A Treatise on Altcoins

Andrew Poelstra

2016-05-25 (commit 29df316)

Status. This document is still under progress and is missing large sections. Existing text should be correct, so any bug reports are welcome at apoelstra@wpsoftware.net.

1 Preamble

Why am I writing this document? Because when I first entered the world of cryptography, there were certain common-sense maxims which were passed around such as *to design a cryptosystem*, *you must first think like a cryptanalyst* or *anybody can make a cryptosystem which they themselves cannot break*. For most people, these maxims could be summarized simply as *don't roll your own crypto*. Anyone who flouted this golden rule, without decades of schooling and experience, was rightly dismissed as a crank or a troll.

Of course, there were a few people who didn't subscribe, and they would spend years repeating their half-baked ideas, conspiracy theories, factoring algorithms and NSA-proof cryptography on sci.crypt. These people were often ridiculed but just as often sincerely advised to seek mental help. I hope for their sake that some of them have since done so. At no point were their ideas taken seriously, used, or, god forbid, invested in.

However, shortly after the turn of the 21st century, Adam Back discovered a novel type of cryptography called *proof-of-work* which enabled a *distributed consensus* cryptosystem. This cryptosystem was used in 2009 by Satoshi Nakomoto to develop the first decentralized *cryptocurrency* — Bitcoin — which was also the first experimental cryptosystem to see billions of dollars poured into it by people who had no understanding of its mechanisms.

By that time, the benefits of doing cryptography in the open had long since been made clear, so Bitcoin's reference implementation was fully open-sourced. This allowed anybody to see the code, and anybody to fork it to develop their own cryptosystems. Of course, "developing your own cryptosystem" is the purview of only cranks and researchers, so it was reasonably assumed that none of these "altcoins", as they were called, could ever be plausibly presented for public use.

Boy, were we ever wrong on that one.

The purpose of this document is twofold:

 If you are a member of the public interested in cryptocurrencies, this document discusses what cryptocurrencies, and cryptosystems in general, are. It discusses the miracles and dangers of modern cryptography, and the serious risks associated with cryptosystem-tweaking by unqualified (and even qualified!) people. Since Bitcoin has introduced direct monetary value to new cryptosystems, it is not only cranks doing stupid things with it, but also liars and thieves. This document also discusses that side of altcoin development.

2. If you are, or are planning to, develop and release an "altcoin" to the public, this document reminds you that you are playing with fire. This sort of behavior was cute on sci.crypt, a community populated mainly by cryptographic experts where there was no risk that your charlatanism would be mistaken for anything legitimate, and where there was no ability to store value in your scheme anyway.

The Bitcoin community differs in both those respects. Your crankery is not cute. You are not a cryptographer, and yet are releasing a homebrew cryptosystem, misrepresenting your own qualifications, and encouraging others to store value in your creation. These actions are incompetent, dishonest and reprehensibly dangerous.

If somehow you are doing this through honest cluelessness, I dream that you'll read this article and realize the error of your ways.

2 What are cryptosystems?

Modern cryptography, as a field, studies the ability and techniques of controlling information flows independently of containing data flow. For example, using public-key cryptography it is possible to broadcast data such that the information contained is only accessible to a single person.

Until the advent of modern cryptography, philosophical questions, such as "where" the information actually is, were considered just that: philosophical questions. Intuitively, if you write some information down, it's right there on the page in front of you, available to anybody who can read it. In light of this intuition, it is something of a miracle that modern cryptography should be able to exist at all. And given that we evolved this intuition which has served us perfectly well until very recently, it should be expected that modern cryptography is an extremely subtle and perilous practice. Indeed, this is the case.

This cryptographic idea of "separating information flow from data flow" can be put on good mathematical footing, and much progress has been made in this direction ¹, though there are still many fundamental open problems². By reading papers in this field, one gets a sense for the difficulty of making concrete statements about such subtle concepts, and for the precision with which one's assumptions must be made.

A *cryptosystem* is a collection of algorithms which work together to achieve some cryptographic goal. A typical cryptosystem consists of three algorithms: key generation, encryption, and decryption. Cryptosystems are typically published alongside security proofs which reduce some "hard" mathematical problem such as finding a discrete logarithm of an elliptic curve group element to "breaking" the cryptosystem (e.g. learning some bits of the input to the encryption algorithm from

¹See, for example, Shafi Goldwasser and Silvio Micali, Probabilistic Encryption, Special issue of Journal of Computer and Systems Sciences, Vol. 28, No. 2, pages 270-299, April 1984. Available online.

²For example, functions such as SHA256d which are easy to calculate but have unpredictable outputs, are called *one-way functions*. However, SHA256d is merely assumed to be one-way, but no proof has been found — in fact, no proof has been found that *any* one-way functions exist!

its output). These proofs are intricate, subtle, and their relation to reality is a subject of intense controversy³. An important thing to note is that these proofs also consider the cryptosystem as a whole: change one algorithm even slightly, and the security proof of another algorithm could be completely invalidated.

Nonetheless, we interact daily with many cryptosystems, most of which we do not even think about. We assume the security of these systems partially because there is legal recourse if they are broken, partially because this is the way that we've always done things, and it seems like security breaches are reasonably uncommon, and partially because we assume that very smart people designed these systems to be hard to break. After all, there's a lot of money riding on them.

Of course, even if these things are true about the cryptosystems we use to check our email, do our banking, buy groceries and store private data, there is no reason to believe in them for novel cryptosystems designed by anonymous people to do unprecedented things. This is largely responsible for the "crank" label assigned to so many pointless projects on sci.crypt.

3 What are cryptocurrencies?

With the advent of modern cryptography, the idea that *information* can be physically real — and valuable — has moved from the dingy halls of philosophy departments to the concrete world of business. We are all familiar with the economic activity enabled by secure communication: negotiations, contracts, transactions, sales and commands can be sent on the public Internet with no fear of forgery or interception. We are also familiar with the financial consequences when secret data is lost or stolen.

Since the advent of cryptographic currency with Bitcoin in January 2009 [3] this notion of valuable information has been made concrete. It is possible to hold and exchange a *fungible* store of value, using public communication media, with cryptographic rather than physical security preventing fraud or theft. Rather than saying "this encryption key is worth \$10,000 because that's what it will cost us if its encrypted data is exposed" one can say "this key is worth \$10,000 but can be broken up, sending only \$20 of it to another party while keeping the rest".

A *cryptocurrency* is such a cryptosystem, designed to facilitate the transfer of scarce goods defined within the system itself. The prototypical example is Bitcoin, which transfers signing authority and maintains a global ledger of value associated to this authority. The primary innovation of Bitcoin was the creation of this ledger, which is updated and verified in a completely decentralized fashion, with all parties agreeing on the atomicity of transactions and their ordering in time.

Out of necessity, cryptocurrencies are enormous cryptosystems and contain many smaller cryptosystems as components. This makes them fearsomely complex, and their security correspondingly difficult to verify, but the fact that Bitcoin has held up for over five years gives evidence that this complexity can be managed.

Adding to the complexity of the cryptosystem itself is the fact that exchanging value necessarily involves economic considerations. Therefore cryptocurrencies must be analyzed not only for

³Alexander Dent, Fundamental Problems in Provable Security and Cryptography, retrieved from the IACR preprint archive, 2006. http://eprint.iacr.org/2006/278.pdf

computational soundness and security, but also for *economic* soundness and security. That is, is the cryptocurrency designed so that the incentives are aligned with the goal of security the system, and *not* with the goal of undermining it?

To illustrate the complexity of Bitcoin, and to give an overview of its workings (which we will cover in more detail in Sections 5 and 6), we have broken the cryptosystem down into its constituent algorithms. The cryptosystem in its entirety is run by every validating "full" node on the network. We assume the existence of a communications network by which nodes are able to exchange data. (In practice, Bitcoin nodes use a peer-to-peer network, and communicate by the "Bitcoin protocol".)

Such a breaking-down is necessarily subjective and gives an incomplete view of the system⁴, but is didactically necessary. It is important to emphasize that this is *one cryptosystem* and the security (economic and computational) of every component is tied to that of every other. Therefore, anyone hoping to change a single component must understand the entire system and have the technical background to analyse and implement the change.

We now give an overview of each component of Bitcoin, leaving detailed cryptographic discussion to future sections.

Setup. When a Bitcoin node is first started it creates two data structures, a weighted hash tree called the *blockchain* and a database called the *utxo set*, both of which are initially empty. Elements of the blockchain are called *blocks*; elements of the utxo set are called *utxos* or *unspent transaction outputs*. The motivation for these terms will become clear.

It then contacts another node to request the highest-weighted path in its blockchain. For each block in this path (which must start with the so-called *genesis block* whose hash is hardcoded into the node), the node runs its Block Verification algorithm, which updates its chainstate.

Relay. Each time a node sees a transaction on the network, it runs its Transaction Verification algorithm. If this passes, the node passes the transaction to each of its peers (after a small delay, to prevent flooding attacks).

Similarly, each time a node sees a block on the network, it runs its Block Verification algorithm. If this passes, and if the new block is part of the highest-weight blockchain path, the node passes the block to each of its peers.

Signature (Script) Evaluation. Since Bitcoin transactions transfer value, a basic requirement for a transaction to be valid is that the previous owner of the coins has signed off on the transfer. So-called *digital signatures* are well-studied cryptographic primitives, and typically consist of a cryptographic proof that the holder of the private half of some keypair has manipulated a message in some distinctive and easily-verified way.

Since Bitcoin transcations are financial transactions, which are often executions of more complicated contracts than "the sole owner of some coin signed off on this spend". Therefore Bitcoin's signature system contains an expressive stack-based scripting language. Often Bitcoin's script is assessed as though it were a programming language; however, its cryptographic function is to be a

⁴Robert Pirsig, Zen and the Art of Motorcycle Maintenance, 1974

digital signature scheme, and therefore its most essential attributes are that script-based signatures are publically verifiable and existentially unforgeable⁵

This language has the capacity to push and pop data, branch on simple conditions, and also execute some traditional cryptographic primitives. Simple Bitcoin transactions may be little more than thin wrappers around these primitives; for example, traditional "pay-to-address" transactions check (a) that a transaction is signed by a traditional ECDSA signature, and (b) that the correct address can be derived from signature's key.

Transaction Verification. At its heart, a Bitcoin transaction is composed of two main parts: the inputs and the outputs. As the inputs refer to the outputs of other transactions, we cover them first.

Both inputs and outputs are constructed from *scripts*, which are instructions in a Bitcoin-specific stack-based programming language. This language is very small and does not support looping, so that it can be consistently implemented and easily audited.

Outputs are fairly simple: they consist of (a) a value, which is the number of Bitcoins the output represents, and (b) a script which reads values from the stack then either passes or fails. A typical script might expect the stack to contain a digital signature, for example. All that is needed to validate outputs is that their scripts use the defined script opcodes.

Inputs are more intricate: they consist of (a) a reference to an output of an existing transaction and (b) a script which places values on the stack. To validate an input, it is first checked that the referenced transaction output has not been spent, *i.e.* it appears in the utxo set. Then that output's script is concatenated to the input's script, and the concatenated script is run using the Script Evaluation algorithm. If the algorithm accepts, the transaction is valid.

Further, the total value of inputs must be greater than or equal to the total value of the outputs (input values are defined as the values of their referenced outputs). If the input total is strictly greater than the output total, the difference is called a *transaction fee* and is recaptured by the network.

There is one exception to this last rule for so-called *coinbase transactions*. These are special transactions which occur once in each block and may be created with no inputs at all. They are the mechanism by which new Bitcoins are brought into circulation. The total output size must be less than or equal to the *block reward* plus the total network fees for all other transactions in the block.

Transaction Generation. To create a transaction, a node selects outputs from its utxo set which it has the capacity to spend (for example, outputs whose scripts expect a digital signature, and the node is in possession of the requisite key.) It chooses enough outputs so that their total value is greater than or equal to the amount desired to spend.

It then creates new outputs which the transaction's recipient has the capacity to spend (typically this requires contacting the recipient through another channel, *e.g.* to obtain the hash of a public key for which the recipient has the corresponding private key), and sets their values so that the total is equal to the amount desired to spend.

⁵Actually, Bitcoin's script is expressible enough to produce signatures which can be forged with varying degrees of ease; an extreme example would be a transactions whose outputs can be spent by anybody at all. On the other hand, pay-to-address transactions should have outputs which are unspendable except by the target address's owner. So "existential unforgeability" is not quite the correct security requirement for Bitcoin signatures. There is something more subtle here.

Any discrepancy between the total input value and total output value is considered as a network fee. To reduce this, the node may add an additional "change" output, which it has the capacity to spend itself.

Block Verification. To verify a block, the Bitcoin node first checks that it is formed correctly and that the correct hash of its contents is in its header. It also checks that the hash of the block is within a small range — the exact range is calculated by observing the timestamps of the block's ancestors and attempting to adjust so that future blocks will be created roughly every ten minutes. See the Block Generation algorithm for more details about this.

It then runs the Transaction Verification algorithm on every transaction in the block, and if any of them fail, the block is invalid.

It is crucial to the Bitcoin cryptosystem that all nodes agree on the result of the Block Verification algorithm — and by extension, that all nodes agree on Transaction Verification, Script Evaluation, and Difficulty Calculation. That is, these algorithms are *consensus algorithms*. More about this will be discussed in Section 6.

The block is weighted in the blockchain according to its difficulty.

Block Generation. Unlike the previous algorithms, Block Generation is not done by most Bitcoin nodes, since it is designed to be very computationally expensive. Today it requires special-purpose hardware to be feasible.

To create a block, a node assembles a list of transactions, which are obtained through the Bitcoin network and all pass the Transaction Verification algorithm. The node also creates a coinbase transaction, which has no inputs and whose outputs can be spent by the node itself.

These transactions are hashed up, and the resulting hash is put alongside a timestamp and nonce in the *block header*. In order that the new block pass the Block Verification algorithm, its hash must fall into a small range defined by the Difficulty Calculation algorithm. To accomplish this, the block is hashed *ad nauseum* for different values of the nonce until a hash is found which falls into the required range. This computation is called a *proof of work*, and Bitcoin's security depends on it satisfying several subtle mathematical properties, which will be discussed in Section 6.

Difficulty Calculation. There are two reasons that Bitcoin blocks are accompanied by a proof-ofwork. One is to create an opportunity cost for extending the blockchain, forcing would-be attackers to commit to a single branch of the chain. The other is to slow the pace of blockchain extensions so that the entire network can be made aware of each block before it is extended. The proof-ofwork also provides a natural way to weight the each block, so that for every path in the blockchain one can compute the "total work". The higher the total work of a path, the more participants are (statistically) required to participate to create it, and therefore "highest total work" is a proxy both for "known to the most people"⁶ and "hardest to forge". For these reasons, Bitcoin nodes are able to

⁶Pieter Wuille pointed out to me that since the hashpower of the Bitcoin network has increased by many orders of magnitudes, it is only a high *recent* total work that can be used as a proxy for visibility. For example, a modern ASIC miner could easily outdo the total work of the first 200k Bitcoin blocks without publishing anything. On the other hand, the timescale on which hashpower changes by orders of magnitude is much larger than the timescale at which we assume the network is synchronous, so by the time an attacker is able to individually out-work a path, that path will have been extended

achieve a distributed consensus on the "real" blockchain by considering reality to be synonymous with greatest total work⁷.

(Since blockchain paths intrinsically order their contained blocks, consensus on the blockchain immediately gives rise to consensus on transaction ordering, which is what Bitcoin actually needs to resolve double-spend incidents consistently.)

To keep the blockrate low enough that the network has time to achieve consensus, while high enough to facilitate transactions at a useful pace, Bitcoin attempts to produce blocks on average every ten minutes. It accomplishes this by a negative feedback loop between the blocktimes (as encoded in the blocks themselves, which are manipulable by dishonest miners) and the block difficulty.

As described in Section 6, Bitcoin's proof-of-work scheme is that of Adam Back's HashCash, and works by requiring the SHA256d hash of valid blocks' headers to lie within a small range. The difficulty parameter is inversely proportional to the size of this range, and assuming that SHA256d values are uniformly random⁸, the size of this range is directly proportional to the probability that any given block will be valid. Miners churn through the space of possible block headers by incrementing a nonce, so that between any two valid blocks, miners may have collectively churned through many quadrillion invalid ones.

The short version of the above: if blocks' timestamps are too close together then the difficulty increases, meaning that the range of valid hashes shrinks. Conversely, if blocks are too far apart the range grows. The result is a negative feedback loop designed to cause blocks to appear every ten minutes on average.

4 Cryptography is hard.

In this section we will step away from the specifics of cryptocurrencies and look at cryptography in general. Modern cryptography is an exciting but extremely subtle field. Theoretical cryptography sits at the intersection of computing science, algebra, statistics, physics and philosophy, while applied cryptography involves all of the above in addition to software engineering, electrical engineering, social science and economics⁹

[tell stories about "obviously" amateur cryptosystems, e.g. altoz, as well as non-obviously amateur ones, e.g. electrum's encrypt-against-key system, as well as professional ones, e.g. ECDSA's fragilities] [tell stories about serious cryptosystems broken in weird ways, eg TLS 1.0 side-channel attack]

beyond his reach and the extension will already by globally visible. Therefore by the time a difference between the visibility implied by total work and that implied by recent total work develops, that difference is long since irrelevant.

⁷This exposition of the function of proof-of-work is far from being a scientific consensus. What is agreed on is that there are mathematical proofs that cryptographically-enforced distributed consensus is impossible, and that Bitcoin evades these impossibility results by introducing economic concepts such as opportunity costs. It is far from clear what the correct formalization of the security (including forces against censorship and centralization) properties required of a proof-of-work are, and whether Bitcoin's proof-of-work achieves those goals. For more information see Andrew Miller's recent work.

⁸This is the so-called *random-oracle assumption* on the hash function, which is that a hash function can be modeled by a mathematically random function. This is physically impossible since the Komolgorov complexity of a true random function would be infinite while that of SHA256d is comparatively very small. But it's an empirical fact that nobody has found a computationally feasible to skew the SHA256d's output distribution away from uniform.

 $^{^{9}}$ Not to mention the revolutionary historical and political implications of the *use* of cryptography; there is much to be said about these aspects of Bitcoin, which we will try to avoid for the purposes of this text.

5 Cryptography of Bitcoin 1: transactions and signatures.

Bitcoin transactions conceptually work as follows: to spend a Bitcoin, you digitally sign a message that contains (a) the amount to be spent, and (b) a public key of the recipient. Then when the recipient wants to spend money, he has to provide a reference to the transaction that gave it to him, and sign a new message with the key from that transaction.

As discussed above, the actual implementation is more involved in order to support splitting up balances, controlling what data is signed, etc. And because financial contracts are often more complex than "the money of party X now belongs to party Y", Bitcoin's signature algorithm actually contains a simple scripting language

[explain transactions and signatures, CHECKSIG etc] [explore dangerous/bad ideas]

- Turing completeness, script expressiveness
- Extrospection

6 Cryptography of Bitcoin 2: distributed consensus.

A *distributed consensus*, as the term is used in Bitcoin, is a consensus (i.e. global agreement) between mutually-distrusting parties who lack identities and were not necessarily present at the time of system set up. We do allow and require *synchronous communication*; that is, there is some maximum duration Δ after which all valid broadcasted data reaches all parties¹⁰[2]. We do not (and cannot, in an untrusted and physically dispersed network) assume that nodes agree on the precise timing or even time-ordering of messages on the network.

For the purposes of cryptocurrency, it is sufficient to achieve distributed consensus on the timeordering of transactions (and nothing else). This implies consensus on the "first transaction which moves these particular funds", which assures the funds' new owner that the network recognizes them as such.

The reason that this consensus is needed is called the *double-spending problem*. That is, in any decentralized digital currency scheme there is the possibility that a spender might send the same money to two different people, and both spends would appear to be valid. Recipients therefore need a way to be assured that there are no conflicts, or that if there are conflicts, that the network will recognize their version as the correct one. A distributed consensus on transaction ordering achieves this: in the case of conflict, everyone agrees that the transaction which came first is valid while all others are not.

(The other problems with digital currency, e.g. authentication and prevention of forgery, are comparatively easy and can be handled with traditional cryptography, as discussed elsewhere in this document.)

¹⁰To obtain a convergent network, we require only that blocks propagate to the entire network before the next block comes in, on average. For efficiency reasons, we actually want propagation to happen much faster than blocks are produced, and this is what occurs in practice.

Economic Assumptions. Our assumptions on the topology of the Bitcoin network make it wellsuited for distributed consensus. For example, it easily evades the classical FLP impossibility result[1] both by being nondeterministic and by having synchronous communication. However, because participants in Bitcoin are anonymous and do not have registered identities, distributed consensus is in fact a difficult problem (*c.f.* the discussion in [2]). Bitcoin is the first apparent solution to this problem, and achieves this only by weakening its requirement from a cryptographic guarantee to a mere economic one. More specifically, it introduces an opportunity cost from outside of the system (expenditure on computing time and energy) and provides rewards within the system, but only if consensus on an unbroken transaction history is maintained.

This dependence on economic assumptions has two serious consequences:

 Rigorous analysis is made much more difficult. Standard cryptographic proofs assume an attacker who does everything within his (clearly-defined) abilities to break a cryptosystem. Bitcoin, by contrast, requires analysis of the attacker's incentives, mixing well-defined mathematics with unclear economics, and introduces dependencies on the actual distribution of wealth and energy in the world.

It is still unclear how to set this analysis on a solid foundation, and it is even unclear whether this distributed consensus mechanism even works. But Bitcoin has been operating in a somewhat decentralized fashion for over five years, and this gives us hope.

 Standard cryptographic proofs attempt to give honest parties an exponential advantage over dishonest ones. This is why we can produce digital signatures which can be honestly produced by RFID keytags while forging them would require more computing power than exists in the universe.

However, with Bitcoin's distributed consensus, an honest party has only linear advantage over an attacker: it is exactly as easy to behave honestly (writing history) as it is to behave dishonestly (rewriting history). This is why Bitcoin is vulnerable to a so-called 51% attack: as soon as a dishonest party is the majority, he has an advantage over the honest parties.

Mechanisms. The way that Bitcoin achieves distributed consensus is by a hash-based *proof of* $work^{11}$. Bitcoin provides a way to prove, for each candidate history, (a) that opportunity cost was forfeited, and (b) how much. This is a proof-of-work. Furthermore, the work proven includes that of all participants who worked on the history¹² [cite whatever of amiller's lottery stuff is public]. The consensus history is the one with most total work (at least as far as it has propagated through the network — our weak synchronicity requirement means that the consensus on the most recent part of the history is uncertain). Since the consensus history is the only one containing spendable rewards for work done, this means (a) that provers have an incentive to work on the same history

¹¹Bitcoin's proof-of-work was invented by Adam Back as part of Hashcash, a precursor to Bitcoin. See Adam Back, *Hashcash – A Denial of Service Counter-Measure*, technical report, August 2002.

¹²In particular, the work done even by miners who don't find blocks is included, in exactly the same sense that gas molecules in a box contribute to its ambient temperature even if they don't happen to collide with the thermometer during measurement. This is not an analogy. The principles are the same. [cite yet-unwritten article about cryptographic thermodynamics]

that other provers are, and (b) individual provers can't take control of the history because they need their peers' contribution.

6.1 Failure Modes

Distributed consensus systems have new and catastrophic failure modes which do not exist in ordinary systems. The reason that consensus failure is catastrophic is that within a cryptocurrency, time itself is determined by consensus. If consensus is lost, at best time stops and the currency is unusable. At worst, there is disagreement on history and fraud becomes possible.

Just as serious than a complete consensus failure is a partial consensus failure: if the network splits into two or more "factions" who disagree on history, then double-spending becomes easy by putting conflicting transactions into the various chains.

Less serious than consensus failure is the risk of bad incentives, which may cause the consensus to become centralized, discourage non-miners from auditing transactions, discourage miners from including transactions or discourage miners from publishing blocks quickly. Of course this list is not exhaustive.

Even the simple-minded changes made to Bitcoin by typical alt-currencies have been enough to cause these problems. Here are some examples:

- Architecture-dependent consensus. In the early altcoin solidcoin 2.0, difficulty changes were computed using floating-point math. This caused users with different systems to disagree on the correct difficulty, resulting in a badly broken consensus. (The original solidcoin had very rapid difficulty changes, which also caused the consensus to break this problem, followed by its inventor's belligerent and cartoonish response¹³, was the original inspiration for this document.)
- *Poor update management*. It's important to realize that every single piece of code which affects the validity of blocks is a piece of code which must be identical across all consensus nodes. Changing this code requires a carefully planned changeover, including public assertions from a majority of stakeholders that they will switch at the right time.

As an extreme example of failing this, in January 2014, the meme-themed altcoin dogecoin pushed a small-seeming change in a point release of their reference client. This change affected the maximum output size of a dogecoin transaction allowed in a block. Since not all nodes updated, as soon as a transaction appeared which was valid under the new rules but not the old ones, the blockchain forked badly. In true dogecoin silliness, the users quickly jumped onto Reddit and started tipping each other madly, knowing that there was no consensus on who was tipping whom¹⁴.

Since dogecoin has a small and largely Reddit-based community, and since their users do not store much value in the currency, the mess was sorted out by the developers deciding a

¹³For example, the proof-of-work in solidcoin 2.0 involved hashing a personal diatribe written by its inventor about another message board user.

¹⁴The relevant thread is http://www.reddit.com/r/dogecoin/comments/1ufl1e/much_concern_dogecoin_ block_chain_has_split/cehm0yw.

"true history" (i.e. by centralizing the consensus, at least at that point) and posting on Reddit that all users needed to update.

As an aside, it should be noted that many cryptocurrencies have a mechanism for the developers of the reference client to send out emergency messages to the network. For dogecoin this was not possible since emergency messages must be digitally signed by the developers — but when dogecoin copied the source code of litecoin, they forgot to set a new signing key!

• *High block frequency*. A common change in altcoins is to increase the block frequency, due to a misunderstanding of the purpose and meaning of transaction confirmations. This increases the frequency of minor blockchain forks (which result in stale blocks and wasted mining power) and also increases the bandwidth and validation costs of non-mining nodes. Both of these tend to increase centralization.

If the block frequency is very high, new blocks will be produced faster than blocks can be transmitted and verified. This destroys consensus since nodes are essentially always seeing competing blockchains in the past. (Remember that consensus time is measured by total blockchain difficulty, roughly, blockheight.) Therefore the first chain that they see will always appear longer than the others, and every node will have a different idea of the best chain.

This happened to feathercoin, which had 60-second blocks. They were unable to achieve distributed consensus, so the network was changed to require developer signatures on all blocks. (They purchased the blocksigning code from Peercoin, whose problems with consensus are detailed in Section 6.4. [TODO citation needed]). Therefore their currency has a centralized consensus; since the point of proof-of-work is to achieve distributed consensus, mining on feathercoin is entirely pointless.

• *Changing block size.* Large blocks are generally good for miners, who have powerful systems and seek transaction fees. In fact, miners with good systems may want large blocks because they are costly to verify, thus muscling out miners with weak systems for whom the verification time is a large cost.

However, large blocks are bad for non-mining nodes, because they require more bandwidth, more verification effort and more storage. Determining the blocksize is therefore a tradeoff between having a high-capacity network and a well-decentralized one.

Many people have suggested finding this balance by adaptively changing the blocksize. The problem with this is that all consensus data is ultimately determined by miners. Therefore they will push the blocksize larger and larger, since they are incentivized to do so, and the network will become centralized.

6.2 Hash Function Changes

A popular and mostly-harmless change is to swap the SHA256d hash function used by Bitcoin for something else. This is covered in detail in *ASICs and Decentralization FAQ* by the author,

available at https://download.wpsoftware.net/bitcoin/asic-faq.pdf. Essentially, the things to watch out for:

- Ease of verification: if it is costly to verify a proof-of-work, unpaid peer-to-peer nodes will stop (or at least stop validating blocks), weakening the network and introducing trust in the miners.
- Progress-freeness: if proof production has any notion of "percent complete", even a probabilistic one, then mining becomes a race and a disproportionate advantage is given to large mining operations. This encourages centralization.
- Optimization-freeness: if a proof-of-work is too complex, it may be possible for individual miners to find secret shortcuts, making mining much cheaper for them and eventually forcing all other miners out of the market. This causes centralization.
- Simplicity: dedicated hardware is inevitable for a hash-grinding based proof-of-work. The more complex the algorithm, the more centralized this hardware development will be. The advantage from centralized hardware will likely also be greater.

It should be noted that scrypt, the most popular alternative to SHA256d, suffers from the first and last of these concerns, as well as being susceptible to a time-memory tradeoff, which is a special case of the third.

6.3 Difficulty Changes

Another popular change to Bitcoin's design involves the difficulty adjustment. In Bitcoin, difficulty as adjusted according to a negative feedback loop which targets a block frequence of every ten minutes. These adjustments occur in discrete steps every 2016 blocks, and are clamped by a factor of four in both directions. The formula for adjusting the difficulty is simple:

$$new target = old target \times \frac{timestamp of last block - timestamp 2016 blocks ago}{2016 \times ten minutes}$$

where the *target* is the numeric value that a blockhash must lie below for the block to be valid.

We see that the difficulty change is entirely determined by the timestamps of two blocks. Furthermore, no validation of these timestamps is done — there couldn't be, since if there were a globally recognized timestamp authority, we wouldn't need a blockchain! In other words, the difficulty is a parameter of the Bitcoin blockchain over which miners have free control¹⁵. Since miners are a small portion of the overall network, and their incentives do not always line up with those of ordinary users, we need to keep this power in check; hence the requirement that (a) 2016 blocks go by between changes, ensuring that large difficulty changes do not happen by accident, and (b) the "factor of four" rule, which ensures that attackers cannot drastically change the block frequency through timestamp manipulation.

¹⁵Things are not quite this bad — nodes receiving blocks in real time check that the timestamps are roughly correct before relaying them, so miners with badly wrong timestamps will be unable to inform a majority of the network of their blocks — but the fact remains that timestamps are not and cannot be authenticated, and are therefore forgeable.

On the other hand, if a significant portion of mining power were to disappear, the slow clamped difficulty changes could cause the block frequency to become much lower than ten minutes for long periods of time. This specter has motivated some altcoin developers to change the adjustment rules, but as we will see, this introduces significant risk — for a benefit that is only visible during a hashpower exodus, a time when the network is likely to have few users, and be prone to other attacks anyway.

Some common changes are:

- Introducing multiple hash functions; this forces separate difficulty calculations for each hash function, since they will all have distinct performance characteristics. Altcoins which do this must not only deal with the following problems, they also run the risk of having wildly different block frequencies for each hash function, producing a "limping blockchain" as they cycle through thom.
- 2. Weakening or even removing the "factor of four" clamping; this is dangerous since it gives miners much more control over the block rate. By increasing the block rate they can give themselves significant advantage by being the first to know about each new block; by decreasing it they can push other miners off the network by increasing the variance of their rewards.
- 3. Increasing the frequency of difficulty changes; this is dangerous because it effectively weakens the clamping (if the difficulty can change by a factor of four every 504 blocks, it can change by a factor of 256 every 2016 blocks), and also lets attackers operate in a "drive-by" fashion where they do not need to control large portions of hashpower for very long in order to execute attacks.

For example, terracoin made both of these mistakes, and was destroyed by a difficultymanipulating attack¹⁶.

- 4. Smoothing out the difficulty changes, *e.g.* by retargeting every block but using the blocktimes of the last 2016 blocks each time; by itself, this has basically no effect on anything. However, to maintain a factor-of-four clamping over 16 blocks, the difficulty change per block must be restricted to a factor of $4^{1/2016} = 1.00069$. To the best of my knowledge, no existing altcoin has such a restrictive bound, so the effective clamping is weakened and the chain is subject to the above attacks.
- 5. Introducing some complex adaptive scheme such as Kimoto Gravity Well (KGW); this is essentially an obfuscated way to introduce some or all of the above changes, and comes with the same risks. It also introduces complexity into consensus code, offering more opportunity for different implementations to do subtly different things and forking the network. Typically these schemes are also computationally expensive and therefore make it much more expensive to determine the longest chain by proof-of-work alone, putting a disproportionate burden on light nodes.

¹⁶See https://bitcointalk.org/index.php?topic=261986.0 for discussion of this event.

For example, megacoin and vertcoin use KGW; these altcoins have not suffered timestampmanipulation attacks, but their scalability has suffered as a result.

It is worth reiterating that a significant drop in hashpower is likely to mean that an altcoin is being deserted and that its block frequency is simply no longer relevant to many users. Introducing risk and complexity to handle this specific case is simply not a reasonable thing to do: under normal circumstances, the changes will have no effect, and in event of an attack, difficulty changes simply increase the likelihood and degree of the attack's success.

6.4 Proof of Stake

Once we have the notion of cryptocurrency and cryptographically-unforgeable transfer of value, a natural extension of this idea is cryptographically-unforgeable proof of ownership. This is the idea behind *proof of stake*. With cryptocurrency, it is possible for a proof-of-stake to not only prove ownership of a precise amount of currency, but also prove that this currency satisfies some property (say, it is locked and unspendable until some contract is satisfied).

In particular, proven stake in a scarce and experimental cryptocurrency can be considered a proof of vested interest in the project's success. By proving stake which is time-locked, it can be used to prove interest in the project's continued (and sustainable) existence.

A popular idea is to use proof-of-stake as a replacement for proof-of-work in producing a distributed consensus. As we will see, this idea is fundamentally flawed.

Failures. It is not well-advertised, but in fact there has never been an example of a cryptocurrency achieving distributed consensus by proof-of-stake. The prototypical proof-of-stake currency, Peercoin, depends on developer signatures to determine block validity: that is, its consensus is not distributed. The same fate has befallen other nominally-PoS currencies such as Blackcoin. In its initial incarnation, NXT was susceptible to a trivial stake-grinding attack (to be described below) and could not achieve any consensus. Since becoming closed-source¹⁷ while spamming technicallyilliterate claims at popular conferences, it has fallen out of scope of this document.

In fact, Peercoin was originally intended to drop the developer signatures once stake had been distributed. They attempted this once and were immediately attacked by stake-grinding. They quietly removed their text showing intention to drop developer signatures and added a small PoW to make stake-grinding less trivial.

Finally, it should be mentioned that developer-signed blocks are known in the PoS community as *checkpoints*. This is a very misleading name because it is already used to describe an anti-denial-of-service measure of Bitcoin's peer-to-peer network; Bitcoin's checkpoints have nothing whatsoever to do with consensus. Therefore claims by PoS advocates that "Bitcoin has checkpoints too" are simply false.

Distributed consensus. Essentially, the idea behind using proof-of-stake as a consensus mechanism is to move the opportunity costs from outside the system to inside the system. The motivation

¹⁷In March 2015, I was contacted by a NXT developer who informed me that NXT is now available under the open-source MIT license.

for this is that using "most proven work" as a criteria for consensus creates an economic incentive to prove as much work as possible. For Bitcoin, which proves thermodynamic work (i.e. a certain amount of irreversible computation was done), there is a physical limit — the Landauer limit — which controls what "as much work as possible" means¹⁸. The result of this limit is a consensus which is extremely resource-intensive, producing entropy and driving us toward the heat death of the universe literally as fast as the laws of physics will allow¹⁹. By moving the opportunity costs into a human-designed cryptocurrency, it should be possible to construct laws which force much smaller limits on resource consumption.

On a lower level, the way that proof of stake works is that currency holders are able to lock their currency for some amount of time, renting "stake" which is cryptographically verifiable. Then to extend the consensus history, rather than attaching a proof of work, each stakeholder digitally signs the extension. For reasons of practicality, typically a small random selection of stakeholders is chosen for each extension, and only a majority of the selection are required to give the extension validity. The chosen stakeholders are given a reward and after some time they are able to unlock their stake if they so desire.

The idea is that rather than depending on the economic inviability of taking control of a history, stakeholders are incentivized to agree on each extension because (a) they are randomly chosen and therefore unlikely to be in collusion, and (b) even if they can collude, they do not want to undermine the system (e.g. by signing many conflicting histories) because they want to recover their stored value when their stake comes unlocked, and (c) they have limited capacity to cause havoc anyway, since for the above reasons the next random selection of stakeholders will probably choose only a single reasonable history to extend.

Begging the question. On a high level, by tying our stake to (temporarily) sacrificed cryptographic resources, we are begging the question of consensus on who is in possession of what resources. Proof of stake advocates attempt to evade this accusation by pointing out that false histories can only be created by stakeholders, and their power is limited to a short interval of time (the time when they are the chosen signers) during which they are incentivized not to do so. Therefore conflicting histories simply will not appear, and we can appeal to synchronicity of the network to obtain consensus on the one existing history.

The problem with this argument is simple: the "short interval of time" is only short as measured by the consensus history, which only corresponds to a short interval in real time if there exists a consensus history. So we are still begging the question. In fact, if a stakeholder later irreversibly sells his stake for some resource outside the system (e.g. at an exchange), he no longer has incentive not to fork the history (or worse, expose his keys and let others fork the history) at the point in

¹⁸This is why we consider the proof of work to be a "proof", by the way: as long as our hash function is strong, the laws of physics prevent cheating.

¹⁹As an aside, it is interesting to note that rather than using a proof-of-work limited by the thermodynamic limit of computations per second, it should be possible to construct a proof-of-work which is limited by the bandwidth of the universe, i.e., the uncertainty principle which puts a lower bound on the size of information storage along with the speed of light which puts bound on how fast information can travel from storage to storage. Since information transfer is reversible, the resultant proof of work should not require large amounts of entropy production. This is the premise behind a *memory-hard* proof-of-work, which is outside the scope of this article. See for example [4]. There are many subtleties to this but the main concern with such a proof-of-work is that it shifts proving costs from marginal expenses to capital ones, which for a currency may cause economic incentives toward an oligarchy.

consensus time when he had control.

This is a bit abstruse. We can illustrate it with an example. Suppose that at some early point in consensus time, a single person has the ability to extend history. (For example, they have control over every key which a new block is required to be signed by.) This may have happened organically, if this person's keys were chosen randomly by the stake-choosing algorithm, but it could also happen if this person tracks down the other keyholders and buys their keys. This may happen much later in consensus time (and real time), so there is no reason to believe these keyholders are still incentivized to keep their keys secret. Alternately, they may have revealed the keys through some honest mistake, the chances of which increase as time passes, backups are lost, etc.

Now, we have a consensus history and an attacker who is able to fork it at some early time. To actually replace the entire consensus history, he needs to produce an alternate history, starting from his fork, which is longer than the existing history. But every block needs a new random selection of signers, so is this possible? The answer is absolutely yes: we have been using this word "random", but in fact we have required consensus on the set of signers (otherwise forks would trivially happen), so even a random selection must be seeded from past consensus history. Therefore, an attacker with enough past signing keys can modify the history he has direct control over, causing future signer selections to always happen in his favour. (It is likely he needs to "grind" through many choices of block before he finds one which lets him keep control of the signer selection. In effect, he has replaced proof-of-stake with proof-of-work, but a centralized one.)

Further, this ability to control the future selection of stakeholders (and even the *set* of stakeholders, by controlling which transactions appear in blocks) has serious consequences. This is because even without a deliberate attacker, the signers who extend the history at every point have an incentive to direct the history toward one in which they have more stake (and therefore more reward), which causes the system to trend toward centralization. They may do this by skewing the stake selection of future blocks, or more insidiously by censoring transactions which (may eventually) increase the set of stakeholders.

Impossibility. Intuitively, it seems impossible to obtain distributed consensus without provably consuming some resource outside of the system, but there is no rigorous argument for this claim.

The problem ultimately comes down to what Greg Maxwell calls *costless simulation*, and Andrew Miller calls *nothing at stake*. If it is costless for signers to create valid blocks, then they are able to cheaply search the blockspace for blocks which direct the history in their favour. No matter how the network is designed to prevent a minority takeover, an attacker can direct history toward a present in which they *are* the majority, as determined by the consensus, even if they are only a single party in physical space.

It would therefore appear that whatever space we want to achieve distributed consensus in (in Bitcoin's case, it is the space of humans, which can we approximate by thermodynamic space since we are autonomous agents within that space), we need to consume resources *in that space* to get the consensus.

Non-fundamental flaws. Aside from the inability of proof-of-stake to produce a distributed consensus — which can be evaded by using a "hybrid" proof-of-work/proof-of-stake system for consensus or even denominating stake in some other currency which has a working consensus, such as Bitcoin — there are incentive problems with using stake to determine block validity.

For example, if stake distribution is determined by stake transactions within the chain, miners have an incentive to censor these transactions to keep their proportion of stake high. The can get around any quotas by simply mining their own stake transactions — and if there are not enough honest stake transactions to meet the quota, the blockchain may halt.

If blocks are invalid without sufficiently many (in all existing proof-of-stake systems, "sufficiently many" means one) signatures, this gives a signers an ability to refuse signatures until some demands are met, effectively taking blocks hostage. This can be used for direct attacks, or to discourage proof-of-work miners (or whoever is producing the actual consensus), weakening the currency's security.

A simple question to ask, from Peter Todd, is *can I use stake to get more stake?* If so, the above problems apply, the consensus will become increasingly centralized, and there is potentially an economic trend toward a feudalism within the currency.

7 Where do I go from here?

I hope this document has provided some perspective on the intellectual magnitude of tackling cryptographic projects. Even experts shy away from developing new cryptosystems, preferring to use tried and true cryptographic primitives which have withstood the test of time and been analysed in depth by thousands of people. However, there are many open problems and exciting research directions in cryptography and the field is remarkably accessible to those willing to invest a few years into learning its history.

As a start, several active and famous cryptographers maintains blogs devoted to presenting cryptographic ideas in accessible ways. Of particular interest are those of Matthew Green and Bruce Schneier. Also current academic research is typically posted to the preprint archive at eprint.iacr.org. It is helpful to skim the abstracts periodically, both to find interesting papers and to get an idea of current trends in cryptographic research.

For an historical account from ancient times through World War II, read David Kahn's tome *The Codebreakers*. This is a very long text, but an enjoyable and engaging read.

Regarding modern cryptography, many classic papers are available online. An incomplete and unordered list of essential reading is:

- Probabilistic Encryption, Goldwasser and Micali, 1984.
- A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, ElGamal, 1985.
- The Decision Diffie-Hellman Problem, Boneh, 1998.
- Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols, Cramer, Damgård, Schoenmakers, 1994.

• *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, Fiat and Shamir, 1986.

It is also worthwhile to read the Wikipedia article on zero-knowledge proofs, (which has plenty of citations, but is more clearly written than any of them).

Dan Boneh also offers excellent courses in cryptography online, for free. They can be accessed at the following URL's.

- https://www.coursera.org/course/crypto
- https://www.coursera.org/course/crypto2

8 Conclusion.

References

- M.A. Fischer, N.A. Lynch, and M.S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. Ass. for Comp. Mach. **32** (1985), no. 2, 374–382.
- [2] A. Miller and J. J. LaViola Jr, Anonymous Byzantine consensus from moderately-hard puzzles: A model for Bitcoin, Tech. Report CS-TR-14-01, UCF, April 2014.
- [3] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009, https://www.bitcoin. org/bitcoin.pdf.
- [4] J. Tromp, Cuckoo cycle: a memory-hard proof-of-work system, https://github.com/ tromp/cuckoo, 2014.