# Zend to Zendoo upgrade
# ShortGuide v1.1 - Mining Pools

2021-11-15

—

---

**What's new:**

With Zendoo we are introducing sidechain support to our zen node network. The Zendoo hard fork is already active on testnet, and will activate on mainnet on December 1st, 2021 at block height 1047600.

Please note that starting with zend v3.0.0 the bmi2 and adx CPU flags are required to run zend. These flags are supported starting from Intel Broadwell and AMD Excavator CPU architectures. You can check if your host is compatible using the following command:

```
( grep -q "adx" /proc/cpuinfo && grep -q "bmi2" /proc/cpuinfo ) &&
echo "v3.0.0 supported" ||
echo "v3.0.0 NOT supported"
```

We recommend running zendoo on a machine with at least 4 CPU cores and high clock speed(>3GHz)/IPC and at least 16GB of memory.

For enabling chain interoperability (eg. bi-directional coin transfers between mainchain and sidechains) we introduced two new transaction types:

- **Version -4**: this transaction type extends the behaviour of a transparent transaction by adding the possibility to interact with sidechains (eg. create a sidechain, send coins to a sidechain, request withdrawal from a sidechain, etc.) For this purpose, additional kinds of outputs and inputs have been added to the transparent transaction. For example, to enable the sending of coins to sidechains, a specific array of outputs was introduced that specifies the amounts transferred and the details of the destination address. Please note: This kind of output does not create UTXOs in the mainchain because the corresponding amount is made available on the sidechain side. This means that the amount sent from the mainchain to a sidechain is burnt on the mainchain. This new kind

of transaction is contained in the same vector of the other previously supported transactions (eg. version 1, 2, -3).

- **Version -5**: this type of transaction - called **certificate** - is specifically meant to transfer coins from a sidechain to a mainchain transparent P2PKH address ("backward transfer"). For this reason, the output amount will usually be greater than the input amount. A certificate contains, as any "standard" transaction, a list of inputs which are used to pay fees, and two lists of outputs: "normal" outputs (typically the change for the inputs used for the fees) and "backward transfer" outputs (users' withdrawals originated in the sidechain). Certificates also contain additional fields that are required for their validation (eg. SNARK proofs, additional data). Certificates are not included in the main transaction vector, but are listed in a separate array (in the JSON block representation the array is called "cert").

The two transactions mentioned above implement the ***Cross Chain Transfer Protocol****. You can find a detailed explanation of their structure in the following document.

For a more complete description of their purpose and behaviour, please refer to our Zendoo whitepaper: https://www.horizen.io/assets/files/Horizen-Sidechain-Zendoo-A_zk-SNARK-Verifiable-Cross-Chain-Transfer-Protocol.pdf

***Cross-Chain Transfer Protocol is the protocol that is used by the mainchain to interact with the sidechains.***

- Mainchain blocks were slightly changed in order to support sidechains. The main differences are:
  - In the block payload, the main list of transactions now can contain the additional transaction type -4.
  - After the main list of transactions, a new vector of "type -5" certificate transactions was added.
  - The merkle tree of transactions of the block, in addition to the main transaction vector, now also includes the list of certificates. The leaves are composed in the following way: the main transaction vector (following the original order) followed by the list of certificates. The "***merkleroot***" field of the block header is now computed considering this new structure.
  - The previously "free-to-use" "***hashReserved***" field of the block header was renamed to "***scTxsCommitment***" and its value has to be consistent with the block. The value now represents the root of the Merkle tree, composed by the sidechain related output and inputs of transactions and certificates contained in the block.

To simplify the calculation of the transactions ***"merkleroot"***, and the sidechain transactions commitment root "***scTxsCommitment"***, we introduced a new RPC API command named "**getblockmerkleroots"**. By appending your coinbase transaction to the transactions vector and passing both transactions and certificates as arguments to the new RPC command, you will be able to calculate the proper merkle roots.

**Typical steps needed to forge a block:**

In order to mine a new block you have to perform the following steps:

- Get a JSON template of the block (certificates and txs) through the RPC command "**getblocktemplate"**.
- Insert the coinbase transaction for the block reward into the main transaction vector as the very first transaction.
- Serialize in hex format the list of raw transactions and the list of raw certificates separately (please refer to the following specification to get more details on how to serialize the new type of transactions and certificates).
- Calculate the transaction Merkle roots (***merkleTree*** and ***scTxsCommitment***) using the RPC command "**getblockmerkleroots"** passing as first argument a JSON array of the raw transactions to be included in the block (including the coinbase transaction), and as second argument a JSON array of raw certificates (that you got from "**getblocktemplate"**). This will return a JSON object that contains two fields, "***scTxsCommitment***" and "***merkleTree***".
- Serialize the block header using the calculated merkle roots.
- Compose the serialized block by concatenating the header, the list of transactions, and the list of certificates.

Please note that:

- The maximum block size has been increased to 4MB (including the header). Only 2MB of the block can be filled by the main transaction vector. Certificates can use up to the full size of the block (4MB).
- The block version changes from 0x20000000 to 0x00000003.
- The RPC call "**getblockmerkleroots"** takes as inputs two arguments, the list of raw transactions and the list of raw certificates to be included in the block. While the second list might be empty (if no certificates are being added to the block), the first one must contain at least the coinbase transaction.

**RPC Commands changes:**

- **getblocktemplate ( "jsonrequestobject" )**
    - New fields:
        - "certificates": it contains the list of certificates included in the block. It is specified at the same level of the "transactions" field.
- **getmininginfo**
    - New fields:

        - “currentblockcert”: it specifies the number of certificates contained in the last block.
        - "pooledcert": it contains the number of certificates that are currently in the mempool.

**Newly introduced RPC Commands:**

- **getblockmerkleroots**
    - Fields:
        - A, array of raw transactions
        - B, array of raw certificates

**3.0.0 release and hard fork:**

We’ve already released Zendoo **3.0.0** and the zendoo hard fork is active on our testnet. The fork activated on testnet at block height **926225**. The fork will activate on mainnet on **December 1st, 2021** at block height **1047600**.

In order to ensure pool compatibility you can test on our testnet. To do so you must use the latest zend release available on our repository https://github.com/HorizenOfficial/zen. You can either build from source or if you prefer you can directly download precompiled binaries at this link: https://github.com/HorizenOfficial/zen/releases/tag/v3.0.0.

**S-nomp compatibility:**

We have made changes to node-stratum-pool to make it compatible with the Zendoo fork. Please check this link for a comparison of the changes: https://github.com/HorizenOfficial/node-stratum-pool/compare/14e315ecebceb5a6424fa7e67c99af3c18eca292...HorizenOfficial:master

These changes can be used as a template to make other mining pool implementations compatible.

A Pull Request to node-stratum-pool was opened here https://github.com/s-nomp/node-stratum-pool/pull/59 and has been merged. Which means that https://github.com/s-nomp/s-nomp is now fully compatible with the zendoo fork on testnet and the upcoming fork on mainnet.