Zend to Zendoo Upgrade
ShortGuide v1.4 - Exchanges

2021-12-05

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 v3.0.3 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 "v3.0.3 supported" ||
echo "v3.0.3 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.

ZencashJS:

A new version of zencashjs (2.0.1) has been released that adds support for decoding of the new version -4/-5 transaction types. Please follow the migration instructions on github to upgrade.

Certificate overview

As mentioned above, certificates are the way users receive funds withdrawn in a specific sidechain. The protocol requires that each sidechain sends a certificate on a regular basis, called "epoch". The epoch length (specified in number of mainchain blocks) is specific to any sidechain. For an epoch there can be multiple certificates specifying the withdrawals related to the past epoch. Such certificates can differ from each other, even for the same sidechain because of the possible presence of forks in the sidechain leading to different histories. Such fork resolution is resolved in a more abstract way by relying on a “quality level” associated with each certificate. For example, in a sidechain that follows the longest chain rule model, the quality will reflect the number of sidechain blocks. Each mainchain node will be able to determine - after a certain given timeframe - which certificate has the highest quality and is thus considered the final certificate for that epoch. Having this mechanism of finality resolution means that the backward transfers cannot be considered spendable immediately after the containing certificate is included in a block. Each epoch has a time window that allows the receival of certificates. During this period, backward transfers of the received certificates are not spendable because each certificate for the same epoch can be superseded by a new one with a higher quality. In such a situation, the backward transfers of the superseded certificate are voided.

In other words, a “certificate” and its “backward transfers” can have 3 different states:

- *Mature*: the “backward transfers” are spendable at a given blockchain height.
- *Immature*: the “backward transfers” are not spendable yet at a given blockchain height.
- *Superseded*: the “backward transfers” will never be spendable independent of any given blockchain height.

As previously mentioned, certificates spend regular inputs for paying fees and have regular outputs to manage the change. These regular inputs and outputs are immediately considered final as soon as the certificate is included in a block, even if the certificate was superseded.

What you need to know

With the introduction of certificates, some of the RPC APIs have been updated to allow filtering / distinguishing non mature amounts (eg. gettxout, listtransactions, etc.). In most cases you would not be interested in seeing immature outputs (filtered by default by most of the RPC APIs). However, consider that for certificates - after some blocks - the same command could return additional data if the certificate matured (eg. gettransaction would also return the mature outputs after having reached the maturity height).

As mentioned in the introduction, new fields were introduced in the structure of transactions with version -4 and -5. For transactions with type -4, the following additional fields have been introduced: “vcsw_ccin”, “vsc_ccout”, “vft_ccout”, “vmbtr_out”. The introduction of these fields may not require a change in your parsing logic because both inputs fields (vcsw_ccin) and outputs fields (vsc_ccout, vft_ccout”, vmbtr_out) do not spend / create mainchain UTXOs but instead, create sidechain transfers. For this reason, you may not need to update your business logic. Please verify your specific implementation and act accordingly.

For certificate transactions with version -5, beyond the other additional fields, please note that the list of outputs includes both “change” outputs (immediately spendable) and backward transfers (whose ability to be spent is subject to the rules described above) that are identified by an additional boolean field named “backwardtransfer”.

You can find a more complete example of both transaction formats at the end of this document.

RPC Command changes

- **getblock "hash|height" (verbose)**
    - With the introduction of transaction types -4 and certificates -5, the API return structure has been updated to show the newly introduced contents (eg. the transaction array will contain the new transaction types with the new fields and an additional array listing the certificates has been added to the block). Please refer to the “What you need to know” section for a brief explanation.
    - New field: "scTxsCommitment" (replaces hashReserved)
    - New field: "cert" → array of the certificates contained in the block
    - New field: "scCumTreeHash”
    - Updated field "merkleroot": 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.
- **getblockexpanded "hash|height" (verbose)**
    - This command has the same output as **getblock** with additional information about “backward transfer” outputs that have become spendable at this block. The certificates that have matured (i.e. become spendable) are included in a new field, “matureCertificate”. It is in the same format as “cert” (see getblock).
    - This command is only available when the “txindex” and “maturityheightindex” flags are enabled in zend.
- **getblockchaininfo**
    - Removed field: "bip9_softforks"

- **getblockheader "hash" (verbose)**
    - New field: "scTxsCommitment" (replaces hashReserved)
    - New field: "scCumTreeHash"(only if verbose)
- **getrawmempool (verbose)**
    - In the returned list, both transaction IDs and certificate IDs will be included. If you run the command in verbose mode, it will also return a new boolean field named “IsCert” and another field named “version”. These fields help you to understand if each ID refers to a transaction or a certificate.
- **gettxout “txid” n (includemempool)**
    - New optional boolean input parameter “includeImmatureBTs”: whether to include immature backward transfers in the result. By default it is false and it **DOES NOT** include immature backward transfers.
    - New field: “certificate” → boolean field that indicates if the input transaction is a certificate or not.
    - New field: “backwardtransfer” → boolean value that indicates if the UTXO is a backward transfer.
    - New field: “maturityHeight” (only if “backwardtransfer” = true) → if the UTXO is confirmed, it will show the height at which it can become spendable; if it is not confirmed, the value is set to -1.
    - New field: “blocksToMaturity” (only if “backwardtransfer” = true) → if the UTXO is confirmed, it will show how many blocks should be mined to reach the maturity height; if it is not confirmed, the value is set to -1.
    - New field: “mature” (only if “backwardtransfer” = true) → boolean value that indicates whether the UTXO is mature (spendable).
- **decoderawtransaction "hexstring"**
    - With the introduction of transaction types -4 and certificates -5, the JSON output has been updated to show the newly introduced contents. Please refer to the “What you need to know” section for a brief explanation.
- **getrawtransaction "txid" (verbose)**
    - With the introduction of transaction types -4 and certificates -5, the JSON output has been updated to show the newly introduced contents. Please refer to the “What you need to know” section for a brief explanation.
- **getbalance ("account" minconf includeWatchonly)**
    - This includes only the MATURE balance. *Please note:* UTXOs referring to mature certificate backward transfers contribute to the balance.
- **gettransaction "txid" (includeWatchonly)**
    - With the introduction of transaction types -4 and certificates -5, the JSON output has been updated to show the newly introduced contents. Please refer to the “What you need to know” section for a brief explanation.

    - New optional input parameter [applicable only if "txid" refers to a certificate]: “includeImmatureBTs” → boolean field (default false) that indicates whether to include immature certificate Backward transfers data in the result. The immature backward transfers output included in the JSON result have "category": "immature".
    - New field: “version” → transaction’s version.
- **listaddressgroupings**
    - A list that includes mature certificate backward transfer balances but skips immature ones.
- **listreceivedbyaddress** (minconf includeempty includeWatchonly)
    - The certificates that are in state superseded (see above) are present in the response.
- **listsinceblock ("blockhash" target-confirmations includeWatchonly includeImmatureBTs)**
    - New optional input parameter: “includeImmatureBTs” → boolean field (default false) that indicates whether to include immature certificate backward transfers.
    - With the introduction of transaction types -4 and certificates -5, the JSON output has been updated to show the newly introduced contents. Please refer to the “What you need to know” section for a brief explanation.
    - The immature “backward transfers” have "category": "immature"
    - Note: "If the input block range does not contain the block where a certificate has been mined, and the certificate became mature in one of the blocks in the range, the output JSON list of this command will contain its backward transfers, and they will be referred to the block at which they became mature."
- **listtransactions ("account" count from includeWatchonly address includeImmatureBTs)**
    - With the introduction of transaction types -4 and certificates -5, the API return structure has been updated to show the newly introduced contents. Please refer to the “What you need to know” section for a brief explanation.
    - New optional input parameter: “includeImmatureBTs” → boolean field (default false) that indicates whether to include immature certificate Backward transfers.
    - New optional input parameter: "address" → string, include only transactions involving this address
- **listunspent (minconf maxconf ["address",...])**
    - Added field: "satoshis"
    - Added field: “isCert” → boolean field that indicates if the UTXO is from a certificate.
    - Immature “backward transfers” are not returned.

How to Use getblockexpanded

Make sure the “txindex” and “maturityheightindex” flags are enabled in Zend and you have either re-synced or re-indexed the blockchain.

In order to decode a block and all transactions, use “getblockexpanded” (instead of “getblock”), with verbosity level 2.

For each certificate in the cert field, save the transaction as you would a regular transaction, but pay special attention to the vouts. The “regular” outputs should be saved. Ignore the “backward transfer” outputs (“backward transfer”: true), because they are not spendable yet in this block. For each certificate in the matureCertificate field, we only care about the “backward transfer” outputs because they have now become spendable. Vins and “regular” outputs can be ignored in mature certificates, as they have been accounted for when they first appeared in the blockchain.

Examples

Example of new transaction type version -4 that creates a new sidechain and sends 1 zen to it

```
:$ zen-cli -testnet  getrawtransaction c205cc95d71dd7f008fe8691a79781b9d0a25ddfa2d34f367839d73b02b3e3a5 1
{
  "txid": "c205cc95d71dd7f008fe8691a79781b9d0a25ddfa2d34f367839d73b02b3e3a5",
  "version": -4,
  "locktime": 937525,
  "vin": [
    {
      "txid": "7150adfda8cdd822572a02020107facd39b7792ee9a37e357ddb7cce33a09a46",
      "vout": 0,
      "scriptSig": {
        "asm":
"3045022100fb94fc8cd4a6443e6491c07b8c11cfffbe100038b5903611f4e43ab098373b1102203b703a8f5d57a585e2c
b7611d60978a3fb7f4d15c8a8fa5ccc55ad8911a5b59201
03c82e97aa4d683af3308b8672f3b2ac2536dbba107d93ca6f1e08672ce6b93f44",
        "hex":
"483045022100fb94fc8cd4a6443e6491c07b8c11cfffbe100038b5903611f4e43ab098373b1102203b703a8f5d57a585e
2cb7611d60978a3fb7f4d15c8a8fa5ccc55ad8911a5b592012103c82e97aa4d683af3308b8672f3b2ac2536dbba107d93
ca6f1e08672ce6b93f44"
      },
      "sequence": 4294967294
    }
  ],
  "vcsw_ccin": [
  ],
  "vout": [
    {
      "value": 2987.99955294,
      "valueZat": 298799955294,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 b42167247213fb4cc9d0ed7cf3f26109e0d09c16 OP_EQUALVERIFY
OP_CHECKSIG 6bfae64c63f7bb479d927b0498a06112d06a2943f622efd0e1f5f7cf37820c00 937235
OP_CHECKBLOCKATHEIGHT",
        "hex":
"76a914b42167247213fb4cc9d0ed7cf3f26109e0d09c1688ac206bfae64c63f7bb479d927b0498a06112d06a2943f622
efd0e1f5f7cf37820c0003134d0eb4",
        "reqSigs": 1,
        "type": "pubkeyhashreplay",
        "addresses": [
          "ztjaQQTWoTH6JKgmTuSux1MvDEQSv24K5c8"
        ]
      }
    }
  ],
  "vsc_ccout": [
    {
      "scid": "1a4d5813b260d0cb456c649b005840e1a1eb6eb2e0f98f3af7d201ea1e95d0b8",
      "n": 0,
      "withdrawalEpochLength": 100,
      "value": 1.00009023,
      "address": "8d8137d57eee250bdd0302fcad05243276ba059556165517c3d919331cd5bdc8",
      "certProvingSystem": "CoboundaryMarlin",
      "wCertVk": "...",
      "vFieldElementCertificateFieldConfig": [
      ],
      "vBitVectorCertificateFieldConfig": [
      ],
      "customData": "fe20c53828c5d01acaf8b473fcbdb08465d422ebab416580b427519209bb1b3500",
      "constant": "a15f637c02977b3c542d3c53a0957f523155f06234d10cdfb123b298826cdf10",
      "ftScFee": 0.00000000,
      "mbtrScFee": 0.00000000,
      "mbtrRequestDataLength": 0
    }
  ],
  "vft_ccout": [
  ],
  "vmbtr_out": [
  ],
  "vjoinsplit": [
  ],
  "blockhash": "0000ee0219c38a4feda2aa5bf0b530bd10bee295bea2dbbcb283a6f6f31e5b5c",
  "confirmations": 620,
  "time": 1634650773,
  "blocktime": 1634650773,
  "hex": "..."
}
```

Example of a new transaction version -5 that includes a backward transfer (vout[1])

```
:$ zen-cli -testnet getrawtransaction "410a983c24f14bc0c2bef11c3274f399b99f32235742da8b3eca1ebb2986146f" 1
{
  "txid": "410a983c24f14bc0c2bef11c3274f399b99f32235742da8b3eca1ebb2986146f",
  "version": -5,
  "vin": [
    {
      "txid": "4f84b0c8932780b9e0d71ab0c4cda0fd0ad262a4298102abb619a03d7735e0bf",
      "vout": 72,
      "scriptSig": {
        "asm":
"3045022100b23c00737b235c486cc8a504e7fd54cd36bb53d2c95d336ef1d775b8bbebbbc302206c5d5eea6260e8c8
4cb7ef0a3dabc9e2a25293fe9eb5a3b4fa169da7584e714201
036cda72878f999d49502b29737e7a708dffa4f6198f7cbec18f07ae3874b06a50",
        "hex":
"483045022100b23c00737b235c486cc8a504e7fd54cd36bb53d2c95d336ef1d775b8bbebbbc302206c5d5eea6260e8
c84cb7ef0a3dabc9e2a25293fe9eb5a3b4fa169da7584e71420121036cda72878f999d49502b29737e7a708dffa4f6198f
7cbec18f07ae3874b06a50"
      },
      "sequence": 4294967295
    }
  ],
  "cert": {
    "scid": "1a4d5813b260d0cb456c649b005840e1a1eb6eb2e0f98f3af7d201ea1e95d0b8",
    "epochNumber": 5,
    "quality": 5,
    "endEpochCumScTxCommTreeRoot":
"a059242590c1337207aa1aab131d9afd67b4385595429aca4d00f90fd2d14e25",
    "scProof": "...",
    "vFieldElementCertificateField": [
    ],
    "vBitVectorCertificateField": [
    ],
    "ftScFee": 0.00000000,
    "mbtrScFee": 0.00000000,
    "totalAmount": 5.00000000
  },
  "vout": [
    {
      "value": 0.99999000,
      "valueZat": 99999000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ec6039c0505e74b8f74fb1e22b77da64d30ce6b3 OP_EQUALVERIFY
OP_CHECKSIG 37be36242d04190743e7e50bb8bd268e5891883dd270ffacb65f7f3675ae0100 937840
OP_CHECKBLOCKATHEIGHT",
        "hex":
"76a914ec6039c0505e74b8f74fb1e22b77da64d30ce6b388ac2037be36242d04190743e7e50bb8bd268e5891883dd2
70ffacb65f7f3675ae010003704f0eb4",
        "reqSigs": 1,
        "type": "pubkeyhashreplay",
        "addresses": [
          "ztphoWCQmyJVuNq2L3SLnRgy2Lw5i5a7hxL"
        ]
      }
    },
    {
      "value": 5.00000000,
      "valueZat": 500000000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ec6039c0505e74b8f74fb1e22b77da64d30ce6b3 OP_EQUALVERIFY
OP_CHECKSIG",
        "hex": "76a914ec6039c0505e74b8f74fb1e22b77da64d30ce6b388ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "ztphoWCQmyJVuNq2L3SLnRgy2Lw5i5a7hxL"
        ]
      },
      "backwardTransfer": true
    }
  ],
  "vjoinsplit": [
  ],
  "blockhash": "00012834f09159b132315e4001406eb567a4e76c8c17045089b2ba1b4ce41065",
  "confirmations": 9,
  "blocktime": 1634740194,
  "hex": "..."
}
```