Intro

After withdrawals are live on Ethereum, special tooling setup will be required to effectively manage exits for 100+ thousands of Lido validators.

We propose a semi-automatic solution which requires Node Operators to pre-sign exit messages and run a dedicated daemon which will broadcast the messages to the network when Lido Oracles publish exit requests on chain inLidoOracle events.

Problem Statement

Ethereum validator exits are going to bevoluntary: there is currently no way for Lido to force a Node Operator to exit one of its validators using withdrawal credentials or other methods without increasing centralisation. It's inefficient and impractical to manually notify Node Operators each time they need to initiate an exit and for them to manually exit validators.

Solutions like having Node Operators pre-sign exit messages and hand over to Lido introduce centralisation dangers as in case of a vulnerability or a software error validators of all Node Operators can be exited.

Assumptions

Validator Key Management

We assume that Lido Node Operators either:

- Have keystore files in BLS12-381 format for their validators
- Run key management software that has an API that allows signing exit messages with (eg Dirk or Web3Signer)

Keystores

We assume that some Node Operators have their validator keys inEIP-2335 BLS12-381 Keystore format, have access to them and their passwords.

We also presume these keystores were likely generated using the official [ethereum/staking-deposit-cli

](https://github.com/ethereum/staking-deposit-cli) (ex eth2cli

) or using custom tooling, but following same output format spec.

External Signers

We assume that a high percentage of node operators (40%+) use an external signer for their validator clients, mainly Dirk or Web3Signer.

Furthermore, we see substantial efforts from Node Operators to research, experiment with and migrate to external signers, so we expect the percentage to rise in the nearest future.

Rough summary of external signer usage of Node Operators, both active setups and future migration intent is counted:

Node Operators

Dirk

Web3Signer

No External Signer

27

26%

19%

59%

In case of proprietary software being used by Node Operators, we assume that Node Operators will be able to use it to sign exit messages with.

Algorithm to Choose Validators for Exit

We presume that algorithm will be deterministic. This way Node Operators can pre-generate and sign a certain number of exit messages ahead of time.

For security purposes, we think that Node Operators should not pre-sign all validator keys at once and sign a small fraction of keys instead, watching closely how many exit messages are left in the daemon to sign more if needed.

Solution

[

Val exits

3240×936 178 KB

](https://europe1.discourse-cdn.com/business20/uploads/lido/original/2X/4/46730d8b093bf26670052e841a5b4128bf6f32a5.png)

We propose a semi-automatic solution, which heavily eases operational burden without security compromises.

Lido will develop a deterministic algorithm which will choose which validators have to exit depending on a variety of factors.

Node Operators will host a new API service which will be able to load all validator keys data of Lido Node Operators, calculate which one will need to be exited next and generate exit messages for them.

Node Operators are going to pre-sign a certain number of validator exit messages that are going to be exited next.

Node Operators will host a daemon which will send these messages to a consensus client whenLido Oracles signal for their validators to exit using LidoOracle contract events. From there, consensus client willbroadcast exit messages and exit will be performed. If/when 0x03 and GMB approaches are implemented, ExitContract.exit(bytes)

will simply be called to initiate an exit.

Proposed new tooling consists of 3 apps:

- Keys API Service to load all Lido validators and calculate next validators to exit on request (Node Operators)
- Ejector daemon to send exit messages when exits are requested (Node Operators)
- Monitoring daemon that checks if requested validators are not exited (Lido)

[

Screenshot 2022-11-21 at 15.57.28

1394×796 66.6 KB

](https://europe1.discourse-cdn.com/business20/uploads/lido/original/2X/4/471edb9fcf0355980eb1d8f3c38697edc7ac8d85.png)

Thanks to Kuhan from Consensys Codefi for this diagram!

Node Operator Flow:

1. Query which validators should be exited next.
2. Generate and sign exit messages for them.
3. Add generated data to the daemon.

Additionally, messages will become invalid and willhave to be regenerated (spec) every second consensus layer fork after generation.

As Lido has no way to force a exit, it is vital to monitor that Node Operators do indeed exit validators we ask for. Lido will host a special monitoring daemon for this.

Notes:

- Lido never has access to signed exit messages.
- Lido can recommend how many keys operators should pre-sign to follow withdrawal demands, but not force it.

Getting Validators Indexes for Exit via Keys API Service

This is an open-source service responsible for getting validator keys data of Lido Node Operators.

Its data providers are independent modules, meaning that in the future it will be able to query not only the Node Operator Registry Contract, but IPFS storage or other storage solutions both on-chain and off-chain.

It has a special endpoint which returns a list of validators with which exit messages should be generated and signed.

http://keys-api-service/exits

{ "0": [{ "validator_index": 123, "public_key": "0x123" }], "1": [{ "validator_index": 123, "public_key": "0x123" }] }

And be filtered by a specific Node Operator index:

http://keys-api-service/exits?operatorId=0

[{ "validator_index": 123, "public_key": "0x123" }, { "validator_index": 123, "public_key": "0x123" }]

Additionally, this endpoint potentially can return ready to sign exit messages:

http://keys-api-service/exits?operatorId=0&messages=true

[{ "epoch": 123, "validator_index": 123 }, { "epoch": 123, "validator_index": 123 }]

Although Lido will host an instance of this service, it is crucial for each Node Operator to run an instance.

Spec Draft

Generating and Signing Exit Messages

We suggest to use an existing ethdo utility when possible for a number of reasons:

1. It supports 2/3 required setups: local keystores and Dirk, leaving only Web3Signer.
2. It can craft exit messages and sign them out of the box, has an option to export signed message instead of broadcasting it straight away.
3. There is one less piece of software for us to develop, test and maintain.
4. Dirk users are very likely to already be familiar with this tool and if not, experience can come in handy for migration to external signers.

But, node operators have the freedom to generate and sign exit messages in a different way as long as output format is valid. This allows them to use their proprietary tooling for generation, but still rely on our ejector.

Keystores - via ethdo

Node operators have an option to decrypt validator key keystores in EIP-2335 JSON format of BLS12-381 keys using ethdo. This is useful when no external signers are used and keystores are available either in a centralized location or on validator machine for importing into a validator client.

Create a new wallet:

ethdo --base-dir=./ethdo wallet create --wallet=test

Add key from the keystore:

ethdo --base-dir=./ethdo account import --account=test/test --keystore=./ethdo/keystore.json --keystore-passphrase=test --passphrase=test

Generate and sign an exit message:

ethdo --base-dir=./ethdo validator exit --account="test/test" --passphrase="test" --json --connection="http://127.0.0.1:5051"

Alternatively, you can do this using the validator private key instead:

ethdo validator exit --key=0x123 --json --connection="http://127.0.0.1:5051"

Docs

Complete Example

Dirk - via ethdo

Dirk is supported in ethdo out of the box.

ethdo --remote=server.example.com:9091 --client-cert=client.crt --client-key=client.key --server-ca-cert=dirk_authority.crt validator exit --account=Validators/1 --json --connection="http://127.0.0.1:5051"

Docs

Web3Signer - via API requests

Web3Signer is not supported by ethdo, so manual message generation and API requests are needed.

Messages themselves are simple:

class VoluntaryExit(Container): epoch: Epoch # Earliest epoch when voluntary exit can be processed validator_index: ValidatorIndex

Then, an API request to a Web3Signer needs to be made:

Docs

After signing, a signed message structure needs to be formed:

class SignedVoluntaryExit(Container): message: VoluntaryExit signature: BLSSignature

Complete example:

const VALIDATOR_INDEX = '123' const VALIDATOR_PUBLIC_KEY = '0x123'

const WEB3SIGNER = 'http://127.0.0.1:9000' const WEB3SIGNER_ENDPOINT = ${WEB3SIGNER}/api/v1/eth2/sign/${VALIDATOR_PUBLIC_KEY}

const CONSENSUS_NODE = 'http://127.0.0.1:5051' const CONSENSUS_FORK_ENDPOINT = ${CONSENSUS_NODE}/eth/v1/beacon/states/finalized/fork const CONSENSUS_GENESIS_ENDPOINT = ${CONSENSUS_NODE}/eth/v1/beacon/genesis

const forkReq = await fetch(CONSENSUS_FORK_ENDPOINT) const forkRes = await forkReq.json() const fork = forkRes.data

const genesisReq = await fetch(CONSENSUS_GENESIS_ENDPOINT) const genesisRes = await genesisReq.json() const genesis_validators_root = genesisRes.data.genesis_validators_root

const voluntary_exit = { epoch: '123', validator_index: VALIDATOR_INDEX, }

const body = { type: 'VOLUNTARY_EXIT', fork_info: { fork, genesis_validators_root, }, voluntary_exit, }

const signerReq = await fetch(WEB3SIGNER_ENDPOINT, { method: 'POST', headers: { 'Content-Type': 'application/json', Accept: 'application/json' }, body: JSON.stringify(body), }) const signature = await signerReq.text()

const signedMessage = { message: voluntary_exit, signature, }

console.log(signedMessage)

Exit Spec, Source

Source: Lodestar

Lighthouse Example

How message is verified in spec

Ejector Daemon

A special single-instance daemon is hosted by Node Operators same way asLido Oracle.

It listens to events ofLidoOracle contract and if it has a signed message for pubkey which needs to be exited, submits signed exit message to a consensus node using Beacon/submitPoolVoluntaryExit.

Requires:

- Folder with signed exit messages.
- Access to a Consensus ETH node which will broadcast exit messages to the network.

Exit message format in the Ejector has no differences to the ETH exit messages specification. So, custom tooling can be used by Node Operators for message generation/signing while still being able to use the Ejector.

class VoluntaryExit(Container): epoch: Epoch # Earliest epoch when voluntary exit can be processed validator_index: ValidatorIndex

class SignedVoluntaryExit(Container): message: VoluntaryExit signature: BLSSignature

On daemon start, message files provided are checked to exist and have correct format.

Consensus client verifies the signed message on submit and will return 400 status code if exit message is invalid.

Daemon also provides metrics like numbers of loaded messages and how many have already been submitted. Node operators are encouraged to monitor these and use in dashboards so they are alerted when signed messages run out.

Monitoring

Special daemon hosted by Lido will listen toLidoOracle contract validator exit events same way a Node Operator daemon does, but will also check that validators are indeed exited in proper time. If not, team is alerted, who will notify the responsible Node Operator.

Requirements:

- ETH Consensus Node or service

Ways to check:

- Beacon/getStateValidators: /eth/v1/beacon/states/head/validators?id=['0x123', …]
- Beacon/getStateValidators: /eth/v1/beacon/states/head/validators?status=['exited', …]
- Beacon/getBlockV2

Alternative Ideas

Fully Manual

We ping operators which validators they need to exit and that's it.

Big operational load both for us and Node Operators.

Fully Automatic All-in-one Daemon

We could develop a daemon to be run by Node Operators that will load keys from validator client storage, decrypt them and on contract events immediately sign and broadcast an exit message.

Prysm example:

import { readFile } from 'fs/promises' import { decrypt } from '@chainsafe/bls-keystore'

const json = JSON.parse(await readFile('all-accounts.keystore.json'))

const pass = '12345678' const decrypted = await decrypt(json, pass)

const decoded = new TextDecoder().decode(decrypted)

console.log(decoded)

Lighthouse: Key management main concepts

Lighthouse: Validator key management

Lighthouse: Importing staking-deposit-cli (ex eth2cli) keystores

- Bug in the daemon will allow exploiter to exit all Lido validators - very risky and risk is not worth it
- Operators would have to spin up an additional daemon on their execution-consensus-validator clusters
- Validator clients store keys differently, we won't create and maintain ~5 adapters

Q&A

- Can we encrypt exit message files so they are not stored in plain text?

Yes, but Node Operators would have to encrypt messages themselves, what brings higher complexity and margin for error than simply generating.

- Can we make use of epoch

values inside exit messages?

Yes, for example we could choose a future epoch

for potentially higher security, but doing so will prevent us to react immediately to changing demand levels for Lido withdrawals.

- Do we need a specific epoch

at all in the exit message?

Yes. If we hardcode a value, then messages generated will expire after two hard forks from that epoch

value. We default to the current epoch

in exit messages instead.

- For how many validators exit messages should be pre-signed?

Current suggestion is that Node Operators should pre-sign 10% of their keys, oldest keys first.

Active discussion is on Lido Research Forums.