Key Splitting Migration Guide.

Executive Summary

This document outlines a comprehensive step-by-step guide and checklist for the key splitting process. Specific implementation details may vary based on your infrastructure and the design of your target Obol DV cluster. Treat this guide as a migration blueprint, which we'll work together to adapt to your infrastructure design. Obol's engineering and customer success teams will validate your plan and support execution. This approach involves splitting an existing validator private key into multiple key shares, customized to your desired quorum and decentralization model. These shares can then be deployed across different machines using LCDVN.

Contents

Why Choose Key Splitting

Advantages

- Minimal downtime (~2 epochs)
- No need to exit validators — existing validators are upgraded in place
- Avoids exit/activation queues and consolidation steps

Risks & Mitigations

- Requires precise key management — duplicate keys across machines can lead to slashing. Obol will provide clear guidance.
- A brief downtime is necessary to ensure safe key transfer. While this results in temporary missed duties, it avoids the greater reward loss and delay associated with validator exits.
- Key shares must be securely distributed to each machine. We recommend using encrypted folders; the Obol team can assist with secure transfer best practices.

Testnet Dry Run

We'll guide you through the full process on testnet (typically Hoodi). The setup includes both vanilla validators and Obol DVs, replicating a mainnet migration scenario at scale — typically with 1,000+ validators. Testing includes:

- Key splitting validation
- Obol DV deployments
- Version upgrade simulations
- Rollback and validator exits

We'll set a target RAVER threshold — e.g., 98% average performance across 1,000 test validators over two weeks — to simulate production standards.

> 💡 *If you've previously run Obol DVs via CSM or Simple DVT, your testing flow will be shorter and focus mainly on verifying key splitting and key security practices.*

STAGE 1

Preparing the Source Validator Inventory and Target Cluster

Start by submitting the required information through the Typeform. This helps us evaluate your validator setup and define a migration strategy aligned with your infrastructure and risk profile.

Source Validator Information

Please complete the Typeform (to be provided) to share details about your current validator setup, including:

- Number of validators / amount of ETH to be migrated
- Current validator distribution (machines, clients, geography, etc.)
- How to identify your validators (e.g., withdrawal address, validator indices, pubkey list)

Target Cluster Configuration

Fill out the second Typeform (to be provided) to specify your desired target Obol DV cluster:

- Cluster name
- Operator / node count and threshold
- Withdrawal configuration:
  - Withdrawal address = Lido CL Vault
  - Fee recipient = Lido EL Vault
  - Incentive accepting address
- Geographic distribution requirements
- Preferred key management architecture
- Any additional technical or operational constraints

After reviewing your infrastructure design and validator details, our solutions architecture team will schedule a meeting with you to craft a tailored migration plan which aligns with your technical requirements and operational preferences.

STAGE 2

Define Migration Strategy

To reduce complexity and avoid human error, we recommend migrating all 1000 keys in a single operation. This approach minimizes coordination overhead and reduces the risk of mistakes during batch migrations. That said, if your infrastructure or operational preferences require a phased rollout, we fully support custom batch-based migration. Our solutions team will work with you to define a safe batching strategy tailored to your needs. No matter the decision the goals are to:

- Minimize operational and slashing risk during migration
- Reduce technical complexity
- Limit reward loss from validator downtime

After our solutions team conducts due diligence on your infrastructure, we will recommend a migration strategy that's tailored to your validator setup.

> ***Example***
>
> *Operator A runs 1,000 validators across 5 instances, with 200 validators per instance.*
> *To minimize human error, our team recommends migrating* ***all 1000 validators into a single cluster.***
> *This ensures clean boundaries and operational simplicity.*
> *That said, if you require a custom approach — for example, batching the 200 validators per instance — we'll work with you to design a migration strategy that fits your specific needs without compromising safety.*

STAGE 3

Splitting the Source Validator keys

3.1 Transferring and Preparing Validator Keystore Files

This section details the secure transfer of your existing encrypted validator keystore files and their associated plaintext passwords to the air-gapped machine, followed by their organization.

> 💡 *This is just an example as the process is subject to how the keys are stored. You can decide how to securely move the keys depending on your key storage infrastructure.*

3.2 Secure Transfer of Existing Keystore Files

This step focuses on safely moving your **keystore-N.json** (encrypted key) and **keystore-N.txt** (plaintext password) files, with an added layer of encryption for transport.

Note: The split key folder can be in different formats depending upon the current clients you use and other variables. We can modify the script of split keys so it adjusts to the way your keys folder is structured.

1. Consolidate & Verify on Source

On your source machine, gather all **keystore-N.json** and **keystore-N.txt** pairs into a single, temporary directory. Ensure each **.json** file has a matching **.txt** password file with the same base name (e.g., **keystore-0.json** and **keystore-0.txt**).

- Generate Checksums (Pre-encryption): Create checksums for all these consolidated files.
    - **Linux/macOS:**Bash

```
cd /path/to/consolidated_keys_on_source
sha256sum keystore-*.json keystore-*.txt > pre_encryption_checksums.txt
```

- Keep **pre_encryption_checksums.txt** separate; it will be used for final verification on the air-gapped machine.

2. Encrypt to Password-Protected Archive

Create a password-protected zip archive of the temporary directory containing your **keystore-*.json**, **keystore-*.txt**, and **pre_encryption_checksums.txt** files.

- Choose a **new, strong, unique password** for this archive that is different from your keystore passwords. This password will need to be manually transferred to the air-gapped machine (e.g., written down securely).

- **Linux/macOS:**Bash

```
zip -e keys_for_transfer.zip keystore-*.json keystore-*.txt pre_encryption_checksums.txt
# You will be prompted for a password.
```

- **Windows:** Use a program like 7-Zip (or similar archiving tool) to create a password-protected **.zip** or **.7z** archive of the folder containing your keys and pre_encryption_checksums.txt. Ensure AES-256 encryption is selected if available.

3. Copy Archive to Trusted Medium

Copy the **keys_for_transfer.zip** (or .7z) archive to a **new, never-before-used, trusted hardware medium** (e.g., fresh USB drive). This medium must never touch an internet-connected machine.

4. Transfer & Decrypt on Air-Gapped Machine

- Insert the trusted medium into your air-gapped machine.
- Create a temporary staging folder on the air-gapped machine.
- Copy the **keys_for_transfer.zip** archive to this staging folder.
- **Decrypt the Archive:** Use the password you set in Step 2 to extract the files.
- **Linux/macOS:**Bash

```
cd /path/to/staging_folder_on_air_gapped
unzip keys_for_transfer.zip
# You will be prompted for the archive password.
```

- **Windows:** Use your chosen archiving tool (e.g., 7-Zip) to extract the contents of the archive using the password.

- After extraction, your **keystore-*.json**, **keystore-*.txt**, and **pre_encryption_checksums.txt** files should now be in the staging folder.

5. Final Checksum Verification

Immediately verify the integrity of the extracted files using the **pre_encryption_checksums.txt** file.

- **Linux/macOS (in staging folder):**Bash (All files should show OK).

```
sha256sum -c pre_encryption_checksums.txt
```

- **Windows (PowerShell, in staging folder):** Manually compare current hashes generated with **Get-FileHash** against the **pre_encryption_checksums.txt.**
- If any checksum fails, **DO NOT PROCEED.** Re-evaluate the transfer process.

6. Securely Wipe Medium

Thoroughly wipe the trusted hardware medium after successful transfer and verification.

3.3 Prepare Keystore Files on Air-Gapped Machine

Organize the transferred and verified files into the required structure.

1. Prepare an air-gapped machine to perform key split

When bringing the keys over, we recommend bringing the Charon binary executable directly to the air-gapped machine using a secure, offline transfer. You can download the appropriate executable from the Charon GitHub Releases page on a connected machine, and transfer only the binary to the air-gapped system.

2. Create Folder

On the air-gapped machine, create the destination directory:

```
mkdir split_keys
```

3. Move File

- Move all verified **keystore-N.json** and **keystore-N.txt** files of the key batch from your temporary staging folder into the **split_keys** directory. You can delete the **pre_encryption_checksums.txt** file from the staging folder after this, as its purpose is fulfilled.
- **Example (from staging folder):**Bash

```
mv keystore-*.json keystore-*.txt split_keys/
```

- Ensure the pairing is correct (e.g., **keystore-0.json** and **keystore-0.txt**).

Resulting Directory Structure

```
├── split_keys
│   ├── keystore-0.json
│   ├── keystore-0.txt
│   ├── keystore-1.json
│   ├── keystore-1.txt
│  ...
│   ├── keystore-N.json
│   └── keystore-N.txt
```

3.4 Split the Keys using the Charon Executable

1. Download the Charon Executable

On a machine with internet access (not your air-gapped machine), navigate to the official Charon GitHub releases page: https://github.com/ObolNetwork/charon/releases

Locate the desired **CHARON_VERSION (e.g., v1.4.3)** and download the **charon-*-linux-amd64.tar.gz** (or **charon-*-linux-arm64.tar.gz** if your air-gapped machine uses an ARM processor) file.

2. Verify the Download

It's highly recommended to verify the integrity of the downloaded **.tar.gz** file using the provided **checksums.txt** file on the releases page.

```
sha256sum --check checksums.txt
```

You will get output like this (the code is on ARM executable). This means there was no AMD executable found. ARM was found. The 3 lines with improper formatting are the one with comments and spaces.

```
sha256sum:  charon-v1.5.1-linux-amd64.tar.gz: No such file or directory
charon-v1.5.1-linux-arm64.tar.gz: OK
sha256sum: WARNING: 3 lines are improperly formatted
```

3. Transfer and Extract the Charon Executable to the Air-Gapped Machine

- **Secure Transfer:** Copy the downloaded **charon-*.tar.gz** file to a new, never-before-used, trusted hardware medium (e.g., fresh USB drive). This medium must never touch an internet-connected machine after this step and transfer to Air-Gapped Machine.

- **Create a directory for Charon**

```
mkdir ~/charon
```

- **Copy and Extract:** Copy the **charon-*.tar.gz** file to the **~/charon** directory and then extract it.

```
cp /path/to/trusted_medium/charon-*.tar.gz ~/charon/
cd ~/charon
tar -xzvf charon-*.tar.gz
```

This will extract the **charon** executable (and potentially other files) into the current directory.

- **Make it Executable**

```
chmod +x charon
```

4. Run the Charon Command to Split Keys

Now, from the **~/charon** directory on your air-gapped machine, you can execute the **charon** command to split your keys. Ensure you **split_keys** directory (prepared in Section 3.1) is at the same level as your **charon** executable, or adjust the **--split-keys-dir** path accordingly.

- Define your Variables and Execute the Command

```
CHARON_VERSION=              # E.g. v1.4.3 (This is just for reference, not directly used in the command below)
CLUSTER_NAME=                # The name of the cluster you want to create.
WITHDRAWAL_ADDRESS=          # The address you want to use for withdrawals
FEE_RECIPIENT_ADDRESS=       # The address you want to use for block reward and MEV payments.
NODES=                       # The number of nodes in the cluster.

./charon create cluster \
--name="${CLUSTER_NAME}" \
 --withdrawal-addresses="${WITHDRAWAL_ADDRESS}" \
--fee-recipient-addresses="${FEE_RECIPIENT_ADDRESS}" \
--split-existing-keys \
--split-keys-dir=/path/to/your/split_keys \
--nodes ${NODES} \
--network mainnet
```

The above command will create **validator_keys** along with **cluster-lock.json** in **./cluster** for each node.

```
****************** WARNING: Splitting keys **********************
 Please make sure any existing validator has been shut down for
 at least 2 finalised epochs before starting the Charon cluster,
 otherwise slashing could occur.
************************************************************

Created Charon cluster:
 --split-existing-keys=true

./cluster/
 ├── node[0-*]/              # Directory for each node
 │   ├── charon-enr-private-key    # Charon networking private key for node authentication
 │   ├── cluster-lock.json         # Cluster lock defines the cluster lock file which is signed by all nodes
 │   ├── validator_keys            # Validator keystores and password
 │   │   ├── keystore-*.json       # Validator private share key for duty signing
 │   │   ├── keystore-*.txt        # Keystore password files for keystore-*.json
```

3.5 Testing the Private Key share recombination [Optional]

You can use the command below to confirm the private key shares can combine in different combinations to create the original key. Refer to this section in docs.

> 💡 *This is not strictly needed now, nor in the case of rollback, only to gain confidence that the keys were split as expected, and that you could recover from a loss of the original keystores.*

The original keystores should be kept as backup rather than solely relying on the ability to re-combine.

```
charon combine --help
Combines the private key shares from a threshold of operators in a distributed validator cluster into a set of validator private keys that can be imported into a standard Ethereum validator client.

Warning: running the resulting private keys in a validator alongside the original distributed validator cluster *will* result in slashing.

Usage:
  charon combine [flags]

Flags:
      --cluster-dir string                Parent directory containing a number of .charon subdirectories from the required threshold of nodes in the cluster. (default ".charon/cluster")
      --execution-client-rpc-endpoint string   The address of the execution engine JSON-RPC API.
      --force                             Overwrites private keys with the same name if present.
  -h, --help                              Help for combine
      --no-verify                         Disables cluster definition and lock file verification.
      --output-dir string                 Directory to output the combined private keys to. (default "./validator_keys")
      --testnet-chain-id uint             Chain ID of the custom test network.
      --testnet-fork-version string       Genesis fork version of the custom test network (in hex).
      --testnet-genesis-timestamp int     Genesis timestamp of the custom test network.
```

3.6 Preparing the private key shares for transfer

You can use the command below to confirm the private key shares can combine in different combinations to create the original key. Refer to this section in docs.

These private key shares will be used to start the Charon cluster. To do that they will need to be transferred later and you should prepare for the secure transfer using the commands below. It will encrypt each folder as a **.zip** to transport them.

```
# For each folder in ./cluster/ encrypt it with a different password
zip -er node1.zip ./cluster/node1/
# Repeat for node2,...,nodeN.
```

- There are other ways to encrypt the split_keys folder as well (e.g., tar cz + gpg).
- Transfer via trusted media (fresh USB) or secure network channel (SCP, etc).

STAGE 4

Setting up the Target Cluster(s)

4.1 Creating the Target Cluster

- Set up / Provision the number of nodes required for your Obol DV clusters. Please refer to this page in docs for **deployment best practices.** If you have any doubt, feel free to reach out to the members of the solutions team in contact with you.
- In your case, you will use **https://github.com/ObolNetwork/lido-charon-distributed-validator-node** as the launcher for nodes built specifically for Lido Clusters.

Once your nodes are synced-up and healthy, you will use the **charon test commands** to run the tests to do required checks.

> 💡 *We highly recommend to verify the results with our team.*

4.2 Monitoring Nodes

Before transferring keys, we need to monitor the cluster to ensure everything is synced and ready for migration. You can monitor the nodes using either your local Grafana or through our centrally hosted Grafana. More details **here.** For centralized monitoring, we will provide you with a Prometheus remote write token that will send metrics and logs to us.

> 💡 *We highly recommend centralized monitoring approach to ensure we can provide fast support, troubleshoot any issues and help improve Charon.*

Set up a contact point for **alerts** immediately to alarm you if something goes wrong. We already have alerts for missed attestations and proposals depending on **duty failure thresholds.**

STAGE 5

Transferring the keys

> 💡 *This is the most crucial step. Please follow closely because if your existing validator is not properly shut down before the Distributed Validator starts, your validator may be slashed due to double signing.*

Step 1:

Shut down the machine/instances where your original vanilla keys exist for the batch of keys that you are migrating. Also delete the backup machines (redundant machines used for maintaining uptime in normal validators). **Do not delete the Vanilla key backups. We will delete them after we can confirm the migration is successful. You will need backups if rollback is required to original vanilla keys.**

Step 2:

If your batch doesn't confirm 1:1 with the validators being shut down, we will provide you with a specific command to shut down the instances according to your requirements.

Step 3:

Monitor the keys that are offline and are not producing any attestations. Monitoring should be done by a trusted source - your own monitoring system that you were using until now or **https://beaconcha.in/**. You can also use the **beaconchain v2 dashboard**.

Step 4:

Once confirmed that keys are offline for at least 2 epochs, at the beginning of 3rd epoch, you can transfer the encrypted zip file generated in stage 3.4. On the target (air-gapped) DV node, verify checksums and decrypt. Wipe the transfer medium after verification.

Step 5:

To minimize reward loss during the migration window, validator key shares must be securely transferred to each node in the DV cluster, followed by manually starting the Charon and validator clients.

- The exact steps may vary depending on your infrastructure and operational setup. In general, this process includes:
  - Securely transferring each encrypted key share to its corresponding machine
  - Decrypting the key share locally on each node
  - Starting Charon and the validator client in a coordinated fashion

> 💡 *We recommend aligning with the Obol engineering team before executing this step to ensure proper coordination and avoid unintentional downtime or slashing risk.*

STAGE 6

Monitoring the DVs

Using the same dashboard(s) and beaconcha.in data, you can monitor the working of your cluster nodes and the on-chain performance of pubkeys to ensure they are back online. After the offline epochs, when the pubkeys are back online you should monitor for at least x more epochs before confirming to continue and roll back. Refer to following checklist on what your should monitor on.

- Beacon chain dashboard
- On-chain, track:
  - **Attestation success** (via beacon node metrics)
  - **Proposal inclusion**
  - **Validator status** (active vs. pending exit)
- Grafana dashboard(s)
- Monitor that the missed duties and successful duties in duties section
- Monitoring the contact points for the alerts set in the previous section
- Refer to the following roll back criteria:
- Criteria 1: If any of the validator is slashed, immediately shut down all validator and Charon processes across the affected cluster to prevent further duties from being performed. Investigate the root cause of the slashing event with the Obol engineering team.
- Criteria 2: If validators miss attestations for 3 more epochs after being back online, reach out to the client support team. We can guide you on if something is wrong with cluster configuration.

If rollback criteria is met refer to the rollback guide to vanilla validators. It provides two options and you can pick depending upon what Lido recommends or your comfort.

- Option 1: Exit DVs and Re-deposit with New Vanilla Validators
- Option 2: Rollback to Original Vanilla Validators.

STAGE 7

Exiting the DVs

In the case of a validator exit, use lido-dv-exit-sidecar in LCDVN, which handles the signing, downloading, and staging of exits.

- To ensure that validators can be exited on demand by Lido or can be exited as part of a rollback mechanism when something goes wrong with migration, the first stage is signing the partial exit messages. The partial exits can be combined later on demand to a full-exit that can be broadcasted. More details here . The lido-dv-exit sidecar handles all of it.

- To **run**, one must provide:

  - a beacon node **(-beacon-node-url)**
  - a directory in which lido-dv-exit will write signed voluntary exits, and where validator-ejector will pick them up **(-ejector-exit-path)**
  - a directory containing Charon's lock file, identity private key and validator shares **(-charon-runtime-dir)**

  - Optionally one can specify an exit epoch, and an instance of the Obol API to be used for coordination purposes. One can also run the program with Docker, provided that the volume pointed to by **--ejector-exit-path** is writable.

  - **lido-dv-exit** supports pushing logs to a Loki instance, while still printing them to stderr. To do so, one must configure the run command with **--loki-addresses** flag, on the **HTTP** Loki port, on the **/loki/api/v1/**push endpoint. For example, if Loki is hosted on the **loki.local** domain:

```
lido-dv-exit run --loki-addresses http://loki.local:3100/loki/api/v1/push # other configuration flags follow
```

You can monitor the partial exit status in grafana dashboard or on launchpad mentioned in step 2 in Obol Docs.

STAGE 8

Claiming Validator Rewards and Obol Incentives

The DV validators will have the same withdrawal address and fee recipients as in the Vanilla setup. As a result, the rewards claiming process will stay the same. Reward sharing with Obol will be done off-chain on a case by case basis. In the future, split contracts can potentially be implemented but will require discussion and on-chain vote.

8.1 Claiming Obol Incentives

You will be earning Obol Incentives as part of running the stake on Obol DVs. The Obol Incentive program is fully funded for 3 years and the calculation will be done according to the formula described in the Obol Docs. The distribution will be done in confirmation with Lido's post . The distribution happens weekly on Mondays.

Regarding DVT provider incentives for any validators operated in this manner in the Curated Module, Node Operators would receive 20% of DVT provider incentives pertaining to said validators, with the other 80% to be directed to the Mellow Decentralized Validator Vault.

There are two ways Obol incentives can be claimed

- Going on the operator page in launchpad and manually claiming it
- Using SDK to claim automatically. An example can be found here. Our team can help you integrate this solution as well.

Appendix

1. Rollback guide

Option 1: Rollback to Original Vanilla Validators

This option allows you to revert your existing validators to a vanilla setup, leveraging your original validator keys or by re-combining Charon shares. This minimizes downtime compared to exiting and re-depositing.

Crucial Note on Key Source:

- **Preferred:** You have securely backed up your original vanilla validator keystore files (**keystore-N.json** and **keystore-N.txt**) in a secure vault **before** migrating to DVs. This is the safest and simplest recovery method.
- **Alternative:** If original keys were not explicitly backed up, you must rely on combining the Charon key shares (**keystore-N.json** and **keystore-N.txt** within each node's **validator_keys** directory) from your DV setup to reconstruct the original validator key. This means keeping all Charon key shares securely in a vault.

Rollback Procedure:

A) Shutdown DV Cluster:

- Stop all Distributed Validator (DV) processes and instances on your machines (**docker compose down**).
- Delete any Docker containers and associated volumes used by the DV setup.
- Delete the DV key shares from your machines. **Ensure you back up any DV-specific configurations or logs if needed for post-mortem analysis before deletion.**

B) Wait for Epochs:

- After shutting down the DV cluster, wait for at least **2 full finalized epochs** (approximately 12.8 minutes per epoch). This crucial waiting period ensures the network registers the DV validators as offline, preventing slashing for double-signing when the vanilla validators come back online.
- **Monitor:** Use a trusted source (e.g., your own monitoring system or beaconcha.in) to confirm that the affected public keys are registered as offline and are not performing duties during this period.

C) Prepare Original Vanilla Keys:

- **If original keys were backed up:** Retrieve your keystore-N.json and keystore-N.txt files from your secure vault.
- **If original keys were NOT backed up** (and you retained Charon shares):
  - On a secure, air-gapped machine, retrieve a quorum of Charon key shares (e.g., **node0/validator_keys/keystore-N.json, node1/validator_keys/keystore-N.json**, etc., up to your threshold) from their secure vault. Use the exact transfer process describe above using checksums and encryption of zipped folder.
  - Use the **charon combine** command to reconstruct the original vanilla keystore: Bash (Adjust input directory and node numbers based on your threshold and how you extract the shares.)

```
charon combine \
 --input-dir ./path/to/extracted_charon_shares/node{0,1,2}/validator_keys \
 --output-dir ./path/for_recovered_vanilla_keys
```

  - The output will be the **keystore-N.json** and **keystore-N.txt** for your vanilla validator.

D) Prepare Vanilla Validator Machines:

Spin up or prepare the machines that will run your vanilla validators again. Ensure they are configured correctly with their respective beacon and validator clients.

E) Transfer & Verify Vanilla Keys:

- Transfer the recovered original vanilla **keystore-N.json** and **keystore-N.txt** files to the respective vanilla validator machines.
- **Crucial:** Before starting the validator client, perform a checksum verification of these transferred files against your known good hashes (either from your original backup or generated during the **charon combine** process) to ensure their integrity.

F) Restart Vanilla:

- Once the vanilla keys are securely in place and verified, restart your vanilla validator clients.
- The validators should come back online and resume duties from the 3rd epoch after the DV shutdown.
- **Monitor:** Continue monitoring the validator performance via your client logs, Grafana dashboards, or public explorers (e.g., beaconcha.in) to confirm they are back online and performing attestations and proposals successfully.

Option 2: Exit DVs and Re-deposit with New Vanilla Validators

This option is for scenarios where you prefer to exit your existing validators and start entirely new ones.

Pre-sign & Broadcast Exit Messages (DV Side):

- Use **charon exit sign** command on your DV cluster to pre-sign partial exit messages for the validators you wish to exit.
- Once the threshold is reached, use **charon exit fetch** to download the signed exits.
- Finally, use **charon exit broadcast** to submit the full exit messages to the network.
- **Monitor:** Confirm that the validators' status changes to **exited** on a beacon explorer (e.g., beaconcha.in).

Wait for Exit Queue:

Validator exits are processed through a queue. The time this takes depends on network congestion. Monitor the queue to confirm your validators have fully exited.

Generate New Vanilla Keys:

- Once your previous validators are fully exited, generate entirely new validator keystores and deposit data for new vanilla validators.
- Perform new ETH deposits for these new validators, following the standard staking process.

Spin Up New Vanilla Validator Machines:

- Prepare and configure new machines to run these new vanilla validators with their respective beacon and validator clients.
- Transfer the newly generated keys to these machines.

Monitor New Validators:

Monitor the activation queue for your new validators. Once activated, observe their performance on-chain to ensure they are attesting and proposing blocks correctly.

2. Estimate Missed rewards

During the time of migration, the vanilla validators will be offline for a minimum ~2 epochs. We can estimate the missed reward using the sheet below.

https://docs.google.com/spreadsheets/d/1jbaekFTmbZcY6caDVS9Vcf2E08cYAvwSuY2I7t-3M8U/edit?gid=2088021440#gid=2088021440