For full disclosure, this post was coauthored by Jason Vranek and Amir Forouzani from Puffer, whose tech is supported by an EF grant. We greatly appreciate the feedback from @justindrake, @domothy, and @adiasg from the EF and @maherlatif_ from Jump Crypto Research.

TLDR

We propose a permissionless liquid staking protocol that allows the bond requirement to be lowered below 16 ETH without in-protocol changes or additional trust assumptions.

Context

Permissionless staking operations require increased capital efficiency to remain competitive with centralized staking operations. However, lowering the Node Operator (NoOp) bond to less than 16 ETH poses challenges, as validators can only exit by signing VoluntaryExit

messages (VEMs) or if their effective balance falls below 16 ETH. This creates issues in a permissionless stake pool as VEMs require knowledge of the ValidatorIndex

, which is unknown until the validator has been activated (and borrowed the pool's ETH). If the NoOp goes offline, they threaten to lose the pool up to 16-B

ETH from inactivity penalties. This is trivially solved by giving a third party (e.g., a DAO) custody of the validator key but introduces a centralization risk.

To address this problem, we propose a client-server architecture where an untrusted aggregator Agg

communicates with N

untrusted NoOps

. Together, Agg

and NoOps

contribute a bond of B = B_{Agg} + N * B_{NoOp}<16

ETH and manage a single validator key on behalf of the pool. Agg

generates an \frac{M}{N}

threshold BLS key, distributes key shares, and aggregates BLS signatures. NoOps

perform validator duties with their BLS key shares and forward their partial signatures to Agg

for aggregation.

The core idea is that the protocol cannot force Agg

or NoOps

to sign a VEM while remaining non-custodial; however, the two parties are incentivized to exit and protect their bonds if the opposite party fails to fulfill their duties. This allows a permissionless liquid staking protocol to lower their bond requirement without in-protocol changes or significant trust assumptions.

Construction

Agg

is running Intel SGX similar to this architecture.

keygen():

Inside SGX, Agg

generates and temporarily saves an \frac{M}{N}

threshold validator key. The N

key shares are securely shared with the NoOps

.

presign():

Inside SGX, after validator activation, Agg

pre-signs a VEM and then deletes the validator key.

aggregate():

Inside SGX, Agg

aggregates partial signatures and only broadcasts the result iff

the signed material will not result in a slashable offense. Before execution, this function will always attempt to run presign(),

removing the key from Agg

's custody.

exit():

Agg

submits their pre-signed VEM to the consensus layer.

NoOps

do not have any hardware assumptions.

validate(): NoOps

run their execution and consensus clients, signing material with their BLS key share.

forward(): NoOps

submit their partial signatures to Agg

.

exit():

NoOps

submit partially-signed VEMs on-chain for aggregation, bypassing Agg

.

Incentives

To properly align incentives, it is expected that B_{Agg}= B_{NoOps} = N*B_{NoOp}

. If either party fails in duty, the validator instance will fail to produce attestations, threatening both bonds. For example, if Agg

goes offline or decides to censor, NoOps

are incentivized to exit()

to protect B_{NoOps}

. Similarly, Agg

can exit()

to protect B_{Agg}

if NoOps

underperform or collude. Either case protects the pool’s ETH from inactivity penalties. While this may result in forgone revenue, this tradeoff directly results from building a non-custodial protocol.

Regarding SGX

The aggregate validator key is in SGX’s custody from the initial keygen()

until the first aggregate()

. If Agg

delays running aggregate()

after the validator is activated (presumably to try and extract the key), they are effectively censoring the NoOps who are incentivized to exit().

SGX is used as a strict security enhancement. Assuming an adversarial Agg

extracts the validator key, B_{Agg}

discourages self-slashing. If Agg

decides to censor and publish their own signatures, NoOps

can easily detect this and exit().

Pros

- actionable:

This mechanism provides a solution for inactivity penalties before0x03 withdrawal credentials are implemented.

- slash-resistant:

Incentives + \frac{M}{N}

threshold signature + SGX-secured Agg

provide excellent slash protection.

- efficient architecture:

O(N)

client-server architecture is simple, low latency, scalable, and requires minimal changes to the NoOp tech stack.

- capital efficiency:

Reduced bond requirement (\frac{32}{B_{Agg} + B_{NoOps}}

x leverage) helps solo validator viability and grow the pool.

- rewards sharing:

\frac{M}{N}

threshold signature allows for more reliable execution rewards sharing, enabling out-of-protocol MEV-Smoothing.

Cons

- inactivity penalty risk:

If both Agg

and NoOps

are offline or collude to deny service, the pool will lose ETH; however, the loss of B_{Agg} + B_{NoOps}

and the opportunity cost of withholding validator work discourages this.

- collusion risk:

If both Agg

and NoOps

collude, they can avoid execution rewards sharing (but cannot steal consensus rewards).

- switching costs:

If Agg

goes offline, NoOps

incur the costs of exiting and rejoining the pool.

- withdrawal friction:

A NoOp will have increased friction when withdrawing if peers disagree.

- lazy nodes:

The threshold signature tolerates up to N-M

failures but also up to N-M

free riders (which can be addressed via tokenomics at the protocol level).

- memetics:

SGX does not have a good reputation.