

DepositSecurityModule

- [Source Code](#)
- [Deployed Contract](#)

Due to front-running vulnerability, we [proposed](#) to establish the Deposit Security Committee dedicated to ensuring the safety of deposits on the Beacon chain:

- monitoring the history of deposits and the set of Lido keys available for the deposit, signing and disseminating messages allowing deposits;
- signing the special message allowing anyone to pause deposits once the malicious Node Operator pre-deposits are detected.

Each member must generate an EOA address to sign messages with their private key. The addresses of the committee members will be added to the smart contract.

To make a deposit, we propose to collect a quorum of 4/6 of the signatures of the committee members. Members of the committee can collude with node operators and steal money by signing bad data that contains malicious pre-deposits. To mitigate this, we propose allowing a single committee member to stop deposits and also enforce space deposits in time (e.g., no more than 150 deposits with 25 blocks in between them) to provide the single honest participant the ability to stop further deposits even if the supermajority colludes.

The guardian himself, or anyone else who has a signed pause message, can call `pauseDeposits` that `pausesDepositSecurityModule`.

To prevent a replay attack, the guardians sign the block number when malicious pre-deposits are observed. After a certain number of blocks (`pauseIntentValidityPeriodBlocks`) message becomes invalid.

Values of the parameters `maxDepositsPerBlock` and `minDepositBlockDistance` are controlled by Lido DAO and must be harmonized with `churnValidatorsPerDayLimit` of [OracleReportSanityChecker](#).

View Methods

getOwner()

Returns the contract's owner address.

function

getOwner ()

external

view

returns

(address) ;

getPauseIntentValidityPeriodBlocks()

Returns `PAUSE_INTENT_VALIDITY_PERIOD_BLOCKS` (see `pauseDeposits`).

function

getPauseIntentValidityPeriodBlocks ()

external

view

returns

(uint256)

getMaxDeposits()

Returns max amount of deposits per block (see `depositBufferedEther`).

function

getMaxDeposits ()

external

view

returns

(uint256)

getMinDepositBlockDistance()

Returns min distance in blocks between deposits (seeddepositBufferedEther).

function

getMinDepositBlockDistance ()

external

view

returns

(uint256)

getGuardianQuorum()

Returns the number of valid guardian signatures required to vet (depositRoot, nonce) pair.

function

getGuardianQuorum ()

external

view

returns

(uint256)

getGuardians()

Returns guardian committee member list.

function

getGuardians ()

external

view

returns

(address []

memory)

isGuardian()

Checks whether the given address is a guardian.

function

isGuardian (address addr)

external

view

returns

(bool)

Parameters

Name	Type	Description
addr	address	Valid ETH-1 address

getGuardianIndex()

Returns index of the guardian, or -1 if the address is not a guardian.

function

getGuardianIndex (address addr)

external

view

returns

(int256)

Parameters

Name	Type	Description
addr	address	Valid ETH-1 address

canDeposit()

Returns whether `LIDO.deposit()` can be called and a deposit can be made for the staking module with `idstakingModuleId`, given that the caller will provide guardian attestations of non-stale deposit root and `nonce` and the number of such attestations will be enough to reach a quorum.

function

canDeposit (uint256 stakingModuleId)

external

view

returns

(bool)

Parameters

Name	Type	Description
stakingModuleId	uint256	Id of the staking module

Methods

setOwner()

Sets new owner.

function

setOwner (address newValue)

external ; note Reverts if any of the following is true:

- `msg.sender`
- is not the owner;
- `newValue`
- is zero address.

Parameters

Name	Type	Description	newValue	address	New owner	address
------	------	-------------	----------	---------	-----------	---------

setPauseIntentValidityPeriodBlocks()

SetspauseIntentValidityPeriodBlocks .

function

setPauseIntentValidityPeriodBlocks (uint256 newValue) note Reverts if any of the following is true:

- msg.sender
- is not the owner;
- newValue
- is 0 (zero).

Parameters

Name	Type	Description	newValue	uint256	Number of blocks after which message becomes invalid
------	------	-------------	----------	---------	--

setMaxDeposits()

SetsmaxDepositsPerBlock .

The value must be harmonized with the parameterchurnValidatorsPerDayLimit of [OracleReportSanityChecker](#) .

function

setMaxDeposits (uint256 newValue) note Reverts if any of the following is true:

- msg.sender
- is not the owner.

Parameters

Name	Type	Description	newValue	uint256	New value of the maxDepositsPerBlock parameter
------	------	-------------	----------	---------	--

setMinDepositBlockDistance()

SetsminDepositBlockDistance .

The value must be harmonized with the parameterchurnValidatorsPerDayLimit of [OracleReportSanityChecker](#) .

function

setMinDepositBlockDistance (uint256 newValue) note Reverts if any of the following is true:

- msg.sender
- is not the owner.

Parameters

Name	Type	Description	newValue	uint256	New value of the min DepositsPerBlock parameter
------	------	-------------	----------	---------	---

setGuardianQuorum()

Sets the number of valid guardian signatures required to vet (depositRoot, nonce) pair (aka "quorum").

function

setGuardianQuorum (uint256 newValue) note Reverts if any of the following is true:

- msg.sender
- is not the owner;

Parameters

Name	Type	Description	newValue	uint256	New quorum value
------	------	-------------	----------	---------	------------------

addGuardian()

Adds a guardian address and sets a new quorum value.

function

addGuardian (address addr ,

uint256 newQuorum) note Reverts if any of the following is true:

- msg.sender
- is not the owner;
- addr
- is already a guardian.

Parameters

Name	Type	Description
addr	address	Guardian address
newQuorum	uint256	New Quorum value

addGuardians()

Adds a set of guardian addresses and sets a new quorum value.

function

addGuardians (address []

memory addresses ,

uint256 newQuorum) note Reverts if any of the following is true:

- msg.sender
- is not the owner;
- any of theaddresses
- is already a guardian.

Parameters

Name	Type	Description
addresses	address[]	Array of Guardian addresses
newQuorum	uint256	New Quorum value

removeGuardian()

Removes a guardian with the given address and sets a new quorum value.

function

removeGuardian (address addr ,

uint256 newQuorum) note Reverts if any of the following is true:

- msg.sender
- is not the owner;
- addr
- is not a guardian.

Parameters

Name	Type	Description
addr	address	Guardian address
newQuorum	uint256	New Quorum value

pauseDeposits()

Pauses deposits for staking module given that both conditions are satisfied (reverts otherwise):

1. The function is called by the guardian with index guardianIndex OR sig
2. is a valid signature by the guardian with index guardianIndex of the data
3. defined below.
4. block.number - blockNumber <= pauseIntentValidityPeriodBlocks

The signature, if present, must be produced for keccak256 hash of the following message (each component taking 32 bytes):

| PAUSE_MESSAGE_PREFIX | blockNumber | stakingModuleId |

If the staking module is not active does nothing. In case of an emergency, the function `pauseDeposits` is supposed to be called by all guardians. Thus only the first call will do the actual change. So the other calls would be OK operations from the point of view of the protocol logic.

function

`pauseDeposits (uint256 blockNumber ,
uint256 stakingModuleId , Signature memory sig)`

Parameters

Name	Type	Description
<code>blockNumber</code>	<code>uint256</code>	Block number with malicious pre-deposits have been observed by the guardian
<code>stakingModuleId</code>	<code>uint256</code>	Id of the staking module to pause deposits for
<code>sig</code>	Signature Short ECDSA guardian signature	as defined in EIP-2098

`unpauseDeposits()`

Unpauses deposits for staking module. If the staking module is not paused, do nothing.

function

`unpauseDeposits (uint256 stakingModuleId)` note Reverts if any of the following is true:

- `msg.sender`
- is not the owner.

Parameters

Name	Type	Description
<code>stakingModuleId</code>	<code>uint256</code>	Id of the staking module

`depositBufferedEther()`

Verifies the deposit security conditions are met and calls `LIDO.deposit(maxDepositsPerBlock, stakingModuleId, depositCalldata)` . Otherwise reverts.

note Reverts if any of the following is true:

1. `IDepositContract.get_deposit_root() != depositRoot;`
2. `StakingModule.getNonce() != nonce;`
3. The number of guardian signatures is less than `getGuardianQuorum()`;
4. An invalid or non-guardian signature received;
5. `block.number - StakingModule.getLastDepositBlock() < minDepositBlockDistance;`
6. `blockhash(blockNumber) != blockHash`. Signatures must be sorted in ascending order by the index of the guardian. Each signature must be produced for the keccak256 hash of the following message (each component taking 32 bytes):

| ATTEST_MESSAGE_PREFIX | blockNumber | blockHash | depositRoot | stakingModuleId | nonce |

function

`depositBufferedEther (uint256 blockNumber , bytes32 blockHash , bytes32 depositRoot , uint256 stakingModuleId , uint256 nonce , bytes`

`calldata depositCalldata , Signature []`

`calldata sortedGuardianSignatures)`

Parameters

Name	Type	Description
<code>blockNumber</code>	<code>uint256</code>	Number of the current deposit block
<code>blockHash</code>	<code>bytes32</code>	Hash of the current deposit block
<code>depositRoot</code>	<code>bytes32</code>	Deposit root of the Ethereum DepositContract
<code>stakingModuleId</code>	<code>uint256</code>	Id of the staking module to deposit with
<code>nonce</code>	<code>uint256</code>	Nonce of key operations of the staking module
<code>depositCalldata</code>	<code>bytes</code>	Staking module deposit calldata
<code>sortedGuardianSignatures</code>	<code>Signature[]</code>	Short ECDSA guardians signatures as defined in EIP-2098 Edit this page Previous wstETH Next Burner