ChainLocks Overview

ChainLocks are a feature provided by the Dash Network which provides certainty when accepting payments. This technology, particularly when used in parallel with InstantSend, creates an environment in which payments can be accepted immediately and without the risk of “Blockchain Reorganization Events”.

The risk of blockchain reorganization is typically addressed by requiring multiple “confirmations” before a transaction can be safely accepted as payment. This type of indirect security is effective, but at a cost of time and user experience. ChainLocks are a solution for this problem.

ChainLocks Process Overview

Every twelve hours a new “LLMQ” (Long-Lasting Masternode Quorum) is formed using a “DKG” (Distributed Key Generation) process. All members of this Quorum are responsible for observing, and subsequently affirming, newly mined blocks:

1. Whenever a block is mined, Quorum Members will broadcast a signed message containing the observed block to the rest of the Quorum.

2. If 60% or more of the Quorum sees the same new block they will collectively form a “CLSIG” (ChainLock Signature) message which will be broadcast to the remainder of the network.

3. When a valid ChainLock Signature is received by a client on the network, it will reject all blocks at the same height that do not match the block specified in that message.

The result is a quick and unambiguous decision on the “correct” blockchain for integrated clients and wallets. **From a security perspective, this also makes reorganizations prior to this block impossible.**

ChainLocks Integration using ZMQ

ChainLock Signatures are created shortly after the related block has been mined. As a result it is recommended that integrated clients use “ZMQ” (ZeroMQ) notification in order to ensure that this information is received as promptly as possible.

This sample code fetches the most recent block using JSON-RPC and will listen for ZMQ notification for every subsequent “chainlock” that occurs when a new block is mined:

```
const zmq = require('zeromq');
const Dashcore = require('@dashevo/dashcore-lib');
const RpcClient = require('@dashevo/dashd-rpc');

/* see: https://raw.githubusercontent.com/snogcel/chainlock-sample/master/index.js */

const rpc = new RpcClient(config); // connect to Dash Core daemon using JSON-RPC

const socket = zmq.socket('sub');

socket.connect('tcp://127.0.0.1:28332'); // connect to Dash Core daemon using ZMQ
Notification
socket.subscribe('rawchainlock'); // subscribe to "rawchainlock"

// Fetch Chain Tip from Dash Core Daemon using JSON-RPC
rpc.getBestBlockHash(function (err, res) {
    if (res.result) {
        rpc.getBlock(res.result, function(err, res) {
            console.log("starting block height:", res.result.height);
            console.log("");
            console.log("hash:", res.result.hash);
            console.log("chainlock:", res.result.chainlock);
            console.log("");
        });
    }
});

// Handle ChainLock Notification using ZMQ
socket.on('message', function(topic, message) {
    if (topic.toString() === 'rawchainlock') {

        let block = new Dashcore.Block(message);

        let chainTip = block.toJSON();

        console.log("* zmq notification: rawchainlock *")
        console.log("hash:", chainTip.header.hash);
        console.log("previous hash:", chainTip.header.prevHash);
        console.log("");

    }
});
```

ChainLocks Integration using JSON-RPC

ChainLock status can also be obtained through direct connection with the dash daemon using JSON-RPC protocol. Information is returned in the “chainlock” attribute; this true/false value is provided in a variety of block and transaction commands which are described below.

★ GetBlock: The GetBlock RPC gets a block with a particular header hash from the local block database either as a JSON object or as a serialized block.

```
dash-cli -testnet getblock \
            00000000045c7adc114443ff61ea6e7f11c2a7bc7651fb5c8c61e55f2fa20d16 \
            1
```


```
{
        "hash": "00000000007b0fb99e36713cf08012482478ee496e6dcb4007ad2e806306e62b",
        "confirmations": 14302,
        "size": 310,
        "height": 86190,
        "version": 536870912,
        "versionHex": "20000000",
        "merkleroot": "25632685ed0d7286901a80961c924c1ddd952e764754dbd8b40d0956413c8b56",
        "tx": [ … ],
        "cbTx": { … },
        "time": 1556114577,
        "mediantime": 1556113720,
        "nonce": 2503323484,
        "bits": "1c0094aa",
        "difficulty": 440.8261075201009,
        "chainwork": "0000000000000000000000000000000000000000000000045ab6f9403a8e7",
        "previousblockhash":
"000000000073a041bac70a7c3e49c29e8bc954071aeee6e00c7ac8064a372e27",
        "nextblockhash": "00000000001c6c962639a1aad4cd069f315560a824d489418dc1f26b50a58aed",
        "chainlock": true
}
```

★ ListSinceBlock: The ListSinceBlock RPC gets all transactions affecting the wallet which have occurred since a particular block, plus the header hash of the most recent block.

```
dash-cli -testnet listsinceblock \
              00000000045c7adc114443ff61ea6e7f11c2a7bc7651fb5c8c61e55f2fa20d16 \
              6 true
```


```
{
       "transactions": [{
              "account": "",
              "address": "ycT8D2LiW5QPDUkvTG1fgACHwgRFiR158P",
              "category": "receive",
              "amount": 143.67890000,
              "label": "",
              "vout": 0,
              "confirmations": 0,
              "instantlock": true,
              "instantlock_internal": true,
              "chainlock": false,
              "trusted": true,
              "txid": "a84b2f871aff315153c2a7553dfcf73ae1936f591fb7346ad0e801995f18a585",
              "walletconflicts": [],
              "time": 1558119292,
              "timereceived": 1558119292
       }],
       "lastblock": "000000000951acfd9da289f501be528891c9225f3ce63fdd005e5feef59be894"
}
```

```
{
       "transactions": [{
              "account": "",
              "address": "ycT8D2LiW5QPDUkvTG1fgACHwgRFiR158P",
              "category": "receive",
              "amount": 143.67890000,
              "label": "",
              "vout": 0,
              "confirmations": 1,
              "instantlock": true,
              "instantlock_internal": false,
              "chainlock": true,
              "blockhash":
"0000000001d0c2b2139aec56e7f81251332457ab68b6242a46014dd0cd7f7909",
              "blockindex": 1,
              "blocktime": 1558119666,
              "txid": "a84b2f871aff315153c2a7553dfcf73ae1936f591fb7346ad0e801995f18a585",
              "walletconflicts": [],
              "time": 1558119292,
              "timereceived": 1558119292
       }],
       "lastblock": "0000000009efe28f5a92fbbf1890781121ed1872bdfb364b1bcebf2d40c0a98a"
}
```

★ ListTransactions: The ListTransactions RPC returns the most recent transactions that affect the wallet.

```
dash-cli listtransactions "" 1 0 true
```


```
[{
        "account": "",
        "address": "ycT8D2LiW5QPDUkvTG1fgACHwgRFiR158P",
        "category": "receive",
        "amount": 143.67890000,
        "label": "",
        "vout": 0,
        "confirmations": 0,
        "instantlock": true,
        "instantlock_internal": true,
        "chainlock": false,
        "trusted": true,
        "txid": "a84b2f871aff315153c2a7553dfcf73ae1936f591fb7346ad0e801995f18a585",
        "walletconflicts": [],
        "time": 1558119292,
        "timereceived": 1558119292
}]
```

```
[{
        "account": "",
        "address": "ycT8D2LiW5QPDUkvTG1fgACHwgRFiR158P",
        "category": "receive",
        "amount": 143.67890000,
        "label": "",
        "vout": 0,
        "confirmations": 1,
        "instantlock": true,
        "instantlock_internal": false,
        "chainlock": true,
        "blockhash": "0000000001d0c2b2139aec56e7f81251332457ab68b6242a46014dd0cd7f7909",
        "blockindex": 1,
        "blocktime": 1558119666,
        "txid": "a84b2f871aff315153c2a7553dfcf73ae1936f591fb7346ad0e801995f18a585",
        "walletconflicts": [],
        "time": 1558119292,
        "timereceived": 1558119292
}]
```

★ GetTransaction: The GetTransaction RPC gets detailed information about an in-wallet transaction.

```
dash-cli -testnet gettransaction \
              a84b2f871aff315153c2a7553dfcf73ae1936f591fb7346ad0e801995f18a585
```


```
{
        "amount": 143.67890000,
        "confirmations": 0,
        "instantlock": true,
        "instantlock_internal": true,
        "chainlock": false,
        "trusted": true,
        "txid": "a84b2f871aff315153c2a7553dfcf73ae1936f591fb7346ad0e801995f18a585",
        "walletconflicts": [],
        "time": 1558119292,
        "timereceived": 1558119292,
        "details": [{
                "account": "",
                "address": "ycT8D2LiW5QPDUkvTG1fgACHwgRFiR158P",
                "category": "receive",
                "amount": 143.67890000,
                "label": "",
                "vout": 0
        }],
        "hex": " … "
}
```

```
{
        "amount": 143.67890000,
        "confirmations": 1,
        "instantlock": true,
        "instantlock_internal": false,
        "chainlock": true,
        "blockhash": "0000000001d0c2b2139aec56e7f81251332457ab68b6242a46014dd0cd7f7909",
        "blockindex": 1,
        "blocktime": 1558119666,
        "txid": "a84b2f871aff315153c2a7553dfcf73ae1936f591fb7346ad0e801995f18a585",
        "walletconflicts": [],
        "time": 1558119292,
        "timereceived": 1558119292,
        "details": [{
                "account": "",
                "address": "ycT8D2LiW5QPDUkvTG1fgACHwgRFiR158P",
                "category": "receive",
                "amount": 143.67890000,
                "label": "",
                "vout": 0
        }],
        "hex": " … "
}
```

Appendix: LLMQ Security Model

Long-Lasting Masternode Quorums (DIP6) are formed using the following process:

1. **Initialization**: The members of the new quorum are determined. This process is fully deterministic and results in exactly the same list seen by all members and observers.

2. **Contribution**: Each member of the LLMQ will generate its own contribution to the DKG and then relay it. At the same time, each member will receive the contributions of all other members and verify the individual contributions.

3. **Complaining**: Members have to keep track of other members which sent invalid secret key contributions. In the complaining phase, a complaint message is created based on this information and relayed to the other members of the LLMQ.

4. **Justification**: Each member that was previously complained about must justify for a valid contribution. Justification is only allowed for members who sent a contribution and is not allowed for members previously marked as bad.

5. **Commitment**: Each member must collect all contributions from all members not marked as bad. The members then build the final quorum verification vector from the individual verification vectors of the members. The members also create their own threshold secret key share from the secret key contributions received by all valid members.

6. **Finalization**: All members of the quorum will collect all locally valid premature commitments and build final commitments from these.

7. **Mining**: After final commitments have been propagated in the network, miners will take the final commitment for a DKG session and mine it into the next block.