

Solutions to Guard Against MEV Theft

Ken Smith (@NextBlockSoln) | Nextblock Solutions
March 2024

DRAFT

This report is released under CC BY 4.0.



Table of Contents

1. Abstract	2
2. Relevant Ethereum Staking Concepts	3
2.1. Staking Rewards	3
2.1.1. Consensus Layer (CL) Rewards	3
2.1.2. Execution Layer (EL) Rewards	3
2.2. MEV	4
2.3. Lottery Blocks	4
2.4. Proposer-Builder Separation (PBS)	5
2.5. EIP-4788: Beacon Block Root in EVM	6
2.6. Exit Messages	6
2.7. EIP-7044: Perpetually Valid Signed Voluntary Exits	7
3. MEV Theft	8
3.1. MEV Theft Strategies	9
3.1.1. Lottery Block Attack	9
3.1.2. Long-Con Attack	11
3.2. Detecting MEV Theft	11
4. MEV Theft Protectorents	11
4.1. Alternative Solutions	12
4.1.1. DVT (Off-Chain)	12
4.1.2. Fee Recipient Enforcement by MEV Relays (Off-Chain)	13
4.1.3. Reduced (or Negative) Node Commission (Off-Chain)	13
4.1.4. Enshrined Proposer-Builder Separation (eEPS) (Enshrined)	14
4.2. Forced Exits	15
4.2.1. Voluntarily Signed Exit Messages (Off-Chain)	16
4.2.2. EIP-7002: Execution Layer Triggerable Exits (Enshrined)	17
5. Appendix A: A compendium of forced-exit designs using Shamir sharing and verifiably encrypted signatures (VES) voluntarily signed exit messages.	19

1. Abstract

<https://ethresear.ch/t/validator-smoothing-commitments/17356>

This paper will present several options that can be used to protect against maximum extractable value (MEV) theft by staking pools composed of permissionless node operators (NO). MEV theft may occur when NOs instruct a proposer-builder separation (PBS) relay to send the winning bid payment to an address under the exclusive control of the NO. MEV theft robs the participating liquid staking token (LST) holders of their portion of the MEV reward. Without an enforceable penalty system to offset the loss of funds, permissionless staking pools may be an efficient vector to conduct profitable MEV theft by adversarial NOs.

In this paper's context, *permissionless staking pools* refer to LST protocols where the NO needs no permission from the protocol to enroll and participate in the validating pool other than the contractual node deposit amount. The node deposit is a financial bond the LST protocol can confiscate if the NO performs an adverse or harmful event. If no harming offense is incurred, the node deposit is returned in full to the NO when they successfully exit the Beacon chain and no longer are required to perform their validating duties.

This paper will provide an overview of recent enhancements to the core Ethereum protocol implemented in the *Dancun* hard fork, proposals for future Ethereum Improvement Proposals (EIPs) that may be considered in later core protocol upgrades (e.g., *Praelectra*), and alternative out-of-protocol MEV theft mitigation strategies that an LST protocol may consider.

We will explore the feasibility of using forced exits as MEV theft protectants. A "forced exit" for an Ethereum validator refers to the process where a decentralized staking pool can initiate the exit of a validator from the network without interaction by the NO. This differs from a voluntary exit, where a NO initiates the request to stop validating duties and begin the exit process from the Beacon chain.

2. Relevant Ethereum Staking Concepts

Before exploring the problem of MEV theft and possible mitigation strategies, we will review some of the relevant Ethereum staking concepts.

2.1. Staking Rewards

In the current Ethereum proof-of-stake model, validators earn rewards for the Consensus Layer (CL) and Execution Layer (EL).

2.1.1. Consensus Layer (CL) Rewards

CL rewards comprise three sub-types. There are rewards earned for proposing a new block in the chain (Block Proposal), rewards for participating in consensus by attesting to valid blocks (Attestation), and rewards for being selected and performing duties as part of validator committees (Sync Committee). These CL rewards provide security incentives for the proof-of-stake Ethereum network. They come from deterministic protocol-established formulas within each subtype. CL rewards do not vary significantly for well-performing nodes and primarily depend on the number of active validators participating in the consensus process and the random luck of the validator to be selected for block proposing and sync committee duties.

The CL reward's economic source is the new ETH issuance by the Ethereum protocol itself. Periodic payments of CL rewards are made from the protocol software to a designated address (the *withdrawal credential*) specified only once by a validator at the time of registration with the Beacon chain.

2.1.2. Execution Layer (EL) Rewards

In the current Ethereum proof-of-stake model, validators can earn additional variable rewards on top of CL rewards. These are called Execution Layer (EL) rewards and are composed of two subtypes: Maximization of Extractable Value (MEV) payments received from certain transactions that result in a profitable change of state of the

blockchain. Priority Fees (a.k.a. tips) are fees blockchain users pay to validators for priority inclusion of their transactions in the block.

The ultimate source of these rewards is Ethereum network users willing to pay for their transaction(s) to be executed in a particular block or a specific sequence within the block, which are thus more variable than the fixed CL rewards. EL rewards are paid to an address referred to as the `feeRecipient`. A validator will specify a `feeRecipient` address when they propose a new block. The NO can specify or change the `feeRecipient` at almost any time.

2.2. Maximal Extractable Value (MEV)

Maximal Extractable Value (MEV) is the total value validators can extract from transaction ordering beyond the standard block rewards and priority fees. MEV arises from validators' ability to arbitrarily include, exclude, or reorder transactions within the blocks they produce. The degree to which a specific set of MEV transactions may extract profits from state changes of the Ethereum Virtual Machine (EVM) depends highly on financial market conditions and user activity on the blockchain.

Most validators participate in MEV extraction by outsourcing their block production. They use services like PBS relays, mainly accomplished using the Flashbot *MEV-Boost* software, to auction their block production responsibilities in exchange for payment determined by a rapid auction bidding process.

2.3. Lottery Blocks

A "lottery block" refers to a rare block proposal opportunity where the MEV value exceeds the node deposit required to form a staking pool validator.

The rarity of lottery blocks and their unpredictable occurrence adds complexity to managing and mitigating risks associated with MEV in LST protocols. This potential for significant reward tempts NOs to act against the interests of the pool, risking the integrity of fair reward distribution mechanisms.

2.4. Proposer-Builder Separation (PBS)

Proposer-builder separation (PBS) allows validators (the “proposer”) to outsource their block-building duties to specialized “builders” who are well-equipped to extract MEV using an auction system administered by MEV relays. MEV relays separate the roles of proposer and builder. Proposers sell their block-production rights to builders who pay for the privilege of choosing the transaction order in a block. Builders compete by bidding an amount to the proposer to have their constructed block selected to be ultimately proposed and inserted into the blockchain. The builder will retain a portion of the MEV for themselves as profit.¹

Most NOs will auction their proposal opportunity to a third-party block builder to maximize the rewards they earn in staking. This auction system has been highly optimized, reliable, and accessible to all NOs via the popular *MEV-Boost*² software addon. *MEV-Boost* is an implementation of proposer-builder separation (PBS) built by Flashbots for proof-of-stake Ethereum. Validators running MEV-Boost maximize their staking reward by selling blockspace to an open market of builders.

When a NO participates in *MEV-Boost*, they register with an MEV relay and specify a specific payment address (`suggested-fee-recipient`) to receive payment from the winning block builder. When participating in PBS, the produced block’s `freeRecipient` is typically set to the builder’s address, not the proposer’s. Instead, the winning block builder pays the bid amount (composed of MEV and inclusion tips) via a standard transaction in the built block.³ As part of their duties, the relay checks the block bid submitted to ensure that the block builder has included a payment equal to the winning bid to the registered `suggested-fee-recipient` in their built block.

When a NO participates in PBS via MEV relays, they forgo their tips as they are redirected to the builder’s `freeRecipient` address. However, the proposer (validator) will ultimately earn these rewards as they are factored into the bid amount the builder offers in the auction process.

¹ <https://ethresear.ch/t/why-enshrine-proposer-builder-separation-a-viable-path-to-epbs/15710>

² <https://boost.flashbots.net/>

³ The proposer payment transaction is typically the last transaction in the block.

2.5. [EIP-4788](#): Beacon Block Root in EVM

[EIP-4788](#)⁴ exposes the hash tree root of parent CL⁵ to the Ethereum Virtual Machine (EVM)⁶ on the EL. EIP-4788 allows decentralized applications (dApps), including contracts of decentralized staking pools, to quickly access this data without trusting an oracle or off-chain data provider. EIP-4788 enables smart contracts to verify proofs about the CL's state in a trust-minimized way. It was included in *Dancun*⁷, the latest hard fork upgrade for the Ethereum blockchain.

Knowledge of the beacon root is essential because it allows a staking protocol to learn the `fee_recipient` contained in the `ExecutionPayload` of the `BeaconBlockBody`. Access to this information is necessary to determine if the NO natively produced the block or if third-party builder services were used.

Further, EIP-4788 allows a smart contract to be knowledgeable of a validator balance to use that value in calculating the amount that a NO has available as an assurance (insurance deposit) to have a risk (at stake) to assure honest MEV extraction.

2.6. Exit Messages

A NO (validator) initiates an exit by signing an exit message using the signing BLS key. This is the same key that the validator uses for day-to-day operations, such as creating and signing attestations and block proposals. Exiting as a validator is irreversible. Once exited, the validator cannot resume validating without setting up a new validator *de nuevo*.

A NO can leave validating responsibilities for various reasons, such as wanting to unlock and withdraw staked ETH, ceasing operations, or reallocating resources. It's important to note that exiting is generally a voluntary action initiated by the NO on a per-validator

⁴ <https://eips.ethereum.org/EIPS/eip-4788>

⁵ The CL is also known as the Beacon Chain.

⁶ The EVM stands for Ethereum Virtual Machine. It is the runtime environment for smart contracts in Ethereum.

⁷ Dancun is a contraction of Deneb, the name of the next CL hard fork named after star names whose first letter is in alphabetical order, and Cancun, the name of the next EL hard fork named a chronicle order of cities that have hosted prominent Ethereum developers conferences (i.e., Devcon III).

basis. There are also involuntary exits Executed in the core Ethereum protocol due to excessive inactivity penalties or slashing violations, which differ from the voluntary exit process.

Before the *Dancun* upgrade, an exit message had an expiration because the function `get_domain`⁸ always returned either the current or prior fork version. So, a signature created two forks prior would only verify if the domain was correct.⁹ The expiry was designed to isolate voluntary exits in case of a hard fork that results in two maintained chains. If a validator operates on both forks and sends an exit request, the message could be replayed on both chains. This limitation was put in place as a safety measure.

2.7. [EIP-7044](#): Perpetually Valid Signed Voluntary Exits

EIP-7044¹⁰ (included in *Dancun*¹¹) locks the voluntary exit signature domain with the *Capella* update to ensure that signed voluntary exits remain valid indefinitely. This enhancement aims to simplify staking operations and improve user experience by removing the need to pre-sign exits for an unbounded number of forks, which is complex when the staking operator and the fund owner are different entities.

EIP-7044 modified the state transition function to fix the signing domain and root on the `CAPELLA_FORK_VERSION`. The update aligns the user experience of voluntary exits with other processes like `BLSToExecutionChanges` and deposits, negating the need for downstream tooling updates with each fork version.

Backward compatibility was maintained in the Consensus Layer, but users must know that pre-signed exits using the Deneb fork domain will no longer be valid. They should ensure exits are signed using the *Capella* fork domain for continued validity. One security consideration is that the modification removes replay protection for voluntary

⁸ https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#get_domain

⁹ For example, an exit message signed for in the first *Altair* fork specifications was valid in both *Altair* and *Bellatrix* (the subsequent upgrade) but is invalid in the *Capella* fork.

¹⁰ EIP-7044: Perpetually Valid Signed Voluntary Exits <https://eips.ethereum.org/EIPS/eip-7044>

¹¹ Ethereum Consensus Specs, Deneb -- The Beacon Chain <https://ethereum.github.io/consensus-specs/specs/deneb/beacon-chain/>

exits after two hard forks, which could affect individual stakers but does not put funds at risk or compromise the chain's security.

3. MEV Theft

MEV theft in the context of LST pools refers to the unauthorized redirection of MEV rewards by NOs to an address they control exclusively at the expense of the rightful LST holders. MEV theft can occur when NOs instruct a PBS relay to send the winning bid payment for block production to an address under their sole control. This redirection undermines the fair distribution of MEV rewards that LST participants in the staking pool should receive.

One of the risks permissionless LST protocols encounter is that a malicious NO could use the lower cost of obtaining a validator to maximize their ability to steal MEV. Although MEV rewards only constitute, on average, about 25% of the predicted earnings of an Ethereum validator, lucky “lottery block” proposers can receive MEV rewards exceeding tens to hundreds of ETH in a single block proposal.



Fig 1. Validator Rewards (source: <https://ultrasound.money/>)

Introducing ultra-low ether bonded ULEB minipools (i.e., minipools with less than 4 ETH as a NO deposit) creates multiple new risks for the RP protocol. One such risk vector is where a malicious NO could use the lowered minipool costs to hijack the proposer payment value (PPV) of potential lucrative maximal extractable value (MEV) block proposals. Because of the lower NO bonding values, it may be cost-effective for a NO to ignore any penalty for cheating assigned by the RP protocol. As the ULEB value decreases, so does the penalty size that RP can assess as a malicious NO.

Further complicating this is that some residual amount of the NO deposit needs to be exempt from a penalty to incentivize the NO to exit once they are done with the MEV heist. The concern is that the NO can hold the validator hostage by refusing to exit it and allowing it to lie abandoned in an active state.

It is important to note that there are existing permissionless staking pools that allow anyone to participate, provided that they make a sufficient deposit of note operator funds to act as a bond to ensure their honest participation. LST protocols already incur a risk of MEV theft, which they have determined acceptable.

3.1. MEV Theft Strategies

There are two conceived strategies that a malicious NO can use to steal MEV. The first strategy is to wait until the malicious NO wins a lottery block where the amount of MEV in the node block exceeds the NO deposit. The second is that the NO immediately redirects all of the EL rewards (MEV) to a privately held address, incurring immediate penalties by the LST protocol but factoring that their small but continued thefts of MEV will ultimately exceed the value of any punitive action by the LST protocol. This strategy is known as the Long-Con.

3.1.1. Lottery Block Attack

In the first scenario, a NO will initially start validating as an honest NO. They will wait for the opportune time when a block proposal opportunity presents itself such that the MEV in the block exceeds the value of the NO deposit (i.e., a lottery block). At that

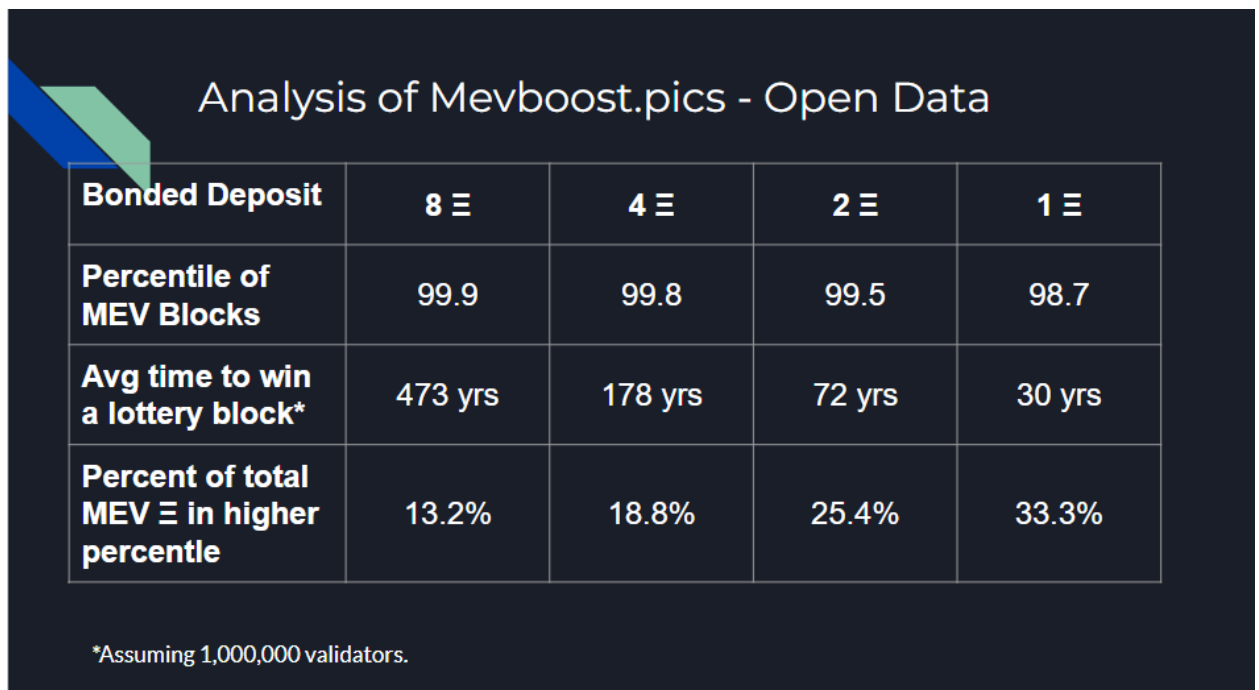
point, the dishonest NO will steal that block for themselves and, from that moment on, operate as a dishonest NO.

Using the etherscan dataset, we can calculate the average number of years for a minipool to win a lottery block. The results of that calculation are found in Table 10.

Table 10
Average Number of Years for a Minipool to Propose a Lottery Block Assuming 1M Validators.

A NO can not change their payment address with a relay after knowing the bids for their proposed slot. Validators may only change their payment address at the beginning of each epoch when the NO registers with a relay. The lottery block strategy could be performed by registering a valid payment address with one relay and a malicious address with a second relay and then choosing the malicious relay only when a lottery block is found.

As shown in Figure 2, a malicious NO could continue to operate honestly until they win one of these lottery blocks.



Bonded Deposit	8 Ξ	4 Ξ	2 Ξ	1 Ξ
Percentile of MEV Blocks	99.9	99.8	99.5	98.7
Avg time to win a lottery block*	473 yrs	178 yrs	72 yrs	30 yrs
Percent of total MEV Ξ in higher percentile	13.2%	18.8%	25.4%	33.3%

*Assuming 1,000,000 validators.

Figure 2: Probabilities of lottery blocks.

3.1.2. Long-Con Attack

The second strategy unfolds with a NO who, from the onset of their validation activities, diverts all MEV to themselves surreptitiously. This approach, dubbed the "long-con attack," envisages a prolonged period of deception where the NO accumulates MEV rewards at the expense of the collective pool, bypassing the equitable distribution of rewards. This strategy's feasibility hinges on the continuous generation of MEV under favorable conditions to the NO's covert operations.

3.2. Detecting MEV Theft

Even when a staking protocol mandates participation in MEV extraction (e.g., the use of *MEV-Boost* and the subscription to one or more of the PBS relays), there may be circumstances where the node produces a local block rather than the third-party block with the highest bid. This makes it difficult to detect when MEV theft has occurred.

Decentralized staking pools will need robust mechanisms to detect MEV theft. Methods for detecting MEV theft are beyond the scope of this report. I refer the reader to the definitive work published by Knoshua in his article about [MEV Proofs](#).¹²

4. MEV Theft Protectorents

Strategies to counteract the temptation for NOs to act dishonestly when encountering a lottery block are essential to ensure the earnings potential of LST holders. We have classified each strategy into two types: those that are executed off-chain and those that are reliant upon some enshrined component of the Ethereum protocol. We have organized the possible solutions into two broad classifications: those that use forced exits and those that employ another methodology.

A discussion of possible solutions to prevent MEV theft begins with alternative solutions.

¹² https://hackmd.io/@knoshua/S19XgucSn?utm_source=preview-mode&utm_medium=rec

4.1. Alternative Solutions

4.1.1. DVT (Off-Chain)

Distributed Validator Technology (DVT) is an innovative approach in the Ethereum staking ecosystem that enhances security and decentralization. In traditional staking, a single entity NO usually operates a validator node. DVT distributes the responsibilities and operations of a single validator node across multiple independent NOs. In this setup, different parts of the validator's operations, such as proposing blocks and attesting to block validity, are handled by different participants. This distribution mitigates risks like single points of failure and slashing penalties due to misbehavior or technical faults in a single node.

DVT-based validators may deter MEV theft because messages to register with an MEV relay need to be signed by a majority of cluster operators. Once the threshold majority of members have signed the registration message, it will be valid. This would be a large deterrent to stealing MEV as the honest cluster members would not sign a relay registration message if it designated a suggested fee recipient other than the correct payment address. However, possible exploit pathways exist when the DVT-based validator is selected as the block proposer.

Most DVT protocols elect a cluster leader in a rotating fashion to query the MEV relay when it is their time to block propose. Instead of provisioning a block proposed by the relay registered with the cluster, the leader could replace the block header with a substitute header that directs the MEV to an address owned by the malicious shareholder. The malicious cluster member would need to either develop search builder capabilities or conspire with others with that capability.

While distributed validator technology (DVT) provides the ability to reduce the NO collateral, it has two major drawbacks. First, there is no practical solution for a NO participating in a DVT share to exit a minipool, making recovery of a cluster member's initial node deposit highly problematic. Second, since the validator rewards earned are shared evenly among the number of NOs in the DVT share, there is no leverage for gaining a larger NO commission.

4.1.2. Fee Recipient Enforcement by MEV Relays (Off-Chain)

MEV-Boost relays could adopt practices that make MEV theft difficult. They could remove the ability for a NO to specify a `suggested-fee-recipient` and instead default the address to the selected `0x01` withdrawal credential without the capability to choose a different reward payment address. However, the malicious NO could avoid this by subscribing to other relays that do not standardize to this format.

Alternatively, relays could specify a delay period, allowing changes in a `suggested-fee-recipient` address only periodically. This would allow time for protocols and other interested parties to detect that the `suggested-fee-recipient` has been changed. But without the ability to penalize the malicious NO, this would have no deterrence.

4.1.3. Reduced (or Negative) Node Commission (Off-Chain)

In addressing the challenge of MEV theft, one strategy is to adopt an approach of "if you can't beat them, join them." This involves a staking pool adopting predictive modeling to estimate the expected average returns from MEV payments, thereby allowing the NO to collect all earned MEV.

This strategy¹³ posits that, on average, the LST holder would receive a total return from consensus layer earnings equivalent to traditional LST returns, where CL and EL rewards are shared proportionally. However, in this model, LST holders would pay a commission to the NO for their validation duties. Under this strategy, a predictive model could estimate the expected returns for LST holders, aiming to match the returns from a traditional LST arrangement. The commission rate for NOs could be adjusted accordingly, potentially leading to a negative commission rate to incentivize the right to validate.

This model has some challenges in that MEV prediction is complicated. MEV opportunities highly depend on having cryptocurrency exchanges and loan liquidation

¹³ See Valdorff's Risk analysis for LEBs, March 2023, https://github.com/Valdorff/rp-thoughts/tree/main/leb_safety

activities performed on Layer 1, as those transactions make up most of all MEV extractable events. A NO may not be willing to participate as a validator, taking on this unknown risk.

Furthermore, even if the future MEV activity was similar to the MEV opportunities experienced in Ethereum since the Merge, there is a significant variance in MEV payments in any given block. The NO assumes greater risk in this scenario, as earnings are likely less than the average MEV payment due to the long-tailed, skewed distribution of MEV occurrences. Some NOs may prefer this high-risk, high-reward approach, akin to a lottery, hoping to propose a block with significant MEV and thus gain a substantial return.

To mitigate these risks, a voluntary MEV smoothing pool could be introduced. In this system, NOs participating in the reduced node commission strategy could opt to pool their MEV payments. Periodically, the pooled rewards would be distributed among participants. However, this introduces a new risk: a malicious NO could redirect MEV payments to an address other than this voluntary smoothing pool.

As a possible solution, MEV relays could serve as both smoothing pool administrators and relays. MEV relays could lock the `suggested-fee-recipient` to the smoothing pool for any validator who wants to participate in the smoothing pool. This would ensure that all participating NOs receive 100% of the MEV.

4.1.4. Enshrined Proposer-Builder Separation (eEPS) (Enshrined)

Enshrined proposer-builder separation (eEPS) (also referred to as “protocol-level proposer-builder separation”) builds on this model but implements this as a core part of the consensus layer specification of the Ethereum protocol. It is conceivable that this opportunity for enshrinement would require that the `feeRecipient` address be locked to the same as the `withdrawal credential`. Such an approach would preclude the need to enforce the MEV theft penalties, as the core protocol will prevent it.

4.2. Forced Exits

For a validator to be exited, the NO must manually trigger it via a voluntary exit message. However, in cases such as staking pools, the owner of the funds being staked is not necessarily the validator. The validator has an opportunity to hold the owner's funds hostage. We could get around this if there were a way for a smart contract to trigger voluntary exits and know that an exit has happened.¹⁴

Forced exits, in combination with a penalty against the node deposit, could be used as a deterrent. Forced exits could be used if the validator acts maliciously, for example, if the NO performs MEV theft.

A second use case for forced exits involves validators who exit the beacon chain with a balance below 32 ETH. This situation can occur when an underperforming NO allows its validator to be offline for an extended period. When a node is offline, the validator(s) leak small amounts of ETH every epoch. The consensus layer protocol will exit any validator once their balance is below 15.75 ETH. However, with LEB4s, only 4 ETH of NO deposit is available to offset this loss, resulting in an 11.75 ETH loss that must be charged against the rETH APY. As an additional injury, it will take more than ten years for an abandoned LEB8 validator to leak to the protocol ejection balance under normal conditions. This leads to an additional lost opportunity cost where ETH locked on the abandoned validator is unproductive.

The ideal of a forced exit emerges as a preferred solution. This is because the forced exit solution is useful to counter either of the MAV theft strategies. A decentralized LST protocol's ability to force-exit a malicious or underperforming minipool solves the MEV-stealing and node-abandonment problems. Without the ability of the LST protocol to “force the exit” of a node, The protocol is severely limited in its capabilities to address MEV theft. With such an ability, a LST protocol could permit ultra-low-ether-bonded (ULEB) validators (e.g., 1 ETH node deposits).

¹⁴ <https://ethresear.ch/t/0x03-withdrawal-credentials-simple-eth1-triggerable-withdrawals/10021>

Decentralized staking pools have only limited mechanisms available to trigger such forced exits to protect the network and the interests of the staking participants. At present, none of the current forced exit options are trustless.

4.2.1. Voluntarily Signed Exit Messages (Off-Chain)

Initial solutions to obtain a force-exit capabilities use voluntarily signed exit messages that are held by the another party beside the NO, usually a LDT protocol custodian.

Providing the LST protocol with copies of these voluntarily signed exit messages introduces a novel approach for validator management. By providing the LST protocol with access to a signed exit message, a validator can ensure a seamless exit process without requiring further actions or permissions from the Node Operator (NO). This capability not only streamlines the exit procedure but also fortifies the security framework around validator operations, enabling a safeguard against potential operational risks or the intent to withdraw from the validating responsibilities for strategic reallocation of resources.

However, the centralized storage or management of these signed exit messages poses significant concerns regarding security and the potential for misuse. The aggregation of such critical and powerful messages within a single entity amplifies the risk of a single point of failure. In the event of unauthorized access or a security breach, the implications could be substantial, leading to unauthorized validator exits and destabilizing the network's validator set. Furthermore, this centralization contradicts the decentralized ethos of blockchain technologies, introducing potential vectors for manipulation or control that could erode trust in the system.

A prudent approach to mitigating these risks involves implementing robust security measures, including encryption, limited access controls, and decentralized storage solutions, to safeguard the signed exit messages. Moreover, exploring decentralized mechanisms for managing these messages, such as distributed ledger technologies or smart contracts with multi-signature capabilities, could further align with the foundational principles of blockchain and Ethereum's decentralized nature.

As an solution, Appendix A proposes several novel arrangements (methods) of existing cryptographic principles to trustlessly permit the the LST protocol to force-exit a minipool without the NO disclosing the validator signing key to any single member in the LST protocol.

4.2.2. EIP-7002: Execution Layer Triggerable Exits (Enshrined)

However a more elegant enshrined solution for forced exits is envisioned as Ethereum Improvement Proposal, EIP-7002 “Execution Layer Triggerable Exits.”¹⁵ EIP-7002 is designed to allow withdrawal credentials, both owned by Externally Owned Accounts (EOAs) and smart contracts (e.g. Megapool contracts), to trustlessly control the destiny of staked ETH. This is achieved by enabling exits that are triggerable by `0x01` withdrawal credentials. This improvement proposes a mechanism that would integrate exit operations directly into the blockchain's block structure, extending block and block header fields to include these exit operations . Specifically, it requires the block body to append a list of exit operations starting from a specific block, referred to as the `FORK_BLOCK`. Similarly, the block header must include a new `exits_root` field, representing the trie root committing to the list of exits within the block body. The validation and processing of blocks will see additional rules to ensure the legitimacy of these exit operations.

EIP-7002 is already proposed and under consideration¹⁶ for tentatively considered for inclusion in for Praelectra¹⁷. The Ethereum community has discussed several EIPs for inclusion in the upcoming Electra upgrade, with EIP-7002 being highlighted for its potential to enhance control with triggered exits, improving security in cases of key loss and enabling more trustless staking pool designs on Ethereum. However, it's important to note that the final decision on which EIPs will be included in the Electra upgrade will be determined through community consensus and further scrutiny within the Ethereum development community.

¹⁵ Detailed information on EIP-7002 and its mechanisms can be found on the official EIP page <https://eips.ethereum.org/EIPS/eip-7002>

¹⁶ <https://www.galaxy.com/insights/research/ethereum-all-core-developers-consensus-call-128/>

¹⁷ *Praelectra* is the name of the next Ethereum upgrade after *Dancun*.

With the ability for EL triggered forced exit, the establishment of Megapool contracts that allow for the aggregation of a NO's node deposit into a single contract, and the fact that a lottery block is such a rare MEV event, LST protocols like Rocket Pool can use the megapooled NO deposit to support multiple validators based on the same aggregate amount of NO deposit. This is because if a NO steals the MEV from any one validator (minipool) in the megapool, the LST protocol could force exit all the malicious NO's validators in the Megapool. The aggregate NO deposit would be sufficient to insure even a huge lottery block of 99.997% of value.

5. Appendix A: A compendium of forced-exit designs using Shamir sharing and verifiably encrypted signatures (VES) voluntarily signed exit messages.

This appendix will outline some approaches for using Shamir sharing and verifiably encrypted signatures (VES) principles to enable Rocket Pool to implement forced exits using voluntarily signed exit messages. We present various technical designs to permit the Rocket Pool (RP) oracle DAO (oDAO) to force-exit any malicious or abandoned validator without the node operator disclosing their signing key to any oDAO member. Since this method uses established cryptographic primitives, development could proceed rapidly using existing software code libraries.

The methods are detailed to aid the protocol's future decisions about which strategy to adopt if voluntarily signed exit messages are needed as a protectant. The methods proposed in this appendix all accomplish the design goal of enabling the oDAO to verify that each oDAO member received sufficient knowledge from the node operator to force-exit a malicious or underperforming minipool from the beacon chain without disclosing to any member the validator's signing key.

Symbol Terminology

The following is a list of symbols used in this report. We have used commonly accepted designations for the defined terms where possible. Where needed in the nomenclature, we have used slightly different representations to add clarity or distinction.

σ	signature of a validator (BLS)
σ_v	exit signature of a validator (BLS)
ω	verifiably encrypted signature (BLS)
key	asymmetric secret encryption key derived using the public wallet addresses of the sender and receiver (ECDSA)
$E_{Key}(msg)$	encrypted ciphertext
m	number of current oDAO members
msg	message
NO	node operator
oDAO	oracle DAO member
pk_{NO}	private key of the node wallet (ECDSA)
pk_{oDAOi}	private key of an individual oDAO member's node wallet (ECDSA)

$pk_{[oDAO]}$	private key of a collective oDAO address generated by MPC (BLS)
PK	public key
PK_{NO}	public key of the NO (ECDSA)
PK_{oDAO}	public key for an individual oDAO member (ECDSA)
$PK_{[oDAO]}$	public key for a collective oDAO address generated by MPC (BLS)
PK_v	public key of a validator (BLS)
PK_{Si}	public key of a Shamir share
S_i	secret Shamir share
sk_v	signing key of an Ethereum validator (BLS)
t	threshold number of Shamir shares needed for reconstitution

Available Inputs

It is envisioned that the following activities will occur after a NO has commanded `node deposit` and has submitted a minimum 1 ETH node deposit to the beacon chain contract to secure a validator address.

- A. A node operator (NO) obtains its Boneh–Lynn–Shacha (BLS) signing key (sk_v) of their validator (also referred to as a minipool in RP vernacular) using the current RP smart stack software.

Example:¹⁸

```
signingKey 2f967c6f9bc9b255c4a7870148b564aaa6b83ddc785b10a59ef0d4af33850f0f
publicKey
03f5faf382a87f0b6fec9c644785495c9eb8965cef77ac4e3e34ab6fc4f0b911b3acb3a41d5405f
26038e584ec9ced80c12f66b2de11819deb915ecf383b5e8f
```

- B. A node operator (NO) obtains its Ethereum node wallet address and its private key (pk_{NO}) using the current RP smart stack software.

Example:¹⁹

```
NO's private key:
45900423816161243849986077981872175393598410351473115919275307430034883597204
NO's public key:
(43439843471715603472918247849000927202021456516716702776655267700482184755287L
,
104570406429338741136020977889914071891342846931366122726360169964764420134508L
)
```

- C. The smart stack published a list of oDAO node wallet addresses.

¹⁸ https://asecuritysite.com/signatures/js_bls

¹⁹ <https://asecuritysite.com/ecc/ecdh2>

Example:²⁰ .

```
Member ID:      rocketpool-1
URL:            https://rocketpool.net
Node address:   0x2354628919e1D53D2a69cF700Cc53C4093977B94
Joined at:      21 Jul 22 00:03 UTC
Last proposal:  ---
RPL bond amount: 1750.000000
Unbonded minipools: 0
```

Common Methods

Secret Sharing of the Shares

Multiple methods discussed in this paper necessitate a secure method to transmit and encrypted ciphertext from the node operator to an individual member of the oDAO ($oDAO_i$). The following describes an Integrated Encryption Scheme (IES) that permits secure communication between the NO and any $oDAO_i$ member.

- S-1. The NO derives the public key (PK)²¹ of each oDAO member from the oDAO member's node wallet address²² using any signed transaction from each oDAO member's wallet.²³ [Alternatively, the PK of the oDAO could be published openly or included in the smart node stack.]

- S-2. The NO creates an encryption key (key) for symmetric data encryption and decryption.

The elliptic curve cryptography (ECC) schema²⁴ does not directly provide an encryption method; instead, an Integrated Encryption Scheme (IES) is needed that allows for the NO to generate a symmetrical encryption key. IES²⁵ is a hybrid encryption scheme that enables the NO to generate a symmetric encryption key (Key) based on the oDAO member's public key (PK_{oDAO_i}) and the NO's private node wallet key (pk_{NO}).

The NO computes the shared encryption $key = pk_{NO} \cdot PK_{oDAO_i}$

Cryptography speaking, this is the same as $key = pk_{NO} \cdot pk_{oDAO_i} \cdot G$ where G is the generator point. The generator point G in the `secp256k1` curve used in Ethereum is a known constant:

Gx = 0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798

Gy = 0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8

²⁰ <https://asecuritysite.com/ecc/ecdh2>

²¹ [Crypto Magic: Recovering Alice's Public Key From An ECDSA Signature](#)

²² An EL node address is a 42-character hexadecimal address derived from the last 20 bytes of the public key controlling the account with 0x appended in front.

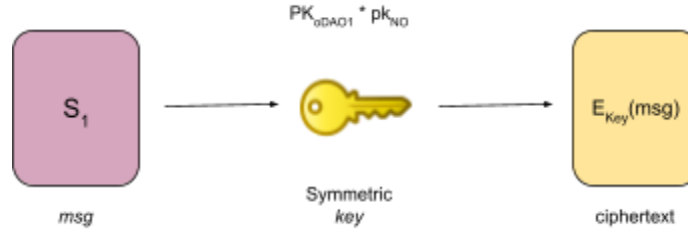
²³ [Recovering a public key from an ECDSA signature in Python](#)

²⁴ Ethereum uses an Elliptic Curve Digital Signature Algorithm (ECDSA) which is a subtype of ECC.

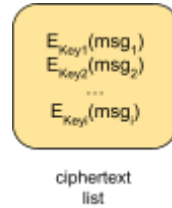
²⁵ [Integrated Encryption Scheme \(IES\) - Elliptic Curve](#)

- S-3. The NO then forms a message (msg) whose contents are the oDAO member's secret Shamir share (e.g., S_i for oDAO member 1) and encrypts it using the shared Key .

The encrypted ciphertext is represented as $E_{Key}(msg)$. The NO repeats this for each Shamir share (S_i) using the unique shared key (key_i) for each oDAO member.²⁶



- S-4. The NO then publishes the ciphertext list of the encrypted secret Shamir shares [$E_{Key1}(msg_1)$, $E_{Key2}(msg_2)$, ..., $E_{Keym}(msg_m)$] to the oDAO members. This can be done on-chain or off-chain. An ideal platform for publishing these ciphertexts is the InterPlanetary File System (IPFS) in a method similar to how oDAO members already publish Merkle trees for reward claims.



- S-5. Each oDAO member derives the public key (PK_{NO}) for the NO using the NO's node wallet address using any signed transaction from the NO wallet. [Alternatively, the NO could attach a copy of their public key PK_{NO} to the encrypted list when they publish the ciphertext.]
- S-6. The NO creates an encryption key (key) for symmetric data encryption and decryption.

Each oDAO member computes a copy of the encryption $Key = pk_{oDAOi} \cdot PK_{NO}$ using their private node wallet key (pk_{oDAOi}) and the NO's public key (PK_{NO}).

²⁶ <https://www.kerryveenstra.com/cryptosystem.html>

- S-7. Each oDAO member attempts to decrypt the list of encrypted messages. Only one of the messages will decrypt using the oDAO member's encryption *key*. The decrypted message will be the Shamir share intended for that specific oDAO member (e.g., S_1).



Commentary by Poupas:

This method is clever and works. However, I'm not very comfortable with using the NO wallet key, whose primary purpose is to sign Ethereum transitions (i.e., it's a "signing key"), for a Diffie-Hellman key exchange also (ECIES). Usually, cryptographic protocols tend to avoid "dual-use" keys, which could create unforeseen vulnerabilities.²⁷ I'm not asserting that this combination, in particular, is insecure, but it's a bit unorthodox. Again: it's probably fine. Also, IES schemes tend to create an ephemeral key to have some forward secrecy²⁸ characteristics. By using two static keys (the NO wallet key and each oDAO member key), we lose FS. If, at some point, an oDAO member wallet key was compromised, this could result in the compromise of all previous encrypted key shares. To solve this, we usually use ephemeral keys. So a key compromise would only compromise a subset of encrypted messages.

IMHO, an alternate scheme could be something like this:

- Each oDAO member publishes an ephemeral X25519 key daily/week to a contract - this implicitly authenticates the oDAO ephemeral keys (since Ethereum transactions are signed).
- Key shares are encrypted using ChaCha20+Poly1305 with an encryption key derived from running Diffie-Hellman with an ephemeral X25519 ephemeral NO key and each oDAO member ephemeral key (described above)²⁹.

If the share is submitted by the NO on-chain, further authentication is not needed.

If submitted off-chain, then the encrypted message would have to be signed with the NO wallet key (this is ok, as the key is a signing key).

oDAO Share Management

²⁷See <https://crypto.stackexchange.com/a/12138>, and <https://security.stackexchange.com/a/8563>

²⁸https://en.wikipedia.org/wiki/Forward_secrecy

²⁹ Example of an audited and ready-to-use library function:

https://libsodium.gitbook.io/doc/public-key_cryptography/sealed_boxes (Note: this link talks "about anonymously send messages", but, in this case, messages are authenticated with the NO wallet key - either by submitting it on-chain or by attaching an explicit ECDSA signature to it.)

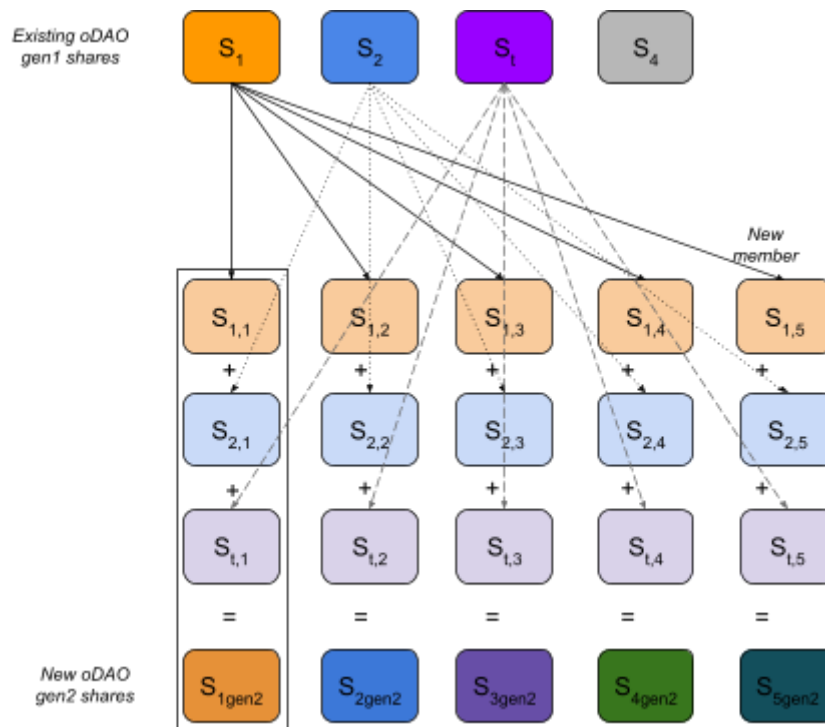
The question arises on how the oDAO can manage their Shamir shares when a member is added or removed from the oracle group. Fortunately, combining Shamir's sharing with BLS cryptography can invoke another synergistic property.

*Resharing*³⁰ allows a threshold (t) of existing oDAO members to add new members by reissuing newly generated Shamir shares to all the current (new and remaining) oDAO members. Resharing also allows changing the required threshold (t). This is helpful to keep the same percentage to reach quorum (e.g., 51%), even if oDAO members join or leave. Further, periodic key rotation and resharing at some fixed interval (e.g., every year) could be used as a proactive security measure to invalidate older shares periodically. This is beneficial if one or more of the oDAO's set of Shamir secrets has been compromised without detection.

Resharing can also be used as a *rekey* method. Rekeying would invalidate all the shares of any member who has left the oDAO. When a member of the oDAO resigns or is kicked by other members of the oDAO, there is a loss of entropy of the secret message. An added benefit of resharing is that if the remaining oDAO members delete their first-generation shares such that a threshold is no longer possible, then any first-generation secret shares held by members who left the oDAO will become useless.

Creating New Threshold Shares:

- R-1. An oDAO member splits their existing (generation 1) share for each validator into m number of new sub-shares.



³⁰ [Non-interactive distributed key generation and key resharing](#) Jens Groth 2021

- R-2. At least a threshold (t) of oDAO members need to perform the Shamir sub-sharing of their first-generation shares and transmit a copy of a sub-share to each of the m oDAO members
- R-3. Each oDAO member will need to aggregate all of the sub-shares received from a t threshold of the existing oDAO members into a second-generation Shamir share.

Note: If a $m - t$ number of oDAO original members destroy their first-generation Shamir shares, then any remaining first-generation shares are also invalidated. It will not be possible to reconstruct the secret message using first-generation shares. Still, it will be possible to reconstitute the secret message using a t threshold of the second-generation shares.

Method 1: Shamir secret sharing of a verified signed VoluntaryExit message signature.

Create a VoluntaryExit Message:

- 1-1. A node operator (NO) generates a valid signature (σ_v) on a VoluntaryExit message using their validator's signing key (sk_v). VoluntaryExit is signed with the validator's usual signing key. RP could decide on a predetermined epoch number to use, thus standardizing the exit message and negating the need to contain the message in any transmittal.

Example: Signed VoluntaryExit message.³¹

```
{
  "message": {
    "epoch": "1",
    "validator_index": "1"
  },
  "signature":
  "0x1b66ac1fb663c9bc59509846d6ec05345bd908eda73e670af888da41af171505cc411d6125
  2fb6cb3fa0017b679f8bb2305b26a285fa2737f175668d0dff91cc1b66ac1fb663c9bc5950984
  6d6ec05345bd908eda73e670af888da41af171505"
}
```

Create Threshold Shares:

- 1-2. A node operator (NO) splits the VoluntaryExit message signature (σ_v) locally on their node into m number of secret Shamir shares³² (S_1, S_2, \dots, S_m), where m = the current number of oDAO members. The NO will set the threshold (t) such that $t = m/2 + 1$.

In usual (t, m) secret sharing schemes, a secret is split into m parts so that any t out of m parts reconstructs the original secret. This method's secret message is a unique exit signature (σ_v) that only endorses the exit message specific to a single validator (validator_index). The NO acts as a central dealer who creates and distributes the

³¹ <https://ethereum.github.io/beacon-APIs/#/Beacon/submitPoolVoluntaryExit>

³² <https://www.zkdocs.com/docs/zkdocs/protocol-primitives/shamir/>

secret Shamir shares. There is no need for multi-party computation (MPC) as the NO is already trusted to know the intact σ_v .

Example:³³ Exit signature (σ_v) as the secret message split into “any 2 of 3” Shamir shares.

```
Secret message:
0x1b66ac1fb663c9bc59509846d6ec05345bd908eda73e670af888da41af171505cc411d61252
fb6cb3fa0017b679f8bb2305b26a285fa2737f175668d0dff91cc1b66ac1fb663c9bc59509846
d6ec05345bd908eda73e670af888da41af171505
Policy. Any 2 from 3

Shares:
oDAO1:
000QzxxMzjuWZeUioVGKc323orx8mzeGIDHMIm0Vz+/9JRJUyr/eH9gdqCu+vhS/titlfHv+OdaPT
fD4RsA9gtAhg0fhRphJcwViZC1YKVK4cwEPce7VGmGF/Sj35fK7qZv5mFnmRjEdaQxErKX3b1vF/k
mbV5JxxfiMI1EXPRUkbjP3d4/DAPtos2/7/cGj/B8S3s7d9IwpeK5poxJn9PjPCl3vgECAXo2NbtP
tzFCvD0PaZCQKl9JWSKc4Y0H+dig+UeRQg==

oDAO2:
001ewW6fLyugiF2b5hlyqBHRjg4SuuBbqBzSuelFcBiDg0rf00PAHYrQc2JBZpfc3dNtSdS5F/V8j
WLGbWnF42Sa70R7vaKurqi4dckOSYXOcSW1/VyaslyomPlAKI+Lbwm0bvxxqrsLLxq4KA/Sz5ZN6WS
uxPlZTJdBG1yiJX3uRenholx8ISQQptCPvWmdTm5I2hCNxHu28RUf37QvxKDPcMLMVoP/aDV6uOa9
p5qv0P6oqaB9dT8cYlQ0zyu/B2DMalmsUg==

oDAO3:
002u0/nrTBv7nZRQKwN7xaVY+9ig+9g9MGvxFWW08AEASuNDAUfiG32GhrhBDxJFYdt1FyVwJZFo7
5SEEZONQb10W0DUK/cGyH2WR+n0qPxUVEh6aKkKSg7fdouYXAjaJJwidRL/19awFUjZ8iQ+esq68I
3PpVo0+akZy6IruchoZeTItq5VkwXqvffSssxDMyZaaxWnIE8DIH1VPwIKTX3VP4Ab4j40+QjLwCr
GmaYZTZA6fHHLJ/jL8/MvMF2BKj133vr7w==
```

Utilizing a threshold signature scheme allows for the validator’s exit signature (σ_v) to be split into a specified number of shares (S_m) amongst a group of oDAO members with a corresponding threshold (t). The threshold establishes how many Shamir shares are required to command a validator to force-exit (a.k.a. send a signed `VoluntaryExit` message to the beacon chain). In this context, no individual oDAO member could unilaterally access the valid exit signature (σ_v), thus creating a multiple-signature process.

Secret Sharing of the Shares:

- 1-3. The NO distributes a cipher text whose message is the Shamir share of the valid exit signature (σ_v) to each of the oDAO members using the Secret Sharing of Shares method described previously. (Steps S-1 thru S-7).

Verification of the Shares

Each oDAO member needs to determine if their share (S_i) of the message exit signature (σ_v) is legitimate without reconstituting the entire signature, as that would allow the threshold member to obtain the complete exit signature, which would allow a single member of the oDAO to force-exit the validator.

³³ [Shamir Secret Sharing](#)

Using the `VoluntaryExit` message being signed and the partial shared signature (S_i), each oDAO member can generate a set of two possible partial public keys. Unfortunately, it is impossible to determine the unique public key with a single signature. An oDAO member would only be able to narrow it down to two possibilities;³⁴ however, they wouldn't know which of the two possibilities is correct.

- 1-4. A set of random oDAO members each derive the two possible partial public keys (PK_{S_i}) from their secret Shamir share (S_i) validator.
- 1-5. It's possible to recover the actual validator public key after combining each combination of partial public keys. $PK_{S_t} = PK_{S_1} + PK_{S_2} + \dots + PK_{S_t}$. However, the number of combinations is 2^m where m is the number of oDAO members. For example, if the oDAO contains 20 members, the number of possible public keys is 1,048,576 combinations - not great, not terrible.

If one of the reconstituted public keys (PK_{S_t}) matches the validator's public key, then the oDAO is assured that the shared signature is valid.

Again, it is important to note that at no point during the signature verification test was the actual signature on the `VoluntaryExit` message ever reconstituted or made known to any member of the oDAO.

Force-exiting:

If a validator is deemed to be acting maliciously or is determined to be underperforming by reaching some set validator balance level, an oDAO member can initiate a `VoluntaryExit` message on behalf of the node operator.

- 1-6. One of the oDAO members proposes a vote to exit the identified validator by forwarding their Shamir share (S_i) of the exit signature (σ_v). Each additional oDAO member who votes yes combines their share to the aggregation. Other oDAO members contribute their shares until the t threshold is met.
- 1-7. Once the threshold (t) of oDAO members has combined their shares, the exit message's original signature (σ_v) is recovered. The oDAO member who completed the threshold now possesses a valid exit signature and can append it to the `VoluntaryExit` message and submit the signed transaction on behalf of the validator to the beacon chain. Once relayed to the beacon chain, the majority of active validators will verify it, and the targeted validator will be marked for exiting by the census layer. Funds will be returned to the specified execution layer withdrawal credential and will be recoverable by the RP protocol.

It is important to note that there is never a need to reconstruct the validating key to sign a forced-exit message to the beacon chain. ODAO members can sign a `VoluntaryExit`

³⁴ The BLS12-381 used in the Ethereum consensus layer uses a cofactor of 1.

message on behalf of the validator by aggregate signing with their Shamir share (S_i), not by reconstituting the validator's private signing key (sk_i). There is no risk at this point for the oDAO members to reveal their Shamir shares as the signed `VoluntaryExit` message being reconstituted is intended to be publicly displayed to the beacon chain.

Method 2: Verifiably encrypted exit message signature under a distributed public key.

This method derives from verifiably encrypted signature security (VESS) described in “[Aggregate and Verifiably Encrypted Signatures from Bilinear Maps](#)” by Boneh, Gentry, Lynn, and Shacham. This method permits a NO to publish a valid signature (σ_v) on a `VoluntaryExit` message as ciphertext so that the exit signature is not visible. Members of the oDAO (or anyone) can verify that the exit signature contained in the encrypted message is valid. The exit signature can only be revealed by adjudication using a trusted private key of an arbitrator.³⁵ In this model, we propose that the arbitrator be a public-private key pair created by the current oDAO members via a distributed key generation ceremony and maintained as Shamir shares to allow for the addition and removal of oDAO members.

A repository containing exploratory code implementing a Verifiably-Encrypted Signature Scheme has already been created by Poupino (<https://github.com/poupas/bls-vess>)

Create an aggregate BLS public key for the oDAO:

- 2-1. Using a distributed key generation (DKG) process, the current collection of oDAO members generates a Shamir share of the BLS public-private key pair.
- 2-2. The oDAO then publishes the BLS public key ($PK_{[oDAO]}$). This public key will serve as the arbiter under which a signed exit message can be encrypted. Note: For this method to work, the oDAO public key must be on the same underlying BLS scheme's key-generation algorithm.
- 2-3. As part of the DKG process, each oDAO member will retain their secret share (S_i) of the BLS private key ($pk_{[oDAO]}$). As oDAO members are removed and added, these keys can be reissued and recalled using the method described in this paper's oDAO Share Management section.

Create a VES:

- 2-4. A NO constructs a verifiably encrypted signature (ω) on a `VoluntaryExit` message (msg) by using their validator's signing key (sk_v) and the oDAO's public key ($PK_{[oDAO]}$).
- 2-5. The NO publishes the `VoluntaryExit` message (msg) and the verifiably encrypted signature (ω) to IPFS.

Verification of the encrypted signature:

³⁵ The original paper refers to this role as the adjudicator.

- 2-6. Members of the oDAO will verify that the verifiably encrypted signature (ω) contains the valid exit signature (σ_v) by using the validator's public key (PK_v), the oDAO public key ($PK_{[oDAO]}$), a plain text copy of the VoluntaryExit message (msg), and the verifiably encrypted signature (ω). If the signature is deemed invalid, oDAO members will vote to scrub the minipool.

Force-exiting:

- 2-7. One oDAO member proposes a vote to exit the identified validator by forwarding $\mu^{S(i)}$, where $S(i)$ is their Shamir share. Note that each oDAO member only shares $\mu^{S(i)}$ and not $S(i)$. Other oDAO members contribute their shares until the t threshold is met.
- 2-8. Once the threshold (t) of oDAO members has combined each $\mu^{S(i)}$ contribution (to recover $\mu^{pk_{[oDAO]}}$) the exit message's original signature (σ_v) is recovered by: $\omega / \mu^{pk_{[oDAO]}}$. The oDAO member who completed the threshold is now in possession of a valid exit signature and can append it to the VoluntaryExit message and submit the signed transaction on behalf of the validator to the beacon chain. Once relayed to the beacon chain, the majority of active validators will verify it, and the targeted validator will be marked for exiting by the census layer. Funds will be returned to the specified execution layer withdrawal credential and will be recoverable by the RP protocol.
- 2-9. Example:³⁶ Sample run output (abridged).

```
NO's pubkey (G1): 08d1...fa35
oDAO Arbiter pubkey (G1): 0df9...03
Adjudicator pubkey (G2): 0236...ae5c
Message: Hello, World
Original signature: 094a...e068
VESig: (001e...ecb6, 0692...5642)
Recovered signature: 094a...e068
Recovered signature matches!
Recovered signature (n-of-m): 094a...e068
```

Method 3: Shamir secret sharing of a verified signing key.

Create Threshold Shares:

- 3-1. A node operator (NO) splits their BLS validator signing key³⁷ (sk_v) locally on their node into m number of secret Shamir shares³⁸ (S_1, S_2, \dots, S_m). Where m = the current number of oDAO members. The NO will set the threshold (t) such that $t = m/2 + 1$.

³⁶ <https://github.com/poupas/bls-vess>

³⁷ <https://kb.beaconcha.in/ethereum-2-keys>

³⁸ <https://www.zkdocs.com/docs/zkdocs/protocol-primitives/shamir/>

Example:³⁹

```
Signing Key:
2f967c6f9bc9b255c4a7870148b564aaa6b83ddc785b10a59ef0d4af33850f0f
Policy. Any 2 from 3

Shares:
oDAO1:
000HkHUhRbxke+VpPjgYXXGhiA7FEAe+08HOo8URh6Ni0VgZD9u3eWMVihvJQq7H3s5Kd4I5nGRph
O/1UjhYDZfbdQuH74=
oDAO2:
001S1CaWSLXrBSihXuPehMMWV+1qXqlx8or2YAtW0m0TUK3oaeDhuoMDY9kPlc0peQqY0CP7cFkgZ
zAtZbt3kz2Ed/Z/E=
oDAO3:
002tGNJXPX696hn75/4+V7hTON8mbjRIgkXS/JFm8bD+B0vP7w45a/qM4GZ5Ez2kakWSvOMG8BB76Y
BB1PT5hqiGivKM760=
```

In usual (t, m) secret sharing schemes, a secret is split into m parts so that any t out of m parts reconstruct the original secret. This method's secret is the validator's signing key (sk_i), and the NO acts as a central dealer who creates and distributes the secret Shamir shares. There is no need for multi-party computation (MPC) as the NO is already trusted to know sk_i .

The method described above could also use verifiable secret sharing (VSS).⁴⁰ However, as described later, we can still authenticate the validity of the shared data by use of BLS signature validation.

Utilizing a threshold signature scheme allows for the validator's signing key (sk_i) to be split into a specified number of shares (S_m) amongst a group of oDAO members with a corresponding threshold (t) for how many of these share signatures are required to command a validator to force-exit (a.k.a. send a sign a `VoluntaryExit` message to the beacon chain). In this context, no individual oDAO member would have unilateral control over a minipool, creating a trustless signature process.

Of particular note, Joao Poupino (@poupas) has coded a proof-of-concept programmed with go (<https://github.com/poupas/rocketpool-split-keys>) that demonstrates a working implementation of the above step.⁴¹

Secret Sharing of the Shares:

³⁹ [Shamir Secret Sharing](#)

⁴⁰ https://en.wikipedia.org/wiki/Verifiable_secret_sharing

⁴¹ An important distinction in the proof-of-concept code versus the method proposed is the mechanism of verifying the authenticity of the share without ever reconstituting the validating key. In Poupino's example code it verifies that a public key composed by interpolating a threshold collection of partial public keys each of which were generated from the Shamir share. Both methods accomplish the objective of determining the legitimacy of the share by verifying that an aggregation of information does not reconstitute the validator key.

- 3-2. The NO distributes a cipher text whose message is the Shamir share of the validators' signing key to each of the oDAO members using the Secret Sharing of Shares method described previously (steps S-1 thru S-7).

Verify Shares:

- 3-3. One of the oDAO members constructs a test message (e.g., "This is a test.") and signs it with their Shamir share (S_i).

A primary benefit of a BLS signature is its homomorphic properties, which means BLS signatures are friendly to aggregation. Because S_i is a Shamir share of a BLS signature, multiple oDAO signatures ($\sigma_1, \sigma_2, \dots, \sigma_m$) can be combined into one aggregate signature (σ). This allows multiple signatures to be combined, enabling messages to be signed without reconstructing the NO private validating key.⁴²

- 3-4. The partially signed message is routed to other members of the oDAO for aggregate signing.⁴³ This can be performed on or off-chain. Other oDAO members sign the message until the threshold is met.

- 3-5. Once the threshold (t) of oDAO members has signed the test message, anyone, presumably an oDAO member, can authenticate the validity of the participating Shamir shares by verifying the polynomial interpolation⁴⁴ of the aggregate signature (σ_i) against the public key of the validator (PK_v).

If the signature is valid, each participating Shamir share (S_i) tested is a verified share of the validator's legitimate signing key. Essentially, we have created a zero-knowledge (zk) proof. Each oDAO member is assured that they have a share of a valid signing for the validator. No reconstruction of the validator key is needed at any point to verify the authenticity of the share.

Alternative Method of Verification:

An alternative approach for steps 3-3 thru 3-5 is illustrated by the proof-of-concept work by Poupino. His method accomplishes the same goal of verifying that the secret message in Shamir's shares is the verified private key. This is performed by forming a public key of the Shamir share (PK_{S_i}) and comparing an aggregation of those partial public keys to the known public key of the validator. The steps are summarized below.

- A. A set of random oDAO members multiply their secret Shamir share (S_i) by G - the generator point on the BLS12-381 curve. This will create a partial public key of the validator. $PK_{S_i} = S_i \cdot G$.

⁴² [Distributed Validator Technology on Eth2](#) Mara Schmiedt & Collin Myers | February 2021

⁴³

<https://stackoverflow.com/questions/67079647/shamirs-secret-sharing-could-each-shard-individually-sign-a-transaction>

⁴⁴ <https://www.dash.org/blog/secret-sharing-and-threshold-signatures-with-bls/>

- B. By using interpolation, it's possible to recover the actual validator public key after combining a threshold (t) of partial public keys. $PK_{St} = PK_{S1} + PK_{S2} + \dots + PK_{St}$.
- C. After reconstituting the public key (PK_{St}), any member can verify that it matches the known public key of the validator (PK_v). $PK_{St} = PK_v$.

Again, it is important to note that at no point was the actual private key of the validator ever reconstituted or made known to any member of the oDAO.

Force-exiting:

If a validator is deemed to be acting maliciously or is determined to be underperforming by reaching some set validator balance level, an oDAO member can initiate a `VoluntaryExit` message on behalf of the node operator.

- 3-6. One of the oDAO members constructs a valid beacon chain `VoluntaryExit` message (e.g., "I quit.") and signs it with their Shamir share (S_i).

Example: Signed `VoluntaryExit` message.⁴⁵

```
{
  "message": {
    "epoch": "1",
    "validator_index": "1"
  },
  "signature":
  "0x1b66ac1fb663c9bc59509846d6ec05345bd908eda73e670af888da41af171505cc411d6125
  2fb6cb3fa0017b679f8bb2305b26a285fa2737f175668d0dff91cc1b66ac1fb663c9bc5950984
  6d6ec05345bd908eda73e670af888da41af171505"
}
```

- 3-7. The partially signed message is routed to other members of the oDAO for aggregate signing.⁴⁶ This can be performed on or off-chain. Other oDAO members sign the message until the t threshold is met.
- 3-8. Once the threshold (t) of oDAO members has signed the test message, it may be relayed to the beacon chain, where once verified by the majority of active validators will be marked for exiting by the census layer. Funds will be returned to the specified execution layer withdrawal credentials and will be recoverable by the RP protocol.

It is important to note that there is never a need to reconstruct the validator signing key (sk_v) to sign a forced-exit message to the beacon chain. ODAO members can sign a `VoluntaryExit` message on behalf of the validator by aggregate signing with their

⁴⁵ <https://ethereum.github.io/beacon-APIs/#/Beacon/submitPoolVoluntaryExit>

⁴⁶ <https://stackoverflow.com/questions/67079647/shamirs-secret-sharing-could-each-shard-individually-sign-a-transaction>

Shamir share (S_i), not by reconstituting the validator's private signing key (sk_i). An oDAO member never needs to reveal their Shamir share to any person.