




Bitcoin's Security Model: A Deep Dive

Jameson Lopp   
© Nov 13, 2016 at 14:57 UTC

FEATURE



When [discussing consensus mechanisms](#) for different cryptocurrencies, one issue that often causes arguments is a lack of understanding (and definition) of the security model that they provide for the historical data in the ledger. While each consensus model aims to prevent various theoretical attacks, it's important to understand the goals for the model.

Every security model has two main parts: assumptions and guarantees. If the assumptions used as inputs hold true, then so should the guarantees that are output by the model.

Let's dig into the security model that appears to be offered to bitcoin users who run a full node.

In search of truth

"One of bitcoin's strengths – the most important in my opinion even – is the low degree of trust you need in others." – Pieter Wuille

The goal of distributed ledgers is to provide an ordered history of events, because in distributed systems you can't simply trust a timestamp.

When a new participant on a blockchain-based network joins, they download any available blocks and consider every valid series of blocks that they see, starting from a hard-coded genesis block.

assumptions made by bitcoin's security model is that the majority of miners are honest – that they are working to secure the network and attempting to undermine it. In practice, this has held true throughout bitcoin's history due to miner incentives, though [some question](#) if it will continue to hold true in the future.

Given this assumption, full node operators can be completely sure of several facts:

- Nobody has inflated the monetary supply except for miners, and only according to a well-defined schedule.
- Nobody ever spent money without having the appropriate private key(s).
- Nobody ever spent the same money twice.

Full node operators can be reasonably sure of several other things. There is a strong guarantee that:

- Any block in the chain was created within approximately two hours of the block's timestamp.
- They are syncing the "true" blockchain history.

At a more technical level, this requires a multitude of checks:

- All blocks follow the consensus rules:
 - Each block is [chained to a parent block](#)
 - Each block met its difficulty target and has [sufficient proof of work](#)
 - Block timestamps fall in a [window relative to recent blocks](#)
 - The Merkle root [matches the block's transactions](#)
 - No blocks were [larger than the allowed maximum size](#)
 - Each block's first (and only first) transaction [is a coinbase transaction](#)
 - Coinbase outputs don't pay more than the [appropriate block reward](#)
 - No blocks contained more than the [allowed signature operations](#)
- All transactions follow the consensus rules:
 - Input and output values [are sane](#)
 - Transactions [only spend unspent outputs](#)
 - All inputs being spent [have valid signatures](#)
 - No coinbase transaction outputs were spent [within 100 blocks of their creation](#).
 - No transactions [spend inputs with a locktime](#) before the block in which they are confirmed.
- [many other rules](#) that would take too long to cover here.



Thermodynamic security

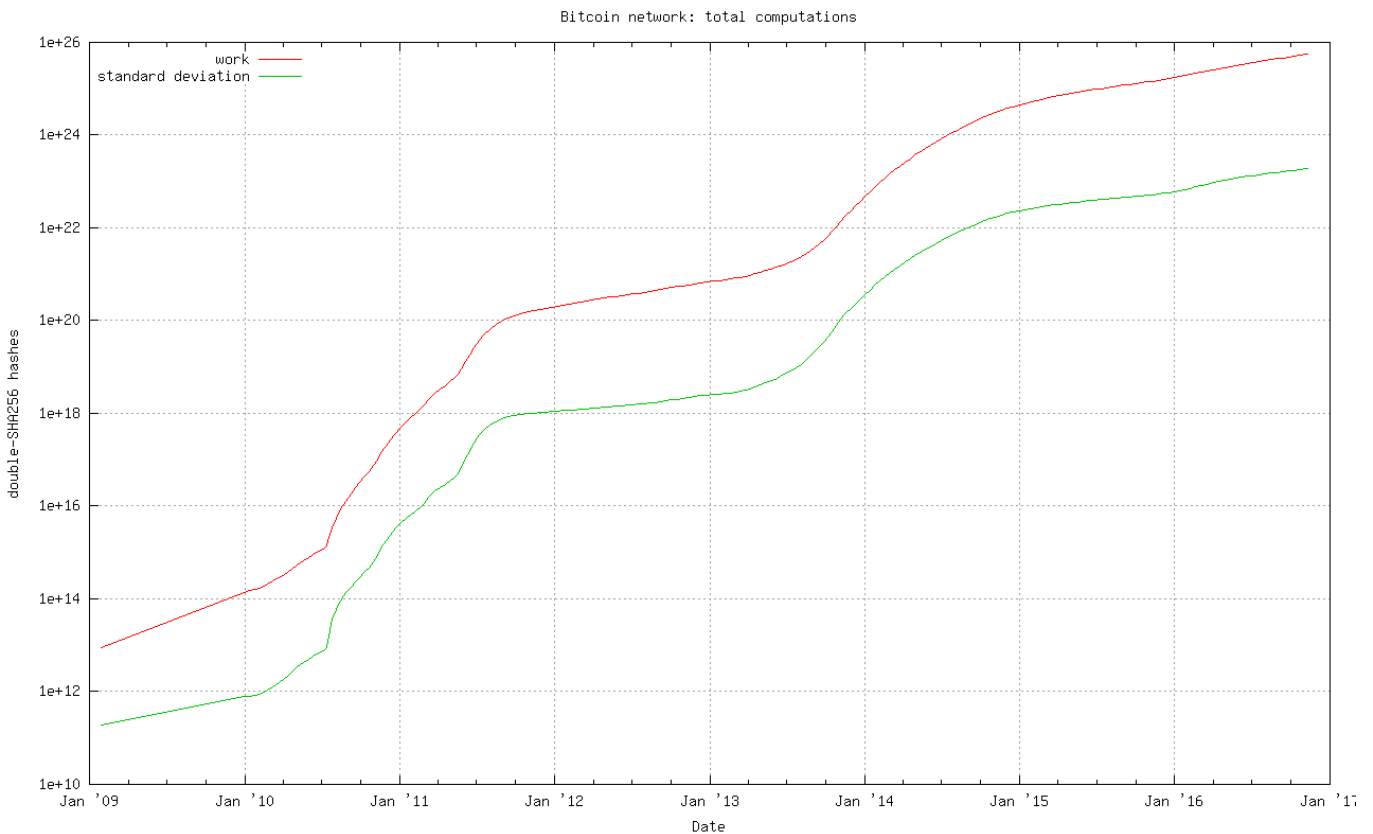
Once a transaction is confirmed in a block it can't be reversed without someone expending a minimum amount of energy to rewrite the chain.

As long as no attacker holds more than 50% of the network's computational power, and honest nodes can communicate quickly, the probability of a transaction being reversed decreases exponentially with the number of confirmations it has received. There are other attacks, [such as selfish mining](#), that can reduce this power requirement, though they appear to be difficult to perpetrate.

	1	2	3	4	5	6	7	8	9	10
2%	0.24%	0.02%	≈0%	≈0%	≈0%	≈0%	≈0%	≈0%	≈0%	≈0%
6%	2.16%	0.42%	0.09%	0.02%	≈0%	≈0%	≈0%	≈0%	≈0%	≈0%
10%	5.98%	1.85%	0.60%	0.20%	0.07%	0.03%	≈0%	≈0%	≈0%	≈0%
14%	11.66%	4.88%	2.11%	0.93%	0.42%	0.19%	0.09%	0.04%	0.02%	≈0%
18%	19.13%	9.94%	5.32%	2.90%	1.60%	0.89%	0.50%	0.28%	0.16%	0.09%
22%	28.27%	17.33%	10.89%	6.95%	4.48%	2.91%	1.91%	1.25%	0.83%	0.55%
26%	38.90%	27.17%	19.36%	13.97%	10.17%	7.45%	5.49%	4.06%	3.01%	2.23%
30%	50.70%	39.33%	30.98%	24.64%	19.73%	15.88%	12.84%	10.41%	8.46%	6.89%
34%	63.23%	53.37%	45.55%	39.14%	33.81%	29.31%	25.49%	22.21%	19.39%	16.95%
38%	75.80%	68.45%	62.25%	56.85%	52.09%	47.85%	44.03%	40.58%	37.45%	34.56%
42%	87.35%	83.09%	79.31%	75.86%	72.68%	69.72%	66.95%	64.33%	61.83%	59.44%
46%	96.26%	94.88%	93.61%	92.41%	91.27%	90.17%	89.10%	88.05%	86.99%	85.82%
48%	98.98%	98.59%	98.23%	97.88%	97.54%	97.21%	96.88%	96.54%	96.15%	95.60%
50%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Source: "Bitcoin's Security Model Revisited" by Yonatan Sompolinsky and Aviv Zohar

Looking at the current cumulative work performed by bitcoin miners, it would take nearly 10^{26} hashes to build an alternative blockchain from genesis with greater cumulative proof of work that full nodes would consider to be the "true" chain.



Source: <http://bitcoin.sipa.be>

To crunch some numbers on the cost involved in such an attack:

An Antminer S9 runs at 0.1 Joule per GH (10^9 hashes)

$$10^{26} \text{ hashes} * 0.1 \text{ J} / 10^9 \text{ hashes} = 10^{15} \text{ joules}$$

$$10^{15} \text{ joules} = 2,777,777,778 \text{ kw hours} * \$0.10 \text{ kw/hour} = \$277,777,778 \text{ worth of electricity to rewrite the entire blockchain}$$

Whereas at time of writing a single block must hit a difficulty target of 253,618,246,641 which would require approximately:

$$253,618,246,641 * 2^{48} / 65535 = 1.09 * 10^{21} \text{ hashes}$$

$$1.09 * 10^{21} \text{ hashes} * 0.1 \text{ J} / 10^9 \text{ hashes} = 1.09 * 10^{11} \text{ joules}$$

$$1.09 * 10^{11} \text{ joules} = 30,278 \text{ kw hours} * \$0.10 \text{ kw/hour} = \$3,028 \text{ worth of electricity per block}$$

This is why we can state that bitcoin is provably thermodynamically secure.

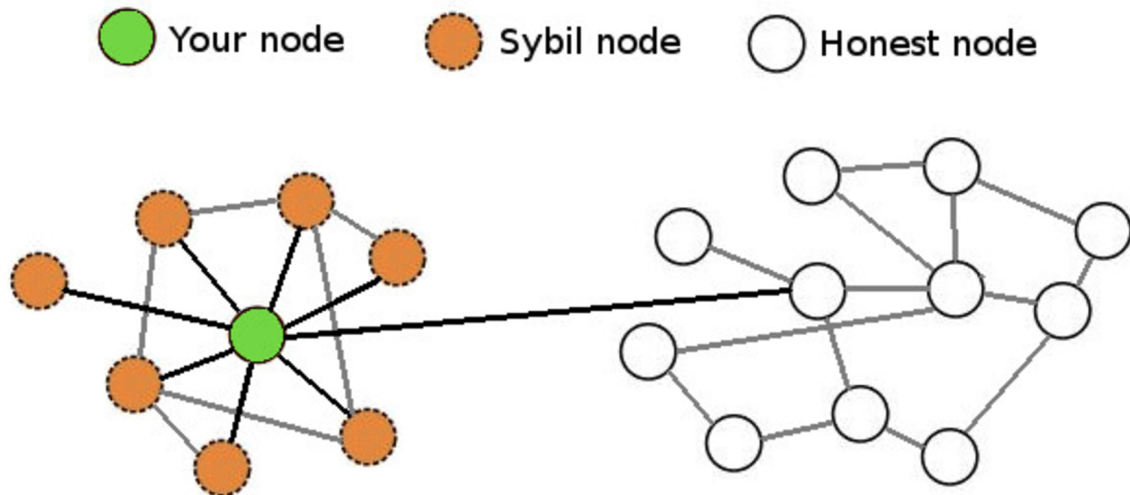
les that you can tweak in the above calculation to decrease the cost, but we can be sure that it will require many millions of dollars in order to rewrite the entire blockchain. However, an attacker with this much hash power would at worst be able to reverse transactions back to 2014 – we'll delve into the reason for this shortly.

Also note that this doesn't take into account the costs required to obtain and operate sufficient mining equipment to carry out such an attack.

Sybil resistance

Because the bitcoin protocol considers the true chain to be the one with the most cumulative proof of work (not the longest chain as is often incorrectly stated,) the result is that a new peer joining the network only needs to connect to a single honest peer in order to find the true chain.

This is also known as "Sybil resistance," which means that it's not possible for someone to launch an attack against a node by creating many dishonest peers that feed it false information.



Pictured here is a near worst-case scenario in which your node is being massively Sybil attacked but still has a single connection with an honest node that is connected to the true bitcoin network. As long as a single honest peer is passing the true blockchain data to your full node, it will be quite clear that any Sybil attackers are attempting to deceive you and your node will ignore them.

Real-time consensus

The bitcoin protocol creates a number of other interesting attributes with regard to maintaining network-wide consensus once your node is at the tip of the blockchain.

The authors of "*Research Perspectives and Challenges for Bitcoin and Cryptocurrencies*" note the following properties that are important to the stability of a cryptocurrency:

Eventual consensus. At any time, all compliant nodes agree upon a prefix of what will become the eventual "true" blockchain.

Exponential convergence. The probability of a fork of depth n is $O(2^{-n})$. This gives users high confidence that a simple "k confirmations" rule will ensure their transactions are settled permanently.

Liveness. New blocks will continue to be added and valid transactions with appropriate fees will be included in the blockchain within a reasonable amount of time.

Correctness. All blocks in the chain with the most cumulative proof of work will only include valid transactions.

Fairness. A miner with $X\%$ of the network's total computational power will mine approximately $X\%$ of blocks.

The authors of the paper note that bitcoin appears to have these properties, at least under the assumption that the majority of miners are remaining honest, which is what the block rewards along with proof of work attempt to incentivize.

There are many other algorithms that can be used to maintain consensus in distributed systems such as:

- Proof of Stake
- Proof of Coin Age
- Proof of Deposit
- Proof of Burn
- Proof of Activity

Proof of Elongated Time

nsensus

- Practical Byzantine Fault Tolerance

These create different security models – the most obvious difference from proof of work being that each of the alternative systems' consensus is driven at the expense of internal resources (coins or reputation) rather than external resources (electricity.) This creates a very different set of incentives for (and trust in) validators on the network which drastically changes the security model.

Security model misunderstandings

A common mistaken assumption is that there is a well-defined security model for bitcoin.

In reality, the bitcoin protocol was and is being built without a formally defined specification or security model. The best that we can do is to study the incentives and behavior of actors within the system in order to better understand and attempt to describe it.

That said, there are a few properties of the bitcoin protocol that are often analyzed incorrectly.

Some blockchains have suffered badly enough from attacks that developers add [centrally broadcasted signed checkpoints](#) into the node software, essentially saying that "block X has been validated by the developers as being on the correct historical chain." This is a point of extreme centralization.

It's worth noting that bitcoin has [13 hard-coded checkpoints](#), but they do not change the security model in the way that broadcasted checkpoints do. The last checkpoint was added to [Bitcoin Core 0.9.3](#) and is at block 295000, which was created on April 9, 2014. This block had a difficulty of 6,119,726,089 which would require approximately:

$$6,119,726,089 * 2^{48} / 65535 = 2.62 * 10^{19} \text{ hashes}$$

$$2.62 * 10^{19} \text{ hashes} * 0.1 \text{ J} / 10^9 \text{ hashes} = 2.62 * 10^9 \text{ joules}$$

$$2.62 * 10^9 \text{ joules} = 728 \text{ kw hours} * \$0.10 \text{ kw/hour} = \$73 \text{ worth of electricity to generate}$$

Thus, if a Sybil attacker completely surrounded a new node that was syncing from scratch, it could create some short blockchains at low heights at almost no cost, but only up to the various checkpointed blocks.

If it partitioned a node off the network that had synced past block 295,000 it would be able to start feeding false blocks at the cost of \$73 per block, at least until it hit a difficulty readjustment. However, the further along the victim node had synced, the greater the cost would be for the attacker to create a chain with more cumulative work.

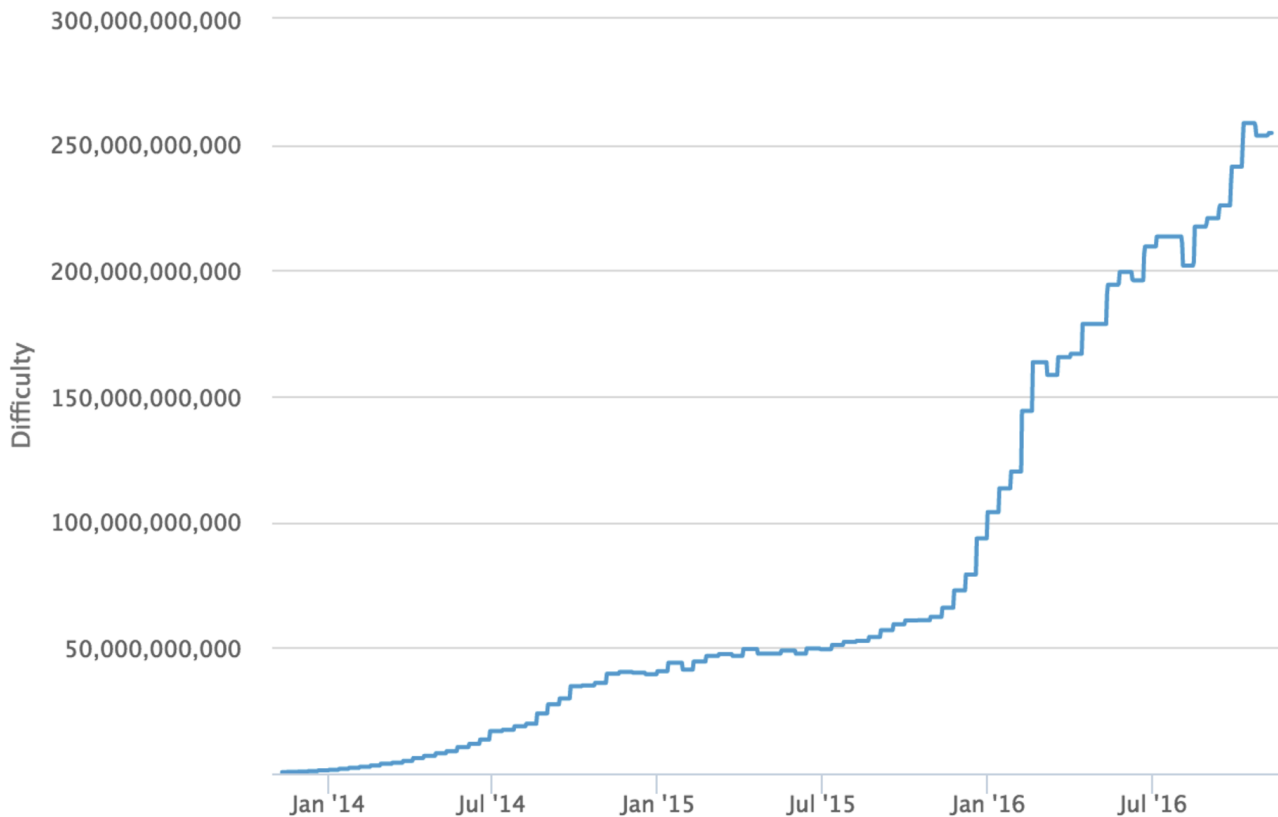
Both [Greg Maxwell](#) and [Pieter Wuille](#) have stated that they hope to someday completely remove checkpoints. Bitcoin Core lead maintainer Wladimir van der Laan noted that checkpoints are a [constant source of confusion](#) to people who seek to understand bitcoin's security model.

An argument could be made that this means a full node is "trusting" the Core devs regarding the validity of the blockchain history up until 9th April, 2014, but the node still checks the Merkle hashes in each block's header, meaning that the soundness of the transaction history is still secured by proof of work. These old checkpoints [enable a performance increase](#) (skipping signature verification) when initially syncing the historical blockchain, though the introduction of libsecp256k1 has made the [performance difference less significant](#).

[Checkpoints remain](#) in place for three purposes:

1. To prevent nodes from [having their memory filled up](#) with valid but low proof-of-work block headers
2. Skipping signatures in earlier blocks (performance improvement)
3. To estimate syncing progress

While this article was being written Greg Maxwell [proposed replacing checkpoints](#) with a [cumulative work check](#) instead. Once a node has a chain that contains more than $5.4 * 10^{24}$ hashes performed, chains with less cumulative work would be rejected. This coincides with the amount of work performed up to approximately block 320,000 in September 2014, at which point individual blocks were of difficulty $\sim 27,000,000,000$.



Source: Blockchain.info

Mining blocks at a difficulty of 27,000,000,000 would require approximately
 $27,000,000,000 * 2^{48} / 65535 = 1.16 * 10^{20}$ hashes
 $1.16 * 10^{20} \text{ hashes} * 0.1 \text{ J} / 10^9 \text{ hashes} = 1.16 * 10^{10}$ joules
 $1.16 * 10^{10} \text{ joules} = 3,222 \text{ kw hours} * \$0.10 \text{ kw/hour} = \$322$ worth of electricity per block

Thus, with this proposed change, if a Sybil attacker completely surrounded a new node that was syncing from scratch, it would be able to start feeding false blocks starting at any block after genesis for basically no cost. If a Sybil attacker completely surrounded a node that synced past block ~320,000 it could start feeding a false chain from that point at the cost of \$322 per block.

In short, either check for securing a node's initial sync is relatively inexpensive to attack if an entity can gain complete control of your node's Internet connection; if they can't, then the node will easily dismiss the attacker's blocks.

On a related note, every blockchain system has its [genesis block hard coded](#) into the node software. You could argue that there is a social contract to the "shared history" that is the ledger – once a block is old enough, there is an understanding amongst everyone on the network that it will never be reverted. As such, when developers take a very old block and create a checkpoint out of it, it is done more so as an agreed-upon sanity check rather than as a dictation of history.

In addition to checkpoints, there's also the matter of how a node bootstraps itself. The current process for bitcoin nodes is to check to see if it has a local database of peers it has previously learned about. If not, then it will query a set of "DNS Seeds" that are [hard-coded into the software](#). These seeds maintain a list of well connected bitcoin nodes that they return to your node.

As we can see from the code, Bitcoin Core 0.13 currently uses DNS Seeds run by Pieter Wuille, Matt Corallo, Luke Dashjr, Christian Decker, Jeff Garzik, and Jonas Schnelli. Anyone can run a DNS seed by using Pieter Wuille's [bitcoin-seeder software](#) or [Matt Corallo's software](#), though in order for it to be used by new nodes you'd have to convince the developers of one of the full node implementations to add your DNS seed host to their software.

It may once again seem like a point of extreme centralization that the bootstrapping process for a new node is reliant upon a mere six DNS seeds. Recall that bitcoin's security model only requires that you connect to a single honest peer in order to be able to withstand Sybil attacks.

As such, a new node only needs to be able to connect to a single DNS seed that isn't compromised and returns IP addresses of honest nodes. However, there is a fallback if for some reason all of the DNS seeds are unreachable – a [hard-coded list](#) of reliable node IP addresses that [gets updated](#) for each release.

The security model for these various initialization parameters is not that the full node operator is trusting X DNS seeds or Y Core developers to feed them honest data, but rather that at least 1 / X DNS seeds is not compromised or 1 / Y Core developers is honest about [reviewing the validity of hard-coded peer changes](#).

Nothing is perfectly secure

At an even deeper level, when you run a full node, you are probably trusting the hardware and software you are running to a certain extent.

Verify the software by [checking the signatures of your binary](#) against those of van der Laan, but it's unlikely that many people bother to do this. As for trustworthy hardware, that's a tough problem. The closest you'll probably come to a secure hardware solution is something like [ORWL](#), which guaranteed to "self destruct" if anyone attempts to tamper with it.



However, given that hardware architectures for CPUs, RAM and other important hardware tend to be proprietary, you can never be 100% sure that they aren't compromised.

Bitcoin's balance of power

The waters become even murkier when you begin to investigate the relationship between different participants in the system.

The purpose of running a full node is to protect your financial sovereignty. This generally means that by installing and running a specific version of software, you are entering into an agreement that you will abide by the rules of that software and that everyone else using the network must also abide by them.

As such, if people want to change the rules in such a way that are not backwards compatible, you must explicitly agree to the rule change by running a new version of the software. On the other hand, backwards compatible rule changes can be implemented and enforced without your consent.

A highly simplified description of the power dynamics in bitcoin:



Jameson Lopp
@lopp

3 branches of BTC "governance:"

- * Full Nodes (can veto miners & devs)
- * Miners (can veto devs)
- * Devs (can help others bypass some vetoes)

4:27 PM - Oct 12, 2016

8
40
64

It's important to note that full node software does not automatically update itself, and this is by design. Automatic updates would greatly shift the balance of power to developers, enabling them to force rule changes upon nodes and miners without their permission.

Unfortunately, while a rule change may be technically backwards compatible, we have come to learn over the years that sufficiently creative soft forks can actually implement changes that are clearly outside the intent of the previous version of rules. Vitalik Buterin [demonstrated this](#) with a description of a way to soft fork bitcoin's block time from 10 minutes to 2 minutes, which would of course also speed up the emission schedule of new bitcoins.

There is one trump card that full nodes have in order to fight back against unwanted soft forks is to hard fork away from the miners who implemented the soft fork. This is difficult to perform (by design) and raises a lot of questions about measuring consensus and finding the economically important nodes.

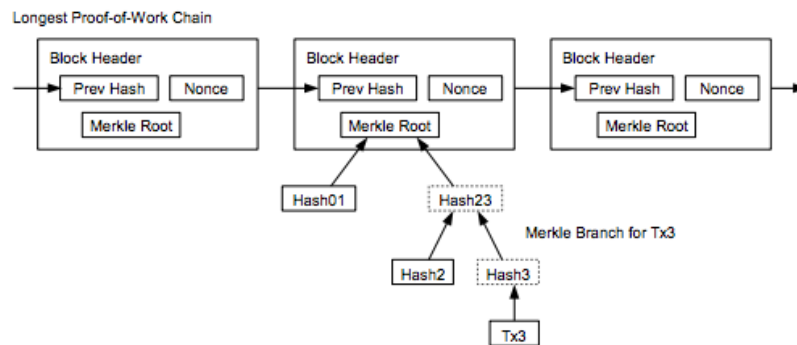
Technically, it could be done by changing the miner algorithm from double SHA256 to a different hash function, thereby rendering all SHA256 ASICs useless for mining bitcoins. It's for this reason that node operators should remain vigilant to changes in the ecosystem and remind miners that they can be replaced if they exceed their authority.

A lot of game theory is involved in discussing miner operations and their threat to bitcoin's security, and I speculated as to how the mining ecosystem may change [in a previous article](#). While bitcoin mining is more centralized than most of us would like, it still seems to work well because bitcoin miners have a lot of capital invested – they can't risk destroying their investment by acting maliciously in a system where everyone is watching.

SDV security

employ a lightweight client to access the network rather than a full node since it requires far fewer resources while still providing

A client employing Simplified Payment Verification (SPV) downloads a complete copy of the headers for all blocks in the entire chain. This means that the download and storage requirements scale linearly with the amount of time since bitcoin was invented. This is described in section 8 of the [bitcoin whitepaper](#).



Satoshi wrote that an SPV client "can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it." SPV assumes that a transaction X blocks deep will be costly to forge.

SPV seems to offer similar guarantees as full node security, but with an additional assumption that any block with a valid header and proof of work always contains valid transactions. Because SPV clients don't check all of the consensus rules noted in the first section of this article, they are making the assumption that the consensus rules are being checked by the node(s) from which they request transactions.

An additional, minor security difference involves peers withholding information from you. When you're running a full node, peers can withhold unconfirmed transactions and blocks from you. However, once you receive a block from any peer, it's not possible for anyone to withhold the transactions in that block from you. On the flip side, it is possible for a peer to give a block header to an SPV client and then withhold information about transactions in that block.

SPV clients can make a query to learn information about transactions affecting a certain address and while it would be costly for peers to lie to them about the existence of fake confirmed transactions (would require mining a block with sufficient PoW) they could lie by omission claim that there were no results for the bloom filter you used to query for transactions. It's also worth noting that SPV is terribly [broken from a privacy standpoint](#) due to flaws with bloom filters.

BitcoinJ has an [excellent write-up](#) of the SPV security model. Regarding unconfirmed transactions, they note:

"In SPV mode, the only reason you have to believe the transaction is valid is the fact that the nodes you connected to relayed the transaction. If an attacker could ensure you were connected to his nodes, this would mean they could feed you a transaction that was completely invalid (spent non-existing money), and it would still be accepted as if it was valid."

SPV security is probably "good enough" for the average user, though it could be improved upon with SPV Fraud Proofs. There has been [some discussion](#) of [this concept](#) but no implemented [proposals](#) for building them into the protocol.

There's no place like 127.0.0.1

If you aren't running a full node (and actually using it to validate transactions) then you're outsourcing at least some level of trust to third parties, resulting in a different security model for your usage of bitcoin. Note that this need not necessitate that all users and businesses build their software directly on top of Bitcoin Core's RPC API.

Some alternate infrastructure configurations might include but are not limited to:

1) Using a mobile wallet such as [Bitcoin Wallet for Android](#), [GreenAddress](#), or [Stash](#) that enables you to configure the wallet to only query your own full node.

SPV synchronization

Enable SPV
Simple Payment Verification allows trusting your transactions and balance thanks to synchronization with Bitcoin network. It adds bandwidth and processing requirements, so you may want to switch it off if on poor connections or low end phones.

Synchronize SPV on mobile
SPV is always synchronized on non mobile networks like WiFi or Ethernet. This setting also synchronizes when using mobile data. This can be expensive depending on your data plan so take care when enabling.

Only connect to trusted node[s] for SPV
Example: 8.90.14.2:8333,40.116.12.85:8333

Settings

Denomination and precision
Unit to show amounts in. This does not affect computations.

Own name
Name of yourself, to be added to payment requests. Try to keep it short.

Auto-close send coins dialog
When the payment is made, the send dialog will close automatically.

Connectivity indicator
Show current number of connected peers in the notification area.

Trusted peer
IP or hostname of single peer to connect to.

2) Building apps on top of SPV node libraries such as BitcoinJ and configuring them to only connect to full nodes that you operate. In BitcoinJ this can be accomplished by defining your own [SeedPeers](#) that you pass to your [PeerGroup](#) during initialization. With libbitcoin you can define a network connection to a specific node [using this example](#).

3) Building a proxy server that is compatible with Bitcoin Core's JSON-RPC API that sends some calls to third party services but also automatically verifies the data they return by making calls to a local full node. For an example, see [BitGo's BitGoD software](#). This hybrid model can give you the best of both worlds: you can leverage advanced features offered by third parties while still retaining your financial sovereignty.

Full nodes for freedom

It's clear that running your own full node offers superior security with the fewest required assumptions. Given that you can build a computer capable of running a reliable full node for only a few hundred dollars, do the math and determine if securing your financial sovereignty is worth the price.

Thanks to Kristov Atlas, Eric Martindale, Andrew Miller, and Kiara Robles for reviewing and providing feedback for this article.

Featured image via [Dan Nott](#) for CoinDesk. Other images as captioned.

The leader in blockchain news, CoinDesk is an independent media outlet that strives for the highest journalistic standards and abides by a [strict set of editorial policies](#). Interested in offering your expertise or insights to our reporting? Contact us at news@coindesk.com.

[Security](#) [Nodes](#)

PREVIOUS ARTICLE



Bitcoin Tests 2016 Highs as Investors Seek Privacy

NEXT ARTICLE



Will Bitcoin Have its Moment in the Trump Era?

RELATED STORIES

Dec 23, 2017 at 13:45 | Taylor Monahan

Poof, Your Money's Gone: Building for Blockchain Users

Dec 21, 2017 at 05:01 | Michael del Castillo

Think Tank Links Rising Bitcoin Price to Terrorist Use