```
BIP: 68
Layer: Consensus (soft fork)
Title: Relative lock-time using consensus-enforced sequence numbers
Author: Mark Friedenbach <mark@friedenbach.org>
        BtcDrak <btcdrak@gmail.com>
        Nicolas Dorier <nicolas.dorier@gmail.com>
        kinoshitajona <kinoshitajona@gmail.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0068
Status: Final
Type: Standards Track
Created: 2015-05-28
```

Abstract

This BIP introduces relative lock-time (RLT) consensus-enforced semantics of the sequence number field to enable a signed transaction input to remain invalid for a defined period of time after confirmation of its corresponding outpoint.

Motivation

Bitcoin transactions have a sequence number field for each input. The original idea appears to have been that a transaction in the mempool would be replaced by using the same input with a higher sequence value. Although this was not properly implemented, it assumes miners would prefer higher sequence numbers even if the lower ones were more profitable to mine. However, a miner acting on profit motives alone would break that assumption completely. The change described by this BIP repurposes the sequence number for new use cases without breaking existing functionality. It also leaves room for future expansion and other use cases.

The transaction nLockTime is used to prevent the mining of a transaction until a certain date. nSequence will be repurposed to prevent mining of a transaction until a certain age of the spent output in blocks or timespan. This, among other uses, allows bi-directional payment channels as used in Hashed Timelock Contracts (HTLCs) and BIP112.

Specification

This specification defines the meaning of sequence numbers for transactions with an nVersion greater than or equal to 2 for which the rest of this specification relies on.

All references to median-time-past (MTP) are as defined by BIP113.

If bit (1 « 31) of the sequence number is set, then no consensus meaning is applied to the sequence number and can be included in any block under all currently possible circumstances.

If bit (1 « 31) of the sequence number is not set, then the sequence number is interpreted as an encoded relative lock-time.

The sequence number encoding is interpreted as follows:

Bit (1 « 22) determines if the relative lock-time is time-based or block based: If the bit is set, the relative lock-time specifies a timespan in units of 512 seconds granularity. The timespan starts from the median-time-past of the output's previous block, and ends at the MTP of the previous block. If the bit is not set, the relative lock-time specifies a number of blocks.

The flag (1«22) is the highest order bit in a 3-byte signed integer for use in bitcoin scripts as a 3-byte PUSHDATA with OP_CHECKSEQUENCEVERIFY (BIP 112).

This specification only interprets 16 bits of the sequence number as relative lock-time, so a mask of 0x0000ffff MUST be applied to the sequence field to extract the relative lock-time. The 16-bit specification allows for a year of relative lock-time and the remaining bits allow for future expansion.

For time based relative lock-time, 512 second granularity was chosen because bitcoin blocks are generated every 600 seconds. So when using block-based or time-based, the same amount of time can be encoded with the available number of bits. Converting from a sequence number to seconds is performed by multiplying by 512 = 2ˆ9, or equivalently shifting up by 9 bits.

When the relative lock-time is time-based, it is interpreted as a minimum block-time constraint over the input's age. A relative time-based lock-time of zero indicates an input which can be included in any block. More generally, a relative time-based lock-time n can be included into any block produced 512 * n seconds after the mining date of the output it is spending, or any block thereafter. The mining date of the output is equal to the median-time-past of the previous block which mined it.

The block produced time is equal to the median-time-past of its previous block.

When the relative lock-time is block-based, it is interpreted as a minimum block-height constraint over the input's age. A relative block-based lock-time of zero indicates an input which can be included in any block. More generally, a relative block lock-time n can be included n blocks after the mining date of the output it is spending, or any block thereafter.

The new rules are not applied to the nSequence field of the input of the coinbase transaction.

Implementation

A reference implementation is provided by the following pull request

https://github.com/bitcoin/bitcoin/pull/7184

```
enum {
    /* Interpret sequence numbers as relative lock-time constraints. */
    LOCKTIME_VERIFY_SEQUENCE = (1 << 0),
};

/* Setting nSequence to this value for every input in a transaction
 * disables nLockTime. */
static const uint32_t SEQUENCE_FINAL = 0xffffffff;

/* Below flags apply in the context of BIP 68*/
/* If this flag set, CTxIn::nSequence is NOT interpreted as a
 * relative lock-time. */
static const uint32_t SEQUENCE_LOCKTIME_DISABLE_FLAG = (1 << 31);

/* If CTxIn::nSequence encodes a relative lock-time and this flag
 * is set, the relative lock-time has units of 512 seconds,
 * otherwise it specifies blocks with a granularity of 1. */
static const uint32_t SEQUENCE_LOCKTIME_TYPE_FLAG = (1 << 22);

/* If CTxIn::nSequence encodes a relative lock-time, this mask is
 * applied to extract that lock-time from the sequence field. */
static const uint32_t SEQUENCE_LOCKTIME_MASK = 0x0000ffff;

/* In order to use the same number of bits to encode roughly the
 * same wall-clock duration, and because blocks are naturally
 * limited to occur every 600s on average, the minimum granularity
 * for time-based relative lock-time is fixed at 512 seconds.
 * Converting from CTxIn::nSequence to seconds is performed by
 * multiplying by 512 = 2^9, or equivalently shifting up by
 * 9 bits. */
static const int SEQUENCE_LOCKTIME_GRANULARITY = 9;

/**
 * Calculates the block height and previous block's median time past at
 * which the transaction will be considered final in the context of BIP 68.
 * Also removes from the vector of input heights any entries which did not
 * correspond to sequence locked inputs as they do not affect the calculation.
 */
static std::pair<int, int64_t> CalculateSequenceLocks(const CTransaction &tx, int flags, std
{
    assert(prevHeights->size() == tx.vin.size());

    // Will be set to the equivalent height- and time-based nLockTime
    // values that would be necessary to satisfy all relative lock-
    // time constraints given our view of block chain history.
    // The semantics of nLockTime are the last invalid height/time, so
    // use -1 to have the effect of any height or time being valid.
    int nMinHeight = -1;
    int64_t nMinTime = -1;

    // tx.nVersion is signed integer so requires cast to unsigned otherwise
    // we would be doing a signed comparison and half the range of nVersion
    // wouldn't support BIP 68.
    bool fEnforceBIP68 = static_cast<uint32_t>(tx.nVersion) >= 2
                      && flags & LOCKTIME_VERIFY_SEQUENCE;

    // Do not enforce sequence numbers as a relative lock time
    // unless we have been instructed to
    if (!fEnforceBIP68) {
        return std::make_pair(nMinHeight, nMinTime);
    }

    for (size_t txinIndex = 0; txinIndex < tx.vin.size(); txinIndex++) {
        const CTxIn& txin = tx.vin[txinIndex];

        // Sequence numbers with the most significant bit set are not
        // treated as relative lock-times, nor are they given any
        // consensus-enforced meaning at this point.
        if (txin.nSequence & CTxIn::SEQUENCE_LOCKTIME_DISABLE_FLAG) {
            // The height of this input is not relevant for sequence locks
            (*prevHeights)[txinIndex] = 0;
            continue;
        }

        int nCoinHeight = (*prevHeights)[txinIndex];

        if (txin.nSequence & CTxIn::SEQUENCE_LOCKTIME_TYPE_FLAG) {
            int64_t nCoinTime = block.GetAncestor(std::max(nCoinHeight-1, 0))->GetMedianTime
            // NOTE: Subtract 1 to maintain nLockTime semantics
            // BIP 68 relative lock times have the semantics of calculating
            // the first block or time at which the transaction would be
            // valid. When calculating the effective block time or height
            // for the entire transaction, we switch to using the
            // semantics of nLockTime which is the last invalid block
            // time or height.  Thus we subtract 1 from the calculated
            // time or height.

            // Time-based relative lock-times are measured from the
            // smallest allowed timestamp of the block containing the
            // txout being spent, which is the median time past of the
            // block prior.
            nMinTime = std::max(nMinTime, nCoinTime + (int64_t)((txin.nSequence & CTxIn::SE
        } else {
            nMinHeight = std::max(nMinHeight, nCoinHeight + (int)(txin.nSequence & CTxIn::SE
        }
    }

    return std::make_pair(nMinHeight, nMinTime);
}

static bool EvaluateSequenceLocks(const CBlockIndex& block, std::pair<int, int64_t> lockPair
{
    assert(block.pprev);
    int64_t nBlockTime = block.pprev->GetMedianTimePast();
    if (lockPair.first >= block.nHeight || lockPair.second >= nBlockTime)
        return false;

    return true;
}

bool SequenceLocks(const CTransaction &tx, int flags, std::vector<int>* prevHeights, const C
{
    return EvaluateSequenceLocks(block, CalculateSequenceLocks(tx, flags, prevHeights, block
}

bool CheckSequenceLocks(const CTransaction &tx, int flags)
{
    AssertLockHeld(cs_main);
    AssertLockHeld(mempool.cs);

    CBlockIndex* tip = chainActive.Tip();
    CBlockIndex index;
    index.pprev = tip;
    // CheckSequenceLocks() uses chainActive.Height()+1 to evaluate
    // height based locks because when SequenceLocks() is called within
    // ConnectBlock(), the height of the block *being*
    // evaluated is what is used.
    // Thus if we want to know if a transaction can be part of the
    // *next* block, we need to use one more than chainActive.Height()
    index.nHeight = tip->nHeight + 1;

    // pcoinsTip contains the UTXO set for chainActive.Tip()
    CCoinsViewMemPool viewMemPool(pcoinsTip, mempool);
    std::vector<int> prevheights;
    prevheights.resize(tx.vin.size());
    for (size_t txinIndex = 0; txinIndex < tx.vin.size(); txinIndex++) {
        const CTxIn& txin = tx.vin[txinIndex];
        CCoins coins;
        if (!viewMemPool.GetCoins(txin.prevout.hash, coins)) {
            return error("%s: Missing input", __func__);
        }
        if (coins.nHeight == MEMPOOL_HEIGHT) {
            // Assume all mempool transaction confirm in the next block
            prevheights[txinIndex] = tip->nHeight + 1;
        } else {
            prevheights[txinIndex] = coins.nHeight;
        }
    }

    std::pair<int, int64_t> lockPair = CalculateSequenceLocks(tx, flags, &prevheights, index
    return EvaluateSequenceLocks(index, lockPair);
}
```

Acknowledgments

Credit goes to Gregory Maxwell for providing a succinct and clear description of the behavior of this change, which became the basis of this BIP text.

This BIP was edited by BtcDrak, Nicolas Dorier and kinoshitajona.

Deployment

This BIP is to be deployed by "versionbits" BIP9 using bit 0.

For Bitcoin **mainnet**, the BIP9 **starttime** will be midnight 1st May 2016 UTC (Epoch timestamp 1462060800) and BIP9 **timeout** will be midnight 1st May 2017 UTC (Epoch timestamp 1493596800).

For Bitcoin **testnet**, the BIP9 **starttime** will be midnight 1st March 2016 UTC (Epoch timestamp 1456790400) and BIP9 **timeout** will be midnight 1st May 2017 UTC (Epoch timestamp 1493596800).

This BIP must be deployed simultaneously with BIP112 and BIP113 using the same deployment mechanism.

Compatibility

The only use of sequence numbers by the Bitcoin Core reference client software is to disable checking the nLockTime constraints in a transaction. The semantics of that application are preserved by this BIP.

As can be seen from the specification section, a number of bits are undefined by this BIP to allow for other use cases by setting bit (1 « 31) as the remaining 31 bits have no meaning under this BIP. Additionally, bits (1 « 23) through (1 « 30) inclusive have no meaning at all when bit (1 « 31) is unset.

Additionally, this BIP specifies only 16 bits to actually encode relative lock-time meaning a further 6 are unused (1 « 16 through 1 « 21 inclusive). This allows the possibility to increase granularity by soft-fork, or for increasing the maximum possible relative lock-time in the future.

The most efficient way to calculate sequence number from relative lock-time is with bit masks and shifts:

```
// 0 <= nHeight < 65,535 blocks (1.25 years)
nSequence = nHeight;
nHeight = nSequence & 0x0000ffff;

// 0 <= nTime < 33,554,431 seconds (1.06 years)
nSequence = (1 << 22) | (nTime >> 9);
nTime = (nSequence & 0x0000ffff) << 9;
```

References

Bitcoin mailing list discussion: https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg07864.html

BIP9: https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki

BIP112: https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki

BIP113: https://github.com/bitcoin/bips/blob/master/bip-0113.mediawiki

Hashed Timelock Contracts (HTLCs): https://github.com/ElementsProject/lightning/raw/master/doc/deployable-lightning.pdf