eth2_network_name": "ETH2_NETWORK_NAME_2", "deposit_message_root": "DEPOSIT_MESSAGE_ROOT_2", "deposit_data_root": "DEPOSIT_DATA_ROOT_2" }] This tool will automatically split the keys into chunks and submit the transactions to your wallet for approval. Transactions will come one by one for signing. Unfortunately, we cannot send a large number of keys in a single transaction. Right now, the chunk size is 50 keys, it's close to the limit of gas per block.

Connect your wallet, clickValidate button, the interface would run required checks. And then clickSubmit keys button.

We now support the following connectors:

- MetaMask and similar injected wallets
- Wallet Connect
- Gnosis Safe
- Ledger HQ

If you want to use Gnosis, there are two ways to connect:

- Add this app as acustom app
- in your safe.
- Use WalletConnect
- to connect to your safe.

When you submit a form, the keys are saved in your browser. This tool checks the new key submits against the previously saved list to avoid duplication. Therefore it is important to use one browser for submitting.

Importing the keys to a Lighthouse validator client

If you've used the forkedstaking-deposit-cli to generate the keys, you can import them to a Lighthouse validator client by running this command:

docker run --rm -it \ --name validator_keys_import \ -v "KEYS_DIR":/root/validator_keys \ -v "DATA_DIR":/root/.lighthouse \ sigp/lighthouse \ lighthouse account validator import \ --reuse-password \ --network "TESTNET_NAME" \ --datadir /root/.lighthouse/data \ --directory /root/validator_keys

Checking the keys of all Lido Node Operators

Key checking works with on-chain data. Make sure key submission transactions are confirmed before checking the keys.

Never vote for increasing the key limits of Node Operators before verifying new keys are present and valid.

Lido CLI

Make sure Python with pip is installed and then run:

pip install lido-cli lido-cli --rpc http://1.2.3.4:8545 validate_network_keys --details This operation checks all Lido keys for validity. This is a CPU-intensive process, for example, a modern desktop with 6 cores, 12 threads and great cooling processes 1k keys in 1—2 seconds.

You would need an RPC endpoint - a local node / RPC provider (eg Alchemy/Infura).

Lido Node Operator Dashboard

You can also check the uploaded keys onMainnet Lido Node Operator Dashboard orTestnet Lido Node Operator Dashboard .

This UI shows a number of submitted, approved and valid keys for each Node Operator, along with all invalid keys in case

there are any.

It is updated every 30 minutes via cron, but update period may change in the future.

Results

You don't see invalid keys

If the new keys are present and valid, Node Operators can vote for increasing the key limit for the Node Operator.

You spot invalid keys

It is urgent to notify Lido team and other Node Operators as soon as possible. For example, in the group chat.

Increasing the Staking Limits with an Easy Track motion

Once new keys are present and valid, a motion can be proposed to increase the staking limit for the Node Operator.

Node Operators Guide to Easy Track Edit this page Previous General Overview Next Execution Layer Rewards Configuration