```
BIP: 325
Layer: Applications
Title: Signet
Author: Karl-Johan Alm <karljohan-alm@garage.co.jp>
        Anthony Towns <aj@erisian.com.au>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0325
Status: Proposed
Type: Standards Track
Created: 2019-03-20
License: CC0-1.0
```

Abstract

A new type of test network where signatures are used in addition to proof of work for block progress, enabling much better coordination and robustness (be reliably unreliable), for persistent, longer-term testing scenarios involving multiple independent parties.

Motivation

Testnet is a great place to try out new things without risking real money, but it is notoriously unreliable. Huge block reorgs, long gaps in between blocks being mined or sudden bursts of blocks in rapid succession mean that realistic testing of software, especially involving multiple independent parties running software over an extended period of time, becomes infeasible in practice.

A new type of test network would be more suitable for integration testing by organizations such as exchanges, or testing of next generation Layer-2 protocols like Eltoo or sidechain pegs. The goal is not to be perfectly reliable but rather to have a predictable amount of unreliability. You want a test network to behave like mainnet (i.e. no thousands of block reorgs) while also making it easier to trigger expected but rare events like a 6-block reorg. Regtest is not suitable for longer-term scenarios involving multiple independent parties because creating blocks costs nothing, so any party can completely control the test network.

Specification

A new type of network ("signet"), which takes an additional consensus parameter called the challenge (scriptPubKey). The challenge can be a simple pubkey (P2PKH style), or a k-of-n multisig, or any other script you would want.

Signet requires all blocks to have a BIP 141 commitment in the coinbase transaction. In order to provide a non-empty solution to the block challenge the block's BIP 141 commitment's optional data must include an additional commitment of the signature/solution for the block:

```
   1-5 bytes - Push the following (4 + x + y) bytes
4 bytes - Signet header (0xecc7daa2)
x bytes - scriptSig
y bytes - scriptWitness
```

In the special case where an empty solution is valid (ie scriptSig and scriptWitness are both empty) this additional commitment can optionally be left out. This special case is to allow non-signet-aware block generation code to be used to test a custom signet chain where the challenge is trivially true.

The scriptSig is serialized by first encoding its length as CompactSize. The scriptWitness stack is serialized as described in BIP 141.

Any push operations that do not start with the 4 byte Signet header are ignored. Multiple push operations with the 4 byte Signet header are ignored except for the first instance of the header.

To sign the block or verify a block signature, two virtual transactions, each with a single input and output are constructed from the block as follows.

The "to_spend" transaction is:

```
   nVersion = 0
nLockTime = 0
vin[0].prevout.hash = 0000...000
vin[0].prevout.n = 0xFFFFFFFF
vin[0].nSequence = 0
vin[0].scriptSig = OP_0 PUSH72[ block_data ]
vin[0].scriptWitness = []
vout[0].nValue = 0
vout[0].scriptPubKey = signet_challenge
```

where block_data is the serialization of the block's nVersion, hashPrevBlock, signet_merkle_root, and nTime. The `signet_merkle_root` is obtained by generating the merkle root of the block transactions, after modifying the coinbase witness commitment by replacing the signet solution with an empty solution (that is, the witness commitment includes a four byte push of the Signet header with no additional solution data, and no prior pushes beginning with the Signet header). This means the merkle root of the block is different from the merkle root in the signet commitment. This is needed, because the signature can never be included in the very message (in this case, a block) that is being signed.

The "to_sign" transaction is:

```
   nVersion = 0
nLockTime = 0
vin[0].prevout.hash = to_spend.txid
vin[0].prevout.n = 0
vin[0].nSequence = 0
vin[0].sigScript = [ signet_solution sigScript (x bytes), if any ]
```

```
vin[0].scriptWitness = [ signet_solution scriptWitness (y bytes), if any ]
vout[0].nValue = 0
vout[0].scriptPubKey = OP_RETURN
```

The scriptSig and/or scriptWitness for `vin[0]` are filled in from the Signet header push above.

To simplify block generation (mining), the signature also does not commit to the block nonce value, so that rolling the nonce to generate proof-of-work does not also require regenerating signatures. When grinding proof of work, the extended nonce cannot be used as it would invalidate the signature. Instead, simply resigning the same (or an updated) block will give a new search space.

A block is considered fully validated only if the to_sign transaction is a valid spend of the to_spend transaction. It is recommended that this verification is done directly before or after the witness commitment verification, as the data required to do both is approximately the same.

There is one other acceptable special case: if a block's challenge is e.g. ‘OP_TRUE‘ (‘0x51‘), where an empty solution would result in success, the block is also considered valid if the signet commitment is absent.

Genesis Block and Message Start

The genesis block is the same for all signet networks, whereas the message start is defined as the first four bytes of the sha256d of the challenge script as a single data push (see below).

Genesis Block

- Time stamp: 1598918400
- Nonce: 52613770
- Difficulty: 0x1e0377ae
- Version: 1

The resulting genesis block hash is 00000008819873e925422c1ff0f99f7cc9bbb232af63a077a480a3633bee1ef6,
and the block hex is 010000000000000000000000000000000000000000000000000000000000000000000000000000003ba3edfd7

Message Start

The message start is defined as the first four bytes of the sha256d of the challenge script, as a single push (i.e. prefixed with the challenge script length). Example:

- Challenge script = 512103ad5e0edad18cb1f0fc0d28a3d4f1f3e445640337489abb10404f2d1e086be43051ae
- Sha256d(len || challenge script) = sha256d(25512103ad...51ae) = 7ec653a59b1912f9db10da2c461ed827d48f9404d5ef0346a6c94aadd4203646
- First four bytes = the message start = 7ec653a5

Compatibility

This specification is backwards compatible in the sense that existing software can use Signet out of the box.

Simply by adding the network parameters for signet (magic number, etc), a client can connect to and use any signet network without further modifications. The block headers have valid proof of work, so clients can trivially check that blocks are "probably" valid.

However, anyone can mine blocks that are accepted by the client for any given signet network. These blocks do not contain the required signatures, however, so any fully validating node will promptly reject them. As such, clients need to either validate the block signature inside the coinbase transaction, or connect to trusted peers.

Other software need not add block signature validation code that they will not use in production. This is adequate for non-production test purposes where the goal is to have a network behave as much like mainnet as possible.

Reference implementation

Pull request at https://github.com/bitcoin/bitcoin/pull/18267

References

1. Original mailing list thread: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2019-March/016734.html
2. Bitcoin Wiki entry: https://en.bitcoin.it/wiki/Signet

Copyright