Hord Liquid Staking SSV Integration
Grant Application

Introduction

Hord.fi was created by the DcentraLab team.

DcentraLab was founded in 2017 with the goal of creating an ecosystem of on-chain products designed to accelerate blockchain mass adoption. Dcentralab is headquartered in Tel Aviv and Belgrade, with additional team members worldwide with over 50 employees.

Since DecentraLab's launch, we've developed different products that produce value, increase transparency, and lead the decentralized revolution. These products include ChainPort, TokensFarm, Hord, and 2Key, with additional products in development.

The Product

Hord's Ethereum staking platform is a highly scalable and fault tolerant infrastructure for running validators and ssv operators, as well as a service (available on app.hord.fi) that allows users to stake ETH in any amount and immediately start earning rewards for participating in the process of validating Ethereum blocks.

With Hord.fi, users can stake their ETH via a secure smart contract and start generating stable APYs, both in native ETH and in $HORD. In exchange for staking ETH, users receive hETH, a liquid ERC20 based token representing the amount of ETH they have staked combined with rewards (attestation, block proposals, auto-compounding and MEV rewards from all validators associated with the pool stream back to the pool).

Our core benefits:

- **No Minimums -** Unlike solo staking or other staking platforms, Hord has no minimums. Users can stake any amount of ETH they'd like and start generating rewards.
- **Easy to Stake -** Hord offers a more hands-off approach and handles the technical details, such as setting up a node on behalf of the user.
- **Compounding Rewards -** Hord offers the most competitive APRs by combining staking rewards, with compounding MEV rewards (Maximum Extractable Value) into the ETH staking pool.

- **Low Fees** - The Hord ETH Staking platform will include low fees to compete with other staking pools.
- **Multi-Chain App** - Hord will enable to onboard with wrapped ETH on other chains as well, combining the liquid staking into a multi-chain available opportunity.
- We aim to combine our own strong infra with SSV decentralization of validators to offer a well balanced approach for highly reliable, scalable and secure validators to work for the hETH holders.

Proposal Details

Outline:

1. **We will run high quality SSV operators for the benefit of the entire SSV network**: We will launch 2 highly scalable, highly redundant kubernetes clusters on AWS in 2 different regions (US, Europe), each will be able to run operators for the SSV network.
2. **We will integrate SSV into app.hord.fi**: We will integrate a distributed approach so that a significant portion of validators launched for the Hord liquid ETH Staking product will be run via SSV network.

Technical Overview

Contracts overview:

The Hord Eth Staking component is built from two main contracts, both of which have been audited. These contracts are upgradable, governed by the multi-sig congress contract DAO of cold storage and vault signers.The Governance component is designed to manage a congress that is able to vote on-chain to invoke some of the restricted functions.

The Hord Staking component allows users to stake their Hord tokens and receive benefits and utility in return, such as immediate withdrawal of ETH when requested by the user.

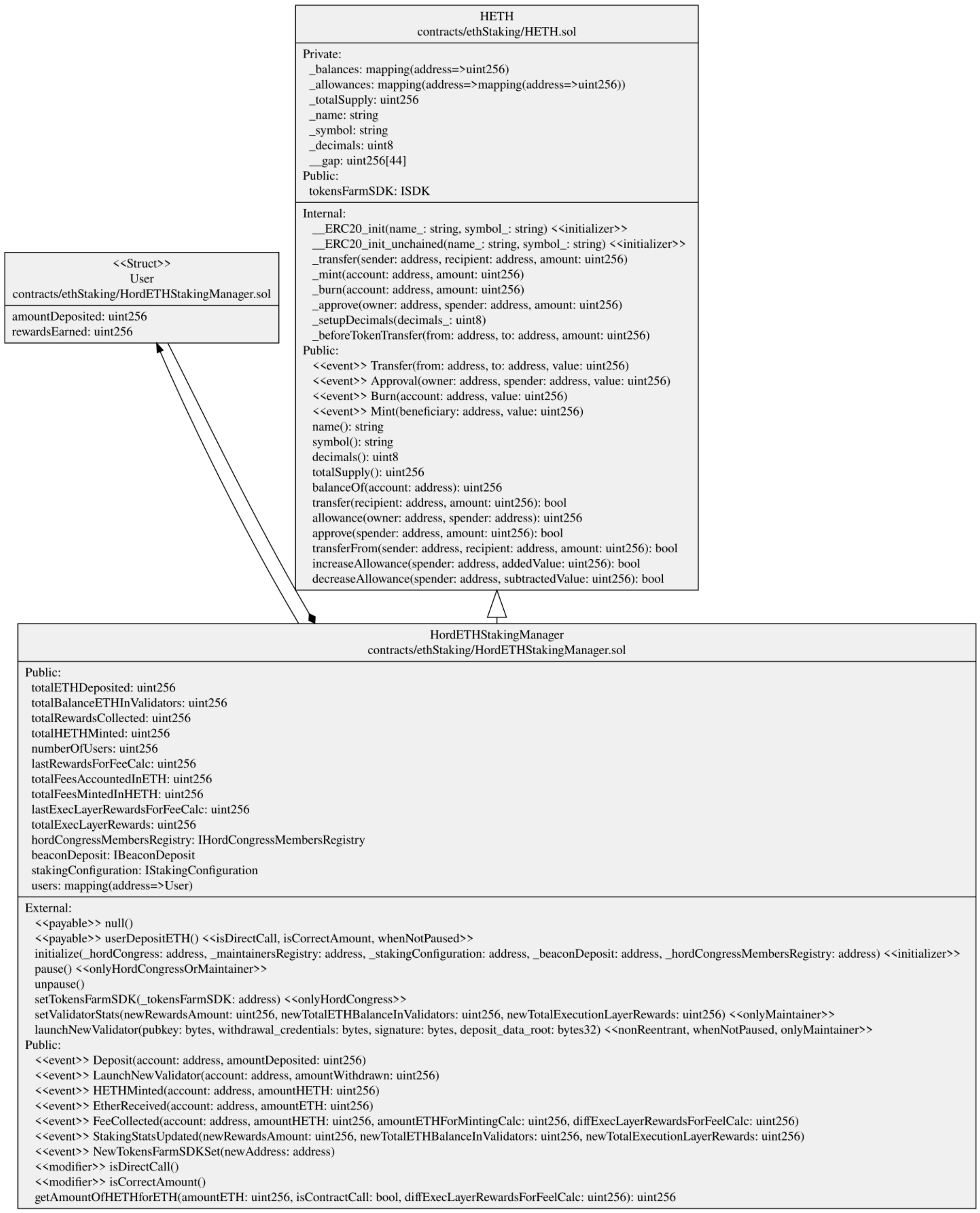
hETH contract:
This ERC20 token represents a user's share in the ETH staking pool. It is the liquid token that is minted to a user who has made a deposit of ETH to the staking pool.

HordEthStakingManager contract:
This contract serves as the Eth staking pool, providing all the necessary functionality to manage it. Users can deposit into the contract by calling the *userDepositETH* function, after which the contract will mint them HETH tokens based on the amount they deposited and the current HETH price (determined by ratio of staked ETH+rewards to the minted hETH).

There are two restricted roles involved in this contract: the Hord congress and the Maintainer. The Maintainer, which is the Hord Backend, is responsible for obtaining off-chain data, such as validator stats from the beacochain api, and feeding it into the stakingManager contract. The Maintainer is also responsible for triggering the *launchNewValidator* function to add a new validator when the stakingManager has enough ETH to create one.

Once triggered by the maintainer, and providing there is enough ETH deposited, the EthStakingManager contract directly deposits the 32 ETH required to the official ETH validator staking contract on Ethereum to trigger the launch of the new validator. This is accomplished by the backend calling the launchNewValidator() function with parameters for the next available validator provided by the highly fault tolerant and scalable validator infrastructure running on the Hord backend. The infrastructure always maintains a large buffer of fresh validators ready to be activated once the required ETH has been deposited onchain.

UML Diagram:



```
HETH
contracts/ethStaking/HETH.sol

Private:
  _balances: mapping(address=>uint256)
  _allowances: mapping(address=>mapping(address=>uint256))
  _totalSupply: uint256
  _name: string
  _symbol: string
  _decimals: uint8
  __gap: uint256[44]
Public:
  tokensFarmSDK: ISDK

Internal:
   __ERC20_init(name_: string, symbol_: string) <<initializer>>
   __ERC20_init_unchained(name_: string, symbol_: string) <<initializer>>
   _transfer(sender: address, recipient: address, amount: uint256)
   _mint(account: address, amount: uint256)
   _burn(account: address, amount: uint256)
   _approve(owner: address, spender: address, amount: uint256)
   _setupDecimals(decimals_: uint8)
   _beforeTokenTransfer(from: address, to: address, amount: uint256)
Public:
   <<event>> Transfer(from: address, to: address, value: uint256)
   <<event>> Approval(owner: address, spender: address, value: uint256)
   <<event>> Burn(account: address, value: uint256)
   <<event>> Mint(beneficiary: address, value: uint256)
   name(): string
   symbol(): string
   decimals(): uint8
   totalSupply(): uint256
   balanceOf(account: address): uint256
   transfer(recipient: address, amount: uint256): bool
   allowance(owner: address, spender: address): uint256
   approve(spender: address, amount: uint256): bool
   transferFrom(sender: address, recipient: address, amount: uint256): bool
   increaseAllowance(spender: address, addedValue: uint256): bool
   decreaseAllowance(spender: address, subtractedValue: uint256): bool
```

```
<<Struct>>
User
contracts/ethStaking/HordETHStakingManager.sol

amountDeposited: uint256
rewardsEarned: uint256
```

```
HordETHStakingManager
contracts/ethStaking/HordETHStakingManager.sol

Public:
  totalETHDeposited: uint256
  totalBalanceETHInValidators: uint256
  totalRewardsCollected: uint256
  totalHETHMinted: uint256
  numberOfUsers: uint256
  lastRewardsForFeeCalc: uint256
  totalFeesAccountedInETH: uint256
  totalFeesMintedInHETH: uint256
  lastExecLayerRewardsForFeeCalc: uint256
  totalExecLayerRewards: uint256
  hordCongressMembersRegistry: IHordCongressMembersRegistry
  beaconDeposit: IBeaconDeposit
  stakingConfiguration: IStakingConfiguration
  users: mapping(address=>User)

External:
   <<payable>> null()
   <<payable>> userDepositETH() <<isDirectCall, isCorrectAmount, whenNotPaused>>
   initialize(_hordCongress: address, _maintainersRegistry: address, _stakingConfiguration: address, _beaconDeposit: address, _hordCongressMembersRegistry: address) <<initializer>>
   pause() <<onlyHordCongressOrMaintainer>>
   unpause()
   setTokensFarmSDK(_tokensFarmSDK: address) <<onlyHordCongress>>
   setValidatorStats(newRewardsAmount: uint256, newTotalETHBalanceInValidators: uint256, newTotalExecutionLayerRewards: uint256) <<onlyMaintainer>>
   launchNewValidator(pubkey: bytes, withdrawal_credentials: bytes, signature: bytes, deposit_data_root: bytes32) <<nonReentrant, whenNotPaused, onlyMaintainer>>
Public:
   <<event>> Deposit(account: address, amountDeposited: uint256)
   <<event>> LaunchNewValidator(account: address, amountWithdrawn: uint256)
   <<event>> HETHMinted(account: address, amountHETH: uint256)
   <<event>> EtherReceived(account: address, amountETH: uint256)
   <<event>> FeeCollected(account: address, amountHETH: uint256, amountETHForMintingCalc: uint256, diffExecLayerRewardsForFeelCalc: uint256)
   <<event>> StakingStatsUpdated(newRewardsAmount: uint256, newTotalETHBalanceInValidators: uint256, newTotalExecutionLayerRewards: uint256)
   <<event>> NewTokensFarmSDKSet(newAddress: address)
   <<modifier>> isDirectCall()
   <<modifier>> isCorrectAmount()
   getAmountOfHETHforETH(amountETH: uint256, isContractCall: bool, diffExecLayerRewardsForFeelCalc: uint256): uint256
```

Validator Setup

Currently, we are running our own validators, but we will branch out to a dynamically configurable ratio of choosing to run an SSV-based validator each X validators that are launched. E.g. start out with 1 of every 3 launched validators as an SSV validator.

Validator Launch flow - Current:

1. Backend pre-creates hundreds of validators.
2. Backend maintains a highly scalable cluster of running validators, with a large buffer of running validators which are not activated yet (ready for immediate activation).
3. As soon as 32 new ETH in aggregate are deposited into the **HordETHStakingManager** Contract, the backend picks up on this, selects the next validator available to activate, and triggers the contract to deposit the ETH for activating that validator.

Validator Launch flow - Proposed:

1. **DWS** Backend will now maintain also the 2 geographically distributed operators in SSV network, and register them for other clients on the network to use with competitive fees.
   a. This means we will run operators that other clients like lido and rocketpool can use
   b. One operator can facilitate multiple validators, so enough to run high sla pods for this on 2 geo zones.
   c. Hord validators will also always assign 2 operator slots to DWS operators
2. Backend will use an updated registry of the most reputable operators in SSV network utilizing different cloud providers than our own and in different geographies than our own infra to maximize the decentralization effect of utilizing SSV within app.hord.fi
3. **DWS**: We will pre-generate a working buffer of validators and use the SSV tooling to split their keys to shares, ready for registration on the ssv network.
   a. We will pre create 100 validators, and store all of their data encrypted on aws ssm.
   b. and then feed in their depositfiles/config json into a new script which will, instead of readying them to be uploaded to the k8s as actual running validators, will instead use the SSV tooling to split their keys to ssv keyshares,
   c. On the db, we will have a new table for ssv keyshares, each of those validators will be marked to be used only on ssv and not to be uploaded to the actual pod cluster, and also there will be a link table between the validtor and the ssv keyshares table
   d. The validator table will keep track of all validators, with special flags stating which are run in house in our k8s cluster and which are on ssv, aswell as the region the validator will be running on.
   e. Once we have 32ETH in the contract, we will fetch data of the next available validator for launch, select operators for the validator, and then generate keyshares for the selected operators.
4. **Deposit Flow: Once 32 new ETH** in aggregate are deposited into the HordETHStakingManager contract, the Hord backend will utilize a configurable ratio, where once in every X new validators, the backend will opt to trigger that validator to be run on the SSV network rather than our own standalone network of validator:

    a. Hord backend decides now is time to launch ssv validator, so would call:
        i. DWS: activateNewValidatorForSSV
    b. DWS: activateNewValidatorForSSV
        i. Get the next non activated ssv validator
        ii. Select our 2 operators
        iii. Use SSV apis to select best 2 operators with high reputation and traction (and also try not to rechoose the same ones over and over again for resilience.
        iv. Generate the keyshares for this validator to the selected operators.
        v. Save the encrypted keyshares in SSM.
        vi. Retrieve the Cluster Snapshot Data for that cluster of operators (step 3 in setup docs)
        vii. Return all of the above in the api call
    c. Hord backend will use this returned data to call the HordETHStakingManager.*launchNewValidatorWithSSV*
    d. The **HordETHStakingManager** contract:
        i. will expose a new function: * *launchNewValidatorWithSSV* that will be callable by the maintainer, with params of the validator public key, list of operator ids, list of encrypted key shares, the cluster snapshot data
            1. This function registers the new validator to the operators on the ssv network on chain, then does a regular deposit of the 32 eth to activate the validator on eth network.
        ii. If optionally the amount to pay upfront in SSV - or pay as you go, but in any case, the contract needs budget of SSV on it to pay the operators.

5. **Maintenance, monitoring and alerts mode:**
    a. Keep track of running validators via ssv dashboard
    b. Alert in case an ssv validator goes down or one of the operators goes down
    c. Exit a validator in case operator becomes faulty or needs to be replaced
    d. Monitor validator costs and SSV funds
6. **Exit flow:**
    a. Once the contract balance is not enough to cover a user's request to withdraw, the backend will select a validator to liquidate.
    b. The backend will select the validator with the lowest amount of rewards.
    c. Once a validator has been selected for exit, the validator will be manually exited.

Dapp:
The dapp is live on mainnet: app.hord.fi
We were focused on user experience to create the easiest way for users to get into the Eth Staking with only basic knowledge. All stats needed are presented there including APR and hETH price. hETH price is designed to only go up relative to ETH because it represents the ETH rewards of the validators as well which aggregate without minting hETH on the other side.

Hord Backend:

Written in Python and running on AWS, the backend (BE) is responsible for obtaining stats from our live validators and feeding that data into the staking manager contract. This data is used to calculate the HETH price in real-time.

The BE is also responsible for invoking the *launchNewValidator* and *launchNewValidatorWithSSV* functions once the 32 ETH requirement is met. Furthermore, it monitors on-chain data to ensure the safety and functionality of the entire system. If necessary, it will take corrective action or notify the Dcentralab team of any system misbehavior.

DcentraLab Devops Infrastructure:
Utilizing python, kubernetes, AWS, terraform, prometheus, kibana, elasticsearch, fluentD. Empowers a highly scalable, redundant and fault tolerant cluster of Ethereum validators and SSV operators, as well as underlying APIs to serve stats and logs on the validators and operators.

Operator Selection

We will be running two separate kubernetes clusters, in two different regions, and each will act as a separate operator provider. So for each of our SSV validators run for hord.fi , 2 will be our own operators, and the other will be chosen by the SSV operator reputation/stats API.

Withdrawals

Describe how you plan to handle user withdrawals (pre and post withdrawals are enabled).

Pre withdrawals:
Users are able to swap their hETH back into ETH on a dex. (UniswapV3 pool) - currently available on mainnet.

Post withdrawals:
The technical spec for this has been internally approved and we're now implementing it.

When a user wants to withdraw, they will trigger the *redeemHETH* function. The user will enter a cooldown period before they can claim their ETH. We will add a buffer of ETH to the **HordETHStakingManager** (currently projected at 32 ETH), to avoid immediate liquidation of validators.

Once a user request to redeem hETH is received, if there is enough ETH on the **HordETHStakingManager** and the withdrawBuffer balance on the **HordETHStakingRedeemsManager** contracts, the claim will be available with a short cooldown and assured balance. If not enough ETH is present, our backend will pick that up, and we will be able to choose whether to top up ETH for covering the withdrawal to avoid liquidation of a validator, or to choose and signal a validator to be liquidated.

In case the validator to be liquidated is an SSV validator, backend will signal Ethereum to exit the validator, then backend will trigger a function on the **HordETHStakingManager -** *removeValidatorFromSSV* that will signal the SSV network that the validator has been exited.

Once we top up the withdraw buffer on the redeem contract, or a validator 32ETH are liquidated back to the **HordETHStakingManager,** the backend will trigger the contract to pass ETH from those funds to close the remaining open debts on the redeem contract, and once cooldown has passed and debts have been closed, those users will be able to withdraw their ETH as well.

SSV Payments

Optimally, all SSV validators will be registered via the **HordETHStakingManager,** and it will contain a working balance of SSV and an approval for the SSV contract to collect the fees. Otherwise, if possible, we will dedicate an ad hoc account/address/contract to facilitate the SSV balance and approval for fee taking by the operators/ssv network.

We are aiming for an economically feasible way to support SSV payments ongoing.. We currently pay for the validators that we run ourselves from our IT budget, and to start with, the SSV payments will be sourced from the same budget, as its payment to run the infra required for the Hord pool. Going forward, to be economically viable, the SSV budget would have to be balanced out ongoing with the fees earned by the protocol, once the fees stabilize in volume we will start to work on automatically swapping some of the fees into SSV for supporting an ongoing SSV payment budget directly on the Hord liquid staking manager contract.

Project Plan

#	Milestone	Deliverables	Est. Effort
1	*POC for running SSV operators*	● *Run SSV operators on ec2 integrated with our geth+prysm+mevboost cluster on k8s*	*3 weeks*
2	*POC for running SSV operator within a dedicated K8S Cluster*	● *Generate dedicated terraform based setup with dynamic scripting to create k8s cluster for running geth+prysm+ssv operators in a fault tolerant, scalable manner*	*3 weeks*
3	*Launch SSV Operator Cluster*	● *Setup integration ready operator Cluster for testnet*	*2 weeks*

	for testnet		
4	*Setup + Launch Devops Infra for Testnet - on DWS (DcentraLab Web Services API)*	● Setup required infra and automatically generating validators, separating to keyshares, and provide api for available next validator ● Monitor operator stats on ssv and expose via api ● Monitor validator owner payment stats and expose via api to enable alerting via webhooks on payment thresholds nearing limits etc..	*3 weeks*
5	*Setup + Launch Backend and Contracts on Testnet*	● Add required functionality on contracts to launch ssv validator and exit a validator ● Add required functionality on backend - interface with the DWS API ● Add monitors and programmatic workers to handle SSV interface via the DWS the contracts and the ssv network	*4 weeks*
6	*Integration on testnet*	● Integrate DWS, Contracts, Backend, SSV on testnet	*3 weeks*
7	*Regression on testnet*	● Full regression on testnet	*3 weeks*
8	*Deploy + Regression on staging*	● Full regression on staging ● SSV version updates fixes ● SSV mainnet updates fixing	*3 weeks*
9	*Deploy + Regression on prod*	● Full Prod regression ● Mainnet integration and fixes	*3 weeks*

Payments

Requesting $50,000 for SSV integration per the above plan.

Terms

100% SSV

Milestone Allocation

Milestone	**Amount**	**Percentage**
Testnet Integration	*$12,500*	*25%*

Mainnet Integration	*$12,500*	*25%*
25 SSV Validators on Mainnet	*$25,000*	*50%*
Total	***$50,000***	***100%***

Open Source

For security reasons our contracts code are usually closed source, as is our backend code, but we will work to externalize an API for operator stats, reputation and performance and a tutorial for setting up SSV operators on K8S, for the benefit of the SSV network.