



AFFIDAVIT OF NATHANIEL E FRANK-WHITE

1. I am a Records Request Processor at the Internet Archive. I make this declaration of my own personal knowledge.
2. The Internet Archive is a website that provides access to a digital library of Internet sites and other cultural artifacts in digital form. Like a paper library, we provide free access to researchers, historians, scholars, and the general public. The Internet Archive has partnered with and receives support from various institutions, including the Library of Congress.
3. The Internet Archive has created a service known as the Wayback Machine. The Wayback Machine makes it possible to browse more than 450 billion pages stored in the Internet Archive's web archive. Visitors to the Wayback Machine can search archives by URL (i.e., a website address). If archived records for a URL are available, the visitor will be presented with a display of available dates. The visitor may select one of those dates, and begin browsing an archived version of the Web. Links on archived files in the Wayback Machine point to other archived files (whether HTML pages or other file types), if any are found for the URL indicated by a given link. For instance, the Wayback Machine is designed such that when a visitor clicks on a hyperlink on an archived page that points to another URL, the visitor will be served the archived file found for the hyperlink's URL with the closest available date to the initial file containing the hyperlink.
4. The archived data made viewable and browsable by the Wayback Machine is obtained by use of web archiving software that automatically stores copies of files available via the Internet, each file preserved as it existed at a particular point in time.
5. The Internet Archive assigns a URL on its site to the archived files in the format `http://web.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL]` aka an "extended URL". Thus, the extended URL `http://web.archive.org/web/19970126045828/http://www.archive.org/` would be the URL for the record of the Internet Archive home page HTML file (`http://www.archive.org/`) archived on January 26, 1997 at 4:58 a.m. and 28 seconds (1997/01/26 at 04:58:28). The date indicated by an extended URL applies to a preserved instance of a file for a given URL, but not necessarily to any other files linked therein. Thus, in the case of a page constituted by a primary HTML file and other separate files (e.g., files with images, audio, multimedia, design elements, or other embedded content) linked within that primary HTML file, the primary HTML file and the other files will each have their own respective extended URLs and may not have been archived on the same dates.
6. Attached hereto as Exhibit A are true and accurate copies of screenshots of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each printout.





7. Attached hereto as Exhibit B are true and accurate copies of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each file.
8. I declare under penalty of perjury that the foregoing is true and correct.

DATE: 03/10/2023

Nathaniel Frank-White
Nathaniel E Frank-White

Please see attached
All Purpose
Jurat form
for additional
Notary Events



JURAT ATTACHMENT

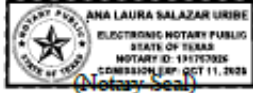
A notary public or other officer completing this certificate verifies only the identity of the individual who signed the document to which this certificate is attached, and not the truthfulness, accuracy, or validity of that document.

STATE OF Texas }

COUNTY OF Harris }

The foregoing instrument was subscribed and sworn before me this date of 03/10/2023 by Nathaniel Frank-White

This notarial act was an online notarization.



Notary's Signature Ana Laura Salazar Uribe

Registration No.: 131757026

Commission Expiration Date: October 11, 2026



EXHIBIT A

Mastering Bitcoin tells you everything you need to know about joining one of the most exciting revolutions since the invention of the web: digital money. Bitcoin is the first successful digital currency. It is instant, global, frictionless and it is changing money forever. Bitcoin is still in its infancy, and yet it has already spawned an economy valued at nearly \$2 billion that is growing exponentially. Established companies like PayPal are considering adding bitcoin as a payment method, and investors are funding a flurry of new startups aiming to stake claims in a new industry that may rival the Internet in terms of scale and impact on daily life.

If you are interested in learning more about the technical operation of bitcoin, or if you are building the next great bitcoin killer-app or business, you will find this book essential reading. From the basic use of a bitcoin wallet to buy a cup of coffee, to running a bitcoin marketplace with hundreds of thousands of transactions, or collaboratively building new financial innovations that will transform our understanding of currency and credit, this book will help you engineer money. You are about to unlock the API to a new economy. This book is your key.

Shape the Book:

[Review](#), or [comment on and edit](#) the book outline.

Then, complete an [anonymous survey](#) about it!

Take the Survey

What's important to you?

What should the bitcoin book contain?

Take the [anonymous survey](#).

Sign-up for Updates

Email Address

Subscribe

Free to Read and Share

Mastering Bitcoin by Andreas M. Antonopoulos will be licensed under a [CC-BY-NC-ND US License](#) after print publication.



Book cover, title and outline are samples for illustration purposes and will change before publication.

Content represents only the views of Andreas M. Antonopoulos.

This site is not endorsed or operated by O'Reilly Media.

Minimum Viable Block Chain

By [Ilya Grigorik](#) on May 05, 2014



Cryptocurrencies, and Bitcoin in particular, have been getting a lot of attention from just about every angle: regulation, governance, taxation, technology, product innovation, and the list goes on. The very concept of a "peer-to-peer (decentralized) electronic cash system" turns many of our previously held assumptions about money and finance on their head.

That said, putting the digital currency aspects aside, an arguably even more interesting and far-reaching innovation is the underlying block chain technology. Regardless of what you think of Bitcoin, or its [altcoin derivatives](#), as a currency and a store of value, behind the scenes they are all operating on the same basic block chain principles [outlined by Satoshi Nakamoto](#):

"We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power... The network itself requires minimal structure."

The block chain is agnostic to any "currency". In fact, it can (and will) be adapted to power many other use cases. As a result, it pays to understand the how and the why behind the "minimum viable block chain":

- **What follows is not** an analysis of the Bitcoin block chain. In fact, I intentionally omit mentioning both the currency aspects, and the many additional features that the Bitcoin block chain is using in production today.
- **What follows is** an attempt to explain, from the ground up, why the particular pieces (digital signatures, proof-of-work, transaction blocks) are needed, and how they all come together to form the "minimum viable block chain" with all of its remarkable properties.

I have learned long ago that writing helps me refine my own sloppy thinking, hence this document. Primarily written for my own benefit, but hopefully helpful to someone else as well. Feedback is always welcome, leave a comment below!

- [Securing transactions with triple-entry bookkeeping](#)
- [Securing transactions with PKI](#)
- [Balance = Σ\(receipts\)](#)
- [Multi-party transfers & verification](#)
- [Double-spending and distributed consensus](#)
 - [Requirements for a distributed consensus network](#)
 - [Protecting the network from Sybil attacks](#)
 - [Proof-of-work as a participation requirement](#)
- [Building the minimum viable block chain](#)
 - [Adding "blocks" & transaction fee incentives](#)
 - [Racing to claim the transaction fees](#)
 - [Resolving chain conflicts](#)
 - [Blocks are never final](#)
- [Properties of the \(minimum viable\) block chain](#)

Securing transactions with triple-entry bookkeeping

Alice and Bob are stamp collectors. It's nothing serious, and they're mostly in it for the social aspects of meeting others, exchanging stories, and doing an occasional trade. If both parties see something they like, they negotiate right there and then and complete the swap. In other words, it's a simple [barter system](#).

Then, one day Bob shows up with a stamp that Alice feels she absolutely must have in her collection. Except there is a problem, because Bob is not particularly interested in anything that Alice has to offer. Distracted, Alice continues negotiating with Bob and they arrive at a solution: they'll do a one-sided transaction where Bob will give Alice his stamp and Alice will promise to repay him in the future.

Both Bob and Alice have known each other for a while, but to ensure that both live up to their promise (well, mostly Alice), they agree to get their transaction "notarized" by their friend Chuck.



They make three copies (one for each party) of the above transaction receipt indicating that Bob gave Alice a "Red stamp". Both Bob and Alice can use their receipts to keep account of their trade(s), and Chuck stores his copy as

evidence of the transaction. Simple setup but also one with a number of great properties:

1. **Chuck can authenticate both Alice and Bob** to ensure that a malicious party is not attempting to fake a transaction without their knowledge.
2. **The presence of the receipt in Chuck's books is proof of the transaction.** If Alice claims the transaction never took place then Bob can go to Chuck and ask for his receipt to disprove Alice's claim.
3. **The absence of the receipt in Chuck's books is proof that the transaction never took place.** Neither Alice nor Bob can fake a transaction. They may be able to fake their copy of the receipt and claim that the other party is lying, but once again, they can go to Chuck and check his books.
4. **Neither Alice nor Bob can tamper with an existing transaction.** If either of them does, they can go to Chuck and verify their copies against the one stored in his books.

What we have above is an implementation of "triple-entry bookkeeping", which is simple to implement and offers good protection for both participants. Except, of course you've already spotted the weakness, right? We've placed a lot of trust in an intermediary. If Chuck decides to collude with either party, then the entire system falls apart.

Moral of the story? **Be (very) careful about your choice of the intermediary!**

Securing transactions with PKI

Dissatisfied with the dangers of using a "reliably intermediary", Bob decides to do some research and discovers that public key cryptography can eliminate the need for an intermediary! This warrants some explanation...

“ Public-key cryptography, also known as asymmetric cryptography, refers to a cryptographic algorithm which requires two separate keys, one of which is secret (or private) and one of which is public. Although different, the two parts of this key pair are mathematically linked. The public key is used to encrypt plaintext or to verify a digital signature; whereas the private key is used to decrypt ciphertext or to create a digital signature.

https://en.wikipedia.org/wiki/Public-key_cryptography

The original intent behind using a third party (Chuck) was to ensure three properties:

- **Authentication:** a malicious party can't masquerade as someone else.
- **Non-repudiation:** participants can't claim that the transaction did not happen after the fact.
- **Integrity:** the transaction receipt can't be modified after the fact.

Turns out, public key cryptography can satisfy all of the above requirements. Briefly, the workflow is as follows:



1. Both Alice and Bob generate a set public-private keypairs.
2. Both Alice and Bob publish their public keys to the world.
3. Alice writes a transaction receipt in plaintext.
4. Alice encrypts the plaintext of the transaction using her private key.
5. Alice prepends a plaintext "signed by" note to the ciphertext.
6. Both Alice and Bob store the resulting output.

Note that step #5 is only required when many parties are involved: if you don't know who signed the message then you don't know whose public key you should be using to decrypt it. This will become relevant very soon...

This may seem like a lot of work for no particular reason, but let's examine the properties of our new receipt:

1. Bob doesn't know Alice's private key, but that doesn't matter because he can look up her public key (which is shared with the world) and use it to decrypt the ciphertext of the transaction.
2. Alice is not really "encrypting" the contents of the transaction. Instead, by using her private key to encode the transaction she is "signing it": anyone can decrypt the ciphertext by using her public key, and because she is the only one in possession of the private key this mechanism guarantees that only she could have generated the ciphertext of the transaction.

How does Bob, or anyone else for that matter, get Alice's public key? There are many ways to handle distribution of public keys - e.g. Alice publishes it on her website. We'll assume that some such mechanism is in place.

As a result, the use of public key infrastructure (PKI) fulfills all of our earlier requirements:

1. Bob can use Alice's public key to authenticate signed transaction by decrypting the ciphertext.
2. Only Alice knows her private key, hence Alice can't deny that the transaction took place - she signed it.
3. Neither Bob nor anyone else can fake or modify a transaction without access to Alice's private key.

Note that for #2, Alice can deny that she is the true owner of the public-private keypair in question - i.e. someone is faking her identity in #1. The keypair to identity association is something your key distribution mechanisms needs to account for.

Both Alice and Bob simply store a copy of the signed transaction and the need for an intermediary is eliminated. The "magic" of public key cryptography is a perfect match for their two-party barter system.

Balance = $\Sigma(\text{receipts})$

With the PKI infrastructure in place, Bob and Alice complete a few additional trades: Alice acquires another stamp from Bob and Bob picks up a stamp from Alice. They each follow the same steps as before to generate signed transactions and append them to their respective ledgers.



The records are secure, but there is a small problem: it's not clear if either party has an outstanding balance. Previously, with just one transaction, it was clear who owed whom (Alice owed Bob) and how much (one red stamp), but with multiple transactions the picture gets really murky. Are all stamps of equal value? If so, then Alice has a negative balance. If not, then it's anyone's guess! To resolve this, Alice and Bob agree on the following:

- Yellow stamp is worth twice the value of a red stamp.
- Blue stamp is equal in value to a red stamp.

Finally, to ensure that their new agreement is secure they regenerate their ledgers by updating each transaction with its relative value. Their new ledgers now look as follows:



With that, computing the final balance is now a simple matter of iterating through all of the transactions and applying the appropriate debits and credits to each side. The net result is that Alice owes Bob 2... *units of value*. What's a "unit of value"? It's an arbitrary [medium of exchange](#) that Alice and Bob have agreed on. Further, since "unit of value" doesn't roll off the tongue, Alice and Bob agree to call 1 unit of value as 1 *chroma* (plural: *chroms*).

All of the above seems trivial, but **the fact that the balance of each party is a function of all of the receipts in the ledger has an important consequence: anyone can compute everyone's balance.** There is no need for any trusted intermediaries and the system is trivial to audit. Anyone can traverse the full ledger, verify the trades, and figure out the outstanding balances of each party.

Multi-party transfers & verification

Next, Bob stumbles across a stamp owned by John that he really likes. He tells John about the secure ledger he is using with Alice and asks him if he would be willing to do a trade where Bob transfers his balance with Alice as a method of payment - i.e. Bob gets the stamp from John, and Alice would owe John the amount she previously owed Bob. John agrees, but now they have dilemma. How exactly does Bob "transfer" his balance to John in a secure and verifiable manner? After some deliberation, they arrive at an ingenious plan:



Bob creates a new transaction by following the same procedure as previously, except that he first computes the SHA-256 checksum (a unique fingerprint) of the encrypted transaction he wants to transfer and then inserts the checksum in the "What" field of the new receipt. In effect, he is linking the new transfer receipt to his previous transaction with Alice, and by doing so, transfers its value to John.

To keep things simple, we'll assume that all transfers "spend" full value of the transaction being transferred. It's not too hard to extend this system to allow fractional transfers, but that's unnecessary complexity at this point.

With the new transaction in place, John makes a copy of the encrypted ledger for his safekeeping (now there are three copies) and runs some checks to verify its integrity:

1. John fetches Alice's and Bob's public keys and verifies the first three transactions.
2. John verifies that Bob is transferring a "valid" transaction:
 - The transaction that is being transferred is addressed to Bob.
 - Bob has not previously transferred the same transaction to anyone else.

If all the checks pass, they complete the exchange and we can compute the new balances by traversing the ledger: Bob has a net zero balance, Alice has a debit of 2 chroms, and John has a credit of 2 chroms (courtesy of Alice). Further, John can now take his new ledger to Alice and ask her for payment, and even though Alice wasn't present for their transaction, that's not a problem:

- Alice can verify the signature of the new transfer transaction using Bob's public key.
- Alice can verify that the transfer transaction is referencing one of her own valid transactions with Bob.

The above transfer and verification process is a pretty remarkable property of the system! Note that to make it all work, we need two enabling technologies: (a) use of PKI, which enables digital signature verification, and (b) the receipt ledger, which enables us to look at the full transaction history to verify balances and to link previous transactions to enable the transfer.

Satisfied with their ingenuity John and Bob part ways: Bob goes home with a new stamp and John with a new ledger. On the surface, everything looks great, but **they've just exposed themselves to a challenging security problem...** Can you spot it?

Double-spending and distributed consensus

Shortly after completing the transaction with John, Bob realizes that they have just introduced a critical flaw into their system and one that he could exploit to his advantage if he acts quickly: both Bob and John have updated their ledgers to include the new transaction, but neither Alice nor anyone else is aware that it has taken place. As a result, **there is nothing stopping Bob from approaching other individuals in his network and presenting them with an old copy of the ledger that omits his transaction with John!** If he convinces them to do a transaction, just as he did with John, then he can "[double-spend](#)" the same transaction as many times as he wants!



Of course, once multiple people show up with their new ledgers and ask Alice for payment, the fraud *will* be detected, but that is of little consolation - Bob has already run away with his loot!

The double-spend attack was not possible when we only had two participants since in order to complete the transaction you'd verify and update both sets of books simultaneously. As a result, all ledgers were always in sync. However, the moment we added an extra participant we introduced the possibility of incomplete and inconsistent ledgers between all the participants, which is why the double-spend is now possible.

In CS speak, a two-party ledger provides "strong consistency", and growing the ledger beyond two parties requires some form of [distributed consensus](#) to mitigate double-spend.

The simplest possible solution to this problem is to require that all parties listed in the ledger must be present at the time when each new transaction is made, such that everyone can update their books simultaneously. An effective strategy for a small-sized group, but also not a scalable one for a large number of participants.

Requirements for a distributed consensus network

Let's imagine that we want to scale our ledger to all stamp collectors around the globe such that anyone can trade their favorite stamps in a secure manner. Obviously, requiring that every participant must be present to register each transaction would never work due to geography, timezones, and other limitations. Can we build a system where we don't need everyone's presence and approval?

1. Geography is not really an issue: we can move communication online.
2. Timezone problems can be solved with software: we don't need each individual to manually update their ledgers. Instead, we can build software that can run on each participant's computer and automatically receive, approve, and add transactions to the ledger on their behalf.

In effect, we could build a peer-to-peer (P2P) network that would be responsible for distributing new transactions and getting everyone's approval! Except, unfortunately that's easier said than done in practice. For example, **while a P2P network can resolve our geography and timezone problems, what happens when even just one of the participants goes offline?** Do we block all transactions until they're back online?

Note that the "how" of building a P2P network is a massive subject in its own right: protocols and signaling, traversing firewalls and NATs, bootstrapping, optimizing how updates are propagated, security, and so on. That said, the low-level mechanics of building such a network are out of scope of our discussion... we'll leave that as an exercise for the reader.



Turns out, [distributed consensus](#) is a well studied problem in computer science, and one that offers some promising solutions. For example, [two-phase commit](#) (2PC) and [Paxos](#) both enable a mechanism where we only need the majority quorum (50%+) of participants to be present to safely commit a new transaction: as long as the majority has accepted the transaction the remainder of the group is guaranteed to eventually converge on the same transaction history.

That said, neither 2PC nor Paxos are sufficient on their own. For example, how would either 2PC or Paxos know the total number of

participants in our P2P stamp-collector network when new participants are joining on a daily basis and others are disappearing without notice? If one of the prior participants is offline, are they offline temporarily, or have they permanently left the network? Similarly, there is another and an even more challenging "[Sybil attack](#)" that we must account for: there is nothing stopping a malicious participant from creating many profiles to gain an unfair share of voting power within our P2P network.

If the number of participants in the system was fixed and their identities were authenticated and verified (i.e. a trusted network), then both 2PC and Paxos would work really well. Alas, that is simply not the case in our ever changing stamp collector P2P network. Have we arrived at a dead end? Well, not quite.

One obvious solution to solve our dilemma is to eliminate the "distributed" part from the problem statement. Instead of building a P2P distributed system we could, instead, build a global registry of all stamp collectors that would record their account information, authenticate them and (try to) ensure that nobody is cheating by creating multiple identities, and most importantly, keep one shared copy of the ledger! Concretely, **we could build a website where these trades can take place, and the website would then take care of ensuring the integrity and correct ordering of all transactions by recording them in its centralized database.**

The above is a practical solution but, let's admit it, an unsatisfying one since it forces us to forfeit the peer-to-peer nature of our ledger system. It places all of the trust in a single centralized system and opens up an entirely new set of questions: what is the uptime, security and redundancy of the system; who maintains the system and what are their incentives; who has administrative access, and so on. **Centralization brings its own set challenges.**

Let's rewind and revisit some of the problems we've encountered with our P2P design:

- Ensuring that every participant is always up to date (strongly consistent system) imposes high coordination costs and affects availability: if a single peer is unreachable the system cannot commit new transactions.
- In practice we don't know the global status of the P2P network: number of participants, whether individuals are temporarily offline or decided to leave the network, etc.
- Assuming we can resolve the above constraints, the system is still open to a Sybil attack where a malicious user can create many fake identities and exercise unfair voting power.

Unfortunately, resolving all of the above constraints is impossible unless we relax some of the requirements: the [CAP theorem](#) tells us that our distributed system can't have strong consistency, availability, and partition tolerance. As a result, in practice our P2P system [must operate under the assumption of weak\(er\) consistency](#) and deal with its implications:

- We must accept that some ledgers will be out of sync (at least temporarily).
- The system must eventually converge on a global ordering (linearizability) of all transactions.
- The system must resolve ledger conflicts in a predictable manner.
- The system must enforce global invariants - e.g. no double-spends.
- The system should be secure against Sybil and similar attacks.

Protecting the network from Sybil attacks

Achieving consensus in a distributed system, say by counting votes of each participant, opens up many questions about the "voting power" of each peer: who is allowed to participate, do certain peers have more voting power, is everyone equal, and how do we enforce these rules?

To keep things simple, let's say everyone's vote is equal. As a first step, we could require that each participant sign their vote with their private key, just as they would a transaction receipt, and circulate it to their peers - signing a vote ensures that someone else can't submit a vote on their behalf. Further, we could make a rule that only one vote is allowed to be submitted. If multiple votes are signed by the same key then all of them are discarded - make up your mind already! So far so good, and now the hard part...



How do we know if any particular peer is allowed to participate in the first place? If all that's needed is just a unique private key to sign a vote, then a malicious user could simply generate an unlimited number of new keys and flood the network. The root problem is that **when forged identities are cheap to generate and use, any voting system is easily subverted.**

To solve this problem we need to make the process of submitting the vote "expensive". Either the cost of generating a new identity must be raised, or the very process of submitting a vote must incur sufficiently high costs. To make this concrete, consider some real-world examples:

- When you show up to vote in your local government election, you are asked to present an ID (e.g. a passport) that is (hopefully) expensive to fake. In theory, nothing stops you from generating multiple fake IDs, but if the costs are high enough (monetary costs of making a fake, risk of being caught, etc), then the cost of running a Sybil attack will outweigh its benefits.
- Alternatively, imagine that you had to incur some other cost (e.g. pay a fee) to submit a vote. If the cost is high enough, then once again, the barrier to running a large-scale Sybil attack is increased.

Note that neither of the above examples "solves" the Sybil attack completely, but they also don't need to: as long as we raise the cost of the attack to be larger than the value gained by successfully subverting the system, then the system is secure and behaves as intended.

Note that we're using a loose definition of "secure". The system is still open for manipulation, and the exact vote count is affected, but the point is that a malicious participant doesn't affect the final outcome.

Proof-of-work as a participation requirement

Any user can easily (and cheaply) generate a new "identity" in our P2P network by generating a new private-public keypair. Similarly, any user can sign a vote with their private key and send it into the P2P network - that's also cheap, as the abundance of spam email in our inboxes clearly illustrates. Hence, submitting new votes is cheap and a malicious user can easily flood the network with as many votes as they wish.

However, what if we made one of the steps above expensive such that you had to expend significantly more effort, time, or money? That's the core idea behind requiring a [proof-of-work](#):

1. The proof-of-work step should be "expensive" for the sender.
2. The proof-of-work step should be "cheap" to verify by everyone else.

There are many possible implementations of such a method, but for our purposes we can re-use the properties provided by the cryptographic hash functions we encountered earlier:



1. It is easy to compute the hash value for any given message.
2. It is expensive to generate a message that has a given hash value.

We can impose a new rule in our system requiring that every signed vote must have a hash value that begins with a particular substring - i.e. require a partial hash collision of, say, two zero prefix. If this seems completely arbitrary, that's because it is - stay with me. Let's walk through the steps to see how this works:

1. Let's say a valid vote statement is a simple string: `"I vote for Bob"`.
2. We can use the same SHA-256 algorithm to [generate a hash value](#) for our vote.
 - `sha256("I vote for Bob") → b28bfa35bcd071a321589fb3a95cac...`
3. The resulting hash value is invalid because it does not start with our required substring of two zeros.
4. We modify the vote statement by appending an arbitrary string and try again:
 - `sha256("I vote for Bob - hash attempt #2") → 7305f4c1b1e7...`
5. The resulting hash value does not satisfy our condition either. We update the value and try again, and again, and... 155 attempts later we finally get:
 - `sha256("I vote for Bob - hash attempt #155") → 008d08b8fe...`

The critical property of the above workflow is that the output of the cryptographic hash function (SHA-256 in this case) is completely different every time we modify the input: the hash value of the previous attempt does not tell us anything about what the hash value of the next attempt when we increment our counter. **As a result, generating a valid vote is not just "hard problem", but also one better described as a lottery where each new attempt gives you a random output.** Also, we can adjust the odds of the lottery by changing the length of the required prefix:

1. Each character of the SHA-256 checksum has 16 possible values: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f`.
2. In order to generate a hash with a valid two zero prefix the sender will need 256 (16^2) attempts on average.
3. Bumping the requirement to 5 zeros will require more than 1,000,000 (16^5) attempts on average... Point being, we can easily increase the cost and make the sender spend more CPU cycles to find a valid hash.

How many SHA256 checksums can we compute on a modern CPU? The cost depends on the size of the message, CPU architecture, and other variables. If you're curious, open up your console and run a benchmark: `$> openssl speed sha`.

The net result is that generating a valid vote is "expensive" for the sender, but is still trivial to verify for the receiver: the receiver hashes the transaction (one operation) and verifies that the checksum contains the required hash collision prefix... Great, so how is this useful for our P2P system? **Above proof-of-work mechanism allows us to adjust the cost of submitting a vote such that the total cost of subverting the system (i.e. spoofing enough valid votes to guarantee a certain outcome) is higher than the value gained by attacking the system.**

Note that the "high cost to generate a message" is a useful property in many other contexts. For example, email spam works precisely because it is incredibly cheap to generate a message. If we could raise the cost of sending an email message - say, by requiring a proof-of-work signature - then we could break the spam business model by raising costs to be higher than profits.

Building the minimum viable block chain

We've covered a lot of ground. Before we discuss how the block chain can help us build a secure distributed ledger, let's quickly recap the setup, the properties, and the unsolved challenges within of our network:



1. Alice and Bob complete a transaction and record it in their respective ledgers.
 1. Once done, Bob has a [PKI-protected IOU](#) from Alice.
2. Bob completes a transaction with John where he [transfers Alice's IOU to John](#). Both Bob and John update their ledgers, but Alice doesn't know about the transaction... yet.

1. **Happy scenario:** John asks Alice to redeem his new IOU; Alice verifies his transaction by fetching Bob's public key; if the transaction is valid she pays John the required amount.
2. **Not so happy scenario:** Bob uses his old ledger that omits his transaction with John to create a [double-spend transaction](#) with Katy. Next, both Katy and John show up at Alice's doorstep and realize that only one of them will get paid.

The double-spend is possible due to the "weak consistency" of the distributed ledger: neither Alice nor Katy know about John and Bob's transaction, which allows Bob to exploit this inconsistency to his advantage. Solution? If the network is small and all participants are known, we can require that each transaction must be "accepted" by the network before it is deemed valid:

- **Unanimous consensus:** whenever a transaction takes place the two parties contact all other participants, tell them about the transaction, and then wait for their "OK" before they commit the transaction. As a result, all of the ledgers are updated simultaneously and double-spend is no longer possible.
- **Quorum consensus:** to improve processing speed and availability of the network (i.e. if someone is offline, we can still process the transaction) we can relax the above condition of unanimous consensus to quorum consensus (50% of the network).

Either of the above strategies would solve our immediate problem for a small network of known and verified participants. However, neither strategy scales to a larger, dynamic network where neither the total number of participants is known at any point in time, nor their identity:

1. We don't know how many people to contact to get their approval.
2. We don't know whom to contact to get their approval.
3. We don't know whom we are calling.

Note that we can use any means of communication to satisfy above workflow: in person, internet, [avian carriers](#), etc!

Lacking identity and global knowledge of all the participants in the network we have to relax our constraints. **While we can't guarantee that any particular transaction is valid, that doesn't stop us from making a statement about the probability of a transaction being accepted as valid:**

- **Zero-confirmation transaction:** we can accept a transaction without contacting any other participants. This places full trust on the integrity of the payer of the transaction - i.e. they won't double-spend.
- **N-confirmation transaction:** we can contact some subset of the (known) participants in the network and get them to verify our transaction. The more peers we contact, the higher the probability that we will catch malicious parties attempting to defraud us.

What is a good value for "N"? The answer depends on the amount being transferred and your trust and relationship with the opposite party. If the amount is small, you may be willing to accept a higher level of risk, or, you may adjust your risk tolerance based on what you know about the other party. Alternatively, you will have to do some extra work to contact other participants to validate your transaction. In either case, there is a tradeoff between the speed with which the transaction is processed (zero-confirmation is instant), the extra work, and the risk of that transaction being invalid.



So far, so good. Except, there is an additional complication that we must consider: our system relies on transaction confirmations from other peers, but nothing stops a malicious user from generating as many fake identities as needed (recall that an "identity" in our system is simply a public-private keypair, which is trivial to generate) to satisfy Katy's acceptance criteria.

Whether Bob decides to execute the attack is a matter of simple economics: if the gain is higher than the cost then he should consider running the attack. Conversely, if Katy can make the cost of running the attack higher than the value of the transaction, then she should be safe (unless Bob has a personal vendetta and/or is willing to lose money on the transaction... but that's out of scope). To make it concrete, let's assume the following:

- Bob is transferring 10 choms to Katy.
- The cost of generating a fake identity and transaction response is 0.001 choms: energy costs to keep the computer running, paying for internet connectivity, etc.

If Katy asks for 1001 confirmations, then it no longer makes (economic) sense for Bob to run the attack. Alternatively, **we could add a proof-of-work requirement for each confirmation and raise the cost for each valid response from 0.001 choms to 1:** finding a valid hash will take CPU time, which translates to a higher energy bill. As a result, Katy would only need to ask for 11 confirmations to get the same guarantee.

Note that Katy also incurs some costs while requesting each confirmation: she has to expend effort to send out the requests and then validate the responses. Further, if the cost of generating a confirmation and verifying it is one-to-one, then Katy will incur the same total cost to verify the transaction as its value... which, of course, makes no economic sense.

This is why the asymmetry of proof-of-work is critical. Katy incurs low cost to send out requests and validate the responses, but the party generating the confirmation needs to expend significantly more effort to generate a valid response.

Great, problem solved, right? Sort of... in the process we've created another economic dilemma. **Our network now incurs a cost to validate each transaction that is of equal or higher value than the transaction itself.** While this acts as an economic deterrent against malicious participants, why would any legitimate participant be willing to incur any costs for someone else? A rational participant simply wouldn't, it doesn't make sense. Doh.

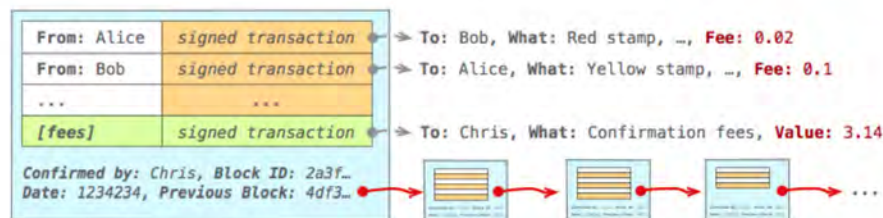
Adding "blocks" & transaction fee incentives

If participants in the network must incur a cost to validate each other's transactions, then we must provide an economic incentive for them to do so. In fact, at a minimum we need to offset their costs, because otherwise an "idle" participant (anyone who is not submitting own transactions) would continue accruing costs on behalf of the network — that wouldn't work. Also a couple of other problems that we need address:

1. If the cost of verifying the transaction is equal to or higher than the value of the transaction itself (to deter malicious participants), then the total transaction value is net-zero, or negative! E.g. Bob transfers 10 chrams to Katy; Katy spends 10 chrams to compensate other peers to validate the transaction; Katy is sad.
2. How does Katy pay for confirmations? If that's its own transaction then we have a recursive problem.

Let's start with the obvious: the transaction fee can't be as high as the value of the transaction itself. Of course, Katy doesn't have to spend the exact value to confirm the transaction (e.g. she can allocate half the value for confirmations), but then it becomes a question of margins: if the remaining margin (value of transaction - verification fees) is high enough, then there is still incentive for fraud. Instead, ideally we would like to incur the lowest possible transaction fees and still provide a strong deterrent against malicious participants. Solution?

We can incentivize participants in the network to confirm transactions by allowing them to pool and confirm multiple transactions at once - i.e. confirm a "block" of transactions. Doing so would also allow them to aggregate transaction fees, thereby lowering the validation costs for each individual transaction.



A block is simply a collection (one or more) of valid transactions - think of it as the equivalent of a page in a physical ledger. In turn, each block contains a reference to a previous block (previous page) of transactions, and the full ledger is a linked sequence of blocks. Hence, **block chain**. Consider the example above:

1. Alice and Bob generate new transactions and announces them to the network.
2. Chris is listening for new transaction notifications, each of which contains a transaction fee that the sender is willing to pay to get it validated and confirmed by the network:
 1. Chris aggregates unconfirmed transactions until he has a direct financial incentive (sum of transaction fees > his cost) to perform the necessary work to validate the pending transactions.
 2. Once over the threshold, Chris first validates each pending transaction by checking that none of the inputs are double-spent.
 3. Once all transactions are validated Chris adds an extra transaction to the pending list (indicated in green in the diagram above) that transfers the sum of advertised transaction fees to himself.
 4. Chris generates a block that contains the list of pending transactions, a reference to the previous block (such that we can traverse the blocks and see the full ledger), and performs the proof-of-work challenge to generate a block hash value that conforms to accepted rules of the network - e.g. partial hash collision of N leading zeros.
 5. Finally, once Chris finds a valid block, he distributes it to all other participants.
3. Both Alice and Bob are listening for new block announcements and look for their transaction in the list:
 1. Alice and Bob verify integrity of the block - i.e. verify proof-of-work and contained transactions.
 2. If the block is valid and their transaction is in the list, then the transaction has been confirmed!

We made a big leap here. Previously we've only had one type of record in our network - the signed transaction. Now we have signed transactions and blocks. The former is generated by the individuals engaging in trade, and the latter is generated by parties interested in collecting fees by validating and confirming transactions.

Also, note that the above scheme requires some minimum volume of transactions in the system to sustain the incentives for individuals creating the blocks: the more transactions there are, the lower the fees have to be for any single transaction.

Phew, ok, Alice has announced a new transaction and received a valid block from Chris confirming it. That's one confirmation, what about the rest? Also, Chris is (hopefully) not the only participant who is incentivized to work on generating the blocks. What if someone else generates a different block at the same time, and which of those blocks is "valid"? This is where it gets interesting...

Racing to claim the transaction fees

The remarkable part about introducing the ability to aggregate fees by verifying a block of transactions is that it creates a role for a new participant in the network who now has a direct financial incentive to secure it. You can now make a profit by validating transactions, and where there is profit to be made, competition follows, which only strengthens the network - a virtuous cycle and a clever piece of social engineering!

That said, the incentive to compete to validate transactions creates another interesting dilemma: how do we coordinate this block generation work in our distributed network? The short answer is, as you may have already guessed, we don't. Let's add some additional rules into our system and examine how they resolve this problem:

1. Any number of participants is allowed to participate ("race") to create a valid block. There is no coordination. Instead, each interested participant listens for new transactions and decides whether and when they want to try to generate a valid block and claim the transaction fees.
2. When a valid block is generated, it is immediately broadcast into the network.
 1. Other peers check the validity of the block (check each transaction and validity of the block itself), and if valid, add it to their ledgers and then finally rebroadcast it to other peers in the network.
 2. Once added, the new block becomes the "topmost block" of their ledger. As a result, if that same peer was also working on generating a block, then they need to abort their previous work and start over: they now need to update their reference to the latest block and also remove any transactions from their unconfirmed list that are contained in the latest block.
 3. Once the above steps are complete, they start working on a new block, with the hope that they'll be the first ones to discover the next valid block, which would allow them to claim the transaction fees.
3. ... repeat the above process until the heat death of the universe.

The lack of coordination between all the participants working on generating the blocks means there will be duplicate work in the network, and that's OK! **While no single participant is guaranteed to claim any particular block, as long as the expected value (probability of claiming the block times the expected payout, minus the costs) of participating in the network is positive, then the system is self-sustaining.**

Note that there is also no consensus amongst the peers on which transactions should be validated next. Each participant aggregates their own list and can use different strategies to optimize their expected payoff. Also, due to the nature of our proof-of-work function (finding a partial hash collision for a SHA-256 checksum of the block), the only way to increase the probability of claiming a block is to expend more CPU cycles.



There is one more caveat that we need to deal with: it's possible that two peers will find a valid block at about the same time and begin propagating through the network - e.g. Kent and Chris in the diagram above. As a result, some fraction of the network may end up accepting Kent's block as topmost block, while the rest will take Chris's block. Now what?

Resolving chain conflicts

Once again, we're going to take a hands-off approach and let the random nature of the block generation process resolve the conflict, albeit with one additional rule: if multiple chains are detected, the participants should immediately switch to and build on top of the longest chain. Let's see how this work in practice:



1. Some peers will start building new blocks on top of Kent's block, others on top of Chris's block.
2. At some point, someone will find a new block and begin propagating it through the network.
 1. When other peers receive the new block, the part of the network that was working with a different topmost block will detect that there is now a longer alternative chain, which means that they need to switch to it - e.g. in the above example, the peers who were working with Chris's block stop their work, drop Chris's block, and switch to the longer (Amy + Kent's) chain.
 2. Any transactions that are part of the discarded block but that are not yet confirmed are placed in the pending list and the process starts over.

It's possible that the race condition can persist for multiple blocks, but eventually one branch will race ahead of the other and the rest of the network will converge on the same longest chain.

Great, we now have a strategy to resolve conflicts between different chains in the network. Specifically, the network promises linearizability of transactions by recording them in a linked list of blocks. But, crucially, it makes no promises

about an individual block "guaranteeing" the state of any transaction. Consider the example above:

- Alice sends out her transaction into the network.
- Chris generates a valid block that confirms her transaction.

Except, there is a fork in the chain and Chris's block is later "removed" as the network converges on Kent's branch of the chain. As a result, even when Alice receives a block with her transaction, she can't be sure that this block won't be undone in the future!

Blocks are never "final"

No block is "final", ever. Any block can be "undone" if a longer chain is detected. In practice, forks should be detected fairly quickly, but there is still always the possibility of an alternative chain. Instead, **the only claim we can make is that the "deeper" any particular block is in the chain, the less likely it is that it will be undone**. Consequently, no transaction can ever be treated as "final" either, we can only make statements about the probability of it being undone.

1. **0-confirmation transaction:** exchange is performed without waiting for any block to include the transaction.
2. **1-confirmation transaction:** latest valid block includes the transaction.
3. **N-confirmation transaction:** there is a valid block that includes the transactions, and there are $N-1$ blocks that have since been built on top of that block.

If you are willing to accept the risk, you always have the option to go with a 0-confirmation transaction: no transaction fees, no need to wait for confirmations. However, you also place a lot of trust in the opposite party.

Alternatively, if you want to lower your risk, then you should wait for one or more blocks to be built on top of the block that includes your transaction. **The longer you wait, the more blocks will be built on top of the block that contains your transaction, the lower the probability of an alternative chain that may undo your transaction.**

By "undo" we mean any scenario where one of the participants can make the network accept an alternative transaction transferring funds to any account other than yours - e.g. you complete the transaction, hand over the widget and get a receipt, but the attacker then injects a transaction that "double-spends" those same funds to an alternative account.

Why does the length of the block chain act as a good proxy for "safety" of a transaction? If an attacker wanted to undo a particular transaction, then they will need to build a chain that begins at a block prior to the one where that transaction is listed, and then build a chain of other blocks that is longer than the one currently used by the network. As a result, the deeper the block, the higher the amount of computational effort that would be required to replace it by creating an alternative chain. The longer the chain the more expensive it is to execute an attack.

How many blocks should you wait for before accepting a transaction? There is no one number, the answer depends on the properties of the network (time to generate each block, propagation latency of the transactions and blocks, size of the network, etc), and the transaction itself: its value, what you know about the other party, your risk profile, and so on.

Properties of the (minimum viable) block chain

1. Individual transactions are secured by PKI.

- Transactions are authenticated: a malicious party can't masquerade as someone else and sign a transaction on their behalf.
 - Authentication is only with respect to the public-private keypair. There is no requirement for "strong authentication" that links the keypair to any other data about the participants. In fact, a single participant can generate and use multiple keypairs! In this sense, the network allows anonymous transactions.
- Non-repudiation: participants can't claim that the transaction did not happen after the fact.
- Integrity: transactions can't be modified after the fact.

2. Once created, transactions are broadcast into the P2P network.

- Participants form a network where transactions and blocks are relayed amongst all the participating peers. There is no central authority.

3. One or more transactions are aggregated into a "block".

- A block validates one or more transactions and claims the transaction fees.
 - This allows the transaction fees to remain small relative to the value of each transaction.
- A valid block must have a valid proof-of-work solution.
 - Valid proof-of-work output is hard to generate and cheap to verify.
 - Proof-of-work is used to raise the cost of generating a valid block to impose a higher cost on running an attack against the network.
- Any peer is allowed to work on generating a valid block, and once a valid block is generated, it is

broadcast into the network.

- Any number of peers can compete to generate a valid block. There is no coordination. When a fork is detected, it is resolved by automatically switching to the longest chain.
- Each block contains a link to a previous valid block, allowing us to traverse the full history of all recorded transactions in the network.

4. Peers listen for new block announcements and merge them into their ledgers.

- Inclusion of the transaction in a block acts as a "confirmation" of that transaction, but that fact alone does not "finalize" any transaction. Instead, we rely on the length of the chain as a proxy for "safety" of the transaction. Each participant can choose their own level of risk tolerance, ranging from 0-confirmation transactions to waiting for any arbitrary number of blocks.

The combination of all of the above rules and infrastructure provides a decentralized, peer-to-peer block chain for achieving distributed consensus of ordering of signed transactions. That's a mouthful, I know, but it's also an ingenious solution to a very hard problem. The individual pieces of the block-chain (accounting, cryptography, networking, proof-of-work), are not new, but the emergent properties of the system when all of them are combined are pretty remarkable.



[Ilya Grigorik](#) is a web performance engineer and developer advocate at Google, where his focus is on making the web fast and driving adoption of performance best practices at Google and beyond.

[Follow @igrigorik](#)

[View Comments](#)

[Tweet](#)



High-Performance Browser Networking (O'Reilly)

What every web developer must know about networking and browser performance: impact of latency and bandwidth, TCP, UDP, and TLS optimization, performance tips for mobile networks, and an under the hood look at performance of HTTP 1.1/2.0, XMLHttpRequest, WebSocket, WebRTC, DataChannel, and other transports.

[Read Online for Free](#)

© 2005-2014 Ilya Grigorik

[subscribe via rss](#)

[about](#)

Navigation

[Main page](#)
[FAQ](#)
[Help](#)
[Forums](#)
[Chatrooms](#)
[Community portal](#)
[Recent changes](#)

Toolbox

[What links here](#)
[Related changes](#)
[Special pages](#)
[Printable version](#)
[Permanent link](#)
[Page information](#)

Contracts

A **distributed contract** is a method of using Bitcoin to form agreements with people via the block chain. Contracts don't make anything possible that was previously impossible, but rather, they allow you to solve common problems in a way that minimizes trust. Minimal trust often makes things more convenient by allowing human judgements to be taken out of the loop, thus allowing complete automation.

By building low trust protocols that interact with Bitcoin, entirely new products can be created:

- **Smart property** is property that can be atomically traded and loaned via the block chain.
- **Transferable virtual property** are digital items that can be traded but not duplicated.
- **Agents** are autonomous programs that maintain their own wallet, which they use to buy server time. Money is obtained by the agent selling services. If demand exceeds supply the agents can spawn children that either survive or die depending on whether they can get enough business.
- **Distributed markets** are a way to implement peer to peer bond and stock trading, allowing Bitcoin to be evolve into a full competitor to the international financial system.
- The **Ripple currency exchange** is a way to implement a distributed currency exchange based on social networks.

This page also lists some smaller examples.

Many of the ideas underlying Bitcoin contracts were first described by Nick Szabó in his seminal paper, [Formalizing and Securing Relationships on Public Networks](#) 📄. These pages were written by [Mike Hearn](#) 📄. Contact him if you have an idea for a new type of contract. You can watch [a video of a talk on contracts](#) 📄 that was presented at the Bitcoin 2012 conference in London.

Contents

- 1 A warning about the mempool transaction replacement mechanism
- 2 Theory
- 3 Example 1: Providing a deposit
- 4 Example 2: Escrow and dispute mediation
- 5 Example 3: Assurance contracts
- 6 Example 4: Using external state
 - 6.1 Trust minimization: challenges
 - 6.2 Trust minimization: multiple independent oracles
 - 6.3 Trust minimization: trusted hardware
 - 6.4 Trust minimization: Amazon AWS oracles
- 7 Example 5: Trading across chains
- 8 Example 6: Pay-for-proof contracts: buying a solution to any pure function
- 9 Example 7: Rapidly-adjusted (micro)payments to a pre-determined party
- 10 Example 8: Multi-party decentralised lotteries
- 11 See Also

A warning about the mempool transaction replacement mechanism

In places this page refers to the ability to use the nSequence field for transaction mempool replacement. This mechanism was disabled in 2010 📄, and more recently the code has been [removed completely](#) 📄, due to concerns over people using it to perform DoS attacks. Implementors should take this into account and try to create contract mechanisms that do not rely on mempool replacement if they wish to have their implementations work with current implementations. If Bitcoin changes in future to allow mempool replacement once again, this page will be updated.

Theory

Every [transaction](#) in Bitcoin has one or more inputs and outputs. Each input/output has a small, pure function associated with it called a [script](#). Scripts can contain signatures over simplified forms of the transaction itself.

Every transaction can have a lock time associated with it. This allows the transaction to be pending and replaceable until an agreed-upon future time, specified either as a block index or as a timestamp (the same field is used for both, but values less than 500 million are interpreted as a block index). If a transaction's lock time has been reached, we say it is final.

Each transaction input has a sequence number. In a normal transaction that just moves value around, the sequence numbers are all UINT_MAX and the lock time is zero. If the lock time has not yet been reached, but all the sequence numbers are UINT_MAX, the transaction is also considered final. Sequence numbers can be used to issue new versions of a transaction without invalidating other inputs signatures, e.g., in the case where each input on a transaction comes from a different party, each input may start with a sequence number of zero, and those numbers can be incremented independently.

Signature checking is flexible because the form of transaction that is signed can be controlled through the use of SIGHASH flags, which are stuck on the end of a signature. In this way, contracts can be constructed in which each party signs only a part of it, allowing other parts to be changed without their involvement. The SIGHASH flags have two parts, a mode and the ANYONECANPAY modifier:

1. **SIGHASH_ALL**: This is the default. It indicates that everything about the transaction is signed, except for the input scripts. Signing the input scripts as well would obviously make it impossible to construct a transaction, so they are always blanked out. Note, though, that other properties of the input, like the connected output and sequence numbers, are signed; it's only the scripts that are not. Intuitively, it means "I agree to put my money in, if everyone puts their money in and the outputs are this".
2. **SIGHASH_NONE**: The outputs are not signed and can be anything. Use this to indicate "I agree to put my money in, as long as everyone puts their money in, but I don't care what's done with the output". This mode allows others to update the transaction by changing their inputs sequence numbers.
3. **SIGHASH_SINGLE**: Like SIGHASH_NONE, the inputs are signed, but the sequence numbers are blanked, so others can create new versions of the transaction. However, the only output that is signed is the one at the same position as the input. Use this to indicate "I agree, as long as my output is what I want; I don't care about the others".

The SIGHASH_ANYONECANPAY modifier can be combined with the above three modes. When set, only that input is signed and the other inputs can be anything.

Scripts can contain the CHECKMULTISIG opcode. This opcode provides n-of-m checking: you provide multiple public keys, and specify the number of valid signatures

that must be present. The number of signatures can be less than the number of public keys. An output can require two signatures to be spent by setting it to something like this:

```
2 <pubkey1> <pubkey2> 2 CHECKMULTISIGVERIFY
```

There are two general patterns for safely creating contracts:

1. Transactions are passed around outside of the P2P network, in partially-complete or invalid forms.
2. Two transactions are used: one (the contract) is created and signed but not broadcast right away. Instead, the other transaction (the payment) is broadcast after the contract is agreed to lock in the money, and then the contract is broadcast.

This is to ensure that people always know what they are agreeing to.

Together, these features let us build interesting new financial tools on top of the block chain.

Example 1: Providing a deposit

Imagine that you open an account on a website (eg, a forum or wiki) and wish to establish your trustworthiness with the operators, but you don't have any pre-existing reputation to leverage. One solution is to buy trust by paying the website some money. But if at some point you close your account, you'd probably like that money back. You may not trust the site enough to give them a deposit that they are tempted to spend. Another risk is that the site might just disappear one day.

The goal is to prove that you made a sacrifice of some kind so the site knows you're not a spambot, but you don't want them to be able to spend the money. And if the operators disappear, you'd eventually like the coins back without needing anything from them.

We can solve this problem with a contract:

1. The user and website send each other a newly-generated public key.
2. The user creates transaction Tx1 (the payment) putting 10 BTC into an output that requires both user and website to sign, but does not broadcast it. They use the key from the previous step for the site.
3. User sends the hash of Tx1 to the website.
4. The website creates a transaction Tx2 (the contract). Tx2 spends Tx1 and pays it back to the user via the address he provided in the first step. Note that Tx1 requires two signatures, so this transaction can't be complete. nLockTime is set to some date in the future (eg, six months). The sequence number on the input is set to zero.
5. Finally, the incomplete (half-signed) transaction is sent back to the user. The user checks that the contract is as expected - that the coins will eventually come back to him - but, unless things are changed, only after six months. Because the sequence number is zero, the contract can be amended in future if both parties agree. The script in the input isn't finished though; there are only zeros where the user's signature should be. He fixes that by signing the contract and putting the new signature in the appropriate spot.
6. The user broadcasts Tx1, then broadcasts Tx2.

At this stage, the 10 BTC are in a state where neither the user nor the website can spend them independently. After six months, the contract will complete and the user will get the coins back, even if the website disappears.

What if the user wishes to close his account early? The website creates a new version of Tx2 with nLockTime set to zero and the input sequence number set to UINT_MAX, then he re-signs it. The site hands the tx back to the user, who signs it as well. The user then broadcasts the transaction, terminating the contract early and releasing the coins.

What if the six months is nearly up and the user wishes to keep his account? The same thing applies: the contract can be resigned with a newer nLockTime, a sequence number 1 higher than the previous and rebroadcast 2³² times. No matter what happens, both parties must agree for the contract to change.

Obviously, if the user turns out to be abusive (i.e., a spammer), the website will not allow an early close of the contract. If too much abuse is getting through, the size of the deposit can be raised or the length of the contract can be increased.

Example 2: Escrow and dispute mediation

A buyer wants to trade with somebody he doesn't know or trust. In the common case where the transaction goes well, the client doesn't want any third parties involved. If something goes wrong though, he'd like a third party to decide who gets the money - perhaps a professional dispute mediation service. Note that this concept can apply to either buyer or seller. The mediator might request proof of postage from the merchant, for example.

In other words, one wants to lock up some coins so a third party has to agree in order for them to be spent:

1. Agree with the merchant on a dispute mediator (e.g., ClearCoin).
2. Ask the merchant for a public key (K1). Ask the mediator for a public key (K2). Create a new key for yourself (K3).
3. Send the merchant K2. The merchant challenges the mediator with a random nonce. The mediator signs the nonce with the private form of K2, thus proving it really belongs to merchant.
4. Create a transaction (Tx1) with an output script as follows and broadcast it:

```
2 <K1> <K2> <K3> 3 CHECKMULTISIGVERIFY
```

Now the coins are locked in such a way that they can only be spent by the following methods:

1. Client and the merchant agree (either a successful trade, or merchant agrees to reimburse client without mediation)
2. Client and the mediator agree (failed trade, mediator sides with client, like a charge-back)
3. The mediator and the merchant agree (goods delivered, merchant gets client's coins despite the dispute)

When signing an input, the contents are set to the connected output. Thus, to redeem this transaction, the client creates a scriptSig containing zeros where the other signature should be, signs it, and then sets one of the slots to his new signature. The partially-complete transaction can then be sent to the merchant or mediator for the second signature.

Example 3: Assurance contracts

An **assurance contract** is a way of funding the creation of a **public good**, that is, a good that, once created, anyone can benefit from for free. The standard example is a lighthouse: whilst everyone may agree that one should be built, it's too expensive for an individual sailor to justify building one, given that it will also benefit all his competitors.

One solution is for everyone to pledge money towards the creation of the public good, such that the pledges are only committed if the total value of all pledges is above the cost of creation. If not enough people contribute, nobody has to pay anything.

Examples where Bitcoin is superior to traditional payment methods for assurance contract fundraising include applications where frequent, small pledges need to be made automatically, for instance internet radio station funding and [web page translation](#). Consider a browser extension that you send a bit of money to. It detects the current language of the page and broadcasts a pledge for a translation into your language to be prepared. If enough users with the extension land on the same page at

the same time (eg, it was linked to from somewhere high traffic), then enough pledges are broadcast to trigger a payment to a company who prepares a high quality translation. When complete it automatically loads in your browser.

We can model this in Bitcoin as follows:

1. An entrepreneur creates a new address and announces that the good will be created if at least 1000 BTC is raised. Anyone can contribute.
2. Each party wishing to pledge creates a new transaction spending some of their coins to the announced address, but they do not broadcast it. The transaction is similar to a regular transaction except for three differences: Firstly, there cannot be any change. If you don't have any outputs of the right size, you must create one first by spending to one of your own addresses. Secondly, the input script signature is signed with SIGHASH_ALL | SIGHASH_ANYONECANPAY. Finally, the output value is set to 1000 BTC. Note that this is not a valid transaction because the output value is larger than the input value.
3. The transaction is uploaded to the entrepreneur's server, which saves it to disk and updates its count of how many coins have been pledged.
4. Once the server has enough coins, it merges the separate transactions together into a new transaction. The new transaction has a single output that simply spends to the announced address - it is the same as the outputs on each contributed transaction. The inputs to the transaction are collected from the contributed pledges.
5. The finished transaction is broadcast, sending the pledged coins to the announced address.

This scheme relies on several aspects of the protocol. The first is the SIGHASH flags used. SIGHASH_ALL is the default and means the entire contents of the transaction are signed, except for the input scripts. SIGHASH_ANYONECANPAY is an additional modifier that means the signature only covers the input it's found in - the other inputs are not signed and thus can be anything.

By combining these flags together, we are able to create a signature that is valid even when other inputs are added, but breaks if the outputs or other properties of the transaction are changed.

The second aspect we exploit is the fact that a transaction in which the output values are larger than the input values is invalid (for obvious reasons). This means it's safe to send the entrepreneur a transaction that spends such coins - it's impossible for him to claim the coins unless he has other inputs that sum to the output value or more.

It's possible to create assurance contracts without using SIGHASH_ANYONECANPAY. Instead, a two step process is used in which pledges are collected without transactions, and once the total value is reached, a transaction with an input for each pledger is created and passed around until all signatures are collected. However, using SIGHASH_ANYONECANPAY and then merging is probably more convenient.

An assurance contract can be prepared for [funding network security](#) for the next block. In this way, mining can be funded even if block space is not scarce.

An elaboration of this concept is described by Tabarrok in his paper, "[The private provision of public goods via dominant assurance contracts](#)". In a dominant assurance contract, if a contract fails (not enough pledges within a set time window) the entrepreneur pays a fee to those who pledged so far. This type of contract attempts to arrange incentives such that taking part is always the right strategy. [A scheme for dominant assurance contracts](#) in Bitcoin has been proposed.

Example 4: Using external state

Scripts are, by design, pure functions. They cannot poll external servers or import any state that may change as it would allow an attacker to outrun the block chain. What's more, the scripting language is extremely limited in what it can do. Fortunately, we can make transactions connected to the world in other ways.

Consider the example of an old man who wishes to give an inheritance to his grandson, either on the grandson's 18th birthday or when the man dies, whichever comes first.

To solve this, the man first sends the amount of the inheritance to himself so there is a single output of the right amount. Then he creates a transaction with a lock time of the grandson's 18th birthday that pays the coins to another key owned by the grandson, signs it, and gives it to him - but does not broadcast it. This takes care of the 18th birthday condition. If the date passes, the grandson broadcasts the transaction and claims the coins. He could do it before then, but it doesn't let him get the coins any earlier, and some nodes may choose to drop transactions in the memory pool with lock times far in the future.

The death condition is harder. As Bitcoin nodes cannot measure arbitrary conditions, we must rely on an *oracle*. An oracle is a server that has a keypair, and signs transactions on request when a user-provided expression evaluates to true.

Here is an example. The man creates a transaction spending his output, and sets the output to:

```
<hash> OP_DROP 2 <sons pubkey> <oracle pubkey> CHECKMULTISIG
```

This is the oracle script. It has an unusual form - it pushes data to the stack then immediately deletes it again. The pubkey is published on the oracle's website and is well-known. The hash is set to be the hash of the user-provided expression stating that he has died, written in a form the oracle knows how to evaluate. For example, it could be the hash of the string:

```
if (has_died('john smith', born_on=1950/01/02)) return (10.0, 1xgRXEH8i86zYzHN2U4KMyRCg4LvwNurp);
```

This little language is hypothetical, it'd be defined by the oracle and could be anything. The return value is an output: an amount of value and an address owned by the grandson.

Once more, the man creates this transaction but gives it directly to his grandson instead of broadcasting it. He also provides the expression that is hashed into the transaction and the name of the oracle that can unlock it.

It is used in the following algorithm:

1. The oracle accepts a measurement request. The request contains the user-provided expression, a copy of the output script, and a partially complete transaction provided by the user. Everything in this transaction is finished except for the scriptSig, which contains just one signature (the grandson's) - not enough to unlock the output.
2. The oracle checks the user-provided expression hashes to the value in the provided output script. If it doesn't, it returns an error.
3. The oracle evaluates the expression. If the result is not the destination address of the output, it returns an error.
4. Otherwise the oracle signs the transaction and returns the signature to the user. Note that when signing a Bitcoin transaction, the input script is set to the connected output script. The reason is that when OP_CHECKSIG runs, the script containing the opcode is put in the input being evaluated, `_not_` the script containing the signature itself. The oracle has never seen the full output it is being asked to sign, but it doesn't have to. It knows the output script, its own public key, and the hash of the user-provided expression, which is everything it needs to check the output script and finish the transaction.
5. The user accepts the new signature, inserts it into the scriptSig and broadcasts the transaction.

If, and only if, the oracle agrees that the man is dead, the grandson can broadcast the two transactions (the contract and the claim) and take the coins.

Oracles can potentially evaluate anything, yet the output script form in the block chain can always be the same. Consider the following possibilities:

```
today() == 2011/09/25 && exchange_rate(mtgoxUSD) >= 12.5 && exchange_rate(mtgoxUSD) <= 13.5
Require exchange rate to be between two values on a given date

google_results_count(site:www.google.com/hostednews 'Mike Hearn' olympic gold medal) > 0
A bet on me doing something that I will never actually do
```

```
// Choose between one of two winners of a bet on the outcome of the Eurovision song contest.
if (eurovision_winner() == 'Azerbaijan')
    return 1Lj9udBDwptFffGSJSC2sohCFudQgSTPD;
else
    return 1JxgRXEHBi86zYzHN2U4KMyRCg4LvwNUrp;
```

The conditions that control whether the oracle signs can be arbitrarily complex, but the block chain never needs to contain more than a single hash.

Trust minimization: challenges

There are various ways to reduce the level of required trust in the oracle.

Going back to our first example, the oracle has not seen the transaction the grandson is trying to unlock, as it was never broadcast, thus, it cannot hold the grandson to ransom because it does not know whether the transaction it's signing for even exists. People can, and should, regularly **challenge** the oracle in an automated fashion to ensure it always outputs what is expected. The challenges can be done without spending any coins because the tx to be signed can be invalid (ie, connected to transactions that don't exist). The oracle has no way to know whether a request to be signed is random or real. How to challenge the oracle with conditions that are not yet true is however an open research question.

Trust minimization: multiple independent oracles

The number of keys in the CHECKMULTISIG can be increased to allow for **n-of-m** oracles if need be. Of course, it is vital to check that the oracles really are independent and not in collusion.

Trust minimization: trusted hardware

Using commodity hardware, you can use **trusted computing** in the form of Intel TXT or the AMD equivalent (SKINIT) to set up a sealed hardware environment and then use the TPM chip to attest that fact to a third party. That third party can verify the hardware was in the required state. Defeating this requires someone to be able to interfere with the execution of a program that may run entirely on the CPU, even in extreme cases without any data traversing the memory bus (you can run entirely using on-die cache if the program is small enough).

Trust minimization: Amazon AWS oracles

Finally, perhaps the most practical approach currently is to use Amazon Web Services. As of November 2013, the closest we have to a working oracle is [this recipe for creating a trusted computing environment using AWS](#) , built in support of [this project for doing selective SSL logging and decryption](#) . The idea is that an oracle, which can be proven trustworthy using the Amazon APIs with Amazon as the root of trust, records encrypted SSL sessions to an online banking interface in such a way that later if there is a dispute about a person-to-person exchange, the logs can be decrypted and the dispute settled.

Example 5: Trading across chains

The Bitcoin technology can be adapted to create [multiple, independent currencies](#). NameCoin is an example of one such currency that operates under a slightly different set of rules, and can also be used to rent names in a namespace. Currencies that implement the same ideas as Bitcoin can be traded freely against each other with limited trust.

For example, imagine a consortium of companies that issue EURcoins, a crypto-currency that is backed 1:1 by deposits in the consortium's bank accounts. Such a currency would have a different set of tradeoffs to Bitcoin: more centralized, but without FX risk. People might wish to trade Bitcoins for EURcoins back and forth, with the companies only getting involved when cashing in/out of the regular banking system.

To implement this, a [protocol proposed by TierNolan](#)  can be used:

1. Party 'A' generates some random data, x (the secret).
2. Party 'A' generates Tx1 (the payment) containing an output with the chain-trade script in it. See below for this script and a discussion of it. It allows coin release either by signing with the two keys (key 'A' and key 'B') or with (secret 'x', key 'B'). This transaction is not broadcast. The chain release script contains hashes, not the actual secrets themselves.
3. Party 'A' generates Tx2 (the contract), which spends Tx1 and has an output going back to key 'A'. It has a lock time in the future and the input has a sequence number of zero, so it can be replaced. 'A' signs Tx2 and sends it to 'B', who also signs it and sends it back.
4. 'A' broadcasts Tx1 and Tx2. Party 'B' can now see the coins but cannot spend them because it does not have an output going to him, and the tx is not finalized anyway.
5. 'B' performs the same scheme in reverse on the alternative chain. The lock time for 'B' should be much larger than the lock time for 'A'. Both sides of the trade are now pending but incomplete.
6. Since 'A' knows the secret, 'A' can claim his coins immediately. However, 'A', in the process of claiming his coin, reveals the secret 'x' to 'B', who then uses it to finish the other side of the trade with ('x', key 'B').

This protocol makes it feasible to do trades automatically in an entirely peer-to-peer manner, which should ensure good liquidity.

The chain-trade script could look like this:

```
IF
  2 <key A> <key B> 2 CHECKMULTISIGVERIFY
ELSE
  <key B> CHECKSIGVERIFY SHA256 <hash of secret x> EQUALVERIFY
ENDIF
```

The contract input script looks like either:

```
<sig A> <sig B> 1
```

or

```
<secret x> <sig B> 0
```

i.e., the first data element selects which phase is being used.

See [Atomic cross-chain trading](#) for details.

Note that whilst EURcoins is a natural idea, there are other ways to implement peer-to-peer currency exchange (Bitcoin to fiat and vice versa), see the [Ripple currency exchange](#) article for more information.

Sergio Demian-Lerner proposed [P2PTradeX](#), a solution requiring the validation rules for one blockchain to be effectively encoded into the validation rules for the other.

Example 6: Pay-for-proof contracts: buying a solution to any pure function

In example 4, we saw how to make payments conditional on the output of some arbitrary program. Those programs are very powerful and can do anything a regular program can do, like fetching web pages. The downside is that a third party is required (an oracle). Although there are techniques that can help reduce the trust needed in the oracle, none can reduce it to zero.

For a restricted class of programs, pure functions, new cryptographic techniques are starting to become available that can actually reduce the trust needed all the way to zero with no third parties. These programs cannot perform any I/O, but in many cases this restriction turns out to be unimportant or can be worked around in other ways, like by giving the program a signed/timestamped document as an input instead of having the program download it.

Read an explanation of the protocol here: [User:Gmaxwell/why_hash_locked](#)

Example 7: Rapidly-adjusted (micro)payments to a pre-determined party

Bitcoin transactions are very cheap relative to traditional payment systems, but still have a cost due to the need for it to be mined and stored. There are some cases in which you want to rapidly and cheaply adjust the amount of money sent to a particular recipient without incurring the cost of a broadcast transaction.

For example, consider an untrusted internet access point, like a WiFi hotspot in a cafe you never visited before. You'd like to pay 0.001 BTC per 10 kilobytes of usage, without opening an account with the cafe. A zero-trust solution means it could be fully automatic, so you could just pre-allocate a budget on your phone's mobile wallet at the start of the month, and then your device automatically negotiates and pays for internet access on demand. The cafe also wants to allow anyone to pay them without the fear of being ripped off.

To do this, the following protocol can be used. This protocol relies upon a **different** behavior of nLockTime to the original design. Starting around 2013 time-locked transactions were made non standard and no longer enter the memory pool, thus cannot be broadcast before the timelock expires. When the behaviour of nLockTime is restored to the original design from Satoshi, a variant of this protocol is required which is discussed below.

We define the client to be the party sending value, and the server to be the party receiving it. This is written from the clients perspective.

1. Create a public key (K1). Request a public key from the server (K2).
2. Create and sign but do not broadcast a transaction (T1) that sets up a payment of (for example) 10 BTC to an output requiring both the server's public key and one of your own to be used. A good way to do this is use OP_CHECKMULTISIG. The value to be used is chosen as an efficiency tradeoff.
3. Create a refund transaction (T2) that is connected to the output of T1 which sends all the money back to yourself. It has a time lock set for some time in the future, for instance a few hours. Don't sign it, and provide the unsigned transaction to the server. By convention, the output script is "2 K1 K2 2 CHECKMULTISIG"
4. The server signs T2 using its public key K2 and returns the signature to the client. Note that it has not seen T1 at this point, just the hash (which is in the unsigned T2).
5. The client verifies the servers signature is correct and aborts if not.
6. The client signs T1 and passes the signature to the server, which now broadcasts the transaction (either party can do this if they both have connectivity). This locks in the money.
7. The client then creates a new transaction, T3, which connects to T1 like the refund transaction does and has two outputs. One goes to K1 and the other goes to K2. It starts out with all value allocated to the first output (K1), ie, it does the same thing as the refund transaction but is not time locked. The client signs T3 and provides the transaction and signature to the server.
8. The server verifies the output to itself is of the expected size and verifies the client's provided signature is correct.
9. When the client wishes to pay the server, it adjusts its copy of T3 to allocate more value to the server's output and less to its ow. It then re-signs the new T3 and sends the signature to the server. It does not have to send the whole transaction, just the signature and the amount to increment by is sufficient. The server adjusts its copy of T3 to match the new amounts, verifies the signature and continues.

This continues until the session ends, or the 1-day period is getting close to expiry. The AP then signs and broadcasts the last transaction it saw, allocating the final amount to itself. The refund transaction is needed to handle the case where the server disappears or halts at any point, leaving the allocated value in limbo. If this happens then once the time lock has expired the client can broadcast the refund transaction and get back all the money.

This protocol has [been implemented in bitcoinj](#).

When nLockTime'd transactions are able to enter the memory pool (once more) and transaction replacement has been re-enabled, a variant on the protocol must be used. In this case, no refund transaction is used. Instead each T3 has a sequence number one higher than the previous and all T3's have a time lock set to the same period as above. Each time a payment is made the sequence number is incremented, ensuring that the last version will take precedence. If the channel protocol is not closed cleanly, this means the value transfer won't commit until the time lock expires. To avoid this both parties can cooperate by signing a T3 that has a sequence number of 0xFFFFFFFF resulting in immediate confirmation regardless of the value of nLockTime.

The lock time and sequence numbers avoid an attack in which the AP provides connectivity, and then the user [double-spends](#) the output back to themselves using the first version of TX2, thus preventing the cafe from claiming the bill. If the user does try this, the TX won't be included right away, giving the access point a window of time in which it can observe the TX broadcast, and then broadcast the last version it saw, overriding the user's attempted double-spend.

The latter protocol that relies on transaction replacement is more flexible because it allows the value allocated to go down as well as up during the lifetime of the channel as long as the client receives signatures from the server, but for many use cases this functionality is not required. Replacement also allows for more complex configurations of channels that involve more than two parties. Elaboration on such use cases is left as an exercise for the reader.

Example 8: Multi-party decentralised lotteries

Using some of the techniques from example 6 and some very advanced scripting, it becomes possible to build a multi-party lottery with no operator. The exact protocol used is explained in the paper "[Secure multiparty computations on Bitcoin](#)". A shorter explanation of how it works may be added to this wiki at some point in the future.

See Also

- [Script](#)

Category: [Technical](#)

[Introduction](#)[Resources](#)[Innovation](#)[Participate](#)[FAQ](#)[English](#)

Bitcoin Core version 0.9.0 released

19 March 2014

Bitcoin Core version 0.9.0 is now available for download at:

<https://bitcoin.org/bin/0.9.0/>

This is a new major version that brings both new features and bug fixes; we urge all users to upgrade.

Please report bugs using the issue tracker at GitHub:

<https://github.com/bitcoin/bitcoin/issues>

[Go back to the version history](#)

[Network Status](#) [Legal](#) [About bitcoin.org](#) © Bitcoin Project 2009-2014 Released under the [MIT license](#)

A community website sponsored by  **The Bitcoin Foundation**

Bitcoin Core version 0.9.0 is now available from:

<https://bitcoin.org/bin/0.9.0/>

This is a new major version release, bringing both new features and bug fixes.

Please report bugs using the issue tracker at github:

<https://github.com/bitcoin/bitcoin/issues>

How to Upgrade

If you are running an older version, shut it down. Wait until it has completely shut down (which might take a few minutes for older versions), uninstall all earlier versions of Bitcoin, then run the installer (on Windows) or just copy over /Applications/Bitcoin-Qt (on Mac) or bitcoind/bitcoin-qt (on Linux).

If you are upgrading from version 0.7.2 or earlier, the first time you run 0.9.0 your blockchain files will be re-indexed, which will take anywhere from 30 minutes to several hours, depending on the speed of your machine.

On Windows, do not forget to uninstall all earlier versions of the Bitcoin client first, especially if you are switching to the 64-bit version.

Windows 64-bit installer

New in 0.9.0 is the Windows 64-bit version of the client. There have been frequent reports of users running out of virtual memory on 32-bit systems during the initial sync. Because of this it is recommended to install the 64-bit version if your system supports it.

NOTE: Release candidate 2 Windows binaries are not code-signed; use PGP and the SHA256SUMS.asc file to make sure your binaries are correct. In the final 0.9.0 release, Windows setup.exe binaries will be code-signed.

OSX 10.5 / 32-bit no longer supported

0.9.0 drops support for older Macs. The minimum requirements are now:

- * A 64-bit-capable CPU (see <http://support.apple.com/kb/ht3696>);
- * Mac OS 10.6 or later (see <https://support.apple.com/kb/ht1633>).

Downgrading warnings

The 'chainstate' for this release is not always compatible with previous releases, so if you run 0.9 and then decide to switch back to a 0.8.x release you might get a blockchain validation error when starting the old release (due to 'pruned outputs' being omitted from the index of unspent transaction outputs).

Running the old release with the -reindex option will rebuild the chainstate data structures and correct the problem.

Also, the first time you run a 0.8.x release on a 0.9 wallet it will rescan the blockchain for missing spent coins, which will take a long time (tens of minutes on a typical machine).

Rebranding to Bitcoin Core

To reduce confusion between Bitcoin-the-network and Bitcoin-the-software we have renamed the reference client to Bitcoin Core.

OP_RETURN and data in the block chain

On OP_RETURN: There was been some confusion and misunderstanding in the community, regarding the OP_RETURN feature in 0.9 and data in the blockchain. This change is not an endorsement of storing data in the blockchain. The OP_RETURN change creates a provably-prunable output, to avoid data storage schemes -- some of which were already deployed -- that were storing arbitrary data such as images as forever-unspendable TX outputs, bloating bitcoin's UTXO database.

Storing arbitrary data in the blockchain is still a bad idea; it is less costly and far more efficient to store non-currency data elsewhere.

Autotools build system

For 0.9.0 we switched to an autotools-based build system instead of individual (q)makefiles.

Using the standard "./autogen.sh; ./configure; make" to build Bitcoin-Qt and bitcoind makes it easier for experienced open source developers to contribute to the project.

Be sure to check doc/build-*.md for your platform before building from source.

Bitcoin-cli

Another change in the 0.9 release is moving away from the bitcoind executable functioning both as a server and as a RPC client. The RPC client functionality ("tell the running bitcoin daemon to do THIS") was split into a separate executable, 'bitcoin-cli'. The RPC client code will eventually be removed from bitcoind, but will be kept for backwards compatibility for a release or two.

``walletpassphrase`` RPC

The behavior of the ``walletpassphrase`` RPC when the wallet is already unlocked has changed between 0.8 and 0.9.

The 0.8 behavior of ``walletpassphrase`` is to fail when the wallet is already unlocked:

```
> walletpassphrase 1000
walletunlocktime = now + 1000
> walletpassphrase 10
Error: Wallet is already unlocked (old unlock time stays)
```

The new behavior of ``walletpassphrase`` is to set a new unlock time overriding the old one:

```
> walletpassphrase 1000
walletunlocktime = now + 1000
> walletpassphrase 10
walletunlocktime = now + 10 (overriding the old unlock time)
```

Transaction malleability-related fixes

This release contains a few fixes for transaction ID (TXID) malleability issues:

- `-nospendzeroconfchange` command-line option, to avoid spending zero-confirmation change
- `IsStandard()` transaction rules tightened to prevent relaying and mining of mutated transactions
- Additional information in `listtransactions/gettransaction` output to report wallet transactions that conflict with each other because they spend the same outputs.
- Bug fixes to the `getbalance/listaccounts` RPC commands, which would report incorrect balances for double-spent (or mutated) transactions.
- New option: `-zapwallettxes` to rebuild the wallet's transaction information

Transaction Fees

This release drops the default fee required to relay transactions across the network and for miners to consider the transaction in their blocks to 0.01mBTC per kilobyte.

Note that getting a transaction relayed across the network does NOT guarantee that the transaction will be accepted by a miner; by default, miners fill their blocks with 50 kilobytes of high-priority transactions, and then with 700 kilobytes of the highest-fee-per-kilobyte transactions.

The minimum relay/mining fee-per-kilobyte may be changed with the `minrelaytxfee` option. Note that previous releases incorrectly used the `mintxfee` setting to determine which low-priority transactions should be considered for inclusion in blocks.

The wallet code still uses a default fee for low-priority transactions of 0.1mBTC per kilobyte. During periods of heavy transaction volume, even this fee may not be enough to get transactions confirmed quickly; the `mintxfee` option may be used to override the default.

0.9.0 Release notes

=====

RPC:

- New notion of 'conflicted' transactions, reported as confirmations: -1
- `'listreceivedbyaddress'` now provides tx ids
- Add raw transaction hex to `'gettransaction'` output
- Updated help and tests for `'getreceivedby(account|address)'`
- In `'getblock'`, accept 2nd 'verbose' parameter, similar to `getrawtransaction`, but defaulting to 1 for backward compatibility
- Add `'verifychain'`, to verify chain database at runtime
- Add `'dumpwallet'` and `'importwallet'` RPCs
- `'keypoolrefill'` gains optional size parameter
- Add `'getbestblockhash'`, to return tip of best chain
- Add `'chainwork'` (the total work done by all blocks since the genesis block) to `'getblock'` output
- Make RPC password resistant to timing attacks
- Clarify help messages and add examples
- Add `'getrawchangeaddress'` call for raw transaction change destinations
- Reject insanely high fees by default in `'sendrawtransaction'`
- Add RPC call `'decodescript'` to decode a hex-encoded transaction script
- Make `'validateaddress'` provide `redeemScript`
- Add `'getnetworkhashps'` to get the calculated network hashrate
- New RPC 'ping' command to request ping, new 'pingtime' and 'pingwait' fields in `'getpeerinfo'` output
- Adding new `'addrlocal'` field to `'getpeerinfo'` output
- Add verbose boolean to `'getrawmempool'`
- Add rpc command `'getunconfirmedbalance'` to obtain total unconfirmed balance
- Explicitly ensure that wallet is unlocked in `'importprivkey'`
- Add check for valid keys in `'importprivkey'`

Command-line options:

- New option: `-nospendzeroconfchange` to never spend unconfirmed change outputs
- New option: `-zapwallettxes` to rebuild the wallet's transaction information
- Rename option `'-tor'` to `'-onion'` to better reflect what it does
- Add `'-disablewallet'` mode to let bitcoind run entirely without wallet (when built with wallet)
- Update default `'-rpcsslcpiphers'` to include TLSv1.2
- make `'-logtimestamps'` default on and rework help-message
- RPC client option: `'-rpcwait'`, to wait for server start
- Remove `'-logtodebugger'`
- Allow `'-noserver'` with bitcoind

Block-chain handling and storage:

- Update leveldb to 1.15
- Check for correct genesis (prevent cases where a datadir from the wrong network is accidentally loaded)
- Allow txindex to be removed and add a reindex dialog
- Log aborted block database rebuilds
- Store orphan blocks in serialized form, to save memory
- Limit the number of orphan blocks in memory to 750
- Fix non-standard disconnected transactions causing mempool orphans
- Add a new checkpoint at block 279,000

Wallet:

- Bug fixes and new regression tests to correctly compute the balance of wallets containing double-spent (or mutated) transactions
- Store key creation time. Calculate whole-wallet birthday.
- Optimize rescan to skip blocks prior to birthday
- Let user select wallet file with `-wallet=foo.dat`
- Consider generated coins mature at 101 instead of 120 blocks
- Improve wallet load time
- Don't count txins for priority to encourage sweeping
- Don't create empty transactions when reading a corrupted wallet
- Fix rescan to start from beginning after `importprivkey`
- Only create signatures with low S values

Mining:

- Increase default `-blockmaxsize/prioritysize` to 750K/50K
- `'getblocktemplate'` does not require a key to create a block template
- Mining code fee policy now matches relay fee policy

Protocol and network:

- Drop the fee required to relay a transaction to 0.01mBTC per kilobyte
- Send tx relay flag with version
- New 'reject' P2P message (BIP 0061, see <https://gist.github.com/gavinandresen/7079034> for draft)
- Dump addresses every 15 minutes instead of 10 seconds
- Relay OP_RETURN data TxOut as standard transaction type
- Remove CENT-output free transaction rule when relaying
- Lower maximum size for free transaction creation
- Send multiple inv messages if `mempool.size > MAX_INV_SZ`
- Split `MIN_PROTO_VERSION` into `INIT_PROTO_VERSION` and `MIN_PEER_PROTO_VERSION`
- Do not treat `fFromMe` transaction differently when broadcasting
- Process received messages one at a time without sleeping between messages
- Improve logging of failed connections
- Bump protocol version to 70002
- Add some additional logging to give extra network insight
- Added new DNS seed from bitcoinstats.com

Validation:

- Log reason for non-standard transaction rejection
- Prune provably-unspendable outputs, and adapt consistency check for it.
- Detect any sufficiently long fork and add a warning
- Call the `-alertnotify` script when we see a long or invalid fork
- Fix multi-block reorg transaction resurrection
- Reject non-canonically-encoded serialization sizes
- Reject dust amounts during validation
- Accept `nLockTime` transactions that finalize in the next block

Build system:

- Switch to autotools-based build system
- Build without wallet by passing `'--disable-wallet'` to configure, this removes the BerkeleyDB dependency
- Upgrade gitian dependencies (libpng, libz, libupnpc, boost, openssl) to more recent versions
- Windows 64-bit build support
- Solaris compatibility fixes
- Check integrity of gitian input source tarballs
- Enable full GCC Stack-smashing protection for all OSes

GUI:

- Switch to Qt 5.2.0 for Windows build
- Add payment request (BIP 0070) support
- Improve options dialog
- Show transaction fee in new send confirmation dialog
- Add total balance in overview page
- Allow user to choose data directory on first start, when data directory is missing, or when the `-choosedatadir` option is passed
- Save and restore window positions
- Add vout index to transaction id in transactions details dialog
- Add network traffic graph in debug window
- Add open URT dialog

- Add Coin Control Features
- Improve receive coins workflow: make the 'Receive' tab into a form to request payments, and move historical address list functionality to File menu.
- Rebrand to 'Bitcoin Core'
- Move initialization/shutdown to a thread. This prevents "Not responding" messages during startup. Also show a window during shutdown.
- Don't regenerate autostart link on every client startup
- Show and store message of normal bitcoin:URI
- Fix richtext detection hang issue on very old Qt versions
- OS X: Make use of the 10.8+ user notification center to display Growl-like notifications
- OS X: Added NSHighResolutionCapable flag to Info.plist for better font rendering on Retina displays.
- OS X: Fix bitcoin-qt startup crash when clicking dock icon
- Linux: Fix Gnome bitcoin: URI handler

Miscellaneous:

- Add Linux script (contrib/qos/tc.sh) to limit outgoing bandwidth
- Add '-regtest' mode, similar to testnet but private with instant block generation with 'setgenerate' RPC.
- Add 'linearize.py' script to contrib, for creating bootstrap.dat
- Add separate bitcoin-cli client

Credits

Thanks to everyone who contributed to this release:

- Andrey
- Ashley Holman
- b6393ce9-d324-4fe1-996b-acf82dbc3d53
- bitsofproof
- Brandon Dahler
- Calvin Tam
- Christian Decker
- Christian von Roques
- Christopher Latham
- Chuck
- coblee
- constantined
- Cory Fields
- Cozz Lovan
- daniel
- Daniel Larimer
- David Hill
- Dmitry Smirnov
- Drak
- Eric Lombrozo
- fanquake
- fcicq
- Florin
- frewil
- Gavin Andresen
- Gregory Maxwell
- gubatron
- Guillermo Céspedes Tabárez
- Haakon Nilsen
- HaltingState
- Han Lin Yap
- harry
- Ian Kelling
- Jeff Garzik
- Johnathan Corgan
- Jonas Schnelli
- Josh Lehan
- Josh Triplett
- Julian Langschaedel
- Kangmo
- Lake Denman
- Luke Dashjr
- Mark Friedenbach
- Matt Corallo
- Michael Bauer
- Michael Ford
- Michagogo
- Midnight Magic
- Mike Hearn
- Nils Schneider
- Noel Tiernan
- Olivier Langlois
- patrick s
- Patrick Strateman
- paveljanik
- Peter Todd
- phantomcircuit
- phelixbtc
- Philip Kaufmann
- Pieter Wuille
- Rav3nPL
- R E Broadley
- regergregregerrge
- Robert Backhaus
- Roman Mindalev
- Rune K. Svendsen
- Ryan Niebur
- Scott Ellis
- Scott Willeke
- Sergey Kazenyuk
- Simon

- Shawn Wilkinson
- Sined
- sje
- Subo1978
- super3
- Tamas Blummer
- theuni
- Thomas Holenstein
- Timon Rapp
- Timothy Stranex
- Tom Geller
- Torstein Husebø
- Vaclav Vobornik
- vhf / victor felder
- Vinnie Falco
- Warren Togami
- Wil Bown
- Wladimir J. van der Laan

Index of /bin/0.9.0/

../		
README.txt	19-Mar-2014 19:10	14725
SHA256SUMS.asc	19-Mar-2014 20:33	1332
SHASUMS.asc	19-Mar-2014 20:33	1210
bitcoin-0.9.0-linux.tar.gz	18-Mar-2014 19:34	35525461
bitcoin-0.9.0-macosx.dmg	19-Mar-2014 20:33	13165861
bitcoin-0.9.0-win.zip	18-Mar-2014 19:34	61455258
bitcoin-0.9.0-win32-setup.exe	18-Mar-2014 19:35	11219184
bitcoin-0.9.0-win64-setup.exe	18-Mar-2014 19:35	12016968

Bitcoin Forum

simple machines forum

April 06, 2014, 06:02:27 AM

Welcome, **Guest**. Please login or register.

Forever
Login with username, password and session length

News: Bitcoin Core **0.9.0** is now available. [Download.](#)

[HOME](#) [HELP](#) [SEARCH](#) [DONATE](#) [LOGIN](#) [REGISTER](#)

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **Alt chains and atomic transfers**

« previous topic next topic »
print

Pages: [1] 2 All

Author	Topic: Alt chains and atomic transfers (Read 3167 times)
TierNolan Sr. Member  Activity: 420  Ignore	 Alt chains and atomic transfers May 02, 2013, 10:35:53 AM #1 One of the big benefits of colored coins is that the transactions are atomic. You can buy 10 shares in a company for 20 BTC and either the transaction happens or it doesn't. You don't get into a situation where you pay (or give the shares) and then the other transaction is reversed for some reason. One way of doing it would be to verify that the other transaction was buried in a second chain. This could involve checking that a token was in a merkle tree. Step 1: verify that the token matches the desired token OP_DUP [target-token] OP_EQUALVERIFY Step 2: build up merkle tree (fixed depth) [3] OP_ROLL OP_IF OP_SWAP OP_ENDIF OP_CAT OP_HASH256 It is assumed the input is [true/false] [child1] [child2] If true, then child1 and child2 are swapped and then the hash is computed. OP_CAT is currently disabled. The output is [parent] The spend would be [true/false] [merkle-node] [true/false] [merkle-node] [true/false] [merkle-node] [token] When this step is finished, then the merkle root would be on the stack. Step 3: scan the alt-chain headers OP_SWAP would be needed before starting to move the merkle root back one. The alt-chain is assumed to have only 3 fields in its header. The input is [nonce] [merkle-root] [previous-hash] OP_CAT OP_CAT OP_HASH256 OP_DUP OP_LESS_THAN [target] OP_VERIFY [2] OP_PICK OP_EQUALVERIFY This computes the hash and checks that it meets difficulty. It then checks that the "previous-hash" for the next header matches the hash of this header. This would be repeated for as many blocks as required. The effect of all of these steps is that it is a script that is released if the token is buried in another chain at least a certain number of blocks deep. To spend, you have to provide the merkle path to the root and then the nonce and merkle root for the headers which build on this one. Depending on the value of the transactions more confirming blocks would be required. The OP_CAT opcode is currently disabled. It isn't really clear what a disabled opcode means. If it isn't accepted, then it effectively doesn't exist. 1LxbG5cKXzTwZg9mjL3gaRE835uNQEtWF

0.0065 BTC / GHs for 5 years. NO FEES!

PB Mining

Advertised sites are not endorsed by the Bitcoin Forum. They may be unsafe, untrustworthy, or illegal in your jurisdiction. [Advertise here.](#)

iddo
Sr. Member



Activity: 343



 **Re: Alt chains and atomic transfers**
May 07, 2013, 10:23:49 AM

#2

I'm not completely sure whether I understood your objective here, and I also didn't I understand some of the specifics.

If the objective is to trade (colored) coins between two chains (e.g. Bitcoin and Litecoin) where the trade is atomic, in the sense that either both parties received the coins that they wanted, or neither party did, then I think that we can use this simple protocol:

Ignore

- 1) Alice has n1 bitcoins, Bob has n2 litecoins, and they want to trade them.
- 2) Bob selects a random password X and broadcasts to the Litecoin network a transaction that spends his n2 litecoins to a script that requires the usual OP_CHECKSIG for Alice's Litecoin address as well as "OP_SHA256 Y OP_EQUAL" where Y=SHA256(X), meaning that Alice (and only Alice) can spend those n2 litecoins, but only if she knew an X that satisfies Y==SHA256(X). This transaction is public, so Alice and everyone else see it on the Litecoin blockchain.
- 3) Alice now broadcasts to the Bitcoin network a similar transaction that allows Bob to spend her n1 bitcoins only if he reveals an X that gives Y==SHA256(X), i.e. a transaction with OP_CHECKSIG for Bob's Bitcoin address and "OP_SHA256 Y OP_EQUAL".
- 4) If Bob spends the n1 bitcoins (e.g. by sending them to another Bitcoin address of his) then he necessarily reveals X, so now Alice can do the same and spend the n2 litecoins.
- 5) To avoid extortion, we can use nlocktime (similarly to how it's used in the Contracts wiki page), so for example the transaction on the Litecoin network in step (2) expires after 2 days (meaning that Bob could spend the coins back to himself after 2 days), and the transaction on the Bitcoin network in step (3) expires after 1 day. This implies that if Bob spends the n1 bitcoins then we can be sure that Alice will be able to spend the n2 litecoins, and if either Alice or Bob abort at any stage then no harm is done.

Sergio_Demian_Lerner

Sr. Member



Activity: 465



Ignore



Re: Alt chains and atomic transfers

May 07, 2013, 12:43:36 PM

3

Check this thread. I think is similar to what you say.

P2PTradeX: P2P Trading between cryptocurrencies (<https://bitcointalk.org/index.php?topic=91843.0>)

TierNolan

Sr. Member



Activity: 420



Ignore



Re: Alt chains and atomic transfers

May 07, 2013, 12:59:34 PM

4

Quote from: iddo on May 07, 2013, 10:23:49 AM

I'm not completely sure whether I understood your objective here, and I also didn't I understand some of the specifics.

Basically, it is a script that verifies that a hash has been included in an alt chain's merkle tree to a certain depth.

Quote

5) To avoid extortion, we can use nlocktime (similarly to how it's used in the Contracts wiki page), so for example the transaction on the Litecoin network in step (2) expires after 2 days (meaning that Bob could spend the coins back to himself after 2 days), and the transaction on the Bitcoin network in step (3) expires after 1 day. This implies that if Bob spends the n1 bitcoins then we can be sure that Alice will be able to spend the n2 litecoins, and if either Alice or Bob abort at any stage then no harm is done.

That isn't how locktime works. Locktime means that the transaction can't be included into the chain until the given time.

Ideally, for an atomic system, it must be possible for it to be cancelled at any stage. This is what the block chain does, in order to establish ordering of transactions.

Step 1: Bob sends coin unlocked by Alice's public key and X for sha(X)

If the transaction ends here, Bob loses his money

Step 2: Alice sends coins unlocked by Bob's public key and X for sha(X)

If the transaction ends here, Alice loses her money. Bob can spend his at some later time.

Step 3: Bob sends X for sha(X) to Alice or to block chain

Transaction is completed.

What is needed is something like

Step 1: Bob sends coin unlocked by [Alice's public key and Proof step 3 happened] or [Proof step 3 didn't happen]

If the transaction ends here, Bob loses his money

Step 2: Alice sends coins unlocked by [Bob's public key and Proof step 3 happened] or [Proof step 3 didn't happen]

If the transaction ends here, Alice loses her money. Bob can spend his at some later time.

Step 3: Bob sends X for sha(X) to Alice or to block chain

Transaction completed.

What you want is an alt chain that gives the proof.

You could have a time-stamping chain. You can submit X to the chain and then it blocks sha(X) being added and vice versa.

The script in the OP was a way to say that the proof must be buried deep in some alt chain.

1LxbG5cKXzTwZg9mJL3gaRE835uNQEtWF

TierNolan

Sr. Member



Activity: 420



Re: Alt chains and atomic transfers

May 07, 2013, 01:00:41 PM

5

Quote from: Sergio_Demian_Lerner on May 07, 2013, 12:43:36 PM

P2PTradeX: P2P Trading between cryptocurrencies (<https://bitcointalk.org/index.php?topic=91843.0>)

Yeah. In fact, having the alt-chain support the system is better than trying for BTC script.

Ignore

1LxbG5cKXzTwZg9mJL3gaRE835uNQEtWF

iddo
Sr. Member
●●●●●

Activity: 343



Ignore

Re: Alt chains and atomic transfers
May 07, 2013, 02:09:23 PM

#6

Quote from: TierNolan on May 07, 2013, 12:59:34 PM

Quote

5) To avoid extortion, we can use nlocktime (similarly to how it's used in the Contracts wiki page), so for example the transaction on the Litecoin network in step (2) expires after 2 days (meaning that Bob could spend the coins back to himself after 2 days), and the transaction on the Bitcoin network in step (3) expires after 1 day. This implies that if Bob spends the n1 bitcoins then we can be sure that Alice will be able to spend the n2 litecoins, and if either Alice or Bob abort at any stage then no harm is done.

That isn't how locktime works. Locktime means that the transaction can't be included into the chain until the given time.

I meant something similar to what gmaxwell described [here](#):

"You first create the ZKP payment transaction which requires (Password+Their_signature) OR (Their signature plus Your signature). You keep this transaction private. You then write a new transaction, the refund transaction, which spends the payment back to you but has an nlocktime set in the future (e.g. 1000 blocks from now). You sign it, and give it to the other party to sign. He is able to sign it without actually seeing the payment transaction (he only sees its hash). When he returns it, you then release the payment transaction. If he does not redeem the payment transaction before the locktime expires you transmit the refund and recover it yourself. Because of the locktime you are unable to steal the payment back right after sending it to him."

Do you agree that the simple protocol that I described gives atomic trade across chains with no possibilities for extortion?

ripper234
Hero Member
●●●●●

Activity: 1092



Ron Gross



Ignore

Re: Alt chains and atomic transfers
May 21, 2013, 08:25:59 AM

#7

I had some difficulty Googling for a clear explanation of atomic cross-chain trading, so I gathered my findings into [this wiki page](#).

I must admit I do not fully understand [Mike Hearn's explanation](#) as presented, I suggest we use the wiki page to develop and clarify the solution as needed.

Please do not pm me, use ron@mastercoin.org instead
Mastercoin Executive Director
Co-founder of the Israeli Bitcoin Association

TierNolan
Sr. Member
●●●●●

Activity: 420



Ignore

Re: Alt chains and atomic transfers
May 21, 2013, 01:31:21 PM

#8

Quote from: ripper234 on May 21, 2013, 08:25:59 AM

I must admit I do not fully understand [Mike Hearn's explanation](#) as presented, I suggest we use the wiki page to develop and clarify the solution as needed.

This basically works by sending the 2 coins to a script that can only be spent if you have the recipient's public key and the x for a matching H(x).

Step 1

A generates H(x)

A sends coins to <B's public key> and x for H(x)

Step 2

B knows that A has sent his coins, so

B sends coins to <A's public key> and x for H(x)

Step 3

A sends x to B.

The can now both spend the coins they received.

If A refuses to perform step 3, when A spends the coins B sent, he has to provide x to the chain. At that point B can spend his coins.

The worst A can do is defect and refuse to unlock the coins he sent B, until he wants to spend the coins he received.

This is potentially a problem.

1LxbG5cKXzTwZg9mJL3gaRE835uNQEtWF

iddo
Sr. Member
●●●●●

Activity: 343



Ignore

Re: Alt chains and atomic transfers
May 21, 2013, 03:36:50 PM

#9

Quote from: TierNolan on May 21, 2013, 01:31:21 PM

The worst A can do is defect and refuse to unlock the coins he sent B, until he wants to spend the coins he received.

This is potentially a problem.

No it's not. I assume that you mean that the problem is that B's coins will remain locked until A agrees to unlock, but this is solved with locktime as described by gmaxwell, see also [here](#).

TierNolan
Sr. Member
●●●●●

Activity: 420



Re: Alt chains and atomic transfers
May 21, 2013, 05:31:26 PM

#10

Quote from: iddo on May 21, 2013, 03:36:50 PM

No it's not. I assume that you mean that the problem is that B's coins will remain locked until A agrees to unlock, but this is solved with locktime as described by gmaxwell, see also [here](#).

Ignore

Sounds reasonable.

A picks a random number x

A creates TX1: "Pay w BTC to <B's public key> If (x for H(x) known and signed by B) or (signed by A & B)"

A creates TX2: "Pay w BTC from TX1 to <A's public key>, locked 48 hours in the future, signed by A"

A sends TX2 to B

B signs TX2 and returns to A

1) A submits TX1 to the network

B creates TX3: "Pay v alt-coins to <A-public-key> If (x for H(x) known and signed by A) or (signed by A & B)"

B creates TX4: "Pay v alt-coins from TX3 to <B's public key>, locked 24 hours in the future, signed by B"

B sends TX4 to A

A signs TX4 and sends back to B

2) B submits TX3 to the network

3) A spends TX3 giving x

4) B spends TX1 using x

This is atomic (with timeout). If the process is halted, it can be reversed no matter when it is stopped.

Before 1: Nothing public has been broadcast, so nothing happens

Between 1 & 2: A can use refund transaction after 48 hours to get his money back

Between 2 & 3: B can get refund after 24 hours. A has 24 more hours to get his refund

After 3: Transaction is completed by 2

- A must spend his new coin within 24 hours or B can claim the refund and keep his coins

- B must spend his new coin within 48 hours or A can claim the refund and keep his coins

For safety, both should complete the process with lots of time until the deadlines.

1LxbG5cKXzTwZg9mjL3gaRE835uNQEtWF

ripper234
Hero Member
★★★★★

Activity: 1092



Ron Gross



Ignore

Re: Alt chains and atomic transfers
May 25, 2013, 09:17:53 PM

#11

Thanks for the great explanation, now it's all clear.

Please do not pm me, use ron@mastercoin.org instead
Mastercoin Executive Director
Co-founder of the Israeli Bitcoin Association

TierNolan
Sr. Member
★★★★★

Activity: 420



Ignore

Re: Alt chains and atomic transfers
August 02, 2013, 08:25:13 PM

#12

I was thinking of trying this out.

Does anyone know if there is a pool like Eligius for Litecoin that accepts non-standard transactions?

1LxbG5cKXzTwZg9mjL3gaRE835uNQEtWF

jl2012
Hero Member
★★★★★

Activity: 714



Ignore

Re: Alt chains and atomic transfers
August 03, 2013, 04:19:56 AM

#13

Quote from: TierNolan on August 02, 2013, 08:25:13 PM

I was thinking of trying this out.

Does anyone know if there is a pool like Eligius for Litecoin that accepts non-standard transactions?

Maybe do it on testnet

Donation address: 1CizPrEjdN4FJcqdlDgVLzT8tgCXxT5ion

TierNolan
Sr. Member
★★★★★

Activity: 420



Ignore

Re: Alt chains and atomic transfers
August 03, 2013, 08:15:41 AM

#14

Quote from: jl2012 on August 03, 2013, 04:19:56 AM

Maybe do it on testnet

Does Litecoin have a testnet, I guess that would work?

Anyway, I was thinking that the feature could be achieved using channels. Both traders would open a channel to handle the trade.

This can be done with standard multi-sig transactions.

LTC fees are 0.1 LTC which is pretty high.

If nodes created channels to 8 other nodes, then you could get a distributed exchange. Each node would act as market maker between the peers it was connected to.

It would tell the other nodes what its bid/ask price was. Persistent nodes could potentially generate profit for their operators.

1LxbG5cKXzTwZg9mJL3gaRE835uNQEtWF

socrates1024
Member



Activity: 113



Andrew Miller



Ignore



Re: Alt chains and atomic transfers
October 10, 2013, 07:31:03 PM

#15

I've been skeptical of atomic cross-chain trades for a while now. In particular I edited the Contracts wiki page to explain that the proposed technique (the "luxgladius" protocol) has a fatal race condition. But now I'm convinced that solution proposed here works and has no race condition. And it's simpler than P2PTradeX, which is good (although P2PTradeX may still be useful for other things). So, here's my writeup of how this works. Unless I realize I missed something, I want to revise the Contracts wiki page to point to TierNolan's description by default one, and remove the disclaimer that cross-trades don't work! <https://en.bitcoin.it/wiki/Contracts>

Quote

This is an explanation of TierNolan's atomic cross transaction technique. It's similar to one by luxgladius, except this protocol avoids a race condition by using asymmetry - only Alice has a secret, and the timeout on one chain is much longer than the other. <https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949>

Alice has 1btc (Bitcoin) and Bob has 1atc (Altcoin). They would like to conduct an atomic exchange, so that after a finite time, either Alice has 1atc and Bob has 1btc, or the transaction was aborted and their positions are both unchanged. We assume that both the Bitcoin and Altcoin chains are loosely synchronized to each other, such that neither chain gets more than 12 hours ahead of the other.

Phase 1: Alice bails in

Alice draws a random secret x . Alice constructs TX1 to bail in a bitcoin. TX1 has a single txout with 1 btc, containing the following scriptPubkey:

Code:

```
OP_IF
  // Refund for A
  2 <pubkeyA> <pubkeyB> 2 OP_CHECKMULTISIGVERIFY
OP_ELSE
  // Ordinary claim for B
  OP_HASH160 <H(x)> OP_EQUAL <pubkeyB> OP_CHECKSIGVERIFY
OP_ENDIF
```

Alice signs TX1 but does not yet publish it. She creates TX2, her refund transaction, which has a locktime of +48 hours. It contains TX1[0] as a txin, and the txout is a standard spend to A. The scriptSig for this txin looks like

Code:

```
<sigB2> <sigA2> 1
```

which proceeds through the main branch of the IF. Alice signs TX2, obtaining sigA2.

Alice transmits TX2 to Bob, and requests that Bob signs and returns it. Once Alice receives Bob's signature sigB2, then Alice publishes TX1, knowing that she will be able to claim her refund after 48 hours, as long as she does not reveal x . Alice is now bailed in.

Phase 2: Bob bails in

Bob received TX2 in the first phase, and therefore knows $H(x)$. Bob also signed Alice's refund transaction, but that will not be active until after 48 hours. If Bob is honest, he'll get what he wants within 24 hours in any case. Bob has also seen that TX1 has been published.

Bob creates TX3, which bails in an Altcoin. TX3 is symmetric with TX1, and has a single txout with 1atc, containing the following scriptPubkey:

Code:

```
OP_IF
  // Refund for B
  2 <pubkeyA> <pubkeyB> 2 OP_CHECKMULTISIGVERIFY
OP_ELSE
  // Ordinary claim for A
  OP_HASH160 <H(x)> OP_EQUAL <pubkeyA> OP_CHECKSIGVERIFY
OP_ENDIF
```

Bob also creates TX4, which is his own Altcoin refund. TX4 is timelocked for only 24 hours, uses TX3[0] as a txin, and the scriptSig looks like

Code:

```
<sigB4> <sigA4> 1
```

which proceeds through the main branch of the if. Bob signs TX4, obtaining sigB, and transmits TX4 to Alice, asking her to sign it. When Bob receives sigA4, he can safely publish TX3, knowing that he will be able to abort and reclaim his coin after 24 hours. Bob is now bailed in.

Phase 3: Alice reveals x and triggers both transactions.

Alice has 24 hours to claim the Altcoin, using the following scriptSig:

Code:

```
<sigA4'> <x> 0
```

which triggers the ELSE branch of the conditional. However in doing so, she must necessarily reveal x . If Alice reveals x within 24 hours, then Bob can also claim his Bitcoin before 48 hours:

Code:

```
<sigB2'> <x> 0
```

This is atomic because:

- each party "bails in" their coins only after receiving a "refund signature" from the other party.
- the refund delay is significantly larger on the Bitcoin chain than on the Altcoin chain, and Alice, who initially knows x , bails in her coin first.

amiller on freenode / 19G6VFcV1qZJxe3Swn28xz3F8gDKTznwEM

[my twitter] [research@umd]

I study Merkle trees, credit networks, and Byzantine Consensus algorithms.

Sergio_Demian_Lerner

Sr. Member

Activity: 465

Ignore

Re: Alt chains and atomic transfers

October 10, 2013, 08:36:59 PM

#16

This socrates1024 protocol seems to be much better than some others I saw. I can't find any drawback.

Sukrim

Hero Member

Activity: 980

Ignore

Re: Alt chains and atomic transfers

October 10, 2013, 10:14:24 PM

#17

As far as I understand, Alice cannot put out a general offer though, she can only trade with Bob and vice versa. This might be a potential drawback. On the other hand this is about atomic transfers, not cross chain trade offers...

What if Alice (because she likes to do arbitrage and is speculating on shifting values) reveals her x very late (after 23:59h and 59 seconds)? This would lead to the case that she doesn't get her altcoin but bob can still claim the Bitcoin, right? Also might it be the case that this leads to some issues in the altcoin network, if half of the nodes see her revealing X thus granting her the altcoin and the other half believe that the time is over and the altcoin should return to Bob?

<https://bitfinex.com/?refcode=x5K9YtL3Zb> <-- leveraged trading of BTCUSD, LTCUSD and LTCBTC (long and short) - 10% discount on fees for the first 30 days with this refcode: **x5K9YtL3Zb**
Mail me at Bitmessage: **BM-BbiHIVv5qh858ULsyRDtpRrG9WjXN3xf** | <https://just-dice.com/> <-- Bitcoin gambling done right!

socrates1024

Member

Activity: 113

Ignore

Re: Alt chains and atomic transfers

October 10, 2013, 10:19:34 PM

#18

Quote from: Sukrim on October 10, 2013, 10:14:24 PM

What if Alice (because she likes to do arbitrage and is speculating on shifting values) reveals her x very late (after 23:59h and 59 seconds)? This would lead to the case that she doesn't get her altcoin but bob can still claim the Bitcoin, right?

Alice can only harm herself by waiting so close to the deadline. This is OK.

amiller on freenode / 19G6VFcV1qZjxe3Swn28xz3F8gDKTznwEM
[my twitter] [research@umd]
I study Merkle trees, credit networks, and Byzantine Consensus algorithms.

Sukrim

Hero Member

Activity: 980

Ignore

Re: Alt chains and atomic transfers

October 11, 2013, 07:36:19 AM

#19

Well, then there's still the issue with altcoins being (far) less secure than Bitcoin:

Alice reveals X, Bob 51% attacks Altcoin to undo/prevent Alice's transaction for the remaining n hours, Bob redeems Bitcoin. Are Altcoins just viewed as "secure" in this scenario and is this a reasonable assumption in reality? This again might be seen as not relevant, as not the protocol but the chain is attacked.

Still it would be nice to allow Alice to trade with anyone, not just a certain Bob.

<https://bitfinex.com/?refcode=x5K9YtL3Zb> <-- leveraged trading of BTCUSD, LTCUSD and LTCBTC (long and short) - 10% discount on fees for the first 30 days with this refcode: **x5K9YtL3Zb**
Mail me at Bitmessage: **BM-BbiHIVv5qh858ULsyRDtpRrG9WjXN3xf** | <https://just-dice.com/> <-- Bitcoin gambling done right!

agorism

Jr. Member

Activity: 36

Ignore

Re: Alt chains and atomic transfers

November 25, 2013, 12:48:05 AM

#20

Socrates1024, I have never written any forth code, but I am pretty sure that how you are using "op_if" is incorrect. See links for details: <http://bitcoin.stackexchange.com/questions/2317/script-op-if-etc-clarifications>
This also talks about if statements in forth: <http://theforthsource.com/guide10.html>

This is my attempt to re-write it:

```
//I am assuming that if <x> is known, that it is already on the stack.
//The first "2" is because of bitcoin's bug. The second "2" is an input for op_checkmultisig
2 <pubkeyMine> <pubkeyOther> 2 OP_CHECKMULTISIG OP_DUP //OP_DUP (duplicate top of stack) is here
//because OP_NOTIF pops an item off the stack.
//If checkmultisig worked, then we want to skip to the end and verify. If checkmultisig did not work,
//then we still have another chance
OP_NOTIF
— OP_DROP //This gets rid of the 0 that OP_checkmultisig gave us. Now <x> should be on top.
— OP_HASH160 <H(x)> OP_EQUAL //If the hash of x is <H(x)>, then this will leave a 1 on top. otherwise, it leaves a 0.
— <pubkeyOther> OP_CHECKSIG //If B has signed, put a 1 on top
— 1 OP_EQUAL OP_EQUAL //If the top 2 items were (1, 1) then, delete them and put a 1 on top. Otherwise delete them
// and put a 0 on top. T
OP_ENDIF
— OP_VERIFY
Now I understand what you were doing Socrates1024
```

Pages: [1] 2 All

print

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **Alt chains and atomic transfers**

« previous topic next topic »

Jump to: => Development & Technical Discussion go



Bitcoin Forum

simple machines forum

April 15, 2014, 06:20:58 AM

Welcome, **Guest**. Please login or register.

Forever

Login with username, password and session length

News: ♦♦ A bug in OpenSSL, used by Bitcoin-QT/Bitcoin Core, could allow your bitcoins to be stolen. Immediately updating Bitcoin Core to 0.9.1 is required in some cases, especially if you're using 0.9.0. Download. [More info.](#)

♦ The same bug also affected the forum. Changing your forum password is recommended.

[HOME](#) [HELP](#) [SEARCH](#) [DONATE](#) [LOGIN](#) [REGISTER](#)

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **P2PTradeX: P2P Trading between cryptocurrencies**

« previous topic next topic »

Pages: [1] 2 All

print

Author Topic: P2PTradeX: P2P Trading between cryptocurrencies (Read 4137 times)

Sergio_Demian_Lerner

Sr. Member



Activity: 467



Ignore

P2PTradeX: P2P Trading between cryptocurrencies

July 05, 2012, 11:49:48 PM

#1

I want to describe a new P2P protocol I'm implementing for an alternate cryptocoin as a proof of concept.

The protocol allows cross-coin p2p trading without a central point. It seems to me that it is the "holly grail" of alternate crypto coins. And this is an idea that can change the cryptocoin ecosystem for good, where all coins trade against each other. The benefit for alternate chains is enormous: they don't need to provide an exchange site, they can trade automatically against Bitcoin. It also means that alternate cryptocurrencies will rely on Bitcoin and will support Bitcoin because they need it to enter the cryptocurrency game.

This is short explanation:

Suppose that there are two crypto-coins, XC and BTC. Each coin XC and BTC has its own blockchain and client. The user A has some XC and want to buy some BTC in return. User B wants the opposite. First both parties find each other (in a central directory or by a P2P protocol) and fix the trade price (A pays "a" XC and B pays "b" BTC back). There are two payments A->B (in XC) and B->A (in BTC). We will call these payments first and second payment, respectively. Both users have an address in the XC system and an address in the Bitcoin system.

The protocol works as follows:

1. User A commits to the first payment of "a" XC to the address of user B in the system XC. This is a special payment with a "contract" that is automatically allowed if a certain "proof" is published as a special transaction in a limited interval after the publication.

2. User B sends b BTC to A via Bitcoin in a standard way. This is the second payment.

The contract specifies that a piece of the Bitcoin blockchain (a branch) should be copied into a special transaction called "proof" into the XC blockchain to prove the second payment has actually taken place.

The contract also specifies:

- The chain branch size (N). This is how much effort in terms of confirmations (PoW) must be added after the block where the second payment is published.
- The hash of the block where the branch should start (root block). The root block should be chosen to be some blocks in the past to avoid choosing a block that will be discarded by a competing branch. For example, if current block is BLK and the previous block of BLK is Prev(BLK), parties can choose a root in Prev^3(BLK) with a length of at least 9 blocks (6 confirmations after current block)
- The maximum number of blocks after the root block where the second payment can appear. This is to prevent the payment being done just after the XC contract interval has expired, thus making the trade one-way only.

3. When the proof transaction (that matches the contract) is published in an XC block, clients automatically accept the first payment (that specifies the contract), thus paying "a" XC from A to B.

Notes:

1) Not the full Bitcoin blockchain branch needs to be included in the XC blockchain. Only the headers of the N blocks, the Merkel branch that proves the existence of the second payment and the transaction of the second payment. Generally less than 800 bytes are required for the full "proof".

2) All this protocol works transparently. The user only setups maximum/minimum trade prices and trade quantities.

3) Generally B is the most interested that the "proof" is published, but if B doesn't do it (and the second payment was done) then A itself can publish the proof.

4) The "proof" is included in a block as a normal transaction and can pay a fee to the miner, but the minimum fee is specified in the contract. Both parties should take care of specifying fees that allow the transactions to be selected by the miners to be included in a block. If the fee is not enough, a higher fee can be specified when the proof is sent. It is recommended that the fee for the miner of the "proof" be much higher than normal fees to give a strong incentive to miners to include this transaction in all competing blocks.

5) The security of the system is relies on the premise that no party can build a blockchain branch longer and faster than the "global" branch in a limited interval (e.g. 20 blocks). Also the proof cannot be constructed in advance, since the root block (which is almost the current block) is specified in the contract.

6) If the trade fails because the interval has elapsed and the second payment was not done, the first payment is canceled.

7) The coins committed in the first payment cannot be used until the contract is canceled or it is accepted.

I have a proof of concept of this protocol working. I will release the code when it's ready. The system is implemented in a way to

understand contracts for all other alternate currencies and unknown ones. The user specifies a "template" for the second payment message, with placeholders for the fields that are unknown (the transaction signature) and fixed values for the remaining fields (amount of money, public key of recipient, etc.). And also a template for the block format (field of linkage, hash algorithm, etc.).

Best regards,
Sergio.

PS: I named the protocol P2PTradeX, because there was no result when I googled that word.

PRE-ORDER CLOUD MINING POWER.

CHEAPEST PRICE.
2 PH/S IN STOCK.

INVEST NOW >

Advertised sites are not endorsed by the Bitcoin Forum. They may be unsafe, untrustworthy, or illegal in your jurisdiction. [Advertise here.](#)

Spekulatius
Hero Member



Activity: 868



Zappa di bibbedi boo!



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies

July 06, 2012, 01:53:29 AM

#2

Nice idea! (dont take my word for it, I have no clue on the technical side what so ever 😊)

If I understand correctly, your protocol only works for digital assets, like crypto currencies. Have you considered working on APIs of **online banks** as well? Since they would be the real "**holy grail**" because changing BTC for traditional currencies is what always concerns people most and where the largest market would be for your system.

☆☆BITCOIN -The Documentary- The first feature-length docu on Bitcoin!☆☆

Be part of it! Be creative! Give us your best ideas, personal contributions and maybe a little donation 😊

Sergio_Demian_Lerner
Sr. Member



Activity: 467



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies

July 06, 2012, 02:07:09 AM

#3

It could in theory be done if banks emitted some kind of digitally signed document when they sent money to an account (which they don't)

Take into account that since bitcoins can be traded against USD or any other fiat money relatively easy now, trading against Bitcoin is "almost" like trading against fiat money.

Sergio.

gmaxwell
Moderator
Hero Member



Activity: 1064



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies

July 06, 2012, 02:17:02 AM

#4

You can do this with far less complexity, and the same extortion risk like this:

Alice and Bob want to trade Acoins and Bcoins. They each pick secrets. Asecret and Bsecret. They each tell each other $H(\text{Asecret}) = A_h$, and $H(\text{Bsecret}) = B_h$.

After agreeing on the amounts they each form transactions

AAlice 1 Acoin -> $A_{\text{Bob_pub}} + H(x) = A_h + H(y) = B_h$

BBob 1 Bcoin -> $B_{\text{Alice_pub}} + H(x) = A_h + H(y) = B_h$

once both transactions are in one or the other reveals their secret out of band. The redemption works for all or none.

As mentioned, there is holdup risk here, which could be solved by adding semi-trusted observer keys to optionally release the funds in the case of defection.

But I think the fragment protocol described here has the same risk. (and the obvious ways to do avoid it make it possible to rip off one party by delaying the mining of the solution).

Sergio_Demian_Lerner
Sr. Member



Activity: 467



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies

July 06, 2012, 02:30:16 AM

#5

Could you clarify this:

Quote from: gmaxwell on July 06, 2012, 02:17:02 AM

After agreeing on the amounts they each form transactions

AAlice 1 Acoin -> $A_{\text{Bob_pub}} + H(x) = A_h + H(y) = B_h$

BBob 1 Bcoin -> $B_{\text{Alice_pub}} + H(x) = A_h + H(y) = B_h$

once both transactions are in one or the other reveals their secret out of band. The redemption works for all or none.

Is " $A_{\text{Bob_pub}} + H(x) = A_h + H(y) = B_h$ " the scriptPub ?

Are the values x and y provided by a scriptSig ?

gmaxwell
Moderator
Hero Member



Activity: 1064



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies

July 06, 2012, 02:31:53 AM

#6

Quote from: Sergio_Demian_Lerner on July 06, 2012, 02:30:16 AM

Could you clarify this:

Is " $A_{\text{Bob_pub}} + H(x) = A_h + H(y) = B_h$ " the scriptPub ?

Are the values x and y provided by a scriptSig ?

Correct. The script sig would push the x and y values and the script pubkey would hash and compare them.

(You can also reduce the extortion risk by using nlocktime, on specially constructed refund transactions. E.g. "Abob_pub + H()+H()") OR "Abob_pub2&&Aalice_pub" and have bob sign a refund txn with far future nlocktime using abob_pub2)

Serenata
Full Member



Activity: 237



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 08:29:03 AM

#7

Trading between cryptocurrencies is a nice idea. In the years to come, it's gonna get more and more useful.

BitcoinX.gr - Το ελληνικό στέκι του Bitcoin

My GPG Key

Spekulatius
Hero Member



Activity: 868



Zappa di bibbedi boo!



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 09:27:27 AM

#8

Quote from: Sergio_Demian_Lerner on July 06, 2012, 02:07:09 AM

It could in theory be done if banks emitted some kind of digitally signed document when they sent money to an account (which they don't)

Take into account that since bitcoins can be traded against USD or any other fiat money relatively easy now, trading against Bitcoin is "almost" like trading against fiat money.

Sergio.

For the time being, its a great advancement if you can get it to work reliably but after all decentralized BTC/fiat exchange feels like the difference between a nice gimmick and the **real deal**.

☆☆BITCOIN -The Documentary- The first feature-length docu on Bitcoin!☆☆

Be part of it! Be creative! Give us your best ideas, personal contributions and maybe a little donation😊

Sergio_Demian_Lerner
Sr. Member



Activity: 467



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 02:31:51 PM

#9

Let's talk about GMAX protocol. The protocol is almost perfect. But as you say, a timeouts would be necessary to prevent the coins getting blocked if one party refuses to reveal the secret.

I see two problems:

1) Suppose the payment A->B is done on cryptocoln X and the B->A payment on cryptocoln Y. If Alice refuses to tell the secret "x" after Bob has told the secret "y", Alice can wait until the last moment when the timeout in Y will elapse and send a transaction that grabs the payment B->A, leaving no time for Bob to send the corresponding transaction in the cryptocoln X. The A->B transaction will timeout, and the money will return to A. Obviously adding a third party would prevent this, but the protocol must be TTF-free.

2) With the scripting system, Alice could encrypt the value x so that it's difficult for Bob to manually find x from the transaction scriptSig. For example the scriptSig reference to x could be "r t + ~" so that $x = \sim(r+t)$. Then the transaction would need special treatment to extract "x" and "y" in cleartext.

These problem are not present in P2PTradeX. I know it's a bit more complex, but it works and has no drawback.

Best regards,
S.

Mike Hearn
Hero Member



Activity: 1218



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 02:46:46 PM

#10

BTW, this idea is already discussed on the Contracts wiki page for some time already. I'd suggest starting new proposals that are variants of those ideas by referencing them, and then explaining how to do better:

https://en.bitcoin.it/wiki/Contracts#Example_5:_Trading_across_chains

To me the interesting thing is not a P2P cryptocurrency exchange (there's only one that matters), but P2P exchange for fiat. It can work by implementing a form of broadcast Ripple whereby people set up trust relationships between each other and settle up fiat asynchronously. Buying or selling BTC would commit the cryptocurrencies instantly, and adjust the debts between people in the exchange graph. You would then get the money owed to you by friends/colleagues whenever and however is most convenient.

By connecting yourself to this social graph of trust relationships, you can aggregate multiple paths through the network to reach as much value as you need. Of course, it doesn't help you if you aren't introduced to Bitcoin by somebody you know, but it's the next step in the evolution of sites like localbitcoin/bitcolnary/etc.

12LMm82ZgAzf7yNDpPydEYxEr4Ap7XtSSK

Sergio_Demian_Lerner
Sr. Member



Activity: 467



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 03:38:56 PM

#11

Dear Mike,
As I posted before, that kind of P2P trading (GMAX/luxgladius) does not work well because of the need for timeouts.

My proposal **is not** based on Example 5. Please read carefully. The only thing in common is that both use the word "Contract".

P2P exchange for fiat is another interesting problem, but it has already been discussed in some other threads. This thread is about P2P cryptocoln exchange.

I think Bitcoin would benefit from any new possible P2P exchange (for example SecondLife linden dollars).

gmaxwell
Moderator
Hero Member
●●●●●●●●

Activity: 1064



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 07:35:13 PM

#12

Quote from: Sergio_Demian_Lerner on July 06, 2012, 02:31:51 PM

The A->B transaction will timeout, and the money will return to A. Obviously adding a third party would prevent this, but the protocol must be TTF-free.

You can't create timeouts in Bitcoin, locktime sets the earliest valid time. So you pre-create the refunds... but yes because the timeout wouldn't be identical in both chains there is some exposure.

I don't see how you avoid extortion in your scheme that wouldn't just apply here. E.g. you could do the pre-created refund tx only on one side in what I described... and have the other side be the send-hash-first side. If they don't you perform the refund after the timeout, thus no race.

(Not that in general I think that using chain fragments is bad— but I think it's interesting in external oracles, and less in chains themselves)

Quote

2) With the scripting system, Alice could encrypt the value x so that it's difficult for Bob to manually find x from the transaction scriptSig. For example the scriptSig reference to x could be "r t + ~" so that $x \approx (r+t)$. Then the transaction would need special treatment to extract "x" and "y" in cleartext.

Both users would txouts before releasing a secret, so any funny business would have to be in the scriptsig. If your scriptpubkey looks like $H(\text{stack}) == 123456...$ Then I could always copy from your scriptsig anything the satisfied that, even if it computed the value. Though yes, a program for this would have to have a pretty general solver to figure out how to extract the data. So long as the abort transactions are sufficiently far out that shouldn't be a problem, and you'd have proof of your counterparty being evil.

Sergio_Demian_Lerner
Sr. Member
●●●●●●●●

Activity: 467



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 08:27:14 PM

#13

Quote from: gmaxwell on July 06, 2012, 07:35:13 PM

I don't see how you avoid extortion in your scheme that wouldn't just apply here.

No extortion is possible.

First let's assume a system where transactions/contracts have a timeout field if the proof is not published in a block. (I'm using here my P2PTradeX terminology of what a contract and a proof is)

Since the contract specifies the length of the branch, it should specify a length whose time is much shorter than the timeout for the contract.

For example, a Alice publishes the following contract:

This contract will pay 100 XC from the prevout X to Bob's address if and only if a "proof" is published before 20 blocks from the publication of this contract.

The contract is this:

- The proof consist of a Bitcoin block branch containing a special tx.
- The block branch must be of length 6 after the block where tx is found,.
- Tx must be a Bitcoin transaction sending 100 BTX to address 37muSN5ZrukVTvyVh3mT5ZcSew9L9CBare (Alice's address)
- The Root of the branch must be 00000000000000ca853e8e3faa30451909ec22db537717653b8bb6949fbe175c (this is the hash of a block 30 minutes in the past)
- The maximum length of the branch should be 10 blocks

Because there is NO SYMMETRY between Alice and Bob, the timeout on XC is setup to be 20 blocks, and the timeout in Bitcoin is setup to be only 10 blocks. If Bob manages to create the tx in Bitcoin in the time specified by the contract, then there is plenty of time (10 blocks) to publish the corresponding proof in the XC system.

As far as I can see, there is no way of extortion.

Best regards, Sergio.

gmaxwell
Moderator
Hero Member
●●●●●●●●

Activity: 1064



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 09:46:05 PM

#14

Quote from: Sergio_Demian_Lerner on July 06, 2012, 08:27:14 PM

This contract will pay 100 XC from the prevout X to Bob's address **if and only if** a "proof" is published before 20 blocks from the publication of this contract.

Ah, I hadn't seen how you handled the non-completion case. So in Bitcoin-style hashchains when that contract was mined it would consume the inputs forever unconditionally. If it did not, you could not have pruning.

Do you plan on breaking pruning or having the satisfaction rules change as a function of height or?

Sergio_Demian_Lerner
Sr. Member
●●●●●●●●

Activity: 467



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
July 06, 2012, 10:15:59 PM

#15

Quote from: gmaxwell on July 06, 2012, 09:46:05 PM

Ah, I hadn't seen how you handled the non-completion case. So in Bitcoin-style hashchains when that contract was mined it would consume the inputs forever unconditionally. If it did not, you could not have pruning.

Do you plan on breaking pruning or having the satisfaction rules change as a function of height or?

Honestly I haven't thought how to implement it on Bitcoin as the system to support these contracts (the first payment). I'm testing it on an alternate cryptocurrency which does not use "prevouts" but "account numbers" and allows contracts to become effective conditionally. Then I can trade against Bitcoin in the way described. To implement this kind of contracts in Bitcoin some rules would have to change...

That's why I said this could be an advantage for ALTERNATE cryptocurrencies, and only indirectly for Bitcoin.

That's why I said this could be an advantage for ALTERNATE cryptocurrencies, and only indirectly, for Bitcoin.

cunicula
Hero Member
★★★★★

Activity: 700



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
November 03, 2012, 06:23:37 AM

#16

I agree with Sergio here. The system sounds like it will work, but it cannot be implemented in bitcoin.

This discussion is related:

<https://bitcointalk.org/index.php?topic=120175.0>

(tl;dr : I am asking why nLockTime can't be modified to allow txns to expire. Due to hold up risk, this is necessary for pure P2P option contracts to work. As I see it Sergio is making an altcoin where txns can expire. This should make P2P option contracts possible without any intermediary.)

and

<https://bitcointalk.org/index.php?topic=121406.0>

(tl;dr : discussion of zero-trust option contracts which rely on an intermediary to serve as "broker" or "notary".)

BTW. Sergio, what system are you using to secure your new cryptocurrency? If you ask me about the 'holy grail' of cryptocurrency, it is a secure cryptocurrency with zero inflation AND simultaneously zero fees. Bitcoin cannot offer this. Ever.

Finding a way to do away with fees seems much, much, more important than cross-blockchain trading.

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

Etlase2
Hero Member
★★★★★

Activity: 756



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
November 03, 2012, 01:05:03 PM

#17

Without fees, how can you possibly solve spam? This is an ongoing and unanswered problem with email. Though the problem has been reduced by *centralized* filtering software. 😊

Decrypts

cunicula
Hero Member
★★★★★

Activity: 700



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
November 03, 2012, 01:18:08 PM

#18

Quote from: Etlase2 on November 03, 2012, 01:05:03 PM

Without fees, how can you possibly solve spam? This is an ongoing and unanswered problem with email. Though the problem has been reduced by *centralized* filtering software. 😊

I don't really mean zero fees. I mean fees like bitcoin has today (which are very very low, but sufficient to deal with spam).

If bitcoin ran just on current txn fees the aggregate hash rate would be around 30 Gigahash; about 1/4 that of the mighty PPCoin. Not secure.

The question is: How can you securely run a decentralized currency worth 100 million US on 30 Gigahash worth of external work input? If you can answer this, then fees can stay this low forever.

Of course if you figure out how to have no implicit or explicit fees at all that is even better (though it seems impossible).

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

lophie
Hero Member
★★★★★

Activity: 546



The change address was the input address!



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
November 03, 2012, 11:18:25 PM

#19

Quote from: Sergio_Demian_Lerner on July 06, 2012, 02:31:51 PM

1) Alice can wait until the last moment when the timeout in Y will elapse and send a transaction that grabs the payment B->A, leaving no time for Bob to send the corresponding transaction in the cryptocoin X.

How about if The secret release has a time limit on it? Like for example the transaction for returning the deposit in the future to the rightful owner is in 10 mins and the transaction for releasing the first secret is in 5 mins. So Alice can wait but it doesn't matter she can only squeeze bob time to 5 mins time frame.

Ask for my public key if you want everything signed 😊
I intent to hurt Jon AKA Ukyo and his family very badly in the future! I will not forget.
CoinDust: Cryptocurrencies SMS services (Exclusively for the Philippines)!

Sergio_Demian_Lerner
Sr. Member
★★★★★

Activity: 467



Ignore

Re: P2PTradeX: P2P Trading between cryptocurrencies
November 04, 2012, 01:39:46 AM

#20

Quote from: cunicula on November 03, 2012, 06:23:37 AM

BTW. Sergio, what system are you using to secure your new cryptocurrency? If you ask me about the 'holy grail' of cryptocurrency, it is a secure cryptocurrency with zero inflation AND simultaneously zero fees. Bitcoin cannot offer this. Ever.

Finding a way to do away with fees seems much, much, more important than cross-blockchain trading.

This should be probably discussed in another thread... It is another under-analyzed important subject...

My estimation is that fee price can be modeled by an equation like this:

Average fee per transaction =
$$a0 * \text{Mining_Electricity_Cost} + a1 * \text{Mining_Hardware_Cost} + a2 * \text{Avg_Transaction_Size} + a3 * \text{Avg_Verification_Time} - a4 * \text{Mining_Block_reward}$$

What I've done to reduce fees in my alternate cryptocoin is to reduce average transaction size (80 bytes average), reduce Verification time (0.1 msec average).

I don't know how to reduce electricity cost of mining, except for Proof-Of-Stake proposals.
Wasn't there a paper that analyzed Bitcoin future fee price? What does it says?

Best regards, Sergio.

Pages: [1] 2 All

[print](#)

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **P2PTradeX: P2P Trading between cryptocurrencies**

[« previous topic](#) [next topic »](#)

Jump to:

Sponsored by Private Internet Access, a Bitcoin-accepting VPN.

Powered by SMF 1.1.19 | SMF © 2006-2009, Simple Machines

WOT

WOT

Bitcoin Forum

simple machines forum

April 04, 2014, 07:17:50 PM

Welcome, **Guest**. Please login or register.

Forever
Login with username, password and session length

News: Bitcoin Core **0.9.0** is now available. [Download.](#)

[HOME](#) [HELP](#) [SEARCH](#) [DONATE](#) [LOGIN](#) [REGISTER](#)

Bitcoin Forum > Other > Alternate cryptocurrencies > Announcements (Altcoins) > **[ANN] Ethereum: Welcome to the Beginning**

« previous topic next topic »

Pages: [1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 ... 77

print

Author Topic: **[ANN] Ethereum: Welcome to the Beginning** (Read 72827 times)

Vitalik Buterin
Sr. Member

Activity: 322



Ignore

[ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 11:33:17 AM

#1

Welcome to the New Beginning

When the grand experiment that is bitcoin began, the anonymous wizard desired to test two parameters- a trustless, decentralized database enjoying security enforced by the austere relentlessness of cryptography and a robust transaction system capable of sending value across the world without intermediaries. Yet the past five years years have painfully demonstrated a third missing feature: a sufficiently powerful Turing-complete scripting language. Up until this point, most innovation in advanced applications such as domain and identity registration, user-issued currencies, smart property, smart contracts, and decentralized exchange has been highly fragmented, and implementing any of these technologies has required creating an entire meta-protocol layer or even a specialized blockchain. Theoretically, however, each and every one of these innovations and more can potentially be made hundreds of times easier to implement, and easier to scale, if only there was a stronger foundational layer with a powerful scripting language for all of these protocols to build upon. And this need is what we seek to satisfy.

Ethereum is a modular, Turing-complete contract scripting system married to a blockchain and developed with a philosophy of simplicity, universality and non-discrimination. Our goal is to provide a platform for decentralized applications - an android of the cryptocurrency world, where all efforts can share a common set of APIs, trustless interactions and no compromises. We ask for the community to join us as volunteers, developers, investors and evangelists seeking to enable a fundamentally different paradigm for the internet and the relationships it provides.

Who is Behind Ethereum?

Our fiduciary team contains 4 members:

- Vitalik Buterin → Inventor of Ethereum, Partner at KryoKit and Cofounder of Bitcoin Magazine
- Anthony Di Iorio → Founder and Executive Director of the Bitcoin Alliance of Canada, Bitcoin Decentral, KryoKit
- Charles Hoskinson → Former CEO of Invictus Innovations and Director of the Bitcoin Education Project
- Mihai Alisie → Founder of Bitcoin Magazine and Egora

We also enjoy a deep bench of developers, marketers, entrepreneurs, evangelists and partners including:

- **Dr. Gavin Wood:** Core C++ Developer
- **Geff Obscura:** Core Go Developer
- **Dr. Emanuele Costa:** Quantitative Analyst; SCRUM Master
- **Joseph Lubin:** Software Engineering, Quantitative Analyst
- **Eric Lombrozo:** Software Architect
- **Max Kaye:** Developer
- **Jonathan Mohan:** Media, Marketing and Evangelism (BitcoinNYC)
- **Wendell Davis:** Strategic Partner and Branding (Hive Wallet)
- **Anthony Donofrio:** Logos, branding, Web Development (Hive Wallet)
- **Taylor Gerring:** Web Development
- **Paul Snow:** Language Development, Software Development
- **Chris Odom:** Strategic Partner, Developer (Open Transactions)
- **Jerry Liu and Bin Lu:** Chinese strategy and translations (<http://www.8btc.com/ethereum>)
- **Hai Nguyen:** Accounting
- **Amir Shetrit:** Business Development (Colored Coins)
- **Steve Dakh:** Developer (KryoKit)
- **Kyle Kurbegovich:** Media (Cointalk)

We will switch our PoW from Dagger to a hybrid PoW/PoS system to be developed via a bountied competition conducted by our university partners and open to the general community for participation. The terms will be announced in late february including judges, specifications and the university partners.

We have partnered with Hive and Kryptokit to assist us in conducting the fundraiser and at time of launch there will be four options to participate in the exodus swap: Kryptokit, Hive, a dedicated python client and directly through the fundraiser website. We will have the testnet operational on February 1st, 2014 at <http://fund.ethereum.org>. We will also post videos on our official youtube channel explaining how one can use each of the methods.

We recommend all those who participate in the fundraiser provide an active email address to receive a cryptographically signed receipt from the exodus address of their swap as a secondary proof of stake in Ethereum's blockchain. The primary proof of stake will be transactions stored by the bitcoin blockchain. You will also be enrolled in our monthly newsletter to help you stay informed about the project and major events.

The Next Step: Some Links

Welcome to the Ethereum Ecosystem:

- <http://ethereum.org/>
- <http://ethereum.org/ethereum.html>
- Enjoy the Code: <https://github.com/ethereum>
- Blog Post: <http://blog.ethereum.org/?p=53/ethereum-now-going-public>
- Coindesk Article: www.coindesk.com/ethererum-launches-cryptocurrency-2-0-network/
- Bitcoin Magazine Article: <http://bitcoinmagazine.com/9671/ethereum-next-generation-cryptocurrency-decentralized-application-platform/>
- (Ethereum Project's Official Youtube Channel)
- (Upvote us on Reddit)

Contact Us:

General Queries
info@ethereum.org

Media Inquiries
charles@ethereum.org

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: OpenPGP.js v.1.20130820
Comment: <http://openpgpjs.org>

```
xsFNBFJugC0BEACv91Xph3RKSGfx4n0ESfGWBNonafhRE+jhwd4lnDM4XYmO
30syetB6QMx/35fPpLHMPa0NjhhMrU8KLz4VC1RZvtoDSmvocMFLxmcJD2W
/IT/1lXp9jv9L67ALovDUdn2QgRCSbDaXaeCutbFcLQxQ8UtrG+FBCSms70
qX9XfaWgD2OdfcDENS3GFpNtoK1YT403s7EG3PgjD/+PoqY9WMjIWoDoulc
WLZSkW82NgfET5T8RsA2pYuf/Eu2GMF24Y6vrD1OTtH7kXDU5pswEocBlItA
DtCbbMAUJhGnHfTYIHDxLbCcL2eHirGXcMgQ20y3dAEoDFAe34D5ERxFe7jy
I07xPL099FeWz4hX7IE4tJ5Yz8F90IKGnAFQpC5+s2zeXJcrFyughPPwSLKj
J8AIXTCib8TbpSt+55pBGP5IXFE+5Nd7nz51mJ28th/4Ce5cgp0MRHXoNxSu
puuQO08wNk4kcZien8pMFypQdR6mHKKAYASCxGsDwfsV1KODKSpabIXT7lla
RV2K7CILMwmRsi7JcF6mvq9nle+3SKNzBDmXaMmmwOIGko4V+dYWh24rmBO
LSBFnGA8AZzBRNpqAKYv6EbPpVgcwk2c58ECCx6EUSLInDc1us0kiYYTRin
SO2csYjtd7tqBl6NbNn5IOH8YHBzRts+N4d9QwARAQABZS9DaGfYbGVzIEhv
c2tpbnNvbIA8Y2hhcmxlcY5ob3NraW5zb25AZ21haWwUy29tPSLBXAQQAQgA
EAUCUm6B7QkQIEPjdxPhVYAAFn+D/4I8XkURnz5pRLw6yt/982B7D5vjgfw
1Qe7h372PfkQ0Tb0LJHmQ0Jywa/V1lHl2aKaHVAmDFI1pJIsDwnvxZj/ff
4tla69vkw9k7UVfZueEYR2D471b2VOKgXyoJOJUqgGYymQ+ogA+ZcM5VLyac
FsbYCWdeQGJZEvkFTzw5USeFJPAPrLst7z5c66aWs3Pph8dHXPk9HmAufclG
NR5/A/UK9aFIPHB8xf74JZeypCLwVg01LA8xvEOgCOayt5KzJO+oUzinMDFS
EXftT6h6yzQevNYIIVmOpeUg0ZiAgHC9UsvVP9cjKp+DLIO1yQ9/EH85CMuZ
14Lg3IB6t09nB9M9VYoAZVp9ooRvWP4wX/G0FENvab+Y/R64Gq8Za2BXKYwx
U1rDgvA2ITat6MU1MbDe+Vzc25FBziz4FZHnWpfJHZUuN2rGQDeMTpah1Zzp
Cg6e80cLi2jP9zU08tBPs98JL8cw6ugvSYsYcQKqTPykRJ6ZG4t1OQKJj0+p
BHN1hi7BewxYkvGaSUqs/fa4YusYbdDyLPQUhnbDhRIRAX/byrQWeKRbFij
CGNjbMNMMLpmV2GUdks4fSapxu6SFNF6AVWP03NsaM4QxKzi7TX8vPWwgyQ4
y6J8KvTHXNF0oFsaQYjUygmjdlwFEXITRW+b12nPO1I3YvwXc4Bt1A==
=gQFQ
```

-----END PGP PUBLIC KEY BLOCK-----

Argumentum ad lunam: the fallacy that because Bitcoin's price is rising really fast the currency must be a speculative bubble and/or Ponzi scheme.

Play - Invest - Win
Trading Fee 0.1%

— PLAYTIN —
Casino - Exchange - Wallet - Trading

Instant Withdrawals
Provably Fair

Advertised sites are not endorsed by the Bitcoin Forum. They may be unsafe, untrustworthy, or illegal in your jurisdiction. [Advertise here.](#)

BitAddict
Sr. Member
●●●●●

Activity: 392



DoctorBitcoin.com



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 11:50:24 AM

#2

I like your project. I will be following it!

Good luck Vitalik!

[DoctorBitcoin.com // iCompra/Venta de Bitcoins!](#)
Selling Bitcoins! Neteller/Monebookers & SEPA
[My GPG public key.](#)

bitkoof
Jr. Member
●

Activity: 37



CoinDesker



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 12:07:48 PM

#3

Good luck - looks very interesting

CoinDesk is now live BTW - <http://www.coindesk.com/ethererum-launches-cryptocurrency-2-0-network/>

fizz919
Full Member
●●●●●

Activity: 155



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 12:33:41 PM

#4

Good luck, have fun!

Best Bitcoin supported browser game:
Minethings: Dig, Trade, and Fight your way to influence!

kikedad
Full Member
●●●●●

Activity: 140



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 12:56:19 PM

#5

Goodluck on the project. ill be investing on the first day of the IPO.

Seth Otterstad
Sr. Member
●●●●●

Activity: 309



Pura Vidal



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 01:20:44 PM

#6

Hi Vitalik, I'll see you at the Miami conference. If you could give your thoughts on these questions, it would be much appreciated. I am very interested in this project.

NECESSITY OF A NEW BLOCKCHAIN

The whitepaper states that colored coins inherit the same limitations as the bitcoin protocol. Metacoins do not, but they cannot be truly secure without having a copy of the blockchain. How big of a problem is this? Hasn't the Electrum model worked for Bitcoin? Is there a big issue with just querying a bunch of supernodes to see if they all give the same result? Is it necessary to have absolute security with every contract, or only very high value transactions? Is it correct that metacoin clients won't need the first 14gb of the blockchain?

BITCOIN BLOCKCHAIN SIZE

Bitcoin Magazine's analysis showed that since storage capacity and internet bandwidth follow Moore's law, the blockchain size will not be a problem to store on phones. How many years from now should we predict that the blockchain size will become irrelevant? If the blockchain size were irrelevant, would ethereum still be necessary? How much demand is there for blockchain contracts at this time, and how fast should we expect demand to accelerate?

ETHEREUM BLOCKCHAIN

How fast should we estimate the ethereum blockchain size will grow? What will be the effects of ethereum blockchain size growth several orders of magnitude greater than Bitcoin? Will this lead to more mining centralisation?

TURING COMPLETENESS

Is it reasonable to think the client can be sandboxed? What will be the effects of these, since the protocol is Turing complete: viruses, malware, adware, keyloggers, scam contracts, wallet stealers

ILLEGAL CONTRACTS

What will be the effects of illegal contracts? An assassination market, for example? Will the Silk Road be able to run on ethereum? Will a database of child porn be able to be stored? If there is child porn stored on the blockchain that is easily accessible, will it be legal to store the blockchain in any country? What will be the effects of running all the clients over Tor by default? Can a 60s block time with huge blocks work over Tor?

SATOSHI'S ARGUMENTS

In 2010, satoshi successfully argued against including Namecoin into bitcoin, suggesting merge-mining instead.

"Piling every proof-of-work quorum system in the world into one dataset doesn't scale.

Bitcoin and BitDNS can be used separately. Users shouldn't have to download all of both to use one or the other. BitDNS users may not want to download everything the next several unrelated networks decide to pile in either.

The networks need to have separate fates. BitDNS users might be completely liberal about adding any large data features since relatively few domain registrars are needed, while Bitcoin users might get increasingly tyrannical about limiting the size of the chain so it's easy for lots of users and small devices." -- Satoshi

You can view the rest of his arguments in the thread here:

<https://bitcointalk.org/index.php?topic=1790.0;all>

DEVELOPER COMMENTS

I have not seen any core bitcoin developers comment on ethereum. Is there somewhere besides the reddit and the thread on bitcointalk about it?

I am not very technical, so sorry if some of my questions are invalid or stated incorrectly. I also posted this on reddit.com/r/ethereum.

Seth Otterstad's Blog @SethOtterstad on twitter Seth on google+
Play poker at SealsWithClubs, the largest bitcoin poker room. Bet sports almost vig-free at BTCsportsmatch
Wires to BitStamp.net from USA are credited in less than one day. .5% currency conversion rates for non-USD wires

ripper234
Hero Member
●●●●●

Activity: 1092



Ron Gross



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 01:31:04 PM

#7

Good luck on the upcoming IPO and more importantly on executing!

My preliminary thoughts are summarized on my blog.

Please do not pm me, use ron@mastercoin.org instead
Mastercoin Executive Director
Co-founder of the Israeli Bitcoin Association

Akka
Hero Member

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 01:34:14 PM

#8

new member
●●●●●

Activity: 644



Ignore

January 23, 2014, 01:34:14 PM
Maybe a nice Idea, but again It's an investment scheme in it due to the limited supply of Ethers.

That are all nice functions, but I don't see, why in order to use them one has to buy into a highly speculative currency. In fact smart property, smart contracts, decentralized exchange, etc. Would all work better if issuers and users of these wouldn't need to buy into a currency that is intended to get more and more expensive in order to use it.

I see really nice developments in crypto and it's cool that so many obviously smart people get involved in it.

But it's utterly disappointing, that everything build is at its core build only to make the early adopters rich. (Same thing with: Mastercoin, ProtoShares, NXT, etc)

What all this people could accomplish if they would help develop Bitcoin? 😞

1Akka1qjBz4XMCSS5yKJFii1UMGbFbMK1q

mamucik666

Jr. Member



Activity: 42



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 01:48:42 PM #9

In my opinion, the idea of giving most coins to the investors, instead of miners, is just plain stupid. This is not the proper way to distribute a currency. It is not fair.

kelpy

Sr. Member



Activity: 370

Archduke of Criptoaerica,
vassal of WallStreetCafe



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 02:24:16 PM #10

Count me IN! Good luck!

Deposita EUR/USD BitStamp / TheRockTrading **LocalRipple**

LocalRipple <https://bitcointalk.org/index.php?topic=327742.0>

Uniqueorn

Member



Activity: 98



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 02:30:50 PM #11

Quote from: **mamucik666** on January 23, 2014, 01:48:42 PM

In my opinion, the idea of giving most coins to the investors, instead of miners, is just plain stupid. This is not the proper way to distribute a currency. It is not fair.

Yeah... because rich people buying high end mining gear and mining everything is sooo much more fair 😞

Vitalik Buterin

Sr. Member



Activity: 322



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 03:08:31 PM #12

> Hasn't the Electrum model worked for Bitcoin?

Yes, it has. But I describe in my whitepaper why it doesn't work for protocols on top of Bitcoin.

> Is there a big issue with just querying a bunch of supernodes to see if they all give the same result?

That is potentially very problematic because it breaks down the no-trust property; instead, you're getting data from a small pool of centralized entities. It's possible, and people do it with blockchain.info, but it somewhat weakens the whole attraction of a cryptocurrency.

> How many years from now should we predict that the blockchain size will become irrelevant?

That's an interesting question that I don't know the answer to. Perhaps if that happens the blockchain size will simply keep on growing, as people keep putting more and more memory-intensive applications on top and reach an equilibrium. In the medium term, I personally am more worried about the possibility that Ethereum's disk and computational load will be too high; I think that is the more pressing question that we at the Ethereum team are more focused on.

> What will be the effects of ethereum blockchain size growth several orders of magnitude greater than Bitcoin?

I don't think the blockchain will be that large. There are ways to combine Ethereum with off-chain protocols to mitigate most of the issues if people do want to put terabytes of data into Ethereum-based systems.

> Is it reasonable to think the client can be sandboxed?

The security model for Ethereum is similar to that of web browsers with Javascript; we'll design our sandbox accordingly.

> What will be the effects of illegal contracts? An assassination market, for example? Will the Silk Road be able to run on ethereum?

Theoretically those use cases are possible, but it is important to point out that you can crowdfund assassinations and create a decentralized Silk Road marketplace with Bitcoin too, even directly on the blockchain using assurance contracts. So the regulatory challenges there are fairly similar to BTC, although we do realize that the legal status of second-generation cryptotolders is less clear; that is why we are interested in working with CODA.

> Will a database of child porn be able to be stored? If there is child porn stored on the blockchain that is easily accessible, will it be legal to store the blockchain in any country?

Just like Bitcoin, the memory fees are prohibitively high to store a JPEG.

> What will be the effects of running all the clients over Tor by default? Can a 60s block time with huge blocks work over Tor?

Maybe. Wait for the testnet and try it yourself.

> [Satoshi's arguments]

The nice thing about having a Turing-complete scripting language is that you can integrate many interesting models for how one currency can be integrated into another:

- (1) The sub-currency can be fully stored as a contract (space heavy, time heavy)
- (2) The Merkle root of the sub-currency state tree can be stored in a contract, and the leaves off-chain. Each update to the tree is validated by the contract (space light, time heavy)
- (3) Two blockchains, but where the Ethereum chain includes an SPV client of the other chain (space light, time light)

If one way of doing things is too inefficient, we can always fall back on other ways.

Quote from: Seth Otterstad on January 23, 2014, 01:20:44 PM

Hi Vitalik, I'll see you at the Miami conference. If you could give your thoughts on these questions, it would be much appreciated. I am very interested in this project.

NECESSITY OF A NEW BLOCKCHAIN

The whitepaper states that colored coins inherit the same limitations as the bitcoin protocol. Metacoins do not, but they cannot be truly secure without having a copy of the blockchain. How big of a problem is this? Hasn't the Electrum model worked for Bitcoin? Is there a big issue with just querying a bunch of supernodes to see if they all give the same result? Is it necessary to have absolute security with every contract, or only very high value transactions? Is it correct that metacoin clients won't need the first 14gb of the blockchain?

BITCOIN BLOCKCHAIN SIZE

Bitcoin Magazine's analysis showed that since storage capacity and internet bandwidth follow Moore's law, the blockchain size will not be a problem to store on phones. How many years from now should we predict that the blockchain size will become irrelevant? If the blockchain size were irrelevant, would that remove the need for ethereum? How much demand is there for blockchain contracts at this time, and how fast should we expect demand to accelerate?

ETHEREUM BLOCKCHAIN

How fast should we estimate the ethereum blockchain size will grow? What will be the effects of ethereum blockchain size growth several orders of magnitude greater than Bitcoin? Will this lead to more mining centralisation?

TURING COMPLETENESS

Is it reasonable to think the client can be sandboxed? What will be the effects of these, since the protocol is Turing complete: viruses, malware, adware, keyloggers, scam contracts, wallet stealers

ILLEGAL CONTRACTS

What will be the effects of illegal contracts? An assassination market, for example? Will the Silk Road be able to run on ethereum? Will a database of child porn be able to be stored? If there is child porn stored on the blockchain that is easily accessible, will it be legal to store the blockchain in any country? What will be the effects of running all the clients over Tor by default? Can a 60s block time with huge blocks work over Tor?

SATOSHI'S ARGUMENTS

In 2010, satoshi successfully argued against including Namecoin into bitcoin, suggesting merge-mining instead.

"Piling every proof-of-work quorum system in the world into one dataset doesn't scale.

Bitcoin and BitDNS can be used separately. Users shouldn't have to download all of both to use one or the other. BitDNS users may not want to download everything the next several unrelated networks decide to pile in either.

The networks need to have separate fates. BitDNS users might be completely liberal about adding any large data features since relatively few domain registrars are needed, while Bitcoin users might get increasingly tyrannical about limiting the size of the chain so it's easy for lots of users and small devices." -- Satoshi

You can view the rest of his arguments in the thread here:

<https://bitcointalk.org/index.php?topic=1790.0;all>

DEVELOPER COMMENTS

I have not seen any core bitcoin developers comment on ethereum. Is there somewhere besides the reddit and the thread on bitcointalk about it?

I am not very technical, so sorry if some of my questions are invalid or stated incorrectly. I also posted this on reddit.com/r/ethereum since it seems like Vitalik responds there more often.

Argumentum ad lunam: the fallacy that because Bitcoin's price is rising really fast the currency must be a speculative bubble and/or Ponzi scheme.

AtlasOno
Sr. Member
●●●●●

Activity: 433



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning

January 23, 2014, 03:23:10 PM

#13

cool

coins101
Full Member
●●●●●
Online

Activity: 140



Ignore

Re: [ANN] Ethereum: Welcome to the Beginning

January 23, 2014, 03:26:00 PM

#14

Quote from: bitkoof on January 23, 2014, 12:07:48 PM

Good luck - looks very interesting

CoinDesk is now live BTW - <http://www.coindesk.com/ethererum-launches-cryptocurrency-2-0-network/>

@CoinDesk says part of the aim is to combat ASIC?

That's what Litecoin tried to do. Once the value went up, vultures moved in to develop ASIC-Script hardware, forcing people to accept them and start pre-ordering.

1. Why won't that happen again with ethererum? Are you offering a centralised or hosted mining?
2. Will there be any need and would you change any code if someone figures out how to develop ASIC? - something the Litecoin developers seem to want to avoid despite the community showing a two-thirds Informal vote for a coding change to avoid ASIC.

ioi
Newbie
#15
Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 03:35:22 PM

Great, looking forward for the fundraising

Activity: 7



Ignore

Terpie
Full Member
#16
Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 03:59:54 PM

Great, another illegal securities offering. How hard is it to hire a lawyer for one hour? I know I'll be investing heavily knowing Sec. 12(a)(1) of the Securities & Exchange Act of 1933 essentially gives me a put right should the value of Ethereum go down whatsoever. And don't think being located outside the US is going to help you.

Activity: 175



Ignore

Seth Otterstad
Sr. Member
#17
Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 04:07:06 PM

Thanks for the excellent response, Vitalik! Since you're a lot smarter than me, I'll just take your word for it that jpeg files will be too expensive to store on ethereum, unless a bitcoin dev comes in here and disputes it. So that takes care of the child porn concern. Could a script could still act as a thumbnail browser and download the images from a Tor server if a cryptocurrency payment was made to an address?

Activity: 309



Ignore

I am also not quite sold on the need for ethereum. It seems to me that the blockchain.info trust model or the "query a bunch of servers" model for low value transactions works just fine for bitcoin. Are there any instances where this has been abused yet? For high value transactions, getting a blockchain copy isn't that big a deal, and will continue to be less and less of a problem as Moore's Law for storage and bandwidth continues to outpace blockchain growth (which is continuing to grow linearly despite massively increased usage).

Quote from: Vitalik Buterin on January 23, 2014, 03:08:31 PM

> What will be the effects of illegal contracts? An assassination market, for example? Will the Silk Road be able to run on ethereum?

Theoretically those use cases are possible, but it is important to point out that you can crowdfund assassinations and create a decentralized Silk Road marketplace with Bitcoin too, even directly on the blockchain using assurance contracts. So the regulatory challenges there are fairly similar to BTC, although we do realize that the legal status of second-generation cryptoledgers is less clear; that is why we are interested in working with CODA.

It seems to me that, just like bitcoin, the first major use case for ethereum will be to make the Silk Road more usable. Instead of having to download Tor Browser Bundle and find a weird random character string to type into a browser, people will just be able to download ethereum and run it to browse and conduct illegal commerce. It will vastly improve the usability of the Silk Road. Can we really expect to be able to run drug marketplaces in the clearnet just because it is decentralized? It seems to me that this will be illegal in every jurisdiction in the world, which is not the case with bitcoin. It seems likely to me that the ethereum website will be seized and people will be prosecuted.

Quote from: Vitalik Buterin on January 23, 2014, 03:08:31 PM

> Is it reasonable to think the client can be sandboxed?

The security model for Ethereum is similar to that of web browsers with Javascript; we'll design our sandbox accordingly.

I don't know a lot about web browser security, but this does not make me feel safe. Exploits are demonstrated often that can break out of the chrome browser sandbox to claim the bug bounty. There is going to be one hell of a reward for finding an exploit and breaking out of the ethereum sandbox.

Seth Otterstad's Blog @SethOtterstad on twitter Seth on google+
Play poker at SealsWithClubs, the largest bitcoin poker room. Bet sports almost vig-free at BTCsportsmatch
Wires to BitStamp.net from USA are credited in less than one day. .5% currency conversion rates for non-USD wires

Vitalik Buterin
Sr. Member
#18
Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 04:40:18 PM

> Could a script could still act as a thumbnail browser and download the images from a Tor server if a cryptocurrency payment was made to an address?

Activity: 322



Ignore

Scripts can't download anything, so no.

> I don't know a lot about web browser security, but this does not make me feel safe. Exploits are demonstrated often that can break out of the chrome browser sandbox to claim the bug bounty. There is going to be one hell of a reward for finding an exploit and breaking out of the ethereum sandbox.

I'll avoid talking more about the legal questions at this point; I am not a lawyer, and we are talking with people who are who are in the process of crafting our legal positions.

Most critical vulnerabilities that we hear of don't actually break out of the sandbox; they are actually of the sort where Javascript code from one website manages to get executed inside of another webpage, and sends user data off to some third party server. There is no equivalent of that here. As for the vulnerabilities that do break into system memory somehow, I think our main defenses will be simplicity and abstraction. We have less than 60 instructions, and there are no operations that affect memory directly; everything is done by making database calls and modifying 256 bit integer objects. Our language has no power, even in theory, to handle input, output, HTTP requests or file system accesses; it's all purely internally focused. If we decide that it's necessary, we also have stronger options like running the scripts with a zero-permission user or even inside a virtual machine.

Argumentum ad lunam: the fallacy that because Bitcoin's price is rising really fast the currency must be a speculative bubble and/or Ponzi scheme.

Seth Otterstad
Sr. Member
#19
Re: [ANN] Ethereum: Welcome to the Beginning
January 23, 2014, 05:09:49 PM

Thanks Vitalik, you made me feel a lot better about the security. Since I am not technical I was under the mistaken impression that Turing-complete meant the script would be able to do absolutely anything on a computer.

Activity: 309



Ignore

pastet89

Member



Online

Activity: 98

GPUcoin - unique business model!



Ignore

Will you make a legal opinion available to investors? I think that ethereum's main first usage will be things that are hard or illegal to do with existing systems (Silk Road), and that when Senator Schumer demands that something be done about it, the website will be seized. Bitcoin got around this problem by functioning only as a currency and externalizing the Silk Road to the Tor network. I don't see how ethereum can compete with bitcoin if it is driven underground.

For what it's worth, I really hope this succeeds. Self-regulating markets are far superior to what we currently have. I am just not convinced this has a reasonable chance of success yet.

Seth Otterstad's Blog @SethOtterstad on twitter Seth on google+
Play poker at [SealsWithClubs](#), the largest bitcoin poker room. Bet sports almost vig-free at [BTCsportsmatch](#)
Wires to [BitStamp.net](#) from USA are credited in less than one day. .5% currency conversion rates for non-USD wires



Re: [ANN] Ethereum: Welcome to the Beginning

January 23, 2014, 05:29:16 PM

#20

Good luck!

I am definately In. 😊

A few questions here:

1.

Quote from: [Vitalik Buterin](#) on January 23, 2014, 11:33:17 AM

Early investors for taking a great risk due to the uncertainty of investment will be rewarded by a bonus of 2*initial investment for the first week and then this bonus decays by 2 percent per day for the remainder of the fundraiser until it reaches a floor of the base exchange rate of 0.001 BTC per 1 Ether

Could you please put some more light on this statement, for example, if I invest 1 BTC within the first week, how much Ethers will I end win at the end of the fundraising period?

2. Where can I find a client for Linux and install it, if it is still not created, how can I get the ethers I am going to buy from the IPO?

Thanks and good luck again! 😊

✧ Buy cheap GPUs with GPU Coin ✧
☐ Win a video card! - Drawing/Contest ☐

Pages: [1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 ... 77

print

Bitcoin Forum > Other > Alternate cryptocurrencies > Announcements (Altcoins) > **[ANN] Ethereum: Welcome to the Beginning**

« previous topic next topic »

Jump to: ==> Announcements (Altcoins)

go

Sponsored by Private Internet Access, a Bitcoin-accepting VPN.

Powered by SMF 1.1.19 | SMF © 2006-2009, Simple Machines



Bitcoin Forum

simple machines forum

April 20, 2014, 10:09:13 AM

Welcome, **Guest**. Please login or register.

Login with username, password and session length

News: Due to the OpenSSL heartbleed bug, changing your forum password is recommended.

HOME HELP SEARCH DONATE LOGIN REGISTER

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **Proof of stake brainstorming**

< previous topic next topic >

Pages: [1]

print

Author

Topic: Proof of stake brainstorming (Read 2172 times)

cunicula
Hero Member
★★★★★

Activity: 700



Ignore

Proof of stake brainstorming
August 15, 2011, 05:12:39 AM

#1

First off, I don't care whether you think proof-of-stake is useful or not. I want this thread to be about implementations only. I won't respond to off-topic posts.

Here is an idea I would like **constructive** feedback on:

Suppose that each wallet has two addresses (****a and ****b). The addresses are publically known to be associated with a single private key. Users sign a block by including a txn in that block that moves X coins from address ****a to address ****b. In order for this signature to be valid, the X coins in address ****a must have at least (c/X) [rounded up] confirmations before they are sent, where c is some constant. This means that the frequency with which a wallet can sign transactions (and thus add to the blockchain) is proportional to the number of coins it holds. The wallet would receive some interest on coins transferred from ****a to ****b to reward participation. Coins would have to be transmitted back from ****b to ****a before they could be sent to other addresses or used to verify additional blocks.

To make sure that blocks have time to propagate across nodes, it may be necessary that blocks arrive 5 or more minutes apart. One way to do force this is through hashing. For example you could require that transfer of coins from ****a to ****b is accompanied by solution to a hashing problem. Difficulty could be set so that blocks arrive at appropriate intervals on average. I am curious about whether and why this time gap is necessary and if there are alternative means of enforcing it besides hashing (i.e. some sort of decentralized timestamp).

Cue for responses:

Is the system above a reasonable method of implementing proof-of-stake or is it misguided in some way? Is it essential to force a time gap between blocks? Are there other decentralized ways to force this time gap besides hashing?

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

mBitCASINO

**WIN BITCOINS IN OUR
24/7 LIVE DEALER CASINO**

Advertised sites are not endorsed by the Bitcoin Forum. They may be unsafe, untrustworthy, or illegal in your jurisdiction. Advertise here.

ByteCoin
Sr. Member
★★★★★

Activity: 413



Ignore

Re: Proof of stake brainstorming
August 15, 2011, 01:14:38 PM

#2

Reading somewhat between the lines, what you appear to be doing is providing a mechanism for tying up portions of a user's coins for a number of blocks, the reward for which is some additional coins. Presumably the "tying up" transaction would contain a hash of a number of third-party transactions which the author approves of and wishes to confirm.

You say it would take c/X confirmations before the funds are untied. Are all confirmations created equal? Let's say I have 50 units of your funds. I can confirm a transaction by tying up all 50 units for c/50 blocks and that provides one confirmation. I could alternatively split it into 50 lots of 1 unit and confirm a transaction 50 times by tying up 50 lots of 1 unit for c/1 blocks.

Minor issues that need clarification are: How do you have two addresses for one private key? Do all transfers tie the funds up for c/X confirmations?

A fundamental problem with the scheme is that people with a lot of funds who never spend them get more funds fastest. I don't think that this is a good way to bootstrap a currency.

It would be worth you editing your post to include a link to an explanation of "proof of stake" and a bit more about your goals by contrast with the existing system.

Interesting idea though...

ByteCoin

Meni Rosenfeld
Donator
Hero Member
★★★★★
Online

Activity: 1148



Re: Proof of stake brainstorming
August 15, 2011, 03:15:48 PM

#3

You need to clarify exactly what is the problem you are trying to solve before analyzing solutions. Heck, I understand the problem and the solution approach and could barely follow you.

I'll assume the problem is this:

We need some rule to determine which branch of the block chain to consider valid. The current rule (using the branch representing the highest total difficulty) is vulnerable to attacks by an entity with high hashrate. Proof-of-stake systems seek to modify the decision rule so that attacking the network can only be done by those who have many bitcoins, and thus, have the least incentive to do so.



So, if this is the goal, we cannot assess how close your suggestion is to achieving it, because you haven't specified what would be the rule to choose between branches.



Ignore

For just the technical part of actually signing a block, my preferred approach is this: Every address is associated with two key pairs. The address itself is a hash of the concatenation of the two public keys. Doing normal bitcoin transactions involve revealing both public keys, as well as signing the transaction with private key A. Signing a block involves revealing both public keys and signing the block hash with private key B. The weight of the signature depends on the number of coins belonging to the address up to the signed block (or alternatively, X blocks back). Key B doesn't need to be as well-protected as key A - If it's compromised, the owner just sends the coins to a different address. So he can maintain voting rights with bitcoins safely stored in a time capsule.

I'll add my thoughts about how to incorporate it in a system later.

1EofoZNBhWQ3kxKkvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
 Bitcoll - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
 Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

cunicula
 Hero Member
 ●●●●●

Activity: 700



Ignore

Re: Proof of stake brainstorming

August 16, 2011, 03:25:33 AM

#4

Thanks for the great comments, Meni and gigabyte. Sorry about being discombobulated. It is because I don't have a full understanding yet. For now, I will keep the first post as is because I like how the discussion is going. Later (once other posters have sorted me out sufficiently), I will edit the intro.

What is important about the current system (and the proof-of-stake solution I am imagining) is that you don't need to aggregate votes from multiple nodes before signing individual blocks. Aggregation emerges from competition to lengthen the blockchain. The longer blockchain is the valid chain.

Here is an analogy which might help.

Currently, miners find a solution to minimization problem (1) with a value under the difficulty criteria. They submit this to sign block 1. After block 1, the minimization problem (1) changes to minimization problem (2). Subsequent signers may choose to confirm the validity of this block (or not). If they choose to confirm validity, they find a solution to the minimization problem (2), and use this to sign block 2. Alternately, they can reject validity. To do this, they find a solution to minimization problem (1) and use this to sign block (1). The new signature creates an alternate sequence of blocks (block 1 and block 2) vs. (block 1 and block 2a). The client treats the longest sequence of blocks as the valid sequence.

Consider an alternative to hashing:

Multiple miners find solutions to minimization problem (1). They find a range of minimum values (some very low-some high). They use these minimum values to vote on block 1. Votes associated with a low minimum value are assigned heavy weight. Votes with a high minimum value are signed low weight. If the value-weighted majority agrees that block 1 is valid, then block 1 is signed. Otherwise, block 1 is not signed.

What is wrong with this alternative? (and by analogy what might be wrong with Meni's proof-of-stake solution)

You need to collect information from many miners before you can add to the chain. Meni's proof-of-stake solution has this same feature (except you need to collect info from many stakeholders). In this thread, Quantum Mechanic proposes proof-of-stake: <https://bitcointalk.org/index.php?topic=27787.0>. Hashcoin (and later Mike Hearn) object that collection of information from many stakeholders before block signing poses severe bandwidth and storage problems. I am not aware of a counterargument. My point is that proof-of-stake can operate the same way as proof-of-work. A miners' frequency of block signing can be determined by stake rather than work. No voting is required. Solutions come from individual miners and the length of a chain of solutions indicates chain validity. Since miners need to wait for confirmations to add solutions, it is not practical for individuals to sign multiple blocks in a row.

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

cunicula
 Hero Member
 ●●●●●

Activity: 700



Ignore

Re: Proof of stake brainstorming

August 16, 2011, 03:39:10 AM

#5

Quote from: ByteCoin on August 15, 2011, 01:14:38 PM

You say it would take c/X confirmations before the funds are untied. Are all confirmations created equal? Let's say I have 50 units of your funds. I can confirm a transaction by tying up all 50 units for c/50 blocks and that provides one confirmation. I could alternatively split it into 50 lots of 1 unit and confirm a transaction 50 times by tying up 50 lots of 1 unit for c/1 blocks.

Yes, all confirmations are equal. Otherwise, you would have incentives to combine money into one wallet or separate money into different wallets. [There is still a minor issue like this related to rounding. Shouldn't be a big deal.]

Quote from: ByteCoin on August 15, 2011, 01:14:38 PM

Minor issues that need clarification are: How do you have two addresses for one private key? Do all transfers tie the funds up for c/X confirmations?

I not sure about the two addresses. Better to ask someone who knows about programming, private, public keys, etc., I'm winging it here. Meni has a suggestion. I have a suggestion below where you don't need two addresses. Instead you have the blockchain escrow coins.

Quote from: ByteCoin on August 15, 2011, 01:14:38 PM

Do all transfers tie the funds up for c/X confirmations?

No you can transfer funds freely, you only need to worry about c/X confirmations if you are trying to sign a block. One way of implementing this is through a central escrow address [shared by everyone], as follows:

(e.g. Signing block 1: Address A coins -> Escrow Address B ... In block 2: Escrow Address B coins + txn fees + any generated reward coins -> Address A

Without inclusion of this send, block 2 cannot be valid. In this way, you don't need special addresses for each user. Escrow Address B just holds coins in escrow for return in the next block. The only purpose here is to prove coin control and reset the number of confirmations on coins in address A. As long as you are not sending coins to escrow address B, you don't need to worry about confirmations.

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

Meni Rosenfeld

Donator
Hero Member



Online

Activity: 1148



Ignore

Re: Proof of stake brainstorming
August 17, 2011, 12:28:47 PM

#6

In the current system, it is not known what the next block will be before it is found. Every miner know a certain set of transactions and tries to find a block including them. When a block is found, which is a relatively rare event, it is broadcast and any remaining floating transactions will remain and can be included in the next block.

If you want several miners to verify a block before it is accepted, it means that they have to agree on the transactions to include before starting to verify it. This is a synchronization nightmare.

My own system is along the following lines.

Hash-based mining will continue more or less as usual. However, branch choice will depend not only on length, but on two additional factors: Signatures, to prevent double spending; and circulation, to prevent interfering with transaction confirmation.

Every 100th block (or 64th, or some other number) will be a signature block. It is mined normally, but once it is found and confirmed with 10 additional blocks, stakeholders (people who have coins as of the signature block) are invited to sign the block's hash with their private key (which can be separate from their transaction private key). Choosing between two branches will be very heavily influenced by the weight of the signatures on the last signature block.

An attacker with high hashrate who tries to build an alternative long branch will be rejected because it conflicts with a signed branch. Thus, someone who makes a large transfer can guarantee its security if he waits for a signature block.

To prevent the results from being skewed by coins whose owners are not interested in signing, the voting weight of an address will decrease for every block it doesn't sign. It will increase back to the normal weight for every block it signs. If coins are moved to a new address, the weight will reset to 0. Also, if an address signs two conflicting blocks, its weight will reset to 0.

If keeping the network secure, and thus maintaining the value of their stake, is not incentive enough, there will be a system to have a part of the transaction fees reserved for stakeholders, or maybe a separate signature fee (paid by those who wish to incentivize stakeholders to sign the next signature block).

This leaves the problem of an attacker trying to disrupt the network by reversing blocks and thus stopping transactions from being confirmed. For this, two more terms can be considered in evaluating a branch:

1. Circulation - how many different coins have circulated in the last 10 blocks. An attacker can try to displace all blocks, except for his own blocks which only include transactions sending money to himself (to make them appear like legitimate block). Unless he has lots of bitcoins, he will need to send the same coins back and forth. Branches where this is detected - that is, the total number of different coins circulated is low - will be penalized. This may be hard to compute, but some sort of useful approximation may be tractable.
2. Inclusivity - A block will be penalized for every transaction it excludes which has a high enough fee and has been floating for a while. A few transactions occasionally missing is ok, but when it becomes systematic it indicates someone trying to block a specific transaction.

1EfofZNBhWQ3kxKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
Bitcoil - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

cunicula

Hero Member



Activity: 700



Ignore

Re: Proof of stake brainstorming
August 17, 2011, 01:36:48 PM

#7

Meni, that is interesting.

Here are some questions about your idea:

Suppose that a malicious individual has access to major hashing power. Their intention is to stop transactions by flooding the network with invalid blocks.

Could they add invalid blocks to every fork they see? If so, it might take a very long time before a sequence of 100 (or 64) valid blocks in a row was created. Would this prevent any fork from ever reaching a new signature checkpoint? Essentially could an attacker get the network stuck between checkpoints in your system? If this is a problem why not require signatures for every single block? Is there a synchronization problem related to adding these signatures?

Here is a revised version of my idea:

Miners who submit blocks must also submit coins for escrow in the blockchain. The escrowed coins must satisfy a 'coin-confirmation constraint'. By 'coin-confirmation constraint', I mean the product of the number of coins sent and the number of confirmations on those coins must be greater than or equal to some number. The blockchain escrows the sent coins and then returns them to the sender when the next block is found. There should not be a synchronization problem here because a single miner both finds the block and includes his coins in that block.

The 'coin-confirmation' threshold would be set relatively low so that say only 10% of extant coins would have to be actively used in mining to keep the system churning out blocks. If miners began to run short of coins, they would want to purchase them to prevent any blocks they find from going to waste. In fact, they would want to keep an oversupply of coins. If they happened to get lucky hashing, they would need to have these extra coins ready for inclusion in blocks. Miners' demand for an excess coin hoard would prevent the system from ever getting stuck.

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

Meni Rosenfeld

Donator
Hero Member



Online

Activity: 1148

Re: Proof of stake brainstorming
August 17, 2011, 02:36:33 PM

#8

Quote from: cunicula on August 17, 2011, 01:36:48 PM

Suppose that a malicious individual has access to major hashing power. Their intention is to stop transactions by flooding the network with invalid blocks.

Could they add invalid blocks to every fork they see? If so, it might take a very long time before a sequence of 100 (or 64) valid blocks in a row was created. Would this prevent any fork from ever reaching a new signature checkpoint? Essentially could an attacker get the network stuck between checkpoints in your system? If this is a problem why not require signatures for every single block? Is there a synchronization problem related to adding these signatures?



Ignore

I'm not sure you are fully aware what a >50% attacker can or cannot do.
He cannot add blocks which don't comply with the protocol, these will simply be rejected.
He cannot add transactions which send coins from addresses he hasn't the key for.
He can add blocks that exclude any or all floating transactions. By putting such blocks in a long enough branch, he can reverse blocks which do include these transactions.

So, whatever he does, eventually 100 blocks will be reached. It doesn't matter if some of these are useless blocks created by the attacker.

A possible problem is that he will cause the difficulty to increase and thus transactions will take more time to enter a block (since some of the blocks are the attacker's and don't include useful transactions), but that is a minor problem.

Having signatures for every block requires a lot of data to be passed around, and doesn't help much.

Quote from: cunicula on August 17, 2011, 01:36:48 PM

Here is a revised version of my idea:

Miners who submit blocks must also submit coins for escrow in the blockchain. The escrowed coins must satisfy a 'coin-confirmation constraint'. By 'coin-confirmation constraint', I mean the product of the number of coins sent and the number of confirmations on those coins must be greater than or equal to some number. The blockchain escrows the sent coins and then returns them to the sender when the next block is found. There should not be a synchronization problem here because a single miner both finds the block and includes his coins in that block.

The 'coin-confirmation' threshold would be set relatively low so that say only 10% of extant coins would have to be actively used in mining to keep the system churning out blocks. If miners began to run short of coins, they would want to purchase them to prevent any blocks they find from going to waste. In fact, they would want to keep an oversupply of coins. If they happened to get lucky hashing, they would need to have these extra coins ready for inclusion in blocks. Miners' demand for an excess coin hoard would prevent the system from ever getting stuck.

This is interesting, but 10% of all coins owned by miners is way too high - especially since we are seeking to keep the network secure while reducing the overhead of mining. And with less than that, I don't think the disincentive to attackers is high enough.

1EofZNBhWQ3kxfKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
Bitcoil - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

cunicula
Hero Member
★★★★★

Activity: 700



Ignore

Re: Proof of stake brainstorming
August 17, 2011, 04:53:34 PM

#9

Okay, thanks that's helpful. How do stakeholders decide whether to sign the block with their private key or not? Can an attacker include double spends in every fork, so that there are no valid chains to sign?

Regarding my suggestion, I don't see any problem with 10% or 20% of coins being locked up in mining. These are just coins that won't circulate freely in the money supply. Since coins don't cost anything to create, there is no additional overhead (i.e., use of non blockchain resources) involved. In fact it should be resource saving because it would reduce demand for electricity and computing hardware (mining-related demand for these items would be partially diverted to bitcoin purchases). It also improves the security of accepting coins based on a small number of confirmations. The scarce resources we need to worry about are storage and bandwidth if anything. The approach I'm suggesting is very economical as far as these resources are concerned.

I'm just seeing a long list of benefits with no obvious downside. Could you explain more clearly what you think the problem is? In particular, can you define what you mean by overhead? And why it is useful to minimize your concept of overhead as opposed to some other one?

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

Meni Rosenfeld
Donator
Hero Member
★★★★★
 Online

Activity: 1148



Ignore

Re: Proof of stake brainstorming
August 28, 2011, 05:49:54 PM

#10

Quote from: cunicula on August 17, 2011, 04:53:34 PM

Okay, thanks that's helpful. How do stakeholders decide whether to sign the block with their private key or not? Can an attacker include double spends in every fork, so that there are no valid chains to sign?

Double-spending means:

1. Buyer (attacker) sends money to seller.
2. Seller gives goods to buyer.
3. Buyer creates a new branch in which the coins are sent back to himself.

The proof-of-stake is only relevant for people willing to wait for it. If stakeholders have several candidate branches, each should choose one randomly or according to some rule. One of the branches will come out ahead as the most signed. For every transaction that was agreed upon, there are two possibilities - either it is included in the signed branch, in which case it is secure forever and the seller can send the goods; or it was replaced by another transaction, in which case the seller doesn't send the goods. So, if you wait for proof-of-stake, you'll never have a payment reversed after you've sent the goods.

Quote from: cunicula on August 17, 2011, 04:53:34 PM

Regarding my suggestion, I don't see any problem with 10% or 20% of coins being locked up in mining.

If I understand your suggestion correctly, it requires that 10% of the coins are *owned* by the miners. This means that miners possess 10% of the wealth of the world (at least, the part of it that is in the form of bitcoins). This means mining is a huge enterprise, instead of a lean service provider. What if I told you that for the internet to work, 10% of the global wealth needs to be at the hands of ISPs?

1EofZNBhWQ3kxfKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
Bitcoil - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

cunicula
Hero Member
★★★★★

Activity: 700



Re: Proof of stake brainstorming
August 29, 2011, 02:57:51 AM

#11

If there is no mechanism for stakeholders to choose between chains, then why include stakeholders at all? Why not just have the miner who finds every 100th block flag it as a checkpoint (and prohibit reorgs past this point)?

As regards my suggestion, your concerns seem to be based on a misunderstanding of opportunity cost. The relevant question is the opportunity cost associated with mining via stakeholding, not the value of the bitcoin used to mine. The majority of bitcoins are just sitting in wallets with lots of confirmations. Using this money to mine has minimal opportunity cost since it is being held as a store of value anyways. I think that people holding bitcoin could be persuaded to mine even if the rate of return on their assets was extremely small (say 1% per annum). Remember that they can 'withdraw' their money from mining and spend it with minimal delay. If people are willing to participate even with a very low rate of return, then the opportunity cost associated with participation is necessarily extremely small. By



Ignore

contrast, the opportunity cost of mining via hashing is quite high because GPUs and electricity are not held as stores of value. Consequently, for computing power-based mining, the value of resources used in the mining process is the opportunity cost. What I am proposing involves a tremendous reduction in mining overhead (properly measured as opportunity cost) not an increase as you suggest.

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

Meni Rosenfeld

Donator
Hero Member



Online

Activity: 1148



Ignore



Re: Proof of stake brainstorming

August 29, 2011, 06:50:42 AM

#12

Quote from: cunicula on August 29, 2011, 02:57:51 AM

If there is no mechanism for stakeholders to choose between chains, then why include stakeholders at all? Why not just have the miner who finds every 100th block flag it as a checkpoint (and prohibit reorgs past this point)?

Because there's no "the" miner. There are several miners, some of whom may be attackers, each of whom may find his own version of, say, block number 143100. Each of these blocks will have its finder flag that "My block is the real one!". How would a node know which branch to use?

If you suggest that a block will be automatically flagged if the branch built upon it is sufficiently ahead of all other branches, you leave the possibility that a node will be introduced first to a wrong version and then refuse to conform to the right version when he finds it.

Blocks aren't good or evil, all you have is synchronization, everyone should agree on the same block. With stakeholder signatures you have that - once a block is signed, everyone can verify this and agree on the block. The only people who have some limited ability to mess this up are stakeholders, who have the least incentive to do so.

Quote from: cunicula on August 29, 2011, 02:57:51 AM

As regards my suggestion, your concerns seem to be based on a misunderstanding of opportunity cost. The relevant question is the opportunity cost associated with mining via stakeholding, not the value of the bitcoin used to mine. The majority of bitcoins are just sitting in wallets with lots of confirmations. Using this money to mine has minimal opportunity cost since it is being held as a store of value anyways. I think that people holding bitcoin could be persuaded to mine even if the rate of return on their assets was extremely small (say 1% per annum). Remember that they can 'withdraw' their money from mining and spend it with minimal delay. If people are willing to participate even with a very low rate of return, then the opportunity cost associated with participation is necessarily extremely small. By contrast, the opportunity cost of mining via hashing is quite high because GPUs and electricity are not held as stores of value. Consequently, for computing power-based mining, the value of resources used in the mining process is the opportunity cost. What I am proposing involves a tremendous reduction in mining overhead (properly measured as opportunity cost) not an increase as you suggest.

More likely, my concerns were based on a misunderstanding of your suggestion. Upon further reflection, it looks like it is not too dissimilar to my own, yet I believe mine is superior. You'll need hashing anyway because it's intractable to collect proof-of-stake for every single block. And there doesn't seem to be an advantage in holding coins in escrow (which is additional overhead) over collecting signatures, with voting weight which depends on the history of the coins.

1EofoZNBhWQ3kxfKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc: WoT
Bitcoin - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

cunicula

Hero Member



Activity: 700



Ignore



Re: Proof of stake brainstorming

August 29, 2011, 07:41:22 AM

#13

I agree that you need to include hashing to prevent some kind of information overload, but that isn't very relevant. You could have a system where 10 terahash is sufficient or one in which 50 gigahash is sufficient. If the systems provide equivalent security then the 50 gigahash system is obviously preferable. There should be very little incentive to amass lots of hashing power in an efficiently designed stakeholding system.

I see what you mean by our schemes being similar. I just feel like mine is more robust. I'm not clear on how people are rewarded for choosing certain checkpoints in your system, how the system handles mistakes (reversing a block with a majority of signatures), or whether the system is vulnerable to disruptive attacks between checkpoints. I think a system which handles all blocks similarly and does not use arbitrary checkpoints is preferable.

Anyways, no one pays attention to these issues so it is a moot point for the time being. Thanks for providing some great feedback.

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

maaku

Hero Member



Activity: 770



Ignore



Re: Proof of stake brainstorming

March 11, 2012, 09:51:25 AM

#14

Mini, instead of having an address represent a pair of keys, what about using the key associated with existing bitcoin addresses to sign a delegation certificate, which basically says "the owner of bitcoin address A approves of any blocks signed with key B". Delegation certificates could even be revoked, obviating the need to transfer money around if the block-verifying key is exposed.

It's a more general solution, doesn't require changing the bitcoin address format, and could be implemented without touching the block chain on an overlay network used by miners.

I'm an independent developer working on bitcoin-core, making my living off community donations.
If you like my work, please consider donating yourself: **13snZ4ZyCzaL7358SmgvHGC9AxsqumNxP**

Meni Rosenfeld

Donator
Hero Member



Online

Activity: 1148



Ignore



Re: Proof of stake brainstorming

March 11, 2012, 10:07:45 AM

#15

Quote from: maaku on March 11, 2012, 09:51:25 AM

Mini, instead of having an address represent a pair of keys, what about using the key associated with existing bitcoin addresses to sign a delegation certificate, which basically says "the owner of bitcoin address A approves of any blocks signed with key B". Delegation certificates could even be revoked, obviating the need to transfer money around if the block-verifying key is exposed.

It's a more general solution, doesn't require changing the bitcoin address format, and could be implemented without touching the block chain on an overlay network used by miners.

Sounds good.

1EofoZNBhWQ3kxfKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc: WoT
Bitcoin - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

cunicula
Hero Member
●●●●●

Activity: 700



Ignore

Re: Proof of stake brainstorming
March 11, 2012, 12:56:46 PM

#16

Quote from: maaku on March 11, 2012, 09:51:25 AM

Mini, instead of having an address represent a pair of keys, what about using the key associated with existing bitcoin addresses to sign a delegation certificate, which basically says "the owner of bitcoin address A approves of any blocks signed with key B". Delegation certificates could even be revoked, obviating the need to transfer money around if the block-verifying key is exposed.

It's a more general solution, doesn't require changing the bitcoin address format, and could be implemented without touching the block chain on an overlay network used by miners.

If the key B can be transferred independently of key A, then this would separate ownership rights from voting rights. Voting rights would be worth very little to individuals because of the tragedy of the commons problem (you just rely on other people not to sell off their voting rights to would-be monopolists/attackers). A free market for pure voting rights is highly undesirable. It is important to ensure that the voter and the coin-owner are the same person.

In a Society in which there is no law, and in theory no compulsion, the only arbiter of behaviour is public opinion. But public opinion, because of the tremendous urge to conformity in gregarious animals, is less tolerant than any system of law. When human beings are governed by "thou shalt not", the individual can practise a certain amount of eccentricity: when they are supposedly governed by "love" or "reason", he is under continuous pressure to make him behave and think in exactly

Pages: [1]

[print](#)

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **Proof of stake brainstorming**

[« previous topic](#) [next topic »](#)

Jump to:

Sponsored by Private Internet Access, a Bitcoin-accepting VPN.

Powered by SMF 1.1.19 | SMF © 2006-2009, Simple Machines



Bitcoin Forum

simple machines forum

April 19, 2014, 10:45:55 PM

Welcome, **Guest**. Please login or register.

Forever
Login with username, password and session length

News: Due to the OpenSSL heartbleed bug, changing your forum password is recommended.



Search

[HOME](#) [HELP](#) [SEARCH](#) [DONATE](#) [LOGIN](#) [REGISTER](#)

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **Proof of stake instead of proof of work**

« previous topic next topic »

Pages: [1] 2 All

print

Author Topic: Proof of stake instead of proof of work (Read 5104 times)

QuantumMechanic

Member

●●●

Activity: 110



Ignore

Proof of stake instead of proof of work

July 11, 2011, 04:12:45 AM

#1

I've got an Idea, and I'm wondering if it's been discussed/ripped apart here yet:

I'm wondering if as bitcoins become more widely distributed, whether a transition from a proof of work based system to a proof of stake one might happen. What I mean by proof of stake is that instead of your "vote" on the accepted transaction history being weighted by the share of computing resources you bring to the network, it's weighted by the number of bitcoins you can prove you own, using your private keys.

For those that don't want to be actively verifying transactions, and so that not all private keys need to be facing the network, votes could be delegated to other addresses via some kind of nonstandard Bitcoin transaction. In this way, voting power would accumulate with trusted delegates instead of miners. New bitcoins and transaction fees could be randomly and periodically distributed to delegates, weighted by the number of votes they've accumulated, thereby incentivising diversity of the delegates and direct voters.

If the implementation could be done, it proved to maintain at least a similar level of privacy and trustworthiness, and it only minimally complicated the UX, I'm thinking that a proof of stake based fork could out-compete a proof of work one due to much lower transaction fees, since its network wouldn't need to support the cost of the miners' computing resources. (Note that the vote delegation scheme has bandwidth/storage overhead that would offset these savings by some amount which would hopefully be relatively small.)

Some other potential improvements this system could offer:

- Possibly quicker, more definite confirmation of transactions, depending on how it can be implemented.
- The "voting power" may be more trustworthy, since it would accumulate in a bottom-up fashion via a network of trust, instead of in the somewhat arbitrary way it accumulates now. (Note the potential problem of vote-buying here.)
- It would remove the physical point of failure of bitcoin mining equipment, which can be confiscated or made illegal to run.
- It could be used to provide stakeholders a means of making their voices heard (via the delegated voting system it establishes) when it comes to proposals for software updates and protocol changes.

Anyway, I just wanted to throw the idea out here to see if there are any obvious reasons why it couldn't be implemented, and to hopefully spark a discussion amongst those better qualified than me.

Cheers.

Black Arrow, Bitmain and More Available Now!

Minersource.net

Advertised sites are not endorsed by the Bitcoin Forum. They may be unsafe, untrustworthy, or illegal in your jurisdiction. [Advertise here.](#)

casascius

Mike Caldwell

VIP

Hero Member

●●●●●

Activity: 1204



The Casascius 1oz 10BTC
Silver Round (w/ Gold B)



Ignore

Re: Proof of stake instead of proof of work

July 11, 2011, 04:15:30 AM

#2

The idea has a huge amount of merit, not necessarily for Bitcoins.

Suppose I start a company and decide to issue its shares as a block chain. Instead of miners getting 50 shares each, the block chain would be programmed where I just issue all the shares up front to the proper shareholders, etc. The block chain is used as a means to buy and sell the shares on the peer-to-peer stock market (which doesn't exist yet). Miners maybe get transaction fees.

In such a case, proof-of-stake voting might be fantastic.

But I don't see what relevance it would have to Bitcoins as they are now known.

Companies claiming they got hacked and lost your coins sounds like fraud so perfect it could be called fashionable. I never believe them. If I ever experience the misfortune of a real intrusion, I declare I have been honest about the way I have managed the keys in Casascius Coins. I maintain no ability to recover or reproduce the keys, not even under limitless duress or total intrusion. Remember that trusting strangers with your coins without any recourse is, as a matter of principle, not a best practice. Don't keep coins online. Use paper wallets instead.

eastendtech

Newbie

●

Activity: 23



Re: Proof of stake instead of proof of work

July 11, 2011, 04:37:14 AM

#3

It would save a lot of computing power, but it doesn't solve the core problems of defeating double spending and attracting more compute power to the network to check against that spending. Even if there were random elements, I think peer-to-peer voting (even if done by the software itself) would be too easy to game and unduly favor dishonesty and cheating. It would make trust = wealth, which is a fallacy because what if all those private keys contained stolen bitcoins, etc?

Ignore

Feel Generous?

12RhX9NSTpBB1sVvbrscKG35kL5kVJkaXz

QuantumMechanic
Member

Activity: 110

Ignore

Re: Proof of stake instead of proof of work
July 11, 2011, 04:40:13 AM

#4

Quote from: **casascius** on July 11, 2011, 04:15:30 AM

The idea has a huge amount of merit, not necessarily for Bitcoins.

Suppose I start a company and decide to issue its shares as a block chain. Instead of miners getting 50 shares each, the block chain would be programmed where I just issue all the shares up front to the proper shareholders, etc. The block chain is used as a means to buy and sell the shares on the peer-to-peer stock market (which doesn't exist yet). Miners maybe get transaction fees.

In such a case, proof-of-stake voting might be fantastic.

But I don't see what relevance it would have to Bitcoins as they are now known.

Company shares sounds like a great application, too. The proof-of-stake alternative seems like it could be a good way to prevent unwarranted outside influence from messing with smaller scale public ledgers like they can with proof-of-work.

This unwarranted outside influence also seems to be why alternative Bitcoin implementations don't seem feasible now - you need enough miners to be on board to secure the network against outside attackers, whereas attackers can only come from within using the proof-of-stake alternative. All it needs is a sufficiently diverse userbase.

This is relevant to Bitcoin as an alternative means of forming consensus on the valid transaction history, or the public ledger at least. But all it would necessarily borrow from Bitcoin is the public ledger contained in the block chain, in order to co-opt its userbase.

Ignore

QuantumMechanic
Member

Activity: 110

Ignore

Re: Proof of stake instead of proof of work
July 11, 2011, 04:52:44 AM

#5

Quote from: **eastendtech** on July 11, 2011, 04:37:14 AM

Its not a good idea. It would make Bitcoins follow an unfair distribution. By proving you have stake, which means, many bitcoins, you would simply be proving that you're rich and then the system would be only rewarding the already wealthy to validate transactions, and there would be little incentive to add computer power to the network, which is what the 50 btc block awards are designed for, to be a reward and a stimulus to add robustness and block validation to the network. It would create a rich-get-richer scenario, and bitcoin distribution is already following a power law.

This alternative wouldn't need computing power - that's the point.

Furthermore, if you want more influence in the current system, all you have to do is buy yourself more computing resources. Morally not much different than buying yourself more bitcoins AFAICT.

Lastly, if the bitcoin-rich (as opposed to the computing power-rich) started "voting" themselves more new bitcoins than the allowed amount, then people would lose confidence in the currency, and the bitcoin-rich would no longer be rich in any meaningful sense. They have more incentive to maintain the currency's value than the current miners do, it seems.

Ignore

Meni Rosenfeld
Donator
Hero Member

Activity: 1148

Ignore

Re: Proof of stake instead of proof of work
July 11, 2011, 04:54:29 AM

#6

This is brilliant. Of course there's still a lot of work to figure out if it's technically feasible and resistant to manipulation, concentration of power etc. But it could solve a lot of the problems Bitcoin will face going forward.

This change is significant enough to warrant a new genesis block rather than forking the current chain. This leaves the problem of initial distribution. Mining serves the purposes of both transaction validation and initial distribution, so if validation rights depend on current possession it will obviously be problematic.

I think this will work best together with, rather than instead of, hash-based mining. Blocks are confirmed with hashing as normal; this is enough for casual double-spend attempts. When a block is old enough, stakeholders can sign it. If a major attacker tries to build an alternative branch, clients will reject it because it conflicts with a branch already signed by stakeholders. This will create a much lower mining requirements for a given level of security.

I'll go as far as saying that if this pans out and is indeed novel, you deserve a place right next to Satoshi in the annals of Bitcoin. (Edit Nov 28 2013: I retract that last statement... Sure, it's a great idea and all, but a place in the hall of fame requires a bit more than that. Also I'm not sure how novel the idea really is.)

1EofZNBhWQ3kxKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
Bitcoin - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

Ignore

gmaxwell
Moderator
Hero Member

Activity: 1078

Ignore

Re: Proof of stake instead of proof of work
July 11, 2011, 05:09:47 AM

#7

Quote from: **Meni Rosenfeld** on July 11, 2011, 04:54:29 AM

I think this will work best together, rather than instead of, hash-based mining. Blocks are confirmed with hashing as normal; this is enough for casual double-spend attempts. When a block is old enough, stakeholders can sign it. If a major attacker tries to build an alternative branch, clients will reject it because it

We already have this in the form of checkpoints in the client software. You don't need voting for it, because there is no ambiguity or controversy about the identity of the block hundreds of steps ago. Better to have a check performed by everyone than just people that happen to have a lot of bitcoin already.

Ignore

Meni Rosenfeld
Donator
Hero Member

Activity: 1148

Ignore

Re: Proof of stake instead of proof of work
July 11, 2011, 05:15:42 AM

#8

Quote from: **gmaxwell** on July 11, 2011, 05:09:47 AM

Quote from: **Meni Rosenfeld** on July 11, 2011, 04:54:29 AM

I think this will work best together, rather than instead of, hash-based mining. Blocks are confirmed with hashing as normal; this is enough for casual double-spend attempts. When a block is old enough, stakeholders can sign it. If a major attacker tries to build an alternative branch, clients will reject it because it

We already have this in the form of checkpoints in the client software. You don't need voting for it, because there is no ambiguity or controversy about the identity of the block hundreds of steps ago. Better to have a check performed by everyone than just people that happen to have a lot of bitcoin already.



This is vulnerable to attacks on the network topology. You can "fake" a Bitcoin node, you can't fake having bitcoins.

I need to look more closely into the current implementation though.



Ignore

1EofoZNbWQ3kxfKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
Bitcoin - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

QuantumMechanic

Member



Activity: 110



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 05:22:35 AM

#9

Quote from: Meni Rosenfeld on July 11, 2011, 04:54:29 AM

This change is significant enough to warrant a new genesis block rather than forking the current chain.

I think a fork is more appropriate, since it incentivises the existing userbase to participate and avoids alienating them, and it solves the initial distribution problem, allowing for a very diverse voting group from the start. I see Bitcoin as having made some progress at solving the social problems (bootstrapping a new currency) just as much as it has solved the technical ones. No need to throw this progress away IMHO.

Quote from: gmaxwell on July 11, 2011, 05:09:47 AM

Quote from: Meni Rosenfeld on July 11, 2011, 04:54:29 AM

I think this will work best together, rather than instead of, hash-based mining. Blocks are confirmed with hashing as normal; this is enough for casual double-spend attempts. When a block is old enough, stakeholders can sign it. If a major attacker tries to build an alternative branch, clients will reject it because it

We already have this in the form of checkpoints in the client software. You don't need voting for it, because there is no ambiguity or controversy about the identity of the block hundreds of steps ago. Better to have a check performed by _everyone_ than just people that happen to have a lot of bitcoin already.

But Ben Laurie would be much happier ☺

Meni Rosenfeld

Donator

Hero Member



Activity: 1148



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 06:55:01 AM

#10

Another thought: Currently there's a problem with very large transfers, since guarding them against double-spending can take days (at least) of confirmations. Paying transaction fees won't help because it can only get it in a block faster, it can't significantly affect block generation rate afterwards.

With the new system (work/stake hybrid approach), senders of large amounts can include a signature fee in addition to the regular miner fee, which goes proportionally to stakeholders who sign the block in which the transaction appears. This will incentivize stakeholders to sign, making the transaction secure without having to wait for more confirmations.

1EofoZNbWQ3kxfKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
Bitcoin - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

hashcoin

Full Member



Activity: 122



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 07:09:06 AM

#11

This is the obvious way to do consensus, that doesn't work because of the massive amount of bandwidth needed to have a vote for *every bitcoin address in existence*. Keep in mind *everyone* needs to do this checking too. The *entire point*, and clever part, of bitcoin, is the idea that one can trade bandwidth/digital signatures for CPU-intensive "hash signatures" that are very small and easy to verify.

In other words, if you asked someone to design a system like bitcoin before bitcoin came out, what you propose is what most would have said...

QuantumMechanic

Member



Activity: 110



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 07:13:11 AM

#12

Quote from: Meni Rosenfeld on July 11, 2011, 06:55:01 AM

Another thought: Currently there's a problem with very large transfers, since guarding them against double-spending can take days (at least) of confirmations. Paying transaction fees won't help because it can only get it in a block faster, it can't significantly affect block generation rate afterwards.

With the new system (work/stake hybrid approach), senders of large amounts can include a signature fee in addition to the regular miner fee, which goes proportionally to stakeholders who sign the block in which the transaction appears. This will incentivize stakeholders to sign, making the transaction secure without having to wait for more confirmations.

I think a hybrid approach would only allow stakeholders to sign whole blocks that have already been found. And then miners could defer authority to the stakeholders when they do, and work forward from that block. This would decrease the confirmation time to 10 minutes on average. And it would remove the need for checkpoints, making Ben Laurie a happier camper.

QuantumMechanic

Member



Activity: 110



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 07:50:58 AM

#13

Quote from: hashcoin on July 11, 2011, 07:09:06 AM

This is the obvious way to do consensus, that doesn't work because of the massive amount of bandwidth needed to have a vote for *every bitcoin address in existence*. Keep in mind *everyone* needs to do this checking too. The *entire point*, and clever part, of bitcoin, is the idea that one can trade bandwidth/digital signatures for CPU-intensive "hash signatures" that are very small and easy to verify.

In other words, if you asked someone to design a system like bitcoin before bitcoin came out, what you propose is what most would have said...

Haha, I thought it felt a bit too obvious. Well at least I have a greater, more justified, appreciation for Bitcoin now.

This whole thing actually came from thinking of a way to solve the checkpoint problem Ben Laurie wrote about. If votes don't need to occur very often, then would the bandwidth/checking cost be such a big deal? Is his issue even really a problem?

Meni Rosenfeld

Donator

Hero Member



Activity: 1148



Re: Proof of stake instead of proof of work

July 11, 2011, 08:50:59 AM

#14

Quote from: hashcoin on July 11, 2011, 07:09:06 AM

This is the obvious way to do consensus, that doesn't work because of the massive amount of bandwidth needed to have a vote for *every bitcoin address in existence*. Keep in mind *everyone* needs to do this checking too. The *entire point*, and clever part, of bitcoin, is the idea that one can trade bandwidth/digital signatures for CPU-intensive "hash signatures" that are very small and easy to verify.



Ignore

Not every Bitcoin address in existence. Just those with a large number of bitcoins. And it needn't be for every block.

This is exactly why I say this should be used in conjunction with hashing. A small amount of proof-of-work provides the skeleton and protection against casual double-spends, while proof-of-stake provides an occasional rock-solid confirmation against massive attacks.

Nobody knows exactly how to prevent miners from including low-fee transactions, and hence, making sure people pay fees. Even if we figure it out, if we want transaction fees to remain low enough, there will be little incentive to mine, making the overall hashing capacity low. Thus the network will be vulnerable to attack, and people will not be able to safely send large amounts.

I see this as a major problem, which the inclusion of proof-of-stake, done correctly, can solve.

1EfoZNBhWQ3kxfKvWkhtMns4AivZArhr | Who am I? | bitcoin-otc WoT
Bitcoi - Exchange bitcoins for ILS (thread) | Israel Bitcoin community homepage (thread)
Analysis of Bitcoin Pooled Mining Reward Systems (thread, summary) | PureMining - Infinite-term, deterministic mining bond

Mike Hearn
Hero Member
●●●●●●

Activity: 1232



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 09:46:48 AM

#15

You need to run some simulations to prove out your theories first. I agree with hashcoin, this idea is one that is intuitively "obvious" but I think it will fail when you run the numbers.

Consider that there are people today who have some non-trivial fraction of the entire chains value under their control (like MtGox). So to have any confidence in a chain you really need a lot of value to sign over it. That means huge numbers of keys.

An ECDSA verification takes around 2-3 msec today:

<https://en.bitcoin.it/wiki/Scalability>

Assuming 2msec you can check 500 signatures per second. 500 keys is obviously not enough to accumulate enough value to have much trust in that block. You'd probably need hundreds of thousands of keys. That's a problem because if we assume a block every ten minutes then we can check at max:

$$10 * 60 * 500 = 300,000 \text{ keys}$$

before we are spending ALL our cycles doing nothing but figuring out how many people signed the block and can no longer keep up with the chain.

This isn't even taking into account signing time, storage or bandwidth.

12LMm82ZgAzf7yNDpPydEYxEr4Ap7XtSSK

QuantumMechanic
Member
●●

Activity: 110



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 09:55:56 AM

#16

If there are relatively few delegates, and if they could each store their votes in single addresses, this could save a significant amount overhead for the network as a whole during voting/checking.

I suppose stakeholders by definition have an incentive to delegate their votes in such a way as to minimize the overhead in the network, but could this alone lead to some stable equilibrium?

Maybe you could also cull some necessary amount of the least influential addresses in the vote in order to help to reduce overhead to a workable amount?

I'm not sure if there's a transaction signing scheme to deal with retracted votes though, so I don't know if such a delegated voting scheme can even work.

Mike Hearn
Hero Member
●●●●●●

Activity: 1232



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 11:31:23 AM

#17

If there are relatively few delegates then what you have is effectively similar to today's payment networks, in which a few big players with pre-established trust relationships come to a consensus on who owns what between themselves. You'd have invented an open source SWIFT.

Bitcoin is somewhat like that as well today due to the (unforeseen) power of the pools. But there are workable proposals for breaking that power and restoring Bitcoin to many thousands of voters again instead of just a few. They haven't been implemented but that's because talk is cheap, whereas working C++ isn't.

By the way, I'm not trying to pour cold water on your idea. Proof of stake doesn't seem like an inherently bad idea to me, and I think everyone agrees proof of work seems wasteful, even if in the end it's less wasteful than the current financial systems. But there's a huge gulf between "how about this?" and a working system that scales.

12LMm82ZgAzf7yNDpPydEYxEr4Ap7XtSSK

QuantumMechanic
Member
●●

Activity: 110



Ignore



Re: Proof of stake instead of proof of work

July 11, 2011, 06:54:28 PM

#18

Quote from: Mike Hearn on July 11, 2011, 11:31:23 AM

If there are relatively few delegates then what you have is effectively similar to today's payment networks, in which a few big players with pre-established trust relationships come to a consensus on who owns what between themselves. You'd have invented an open source SWIFT.

It sounds like it could scale to more than just a few. And the voting power might flow much more easily away from corrupted players than with SWIFT due to the almost zero barriers to retracting votes, and to enter into the market to supplant the established delegates.

Remember, with delegated voting you don't necessarily give your vote directly to who will eventually cast it - you can just pass it on to someone you trust, hopefully solving the problem of rational ignorance/apathy. That or the bad guys will just buy up all the votes 😊 Or maybe the good guys will?

I guess the relevant question is if this would do a *good enough* job of decentralizing the "voting power" and making it accountable? If so, the benefits listed above (that have survived) seem pretty significant. It might at least be a good approach to smaller blockchains that don't expect to attract enough outside computing power to keep them secure.

Quote from: Mike Hearn on July 11, 2011, 11:31:23 AM

But there are workable proposals for breaking that power and restoring Bitcoin to many thousands of voters again instead of just a few. They haven't been

One thing is, members propose for creating that panel and selecting someone to take decisions or take action instead of just a coin they have a seat implemented but that's because talk is cheap, whereas working C++ isn't.

That is good to hear! I understand that proposals from people who obviously aren't qualified to program the shit they propose, not to mention that most of what they propose is indeed shit, probably gets pretty annoying.

Quote from: Mike Hearn on July 11, 2011, 11:31:23 AM

By the way, I'm not trying to pour cold water on your idea.

Not at all! I quite appreciate your thoughtful responses!

Mike Hearn
Hero Member


Activity: 1232



Ignore

 Re: Proof of stake instead of proof of work
July 11, 2011, 07:53:28 PM

#19

By the way, if you're interested in delegated voting, I wrote a paper on using it in a democracy some time ago:

<https://docs.google.com/document/pub?id=1jldmNJHWAtsPLCUD7EPpm8jOEV93kSXbZOMycqCWOyA>

I think delegating voting is a great idea. I'm less convinced that it would work for replacing hashing in the block chain, but it might.

Some more problems for you to consider:

- If I control a key with 10 delegators, then I spend that key to 10 new keys, what happens to the delegations? Does it make sense for a key to have fan-out (ie 1 key to delegate to 10 other keys)? Do you split the value evenly or not?
- How do you efficiently detect and resolve delegation loops? ie, key A delegates to B, B delegates to C and C then delegates to A? You have to do this efficiently because otherwise setting up new delegations is a way to DoS the system. Note that detecting cycles in a large graph is slow, Tarjan's algorithm is $O(\text{number of edges})$. If each key in the system takes part then every new address in the system potentially makes setting up new delegations harder. As the number of keys in the system grows without bound over time, that could be an issue
- Keys are owned by end users who may or may not know/care about the inner workings of their currency. How do you get new users to delegate appropriately?

It should be possible to run a parallel block chain based on proof-by-stake, but using the same transactions. It could be done with an adapted Bitcoin software. Once such a chain was up and running you could compare the results vs the existing chain.

I don't have the time/energy to do this myself.

12LMm82ZgAzf7yNDpPydEYxEr4Ap7XtSSK

casascius
Mike Caldwell
VIP
Hero Member


Activity: 1204



The Casascius 10z 10BTC
Silver Round (w/ Gold B)



Ignore

 Re: Proof of stake instead of proof of work
July 11, 2011, 10:48:44 PM

#20

I was thinking a future Bitcoin client would need a screen of trusted certificates - sort of how your web browser already has one for SSL (well hidden), that helps it decide whose signatures are trusted, and can ultimately be controlled by the user.

Today, it is already a major pain in the ass to redownload the block chain, which takes hours, presumably while every single signature for every transaction is checked. Yet, if I trusted a certificate from Gavin Andresen, and Gavin published a checkpoint certificate vouching for the correct hash of block 130,000 (for example), and this was relayed in a future version of the p2p protocol, then my client could just accept all blocks before that as valid without checking every bloody signature in every block. And if I as a user didn't like or trust that behavior, I would merely remove Gavin's public key from my trusted certs list, or add a key of someone I trust more, the same way I can do this with SSL CA certs in my browser.

Companies claiming they got hacked and lost your coins sounds like fraud so perfect it could be called fashionable. I never believe them. If I ever experience the misfortune of a real intrusion, I declare I have been honest about the way I have managed the keys in Casascius Coins. I maintain no ability to recover or reproduce the keys, not even under limitless duress or total intrusion. Remember that trusting strangers with your coins without any recourse is, as a matter of principle, not a best practice. Don't keep coins online. Use paper wallets instead.

Pages: [1] 2 All

print

Bitcoin Forum > Bitcoin > Development & Technical Discussion (Moderator: gmaxwell) > **Proof of stake instead of proof of work**

« previous topic next topic »

Jump to: => Development & Technical Discussion go

Sponsored by Private Internet Access, a Bitcoin-accepting VPN.

Powered by SMF 1.1.19 | SMF © 2006-2009, Simple Machines



GitHub

This repository ▾

Search or type a command



Explore

Features

Enterprise

Blog

Sign up

Sign in

PUBLIC



ethereum / wiki

★ Star

390

🍴 Fork

47

Home

stephantual edited this page on Jun 24 · 18 revisions

Select your language:

- [English](#)
- [Japanese](#) - TBD
- [Romanian](#) - TBD
- [German](#) - TBD
- [Chinese](#) - TBD
- [Spanish](#) - TBD
- [French](#) - TBD
- [فارسی \(Persian\)](#) - TBD

▼ Pages 42

Find a Page...

- [\[Chinese\] Ethereum TOC](#)
- [\[English\] Block Protocol 2.0](#)
- [\[English\] Clearinghouse](#)
- [\[English\] CLL](#)
- [\[English\] Dagger](#)
- [\[English\] Ethereum TOC](#)
- [\[English\] Layers](#)
- [\[English\] Patricia Tree](#)
- [\[English\] RLP](#)
- [\[English\] White Paper](#)
- [\[English\] Wire Protocol](#)
- [\[French\] Ethereum TOC](#)
- [\[German\] Clearinghaus](#)
- [\[German\] Ethereum TOC](#)
- [\[German\] RLP](#)

[Show 27 more pages...](#)

Clone this wiki locally

<https://github.com/ethereum/>





◀ [English] White Paper

vbuterin edited this page on Apr 24 · 20 revisions

A Next-Generation Smart Contract and Decentralized Application Platform

When Satoshi Nakamoto first set the Bitcoin blockchain into motion in January 2009, he was simultaneously introducing two radical and highly untested concepts. The first is the "bitcoin", a decentralized peer-to-peer online currency that maintains a value without any backing, [intrinsic value](#) or central issuer. So far, the "bitcoin" as a currency unit has taken up the bulk of the public attention, both in terms of the political aspects of a currency without a central bank and its extreme upward and downward volatility in price. However, there is also another, equally important, part to Satoshi's grand experiment: the concept of a proof of work-based blockchain to allow for public agreement on the order of transactions. Bitcoin as an application can be described as a first-to-file system: if one entity has 50 BTC, and simultaneously sends the same 50 BTC to A and to B, only the transaction that gets confirmed first will process. There is no intrinsic way of determining from two transactions which came earlier, and for decades this stymied the development of decentralized digital currency. Satoshi's blockchain was the first credible decentralized solution. And now, attention is rapidly starting to shift toward this second part of Bitcoin's technology, and how the blockchain concept can be used for more than just money.

Commonly cited applications include using on-blockchain digital assets to represent custom currencies and financial instruments ("[colored coins](#)"), the ownership of an underlying physical device ("[smart property](#)"), non-fungible assets such as domain names ("Namecoin") as well as more advanced applications such as decentralized exchange, financial derivatives, peer-to-peer gambling and on-blockchain identity and reputation systems. Another important area of inquiry is "smart contracts" - systems which automatically move digital assets according to arbitrary pre-specified rules. For example, one might have a treasury contract of the form "A can withdraw up to X currency units per day, B can withdraw up to Y per day, A and B together can withdraw anything, and A can shut off B's ability to withdraw". The logical extension of this is [decentralized autonomous organizations](#) (DAOs) - long-term smart contracts that contain the assets and encode the bylaws of an entire organization. What Ethereum intends to provide is a blockchain with a built-in fully fledged Turing-complete programming language that can be used to create "contracts" that can be used to encode arbitrary state transition functions, allowing users to create any of the systems described above, as well as many others that we have not yet imagined, simply by writing up the logic in a few lines of code.

Table of Contents

- [History](#)
 - [Bitcoin As A State Transition System](#)
 - [Mining](#)
 - [Merkle Trees](#)
 - [Alternative Blockchain Applications](#)
 - [Scripting](#)
- [Ethereum](#)
 - [Ethereum Accounts](#)
 - [Messages and Transactions](#)
 - [Ethereum State Transition Function](#)
 - [Code Execution](#)

▼ Pages 34

Find a Page...

- [\[Chinese\] Ethereum TOC](#)
- [\[English\] Block Protocol 2.0](#)
- [\[English\] Clearinghouse](#)
- [\[English\] CLL](#)
- [\[English\] Dagger](#)
- [\[English\] Ethereum TOC](#)
- [\[English\] Layers](#)
- [\[English\] Patricia Tree](#)
- [\[English\] RLP](#)
- [\[English\] Serpent programming language operations](#)
- [\[English\] White Paper](#)
- [\[English\] Wire Protocol](#)
- [\[French\] Ethereum TOC](#)
- [\[German\] Clearinghaus](#)
- [\[German\] Ethereum TOC](#)

Show 19 more pages...

Clone this wiki locally

<https://github.com/ethereum/>



- Blockchain and Mining
- Applications
 - Token Systems
 - Financial derivatives
 - Identity and Reputation Systems
 - Decentralized File Storage
 - Decentralized Autonomous Organizations
 - Further Applications
- Miscellanea And Concerns
 - Modified GHOST Implementation
 - Fees
 - Computation And Turing-Completeness
 - Currency And Issuance
 - Mining Centralization
 - Scalability
- Conclusion
- References and Further Reading

Introduction to Bitcoin and Existing Concepts

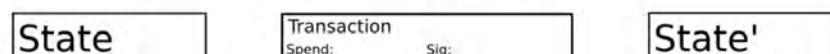
History

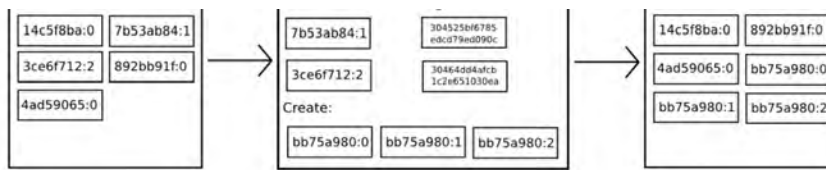
The concept of decentralized digital currency, as well as alternative applications like property registries, has been around for decades. The anonymous e-cash protocols of the 1980s and the 1990s, mostly reliant on a cryptographic primitive known as Chaumian blinding, provided a currency with a high degree of privacy, but the protocols largely failed to gain traction because of their reliance on a centralized intermediary. In 1998, Wei Dai's [b-money](#) became the first proposal to introduce the idea of creating money through solving computational puzzles as well as decentralized consensus, but the proposal was scant on details as to how decentralized consensus could actually be implemented. In 2005, Hal Finney introduced a concept of ["reusable proofs of work"](#), a system which uses ideas from b-money together with Adam Back's computationally difficult Hashcash puzzles to create a concept for a cryptocurrency, but once again fell short of the ideal by relying on trusted computing as a backend.

Because currency is a first-to-file application, where the order of transactions is often of critical importance, decentralized currencies require a solution to decentralized consensus. The main roadblock that all pre-Bitcoin currency protocols faced is the fact that, while there had been plenty of research on creating secure Byzantine-fault-tolerant multiparty consensus systems for many years, all of the protocols described were solving only half of the problem. The protocols assumed that all participants in the system were known, and produced security margins of the form "if N parties participate, then the system can tolerate up to N/4 malicious actors". The problem is, however, that in an anonymous setting such security margins are vulnerable to sybil attacks, where a single attacker creates thousands of simulated nodes on a server or botnet and uses these nodes to unilaterally secure a majority share.

The innovation provided by Satoshi is the idea of combining a very simple decentralized consensus protocol, based on nodes combining transactions into a "block" every ten minutes creating an ever-growing blockchain, with proof of work as a mechanism through which nodes gain the right to participate in the system. While nodes with a large amount of computational power do have proportionately greater influence, coming up with more computational power than the entire network combined is much harder than simulating a million nodes. Despite the Bitcoin blockchain model's crudeness and simplicity, it has proven to be good enough, and would over the next five years become the bedrock of over two hundred currencies and protocols around the world.

Bitcoin As A State Transition System





From a technical standpoint, the Bitcoin ledger can be thought of as a state transition system, where there is a "state" consisting of the ownership status of all existing bitcoins and a "state transition function" that takes a state and a transaction and outputs a new state which is the result. In a standard banking system, for example, the state is a balance sheet, a transaction is a request to move \$X from A to B, and the state transition function reduces the value in A's account by \$X and increases the value in B's account by \$X. If A's account has less than \$X in the first place, the state transition function returns an error. Hence, one can formally define:

```
APPLY(S,TX) -> S' or ERROR
```

In the banking system defined above:

```
APPLY({ Alice: $50, Bob: $50 }, "send $20 from Alice to Bob") = { Alice: $30, Bob: $70 }
```

But:

```
APPLY({ Alice: $50, Bob: $50 }, "send $70 from Alice to Bob") = ERROR
```

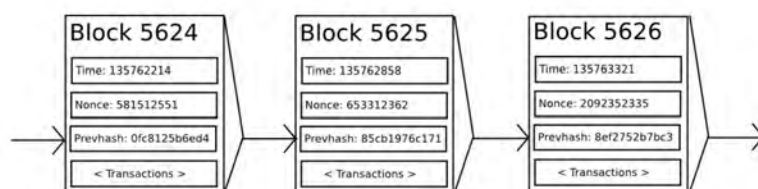
The "state" in Bitcoin is the collection of all coins (technically, "unspent transaction outputs" or UTXO) that have been minted and not yet spent, with each UTXO having a denomination and an owner (defined by a 20-byte address which is essentially a cryptographic public key^[1]). A transaction contains one or more inputs, with each input containing a reference to an existing UTXO and a cryptographic signature produced by the private key associated with the owner's address, and one or more outputs, with each output containing a new UTXO to be added to the state.

The state transition function $APPLY(S, TX) \rightarrow S'$ can be defined roughly as follows:

- For each input in TX :
 - If the referenced UTXO is not in S , return an error.
 - If the provided signature does not match the owner of the UTXO, return an error.
- If the sum of the denominations of all input UTXO is less than the sum of the denominations of all output UTXO, return an error.
- Return S with all input UTXO removed and all output UTXO added.

The first half of the first step prevents transaction senders from spending coins that do not exist, the second half of the first step prevents transaction senders from spending other people's coins, and the second step enforces conservation of value. In order to use this for payment, the protocol is as follows. Suppose Alice wants to send 11.7 BTC to Bob. First, Alice will look for a set of available UTXO that she owns that totals up to at least 11.7 BTC. Realistically, Alice will not be able to get exactly 11.7 BTC; say that the smallest she can get is 6+4+2=12. She then creates a transaction with those three inputs and two outputs. The first output will be 11.7 BTC with Bob's address as its owner, and the second output will be the remaining 0.3 BTC "change", with the owner being Alice herself.

Mining



If we had access to a trustworthy centralized service, this system would be trivial to implement; it could simply be coded exactly as described. However, with Bitcoin we are trying to build a

decentralized currency system, so we will need to combine the state transaction system with a consensus system in order to ensure that everyone agrees on the order of transactions. Bitcoin's decentralized consensus process requires nodes in the network to continuously attempt to produce packages of transactions called "blocks". The network is intended to produce roughly one block every ten minutes, with each block containing a timestamp, a nonce, a reference to (ie. hash of) the previous block and a list of all of the transactions that have taken place since the previous block. Over time, this creates a persistent, ever-growing, "blockchain" that constantly updates to represent the latest state of the Bitcoin ledger.

The algorithm for checking if a block is valid, expressed in this paradigm, is as follows:

1. Check if the previous block referenced by the block exists and is valid.
2. Check that the timestamp of the block is greater than that of the previous block^[2] and less than 2 hours into the future
3. Check that the proof of work on the block is valid.
4. Let $s[e]$ be the state at the end of the previous block.
5. Suppose TX is the block's transaction list with n transactions. For all i in $0 \dots n-1$, set $s[i+1] = \text{APPLY}(s[i], TX[i])$. If any application returns an error, exit and return false.
6. Return true, and register $s[n]$ as the state at the end of this block.

Essentially, each transaction in the block must provide a state transition that is valid. Note that the state is not encoded in the block in any way; it is purely an abstraction to be remembered by the validating node and can only be (securely) computed for any block by starting from the genesis state and sequentially applying every transaction in every block. Note that the order in which the miner includes transactions into the block matters; if there are two transactions A and B in a block such that B spends a UTXO created by A, then the block will be valid if A comes before B but not otherwise.

The interesting part of the block validation algorithm is the concept of "proof of work": the condition is that the SHA256 hash of every block, treated as a 256-bit number, must be less than a dynamically adjusted target, which as of the time of this writing is approximately 2^{192} . The purpose of this is to make block creation computationally "hard", thereby preventing sybil attackers from remaking the entire blockchain in their favor. Because SHA256 is designed to be a completely unpredictable pseudorandom function, the only way to create a valid block is simply trial and error, repeatedly incrementing the nonce and seeing if the new hash matches. At the current target of 2^{192} , this means an average of 2^{64} tries; in general, the target is recalibrated by the network every 2016 blocks so that on average a new block is produced by some node in the network every ten minutes. In order to compensate miners for this computational work, the miner of every block is entitled to include a transaction giving themselves 25 BTC out of nowhere. Additionally, if any transaction has a higher total denomination in its inputs than in its outputs, the difference also goes to the miner as a "transaction fee". Incidentally, this is also the only mechanism by which BTC are issued; the genesis state contained no coins at all.

In order to better understand the purpose of mining, let us examine what happens in the event of a malicious attacker. Since Bitcoin's underlying cryptography is known to be secure, the attacker will target the one part of the Bitcoin system that is not protected by cryptography directly: the order of transactions. The attacker's strategy is simple:

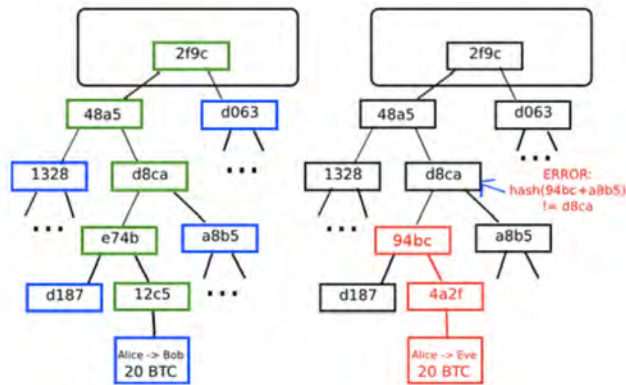
1. Send 100 BTC to a merchant in exchange for some product (preferably a rapid-delivery digital good)
2. Wait for the delivery of the product
3. Produce another transaction sending the same 100 BTC to himself
4. Try to convince the network that his transaction to himself was the one that came first.

Once step (1) has taken place, after a few minutes some miner will include the transaction in a block, say block number 270000. After about one hour, five more blocks will have been added to the chain after that block, with each of those blocks indirectly pointing to the transaction and thus "confirming" it. At this point, the merchant will accept the payment as finalized and deliver the product; since we are assuming this is a digital good, delivery is instant. Now, the attacker creates another transaction sending the 100 BTC to himself. If the attacker simply releases it into the wild, the transaction will not be processed; miners will attempt to run $\text{APPLY}(s, TX)$ and notice that TX consumes a UTXO which is no longer in the state. So instead, the attacker creates a "fork" of the blockchain, starting by mining another version of block 270000 pointing to the same block 269999 as a parent but with the new transaction in place of the old one.

the same block 200000 as a parent but with the new transaction in place of the old one.

Because the block data is different, this requires redoing the proof of work. Furthermore, the attacker's new version of block 270000 has a different hash, so the original blocks 270001 to 270005 do not "point" to it; thus, the original chain and the attacker's new chain are completely separate. The rule is that in a fork the longest blockchain is taken to be the truth, and so legitimate miners will work on the 270005 chain while the attacker alone is working on the 270000 chain. In order for the attacker to make his blockchain the longest, he would need to have more computational power than the rest of the network combined in order to catch up (hence, "51% attack").

Merkle Trees



Left: it suffices to present only a small number of nodes in a Merkle tree to give a proof of the validity of a branch.

Right: any attempt to change any part of the Merkle tree will eventually lead to an inconsistency somewhere up the chain.

An important scalability feature of Bitcoin is that the block is stored in a multi-level data structure. The "hash" of a block is actually only the hash of the block header, a roughly 200-byte piece of data that contains the timestamp, nonce, previous block hash and the root hash of a data structure called the Merkle tree storing all transactions in the block. A Merkle tree is a type of binary tree, composed of a set of nodes with a large number of leaf nodes at the bottom of the tree containing the underlying data, a set of intermediate nodes where each node is the hash of its two children, and finally a single root node, also formed from the hash of its two children, representing the "top" of the tree. The purpose of the Merkle tree is to allow the data in a block to be delivered piecemeal: a node can download only the header of a block from one source, the small part of the tree relevant to them from another source, and still be assured that all of the data is correct. The reason why this works is that hashes propagate upward: if a malicious user attempts to swap in a fake transaction into the bottom of a Merkle tree, this change will cause a change in the node above, and then a change in the node above that, finally changing the root of the tree and therefore the hash of the block, causing the protocol to register it as a completely different block (almost certainly with an invalid proof of work).

The Merkle tree protocol is arguably essential to long-term sustainability. A "full node" in the Bitcoin network, one that stores and processes the entirety of every block, takes up about 15 GB of disk space in the Bitcoin network as of April 2014, and is growing by over a gigabyte per month. Currently, this is viable for some desktop computers and not phones, and later on in the future only businesses and hobbyists will be able to participate. A protocol known as "simplified payment verification" (SPV) allows for another class of nodes to exist, called "light nodes", which download the block headers, verify the proof of work on the block headers, and then download only the "branches" associated with transactions that are relevant to them. This allows light nodes to determine with a strong guarantee of security what the status of any Bitcoin transaction, and their current balance, is while downloading only a very small portion of the entire blockchain.

Alternative Blockchain Applications

The idea of taking the underlying blockchain idea and applying it to other concepts also has a long history. In 2005 Nick Szabo came out with the concept of "secure property titles with owner

authority", a document describing how "new advances in replicated database technology" will allow for a blockchain-based system for storing a registry of who owns what land, creating an elaborate framework including concepts such as homesteading, adverse possession and Georgian land tax. However, there was unfortunately no effective replicated database system available at the time, and so the protocol was never implemented in practice. After 2009, however, once Bitcoin's decentralized consensus was developed a number of alternative applications rapidly began to emerge.

- **Namecoin** - created in 2010, Namecoin is best described as a decentralized name registration database. In decentralized protocols like Tor, Bitcoin and BitMessage, there needs to be some way of identifying accounts so that other people can interact with them, but in all existing solutions the only kind of identifier available is a pseudorandom hash like `1LW79wp5Z8qaHW1jLSTC1BCrhQYtHagUwy`. Ideally, one would like to be able to have an account with a name like "george". However, the problem is that if one person can create an account named "george" then someone else can use the same process to register "george" for themselves as well and impersonate them. The only solution is a first-to-file paradigm, where the first registerer succeeds and the second fails - a problem perfectly suited for the Bitcoin consensus protocol. Namecoin is the oldest, and most successful, implementation of a name registration system using such an idea.
- **Colored coins** - the purpose of colored coins is to serve as a protocol to allow people to create their own digital currencies - or, in the important trivial case of a currency with one unit, digital tokens, on the Bitcoin blockchain. In the colored coins protocol, one "issues" a new currency by publicly assigning a color to a specific Bitcoin UTXO, and the protocol recursively defines the color of other UTXO to be the same as the color of the inputs that the transaction creating them spent (some special rules apply in the case of mixed-color inputs). This allows users to maintain wallets containing only UTXO of a specific color and send them around much like regular bitcoins, backtracking through the blockchain to determine the color of any UTXO that they receive.
- **Metacoins** - the idea behind a metacoin is to have a protocol that lives on top of Bitcoin, using Bitcoin transactions to store metacoin transactions but having a different state transition function, `APPLY'`. Because the metacoin protocol cannot prevent invalid metacoin transactions from appearing in the Bitcoin blockchain, a rule is added that if `APPLY'(S,TX)` returns an error, the protocol defaults to `APPLY'(S,TX) = S`. This provides an easy mechanism for creating an arbitrary cryptocurrency protocol, potentially with advanced features that cannot be implemented inside of Bitcoin itself, but with a very low development cost since the complexities of mining and networking are already handled by the Bitcoin protocol.

Thus, in general, there are two approaches toward building a consensus protocol: building an independent network, and building a protocol on top of Bitcoin. The former approach, while reasonably successful in the case of applications like Namecoin, is difficult to implement; each individual implementation needs to bootstrap an independent blockchain, as well as building and testing all of the necessary state transition and networking code. Additionally, we predict that the set of applications for decentralized consensus technology will follow a power law distribution where the vast majority of applications would be too small to warrant their own blockchain, and we note that there exist large classes of decentralized applications, particularly decentralized autonomous organizations, that need to interact with each other.

The Bitcoin-based approach, on the other hand, has the flaw that it does not inherit the simplified payment verification features of Bitcoin. SPV works for Bitcoin because it can use blockchain depth as a proxy for validity; at some point, once the ancestors of a transaction go far enough back, it is safe to say that they were legitimately part of the state. Blockchain-based meta-protocols, on the other hand, cannot force the blockchain not to include transactions that are not valid within the context of their own protocols. Hence, a fully secure SPV meta-protocol implementation would need to backward scan all the way to the beginning of the Bitcoin blockchain to determine whether or not certain transactions are valid. Currently, all "light" implementations of Bitcoin-based meta-protocols rely on a trusted server to provide the data, arguably a highly suboptimal result especially when one of the primary purposes of a cryptocurrency is to eliminate the need for trust.

Even without any extensions, the Bitcoin protocol actually does facilitate a weak version of a concept of "smart contracts". UTXO in Bitcoin can be owned not just by a public key, but also by a more complicated script expressed in a simple stack-based programming language. In this paradigm, a transaction spending that UTXO must provide data that satisfies the script. Indeed, even the basic public key ownership mechanism is implemented via a script: the script takes an elliptic curve signature as input, verifies it against the transaction and the address that owns the UTXO, and returns 1 if the verification is successful and 0 otherwise. Other, more complicated, scripts exist for various additional use cases. For example, one can construct a script that requires signatures from two out of a given three private keys to validate ("multisig"), a setup useful for corporate accounts, secure savings accounts and some merchant escrow situations. Scripts can also be used to pay bounties for solutions to computational problems, and one can even construct a script that says something like "this Bitcoin UTXO is yours if you can provide an SPV proof that you sent a Dogecoin transaction of this denomination to me", essentially allowing decentralized cross-cryptocurrency exchange.

However, the scripting language as implemented in Bitcoin has several important limitations:

- **Lack of Turing-completeness** - that is to say, while there is a large subset of computation that the Bitcoin scripting language supports, it does not nearly support everything. The main category that is missing is loops. This is done to avoid infinite loops during transaction verification; theoretically it is a surmountable obstacle for script programmers, since any loop can be simulated by simply repeating the underlying code many times with an if statement, but it does lead to scripts that are very space-inefficient. For example, implementing an alternative elliptic curve signature algorithm would likely require 256 repeated multiplication rounds all individually included in the code.
- **Value-blindness** - there is no way for a UTXO script to provide fine-grained control over the amount that can be withdrawn. For example, one powerful use case of an oracle contract would be a hedging contract, where A and B put in \$1000 worth of BTC and after 30 days the script sends \$1000 worth of BTC to A and the rest to B. This would require an oracle to determine the value of 1 BTC in USD, but even then it is a massive improvement in terms of trust and infrastructure requirement over the fully centralized solutions that are available now. However, because UTXO are all-or-nothing, the only way to achieve this is through the very inefficient hack of having many UTXO of varying denominations (eg. one UTXO of 2^k for every k up to 30) and having O pick which UTXO to send to A and which to B.
- **Lack of state** - UTXO can either be spent or unspent; there is no opportunity for multi-stage contracts or scripts which keep any other internal state beyond that. This makes it hard to make multi-stage options contracts, decentralized exchange offers or two-stage cryptographic commitment protocols (necessary for secure computational bounties). It also means that UTXO can only be used to build simple, one-off contracts and not more complex "stateful" contracts such as decentralized organizations, and makes meta-protocols difficult to implement. Binary state combined with value-blindness also mean that another important application, withdrawal limits, is impossible.
- **Blockchain-blindness** - UTXO are blind to blockchain data such as the nonce and previous hash. This severely limits applications in gambling, and several other categories, by depriving the scripting language of a potentially valuable source of randomness.

Thus, we see three approaches to building advanced applications on top of cryptocurrency: building a new blockchain, using scripting on top of Bitcoin, and building a meta-protocol on top of Bitcoin. Building a new blockchain allows for unlimited freedom in building a feature set, but at the cost of development time and bootstrapping effort. Using scripting is easy to implement and standardize, but is very limited in its capabilities, and meta-protocols, while easy, suffer from faults in scalability. With Ethereum, we intend to build a generalized framework that can provide the advantages of all three paradigms at the same time.

Ethereum

The intent of Ethereum is to merge together and improve upon the concepts of scripting, altcoins and on-chain meta-protocols, and allow developers to create arbitrary consensus-based applications that have the scalability, standardization, feature-completeness, ease of development and interoperability offered by these different paradigms all at the same time.

Ethereum does this by building what is essentially the ultimate abstract foundational layer: a

Ethereum does this by building what is essentially the ultimate abstract computational layer on a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions. A bare-bones version of Namecoin can be written in two lines of code, and other protocols like currencies and reputation systems can be built in under twenty. Smart contracts, cryptographic "boxes" that contain value and only unlock it if certain conditions are met, can also be built on top of our platform, with vastly more power than that offered by Bitcoin scripting because of the added powers of Turing-completeness, value-awareness, blockchain-awareness and state.

Ethereum Accounts

In Ethereum, the state is made up of objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts. An Ethereum account contains four fields:

- The **nonce**, a counter used to make sure each transaction can only be processed once
- The account's current **ether balance**
- The account's **contract code**, if present
- The account's **storage** (empty by default)

"Ether" is the main internal crypto-fuel of Ethereum, and is used to pay transaction fees. In general, there are two types of accounts: externally owned accounts, controlled by private keys, and contract accounts, controlled by their contract code. An externally owned account has no code, and one can send messages from an externally owned account by creating and signing a transaction; in a contract account, every time the contract account receives a message its code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn.

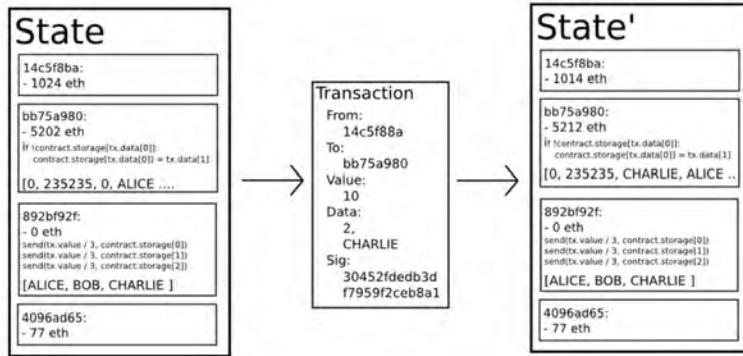
Messages and Transactions

"Messages" in Ethereum are somewhat similar to transactions in Bitcoin, but with three important differences. First, an Ethereum message can be created either by an external entity or a contract, whereas a Bitcoin transaction can only be created externally. Second, there is an explicit option for Ethereum messages to contain data. Finally, the recipient of an Ethereum message, if it is a contract account, has the option to return a response; this means that Ethereum messages also encompass the concept of functions.

The term "transaction" is used in Ethereum to refer to the signed data package that stores a message to be sent from an externally owned account. Transactions contain the recipient of the message, a signature identifying the sender, the amount of ether and the data to send, as well as two values called `STARTGAS` and `GASPRICE`. In order to prevent exponential blowup and infinite loops in code, each transaction is required to set a limit to how many computational steps of code execution it can spawn, including both the initial message and any additional messages that get spawned during execution. `STARTGAS` is this limit, and `GASPRICE` is the fee to pay to the miner per computational step. If transaction execution "runs out of gas", all state changes revert - except for the payment of the fees, and if transaction execution halts with some gas remaining then the remaining portion of the fees is refunded to the sender. There is also a separate transaction type, and corresponding message type, for creating a contract; the address of a contract is calculated based on the hash of the account nonce and transaction data.

An important consequence of the message mechanism is the "first class citizen" property of Ethereum - the idea that contracts have equivalent powers to external accounts, including the ability to send message and create other contracts. This allows contracts to simultaneously serve many different roles: for example, one might have a member of a decentralized organization (a contract) be an escrow account (another contract) between an paranoid individual employing custom quantum-proof Lamport signatures (a third contract) and a co-signing entity which itself uses an account with five keys for security (a fourth contract). The strength of the Ethereum platform is that the decentralized organization and the escrow contract do not need to care about what kind of account each party to the contract is.

Ethereum State Transition Function



The Ethereum state transition function, $\text{APPLY}(S, TX) \rightarrow S'$ can be defined as follows:

1. Check if the transaction is well-formed (ie. has the right number of values), the signature is valid, and the nonce matches the nonce in the sender's account. If not, return an error.
2. Calculate the transaction fee as $\text{STARTGAS} * \text{GASPRICE}$, and determine the sending address from the signature. Subtract the fee from the sender's account balance and increment the sender's nonce. If there is not enough balance to spend, return an error.
3. Initialize $\text{GAS} = \text{STARTGAS}$, and take off a certain quantity of gas per byte to pay for the bytes in the transaction.
4. Transfer the transaction value from the sender's account to the receiving account. If the receiving account does not yet exist, create it. If the receiving account is a contract, run the contract's code either to completion or until the execution runs out of gas.
5. If the value transfer failed because the sender did not have enough money, or the code execution ran out of gas, revert all state changes except the payment of the fees, and add the fees to the miner's account.
6. Otherwise, refund the fees for all remaining gas to the sender, and send the fees paid for gas consumed to the miner.

For example, suppose that the contract's code is:

```
if !contract.storage[msg.data[0]]:
    contract.storage[msg.data[0]] = msg.data[1]
```

Note that in reality the contract code is written in the low-level EVM code; this example is written in Serpent, our high-level language, for clarity, and can be compiled down to EVM code.

Suppose that the contract's storage starts off empty, and a transaction is sent with 10 ether value, 2000 gas, 0.001 ether gasprice, and two data fields: `[ 2, 'CHARLIE' ]`^[3]. The process for the state transition function in this case is as follows:

1. Check that the transaction is valid and well formed.
2. Check that the transaction sender has at least $2000 * 0.001 = 2$ ether. If it is, then subtract 2 ether from the sender's account.
3. Initialize $\text{gas} = 2000$; assuming the transaction is 170 bytes long and the byte-fee is 5, subtract 850 so that there is 1150 gas left.
4. Subtract 10 more ether from the sender's account, and add it to the contract's account.
5. Run the code. In this case, this is simple: it checks if the contract's storage at index `2` is used, notices that it is not, and so it sets the storage at index `2` to the value `CHARLIE`. Suppose this takes 187 gas, so the remaining amount of gas is $1150 - 187 = 963$.
6. Add $963 * 0.001 = 0.963$ ether back to the sender's account, and return the resulting state.

If there was no contract at the receiving end of the transaction, then the total transaction fee would simply be equal to the provided `GASPRICE` multiplied by the length of the transaction in bytes, and the data sent alongside the transaction would be irrelevant. Additionally, note that contract-initiated messages can assign a gas limit to the computation that they spawn, and if the sub-computation runs out of gas it gets reverted only to the point of the message call. Hence, just like transactions, contracts can secure their limited computational resources by setting strict limits on the sub-computations that they spawn.

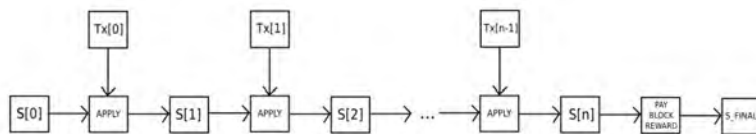
The code in Ethereum contracts is written in a low-level, stack-based bytecode language, referred to as "Ethereum virtual machine code" or "EVM code". The code consists of a series of bytes, where each byte represents an operation. In general, code execution is an infinite loop that consists of repeatedly carrying out the operation at the current program counter (which begins at zero) and then incrementing the program counter by one, until the end of the code is reached or an error or `STOP` or `RETURN` instruction is detected. The operations have access to three types of space in which to store data:

- The **stack**, a last-in-first-out container to which values can be pushed and popped
- **Memory**, an infinitely expandable byte array
- The contract's long-term **storage**, a key/value store. Unlike stack and memory, which reset after computation ends, storage persists for the long term.

The code can also access the value, sender and data of the incoming message, as well as block header data, and the code can also return a byte array of data as an output.

The formal execution model of EVM code is surprisingly simple. While the Ethereum virtual machine is running, its full computational state can be defined by the tuple `(block_state, transaction, message, code, memory, stack, pc, gas)`, where `block_state` is the global state containing all accounts and includes balances and storage. Every round of execution, the current instruction is found by taking the `pc`th byte of `code`, and each instruction has its own definition in terms of how it affects the tuple. For example, `ADD` pops two items off the stack and pushes their sum, reduces `gas` by 1 and increments `pc` by 1, and `STORE` pushes the top two items off the stack and inserts the second item into the contract's storage at the index specified by the first item. Although there are many ways to optimize Ethereum via just-in-time compilation, a basic implementation of Ethereum can be done in a few hundred lines of code.

Blockchain and Mining



The Ethereum blockchain is in many ways similar to the Bitcoin blockchain, although it does have some differences. The main difference between Ethereum and Bitcoin with regard to the blockchain architecture is that, unlike Bitcoin, Ethereum blocks contain a copy of both the transaction list and the most recent state. Aside from that, two other values, the block number and the difficulty, are also stored in the block. The block validation algorithm in Ethereum is as follows:

1. Check if the previous block referenced exists and is valid.
2. Check that the timestamp of the block is greater than that of the referenced previous block and less than 15 minutes into the future
3. Check that the block number, difficulty, transaction root, uncle root and gas limit (various low-level Ethereum-specific concepts) are valid.
4. Check that the proof of work on the block is valid.
5. Let `S[0]` be the `STATE_ROOT` of the previous block.
6. Let `TX` be the block's transaction list, with `n` transactions. For all `i` in `0...n-1`, set `S[i+1] = APPLY(S[i], TX[i])`. If any applications returns an error, or if the total gas consumed in the block up until this point exceeds the `GASLIMIT`, return an error.
7. Let `S_FINAL` be `S[n]`, but adding the block reward paid to the miner.
8. Check if `S_FINAL` is the same as the `STATE_ROOT`. If it is, the block is valid; otherwise, it is not valid.

The approach may seem highly inefficient at first glance, because it needs to store the entire state with each block, but in reality efficiency should be comparable to that of Bitcoin. The reason is that the state is stored in the tree structure, and after every block only a small part of the tree needs to be changed. Thus, in general, between two adjacent blocks the vast majority of the tree should be the same, and therefore the data can be stored once and referenced twice.

of the tree should be the same, and therefore the data can be stored once and referenced twice using pointers (ie. hashes of subtrees). A special kind of tree known as a "Patricia tree" is used to accomplish this, including a modification to the Merkle tree concept that allows for nodes to be inserted and deleted, and not just changed, efficiently. Additionally, because all of the state information is part of the last block, there is no need to store the entire blockchain history - a strategy which, if it could be applied to Bitcoin, can be calculated to provide 10-20x savings in space.

Applications

In general, there are three types of applications on top of Ethereum. The first category is financial applications, providing users with more powerful ways of managing and entering into contracts using their money. This includes sub-currencies, financial derivatives, hedging contracts, savings wallets, wills, and ultimately even some classes of full-scale employment contracts. The second category is semi-financial applications, where money is involved but there is also a heavy non-monetary side to what is being done; a perfect example is self-enforcing bounties for solutions to computational problems. Finally, there are applications such as online voting and decentralized governance that are not financial at all.

Token Systems

On-blockchain token systems have many applications ranging from sub-currencies representing assets such as USD or gold to company stocks, individual tokens representing smart property, secure unforgeable coupons, and even token systems with no ties to conventional value at all, used as point systems for incentivization. Token systems are surprisingly easy to implement in Ethereum. The key point to understand is that all a currency, or token system, fundamentally is is a database with one operation: subtract X units from A and give X units to B, with the proviso that (1) X had at least X units before the transaction and (2) the transaction is approved by A. All that it takes to implement a token system is to implement this logic into a contract.

The basic code for implementing a token system in Serpent looks as follows:

```
from = msg.sender
to = msg.data[0]
value = msg.data[1]

if contract.storage[from] >= value:
    contract.storage[from] = contract.storage[from] - value
    contract.storage[to] = contract.storage[to] + value
```

This is essentially a literal implementation of the "banking system" state transition function described further above in this document. A few extra lines of code need to be added to provide for the initial step of distributing the currency units in the first place and a few other edge cases, and ideally a function would be added to let other contracts query for the balance of an address. But that's all there is to it. Theoretically, Ethereum-based token systems acting as sub-currencies can potentially include another important feature that on-chain Bitcoin-based meta-currencies lack: the ability to pay transaction fees directly in that currency. The way this would be implemented is that the contract would maintain an ether balance with which it would refund ether used to pay fees to the sender, and it would refill this balance by collecting the internal currency units that it takes in fees and reselling them in a constant running auction. Users would thus need to "activate" their accounts with ether, but once the ether is there it would be reusable because the contract would refund it each time.

Financial derivatives and Stable-Value Currencies

Financial derivatives are the most common application of a "smart contract", and one of the simplest to implement in code. The main challenge in implementing financial contracts is that the majority of them require reference to an external price ticker; for example, a very desirable application is a smart contract that hedges against the volatility of ether (or another cryptocurrency) with respect to the US dollar, but doing this requires the contract to know what the value of ETH/USD is. The simplest way to do this is through a "data feed" contract maintained by a specific party (eg. NASDAQ) designed so that that party has the ability to update the contract as needed, and providing an interface that allows other contracts to send a

message to that contract and get back a response that provides the price.

Given that critical ingredient, the hedging contract would look as follows:

1. Wait for party A to input 1000 ether.
2. Wait for party B to input 1000 ether.
3. Record the USD value of 1000 ether, calculated by querying the data feed contract, in storage, say this is \$x.
4. After 30 days, allow A or B to "reactivate" the contract in order to send \$x worth of ether (calculated by querying the data feed contract again to get the new price) to A and the rest to B.

Such a contract would have significant potential in crypto-commerce. One of the main problems cited about cryptocurrency is the fact that it's volatile; although many users and merchants may want the security and convenience of dealing with cryptographic assets, they many not wish to face that prospect of losing 23% of the value of their funds in a single day. Up until now, the most commonly proposed solution has been issuer-backed assets; the idea is that an issuer creates a sub-currency in which they have the right to issue and revoke units, and provide one unit of the currency to anyone who provides them (offline) with one unit of a specified underlying asset (eg. gold, USD). The issuer then promises to provide one unit of the underlying asset to anyone who sends back one unit of the crypto-asset. This mechanism allows any non-cryptographic asset to be "uplifted" into a cryptographic asset, provided that the issuer can be trusted.

In practice, however, issuers are not always trustworthy, and in some cases the banking infrastructure is too weak, or too hostile, for such services to exist. Financial derivatives provide an alternative. Here, instead of a single issuer providing the funds to back up an asset, a decentralized market of speculators, betting that the price of a cryptographic reference asset (eg. ETH) will go up, plays that role. Unlike issuers, speculators have no option to default on their side of the bargain because the hedging contract holds their funds in escrow. Note that this approach is not fully decentralized, because a trusted source is still needed to provide the price ticker, although arguably even still this is a massive improvement in terms of reducing infrastructure requirements (unlike being an issuer, issuing a price feed requires no licenses and can likely be categorized as free speech) and reducing the potential for fraud.

Identity and Reputation Systems

The earliest alternative cryptocurrency of all, [Namecoin](#), attempted to use a Bitcoin-like blockchain to provide a name registration system, where users can register their names in a public database alongside other data. The major cited use case is for a [DNS](#) system, mapping domain names like "bitcoin.org" (or, in Namecoin's case, "bitcoin.bit") to an IP address. Other use cases include email authentication and potentially more advanced reputation systems. Here is the basic contract to provide a Namecoin-like name registration system on Ethereum:

```
if !contract.storage[tx.data[0]]:
    contract.storage[tx.data[0]] = tx.data[1]
```

The contract is very simple; all it is is a database inside the Ethereum network that can be added to, but not modified or removed from. Anyone can register a name with some value, and that registration then sticks forever. A more sophisticated name registration contract will also have a "function clause" allowing other contracts to query it, as well as a mechanism for the "owner" (ie. the first registerer) of a name to change the data or transfer ownership. One can even add reputation and web-of-trust functionality on top.

Decentralized File Storage

Over the past few years, there have emerged a number of popular online file storage startups, the most prominent being Dropbox, seeking to allow users to upload a backup of their hard drive and have the service store the backup and allow the user to access it in exchange for a monthly fee. However, at this point the file storage market is at times relatively inefficient; a cursory look at various [existing solutions](#) shows that, particularly at the "uncanny valley" 20-200 GB level at which neither free quotas nor enterprise-level discounts kick in, monthly prices for mainstream file storage costs are such that you are paying for more than the cost of the entire hard drive in a

single month. Ethereum contracts can allow for the development of a decentralized file storage ecosystem, where individual users can earn small quantities of money by renting out their own hard drives and unused space can be used to further drive down the costs of file storage.

The key underpinning piece of such a device would be what we have termed the "decentralized Dropbox contract". This contract works as follows. First, one splits the desired data up into blocks, encrypting each block for privacy, and builds a Merkle tree out of it. One then makes a contract with the rule that, every N blocks, the contract would pick a random index in the Merkle tree (using the previous block hash, accessible from contract code, as a source of randomness), and give X ether to the first entity to supply a transaction with a simplified payment verification-like proof of ownership of the block at that particular index in the tree. When a user wants to re-download their file, they can use a micropayment channel protocol (eg. pay 1 szabo per 32 kilobytes) to recover the file; the most fee-efficient approach is for the payer not to publish the transaction until the end, instead replacing the transaction with a slightly more lucrative one with the same nonce after every 32 kilobytes.

An important feature of the protocol is that, although it may seem like one is trusting many random nodes not to decide to forget the file, one can reduce that risk down to near-zero by splitting the file into many pieces via secret sharing, and watching the contracts to see each piece is still in some node's possession. If a contract is still paying out money, that provides a cryptographic proof that someone out there is still storing the file.

Decentralized Autonomous Organizations

The general concept of a "decentralized autonomous organization" is that of a virtual entity that has a certain set of members or shareholders which, perhaps with a 67% majority, have the right to spend the entity's funds and modify its code. The members would collectively decide on how the organization should allocate its funds. Methods for allocating a DAO's funds could range from bounties, salaries to even more exotic mechanisms such as an internal currency to reward work. This essentially replicates the legal trappings of a traditional company or nonprofit but using only cryptographic blockchain technology for enforcement. So far much of the talk around DAOs has been around the "capitalist" model of a "decentralized autonomous corporation" (DAC) with dividend-receiving shareholders and tradable shares; an alternative, perhaps described as a "decentralized autonomous community", would have all members have an equal share in the decision making and require 67% of existing members to agree to add or remove a member. The requirement that one person can only have one membership would then need to be enforced collectively by the group.

A general outline for how to code a DAO is as follows. The simplest design is simply a piece of self-modifying code that changes if two thirds of members agree on a change. Although code is theoretically immutable, one can easily get around this and have de-facto mutability by having chunks of the code in separate contracts, and having the address of which contracts to call stored in the modifiable storage. In a simple implementation of such a DAO contract, there would be three transaction types, distinguished by the data provided in the transaction:

- $[0, i, k, v]$ to register a proposal with index i to change the address at storage index k to value v
- $[0, i]$ to register a vote in favor of proposal i
- $[2, i]$ to finalize proposal i if enough votes have been made

The contract would then have clauses for each of these. It would maintain a record of all open storage changes, along with a list of who voted for them. It would also have a list of all members. When any storage change gets to two thirds of members voting for it, a finalizing transaction could execute the change. A more sophisticated skeleton would also have built-in voting ability for features like sending a transaction, adding members and removing members, and may even provide for [Liquid Democracy](#)-style vote delegation (ie. anyone can assign someone to vote for them, and assignment is transitive so if A assigns B and B assigns C then C determines A's vote). This design would allow the DAO to grow organically as a decentralized community, allowing people to eventually delegate the task of filtering out who is a member to specialists, although unlike in the "current system" specialists can easily pop in and out of existence over time as individual community members change their alignments.

An alternative model is for a decentralized corporation, where any account can have zero or

more shares, and two thirds of the shares are required to make a decision. A complete skeleton would involve asset management functionality, the ability to make an offer to buy or sell shares, and the ability to accept offers (preferably with an order-matching mechanism inside the contract). Delegation would also exist Liquid Democracy-style, generalizing the concept of a "board of directors".

Further Applications

1. Savings wallets. Suppose that Alice wants to keep her funds safe, but is worried that she will lose or someone will hack her private key. She puts ether into a contract with Bob, a bank, as follows:

- Alice alone can withdraw a maximum of 1% of the funds per day.
- Bob alone can withdraw a maximum of 1% of the funds per day, but Alice has the ability to make a transaction with her key shutting off this ability.
- Alice and Bob together can withdraw anything.

Normally, 1% per day is enough for Alice, and if Alice wants to withdraw more she can contact Bob for help. If Alice's key gets hacked, she runs to Bob to move the funds to a new contract. If she loses her key, Bob will get the funds out eventually. If Bob turns out to be malicious, then she can turn off his ability to withdraw.

2. Crop insurance. One can easily make a financial derivatives contract but using a data feed of the weather instead of any price index. If a farmer in Iowa purchases a derivative that pays out inversely based on the precipitation in Iowa, then if there is a drought, the farmer will automatically receive money and if there is enough rain the farmer will be happy because their crops would do well.

3. A decentralized data feed. For financial contracts for difference, it may actually be possible to decentralize the data feed via a protocol called "[SchellingCoin](#)". SchellingCoin basically works as follows: N parties all put into the system the value of a given datum (eg. the ETH/USD price), the values are sorted, and everyone between the 25th and 75th percentile gets one token as a reward. Everyone has the incentive to provide the answer that everyone else will provide, and the only value that a large number of players can realistically agree on is the obvious default: the truth. This creates a decentralized protocol that can theoretically provide any number of values, including the ETH/USD price, the temperature in Berlin or even the result of a particular hard computation.

4. Smart multisignature escrow. Bitcoin allows multisignature transaction contracts where, for example, three out of a given five keys can spend the funds. Ethereum allows for more granularity; for example, four out of five can spend everything, three out of five can spend up to 10% per day, and two out of five can spend up to 0.5% per day. Additionally, Ethereum multisig is asynchronous - two parties can register their signatures on the blockchain at different times and the last signature will automatically send the transaction.

5. Cloud computing. The EVM technology can also be used to create a verifiable computing environment, allowing users to ask others to carry out computations and then optionally ask for proofs that computations at certain randomly selected checkpoints were done correctly. This allows for the creation of a cloud computing market where any user can participate with their desktop, laptop or specialized server, and spot-checking together with security deposits can be used to ensure that the system is trustworthy (ie. nodes cannot profitably cheat). Although such a system may not be suitable for all tasks; tasks that require a high level of inter-process communication, for example, cannot easily be done on a large cloud of nodes. Other tasks, however, are much easier to parallelize; projects like SETI@home, folding@home and genetic algorithms can easily be implemented on top of such a platform.

6. Peer-to-peer gambling. Any number of peer-to-peer gambling protocols, such as Frank Stajano and Richard Clayton's [Cyberdice](#), can be implemented on the Ethereum blockchain. The simplest gambling protocol is actually simply a contract for difference on the next block hash, and more advanced protocols can be built up from there, creating gambling services with near-zero fees that have no ability to cheat.

7. Prediction markets. Provided an oracle or SchellingCoin, prediction markets are also easy to implement, and prediction markets together with SchellingCoin may prove to be the first

mainstream application of [futarchy](#) as a governance protocol for decentralized organizations.

8. **On-chain decentralized marketplaces**, using the identity and reputation system as a base.

Miscellanea And Concerns

Modified GHOST Implementation

The "Greedy Heaviest Observed Subtree" (GHOST) protocol is an innovation first introduced by Yonatan Sompolinsky and Aviv Zohar in [December 2013](#). The motivation behind GHOST is that blockchains with fast confirmation times currently suffer from reduced security due to a high stale rate - because blocks take a certain time to propagate through the network, if miner A mines a block and then miner B happens to mine another block before miner A's block propagates to B, miner B's block will end up wasted and will not contribute to network security. Furthermore, there is a centralization issue: if miner A is a mining pool with 30% hashpower and B has 10% hashpower, A will have a risk of producing a stale block 70% of the time (since the other 30% of the time A produced the last block and so will get mining data immediately) whereas B will have a risk of producing a stale block 90% of the time. Thus, if the block interval is short enough for the stale rate to be high, A will be substantially more efficient simply by virtue of its size. With these two effects combined, blockchains which produce blocks quickly are very likely to lead to one mining pool having a large enough percentage of the network hashpower to have de facto control over the mining process.

As described by Sompolinsky and Zohar, GHOST solves the first issue of network security loss by including stale blocks in the calculation of which chain is the "longest"; that is to say, not just the parent and further ancestors of a block, but also the stale descendants of the block's ancestor (in Ethereum jargon, "uncles") are added to the calculation of which block has the largest total proof of work backing it. To solve the second issue of centralization bias, we go beyond the protocol described by Sompolinsky and Zohar, and also provide block rewards to stales: a stale block receives 87.5% of its base reward, and the nephew that includes the stale block receives the remaining 12.5%. Transaction fees, however, are not awarded to uncles.

Ethereum implements a simplified version of GHOST which only goes down one level. Specifically, a stale block can only be included as an uncle by the direct child of one of its direct siblings, and not any block with a more distant relation. This was done for several reasons. First, unlimited GHOST would include too many complications into the calculation of which uncles for a given block are valid. Second, unlimited GHOST with compensation as used in Ethereum removes the incentive for a miner to mine on the main chain and not the chain of a public attacker. Finally, calculations show that single-level GHOST has over 80% of the benefit of unlimited GHOST, and provides a stale rate comparable to the 2.5 minute Litecoin even with a 40-second block time. However, we will be conservative and still retain a Primecoin-like 60-second block time because individual blocks may take a longer time to verify.

Fees

Because every transaction published into the blockchain imposes on the network the cost of needing to download and verify it, there is a need for some regulatory mechanism, typically involving transaction fees, to prevent abuse. The default approach, used in Bitcoin, is to have purely voluntary fees, relying on miners to act as the gatekeepers and set dynamic minimums. This approach has been received very favorably in the Bitcoin community particularly because it is "market-based", allowing supply and demand between miners and transaction senders determine the price. The problem with this line of reasoning is, however, that transaction processing is not a market; although it is intuitively attractive to construe transaction processing as a service that the miner is offering to the sender, in reality every transaction that a miner includes will need to be processed by every node in the network, so the vast majority of the cost of transaction processing is borne by third parties and not the miner that is making the decision of whether or not to include it. Hence, tragedy-of-the-commons problems are very likely to occur.

However, as it turns out this flaw in the market-based mechanism, when given a particular inaccurate simplifying assumption, magically cancels itself out. The argument is as follows. Suppose that:

1. A transaction leads to k operations, offering the reward kR to any miner that includes it where R is set by the sender and k and R are (roughly) visible to the miner beforehand.
2. An operation has a processing cost of C to any node (ie. all nodes have equal efficiency)
3. There are N mining nodes, each with exactly equal processing power (ie. $1/N$ of total)
4. No non-mining full nodes exist.

A miner would be willing to process a transaction if the expected reward is greater than the cost. Thus, the expected reward is kR/N since the miner has a $1/N$ chance of processing the next block, and the processing cost for the miner is simply C . Hence, miners will include transactions where $kR/N > C$, or $R > NC$. Note that R is the per-operation fee provided by the sender, and is thus a lower bound on the benefit that the sender derives from the transaction, and NC is the cost to the entire network together of processing an operation. Hence, miners have the incentive to include only those transactions for which the total utilitarian benefit exceeds the cost.

However, there are several important deviations from those assumptions in reality:

1. The miner does pay a higher cost to process the transaction than the other verifying nodes, since the extra verification time delays block propagation and thus increases the chance the block will become a stale.
2. There do exist nonmining full nodes.
3. The mining power distribution may end up radically inegalitarian in practice.
4. Speculators, political enemies and crazies whose utility function includes causing harm to the network do exist, and they can cleverly set up contracts where their cost is much lower than the cost paid by other verifying nodes.

(1) provides a tendency for the miner to include fewer transactions, and (2) increases NC ; hence, these two effects at least partially cancel each other out. (3) and (4) are the major issue; to solve them we simply institute a floating cap: no block can have more operations than BLK_LIMIT_FACTOR times the long-term exponential moving average. Specifically:

```
blk.oplimit = floor((blk.parent.oplimit * (EMA_FACTOR - 1) + floor(parent.opcount * BLK_LIMIT_FA
```

BLK_LIMIT_FACTOR and EMA_FACTOR are constants that will be set to 65536 and 1.5 for the time being, but will likely be changed after further analysis.

Computation And Turing-Completeness

An important note is that the Ethereum virtual machine is Turing-complete; this means that EVM code can encode any computation that can be conceivably carried out, including infinite loops. EVM code allows looping in two ways. First, there is a `JUMP` instruction that allows the program to jump back to a previous spot in the code, and a `JUMPI` instruction to do conditional jumping, allowing for statements like `while x < 27: x = x * 2`. Second, contracts can call other contracts, potentially allowing for looping through recursion. This naturally leads to a problem: can malicious users essentially shut miners and full nodes down by forcing them to enter into an infinite loop? The issue arises because of a problem in computer science known as the halting problem: there is no way to tell, in the general case, whether or not a given program will ever halt.

As described in the state transition section, our solution works by requiring a transaction to set a maximum number of computational steps that it is allowed to take, and if execution takes longer computation is reverted but fees are still paid. Messages work in the same way. To show the motivation behind our solution, consider the following examples:

- An attacker creates a contract which runs an infinite loop, and then sends a transaction activating that loop to the miner. The miner will process the transaction, running the infinite loop, and wait for it to run out of gas. Even though the execution runs out of gas and stops halfway through, the transaction is still valid and the miner still claims the fee from the attacker for each computational step.
- An attacker creates a very long infinite loop with the intent of forcing the miner to keep computing for such a long time that by the time computation finishes a few more blocks will have come out and it will not be possible for the miner to include the transaction to claim the fee. However, the attacker will be required to submit a value for `STARTGAS` limiting the

too. However, the attacker will be required to submit a value for `gasLimit`, limiting the number of computational steps that execution can take, so the miner will know ahead of time that the computation will take an excessively large number of steps.

- An attacker sees a contract with code of some form like `send(A,contract.storage[A]); contract.storage[A] = 0`, and sends a transaction with just enough gas to run the first step but not the second (ie. making a withdrawal but not letting the balance go down). The contract author does not need to worry about protecting against such attacks, because if execution stops halfway through the changes get reverted.
- A financial contract works by taking the median of nine proprietary data feeds in order to minimize risk. An attacker takes over one of the data feeds, which is designed to be modifiable via the variable-address-call mechanism described in the section on DAOs, and converts it to run an infinite loop, thereby attempting to force any attempts to claim funds from the financial contract to run out of gas. However, the financial contract can set a gas limit on the message to prevent this problem.

The alternative to Turing-completeness is Turing-incompleteness, where `JUMP` and `JUMPI` do not exist and only one copy of each contract is allowed to exist in the call stack at any given time. With this system, the fee system described and the uncertainties around the effectiveness of our solution might not be necessary, as the cost of executing a contract would be bounded above by its size. Additionally, Turing-incompleteness is not even that big a limitation; out of all the contract examples we have conceived internally, so far only one required a loop, and even that loop could be removed by making 26 repetitions of a one-line piece of code. Given the serious implications of Turing-completeness, and the limited benefit, why not simply have a Turing-incomplete language? In reality, however, Turing-incompleteness is far from a neat solution to the problem. To see why, consider the following contracts:

```
C0: call(C1); call(C1);
C1: call(C2); call(C2);
C2: call(C3); call(C3);
...
C49: call(C50); call(C50);
C50: (run one step of a program and record the change in storage)
```

Now, send a transaction to A. Thus, in 51 transactions, we have a contract that takes up 2^{50} computational steps. Miners could try to detect such logic bombs ahead of time by maintaining a value alongside each contract specifying the maximum number of computational steps that it can take, and calculating this for contracts calling other contracts recursively, but that would require miners to forbid contracts that create other contracts (since the creation and execution of all 26 contracts above could easily be rolled into a single contract). Another problematic point is that the address field of a message is a variable, so in general it may not even be possible to tell which other contracts a given contract will call ahead of time. Hence, all in all, we have a surprising conclusion: Turing-completeness is surprisingly easy to manage, and the lack of Turing-completeness is equally surprisingly difficult to manage unless the exact same controls are in place - but in that case why not just let the protocol be Turing-complete?

Currency And Issuance

The Ethereum network includes its own built-in currency, ether, which serves the dual purpose of providing a primary liquidity layer to allow for efficient exchange between various types of digital assets and, more importantly, of providing a mechanism for paying transaction fees. For convenience and to avoid future argument (see the current mBTC/uBTC/satoshi debate in Bitcoin), the denominations will be pre-labelled:

- 1: wei
- 10^3 : lovelace
- 10^6 : babbage
- 10^9 : shannon
- 10^{12} : szabo
- 10^{15} : finney
- 10^{18} : ether

This should be taken as an expanded version of the concept of "dollars" and "cents" or "BTC" and "satoshi". In the near future, we expect "ether" to be used for ordinary transactions, "finney" for microtransactions and "szabo" and "wei" for technical discussions around fees and protocol

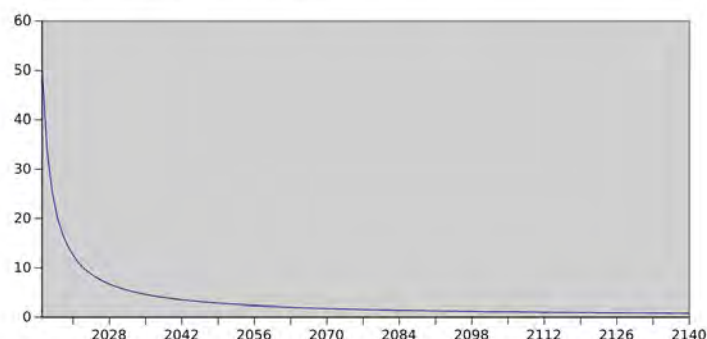
implementation; the remaining denominations may become useful later and should not be included in clients at this point.

The issuance model will be as follows:

- Ether will be released in a currency sale at the price of 1000-2000 ether per BTC, a mechanism intended to fund the Ethereum organization and pay for development that has been used with success by other platforms such as Mastercoin and NXT. Earlier buyers will benefit from larger discounts. The BTC received from the sale will be used entirely to pay salaries and bounties to developers and invested into various for-profit and non-profit projects in the Ethereum and cryptocurrency ecosystem.
- 0.3x the total amount sold will be allocated to the organization to compensate early contributors, pay ETH-denominated expenses before the genesis block and as a long-term reserve. This amount is distributed according to an exponentially decreasing formula; every month, up to 5.6% of the remaining endowment can be distributed among developers and others who participated in project development, and this distribution starts in December.
- 0.3x the total amount sold will be allocated to miners per year forever after that point

Group	At launch	After 1 year	After 5 years
Currency units	1.3X	1.6X	2.8X
Purchasers	76.9%	62.5%	35.7%
Reserve spent pre-sale	5.77%	4.69%	2.68%
Reserve used post-sale	17.3%	14.1%	8.04%
Miners	0%	23.1%	53.6%

Long-Term Supply Growth Rate (percent)



Despite the linear currency issuance, just like with Bitcoin over time the supply growth rate nevertheless tends to zero

The two main choices in the above model are (1) the existence and size of an endowment pool, and (2) the existence of a permanently growing linear supply, as opposed to a capped supply as in Bitcoin. The justification of the endowment pool is as follows. If the endowment pool did not exist, and the linear issuance reduced to 0.231x to provide the same inflation rate, then the total quantity of ether would be 23% less and so each unit would be 30% more valuable. Hence, in the equilibrium 30% more ether would be purchased in the sale, so each unit would once again be exactly as valuable as before. The organization would also then have 1.3x as much BTC, which can be considered to be split into two slices: the original BTC, and the additional 0.3x. Hence, this situation is *exactly equivalent* to the endowment, but with one important difference: the organization holds purely BTC, and so is not incentivized to support the value of the ether unit.

The permanent linear supply growth model reduces the risk of what some see as excessive wealth concentration in Bitcoin, and gives individuals living in present and future eras a fair chance to acquire currency units, while at the same time retaining a strong incentive to obtain and hold ether because the "supply growth rate" as a percentage still tends to zero over time. We also theorize that because coins are always lost over time due to carelessness, death, etc, and coin loss can be modeled as a percentage of the total supply per year, that the total currency supply in circulation will in fact eventually stabilize at a value equal to the annual

issuance divided by the loss rate (eg. at a loss rate of 1%, once the supply reaches $30X$ then $0.3X$ will be mined and $0.3X$ lost every year, creating an equilibrium).

Mining Centralization

The Bitcoin mining algorithm basically works by having miners compute SHA256 on slightly modified versions of the block header millions of times over and over again, until eventually one node comes up with a version whose hash is less than the target (currently around 2^{192}). However, this mining algorithm is vulnerable to two forms of centralization. First, the mining ecosystem has come to be dominated by ASICs (application-specific integrated circuits), computer chips designed for, and therefore thousands of times more efficient at, the specific task of Bitcoin mining. This means that Bitcoin mining is no longer a highly decentralized and egalitarian pursuit, requiring millions of dollars of capital to effectively participate in. Second, most Bitcoin miners do not actually perform block validation locally; instead, they rely on a centralized mining pool to provide the block headers. This problem is arguably worse: as of the time of this writing, the top three mining pools indirectly control roughly 50% of processing power in the Bitcoin network, although this is mitigated by the fact that miners can switch to other mining pools if a pool or coalition attempts a 51% attack.

The current intent at Ethereum is to use a mining algorithm where miners are required to fetch random data from the state, compute some randomly selected transactions from the last N blocks in the blockchain, and return the hash of the result. This has two important benefits. First, Ethereum contracts can include any kind of computation, so an Ethereum ASIC would essentially be an ASIC for general computation - ie. a better CPU. Second, mining requires access to the entire blockchain, forcing miners to store the entire blockchain and at least be capable of verifying every transaction. This removes the need for centralized mining pools; although mining pools can still serve the legitimate role of evening out the randomness of reward distribution, this function can be served equally well by peer-to-peer pools with no central control.

This model is untested, and there may be difficulties along the way in avoiding certain clever optimizations when using contract execution as a mining algorithm. However, one notably interesting feature of this algorithm is that it allows anyone to "poison the well", by introducing a large number of contracts into the blockchain specifically designed to stymie certain ASICs. The economic incentives exist for ASIC manufacturers to use such a trick to attack each other. Thus, the solution that we are developing is ultimately an adaptive economic human solution rather than purely a technical one.

Scalability

One common concern about Ethereum is the issue of scalability. Like Bitcoin, Ethereum suffers from the flaw that every transaction needs to be processed by every node in the network. With Bitcoin, the size of the current blockchain rests at about 15 GB, growing by about 1 MB per hour. If the Bitcoin network were to process Visa's 2000 transactions per second, it would grow by 1 MB per three seconds (1 GB per hour, 8 TB per year). Ethereum is likely to suffer a similar growth pattern, worsened by the fact that there will be many applications on top of the Ethereum blockchain instead of just a currency as is the case with Bitcoin, but ameliorated by the fact that Ethereum full nodes need to store just the state instead of the entire blockchain history.

The problem with such a large blockchain size is centralization risk. If the blockchain size increases to, say, 100 TB, then the likely scenario would be that only a very small number of large businesses would run full nodes, with all regular users using light SPV nodes. In such a situation, there arises the potential concern that the full nodes could band together and all agree to cheat in some profitable fashion (eg. change the block reward, give themselves BTC). Light nodes would have no way of detecting this immediately. Of course, at least one honest full node would likely exist, and after a few hours information about the fraud would trickle out through channels like Reddit, but at that point it would be too late: it would be up to the ordinary users to organize an effort to blacklist the given blocks, a massive and likely infeasible coordination problem on a similar scale as that of pulling off a successful 51% attack. In the case of Bitcoin, this is currently a problem, but there exists a blockchain modification [suggested by Peter Todd](#) which will alleviate this issue.

In the near term, Ethereum will use two additional strategies to cope with this problem. First.

because of the blockchain-based mining algorithms, at least every miner will be forced to be a full node, creating a lower bound on the number of full nodes. Second and more importantly, however, we will include an intermediate state tree root in the blockchain after processing each transaction. Even if block validation is centralized, as long as one honest verifying node exists, the centralization problem can be circumvented via a verification protocol. If a miner publishes an invalid block, that block must either be badly formatted, or the state $s[n]$ is incorrect. Since $s[0]$ is known to be correct, there must be some first state $s[i]$ that is incorrect where $s[i-1]$ is correct. The verifying node would provide the index i , along with a "proof of invalidity" consisting of the subset of Patricia tree nodes needing to process $\text{APPLY}(s[i-1], \text{TX}[i]) \rightarrow s[i]$. Nodes would be able to use those nodes to run that part of the computation, and see that the $s[i]$ generated does not match the $s[i]$ provided.

Another, more sophisticated, attack would involve the malicious miners publishing incomplete blocks, so the full information does not even exist to determine whether or not blocks are valid. The solution to this is a challenge-response protocol: verification nodes issue "challenges" in the form of target transaction indices, and upon receiving a node a light node treats the block as untrusted until another node, whether the miner or another verifier, provides a subset of Patricia nodes as a proof of validity.

Conclusion

The Ethereum protocol was originally conceived as an upgraded version of a cryptocurrency, providing advanced features such as on-blockchain escrow, withdrawal limits, financial contracts, gambling markets and the like via a highly generalized programming language. The Ethereum protocol would not "support" any of the applications directly, but the existence of a Turing-complete programming language means that arbitrary contracts can theoretically be created for any transaction type or application. What is more interesting about Ethereum, however, is that the Ethereum protocol moves far beyond just currency. Protocols around decentralized file storage, decentralized computation and decentralized prediction markets, among dozens of other such concepts, have the potential to substantially increase the efficiency of the computational industry, and provide a massive boost to other peer-to-peer protocols by adding for the first time an economic layer. Finally, there is also a substantial array of applications that have nothing to do with money at all.

The concept of an arbitrary state transition function as implemented by the Ethereum protocol provides for a platform with unique potential; rather than being a closed-ended, single-purpose protocol intended for a specific array of applications in data storage, gambling or finance, Ethereum is open-ended by design, and we believe that it is extremely well-suited to serving as a foundational layer for a very large number of both financial and non-financial protocols in the years to come.

Notes and Further Reading

Notes

1. A sophisticated reader may notice that in fact a Bitcoin address is the hash of the elliptic curve public key, and not the public key itself. However, it is in fact perfectly legitimate cryptographic terminology to refer to the pubkey hash as a public key itself. This is because Bitcoin's cryptography can be considered to be a custom digital signature algorithm, where the public key consists of the hash of the ECC pubkey, the signature consists of the ECC pubkey concatenated with the ECC signature, and the verification algorithm involves checking the ECC pubkey in the signature against the ECC pubkey hash provided as a public key and then verifying the ECC signature against the ECC pubkey.
2. Technically, the median of the 11 previous blocks.
3. Internally, 2 and "CHARLIE" are both numbers, with the latter being in big-endian base 256 representation. Numbers can be at least 0 and at most $2^{256}-1$.

Further Reading

1. Intrinsic value: <http://bitcoinmagazine.com/8640/an-exploration-of-intrinsic-value-what-it-is->

[why-bitcoin-doesnt-have-it-and-why-bitcoin-does-have-it/](#)

2. Smart property: https://en.bitcoin.it/wiki/Smart_Property
3. Smart contracts: <https://en.bitcoin.it/wiki/Contracts>
4. B-money: <http://www.weidai.com/bmoney.txt>
5. Reusable proofs of work: <http://www.finney.org/~hal/rpow/>
6. Secure property titles with owner authority: <http://szabo.best.vwh.net/securetitle.html>
7. Bitcoin whitepaper: <http://bitcoin.org/bitcoin.pdf>
8. Namecoin: <https://namecoin.org/>
9. Zooko's triangle: http://en.wikipedia.org/wiki/Zooko's_triangle
10. Colored coins whitepaper:
https://docs.google.com/a/buterin.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0IlzrTLuoWu2z1BE/edit
11. Mastercoin whitepaper: <https://github.com/mastercoin-MSC/spec>
12. Decentralized autonomous corporations, Bitcoin Magazine:
<http://bitcoinmagazine.com/7050/bootstrapping-a-decentralized-autonomous-corporation-part-i/>
13. Simplified payment verification:
<https://en.bitcoin.it/wiki/Scalability#Simplifiedpaymentverification>
14. Merkle trees: http://en.wikipedia.org/wiki/Merkle_tree
15. Patricia trees: http://en.wikipedia.org/wiki/Patricia_tree
16. GHOST: http://www.cs.huji.ac.il/~avivz/pubs/13/btc_scalability_full.pdf
17. StorJ and Autonomous Agents, Jeff Garzik: <http://garzikrants.blogspot.ca/2013/01/storj-and-bitcoin-autonomous-agents.html>
18. Mike Hearn on Smart Property at Turing Festival: <http://www.youtube.com/watch?v=Pu4PAMFPo5Y>
19. Ethereum RLP: <https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-RLP>
20. Ethereum Merkle Patricia trees: <https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-Patricia-Tree>
21. Peter Todd on Merkle sum trees:
<http://sourceforge.net/p/bitcoin/mailman/message/31709140/>



Ethereum: A Next-Generation Generalized Smart Contract and Decentralized Application Platform

In the last few months, there has been a great amount of interest into the area of using Bitcoin-like blockchains, the mechanism that allows for the entire world to agree on the state of a public ownership database, for more than just money. Commonly cited applications include "colored coins", the idea of using on-blockchain digital assets to represent custom currencies and financial instruments, "smart property", physical objects such as cars which track a colored coin on a blockchain to determine their present legitimate owner, as well as more advanced applications such as decentralized exchange, financial derivatives, peer-to-peer gambling and on-blockchain identity and reputation systems. Perhaps the most ambitious of all is the concept of "autonomous agents" or "[decentralized autonomous corporations](#)" - autonomous entities that operate on the blockchain without any central control whatsoever, eschewing all dependence on legal contracts and organizational bylaws in favor of having resources and funds autonomously managed by a self-enforcing smart contract on a cryptographic blockchain.

However, most of these applications are difficult to implement today, simply because the scripting systems of Bitcoin, and even proto-cryptocurrency-2.0 alternatives such as the Bitcoin-based colored coins protocol and so-called "metacoins", are far too limited to allow the kind of arbitrarily complex computation that DACs require. What this project intends to do is take the innovations that such protocols bring, and generalize them - create a fully-fledged, Turing-complete (but heavily fee-regulated) cryptographic ledger that allows participants to encode arbitrarily complex contracts, autonomous agents and relationships that will be mediated entirely by the blockchain. Rather than being limited to a specific set of transaction types, users will be able to use Ethereum as a sort of "Minecraft of crypto-finance" - that is to say, one will be able to implement any feature that one desires simply by coding it in the protocol's internal scripting language. Custom currencies, financial derivatives, identity systems and decentralized organizations will all be easy to do, and it will also be possible to construct transaction types that even the Ethereum developers did not imagine. Altogether, we believe that this design is a solid step toward the realization of "cryptocurrency 2.0"; we hope that Ethereum will be to cryptocurrency what Web 2.0 was to the World Wide Web circa 1995.

Why A New Protocol

When one wants to create a new application, especially one in so delicate an area as cryptography or cryptocurrency, the immediate, and correct, first instinct is to use existing protocols as much as possible. There is no need to create a new currency, or even a new protocol, when the problem can be solved entirely by using existing technologies. Indeed, the puzzle of attempting to solve the problems of [smart property](#), [smart contracts](#) and [decentralized autonomous corporations](#) on top of Bitcoin is how our interest in cryptocurrency 2.0 technologies originally started. Over the course of our research, however, it became evident that while the Bitcoin protocol is more than adequate for currency, basic multisignature escrow and certain simple versions of smart contracts, there are fundamental limitations that make it non-viable for anything beyond a certain very limited scope of features.

Colored Coins

The first attempt to implement a system for managing smart property and custom currencies and assets on top of a blockchain was built as a sort of overlay protocol on top of Bitcoin itself, with many advocates making a comparison to the way that HTTP serves as a layer on top of TCP in the [internet protocol stack](#). The [colored coins protocol](#) is roughly defined as follows:

1. A colored coin issuer determines that a given transaction output $H:i$ (H being the transaction hash and i the output index) represents a certain asset, and publishes a "color definition" specifying this transaction output alongside what it represents (eg. 1 satoshi from $H:i = 1$ ounce of gold redeemable at [amagimetals.com](#)).
2. Others "install" the color definition file in their colored coin clients.
3. When the color is first released, output $H:i$ is the only transaction output to have that color.
4. If a transaction spends inputs with color X , then its outputs will also have color X . For example, if the owner of $H:i$ immediately makes a transaction to split that output among five addresses, then those transaction outputs will all also have color X . If a transaction has inputs of different colors, then a "color transfer rule" or "color kernel" determines which colors which outputs are (eg. a very naive implementation may say that output 0 has the same color as input 0, output 1 the same color as input 1, etc).
5. When a colored coin client notices that it received a new transaction output, it uses a back-tracing algorithm based on the color kernel to determine the color of the output. Because the rule is deterministic, all clients will agree on what color (or colors) each output has.

However, the protocol has several fundamental flaws:

1. **Difficulty of simplified payment verification** - Bitcoin's [Merkle tree](#) construction allows for a protocol known as "[simplified payment verification](#)", where a client that does not download the full blockchain can quickly determine the validity of a transaction output by asking other nodes to provide a series of hashes starting from the transaction hash, going up the Merkle tree and culminating at the root hash in the block header. The client will still need to download the block headers to be secure, but the amount of data bandwidth and verification time required drops by a factor of nearly a thousand. With colored coins, this is much harder. The reason is that one cannot determine the color of a transaction output simply by looking up the Merkle tree; rather, one needs to employ the backward scanning algorithm, fetching potentially hundreds of transactions and requesting a Merkle tree validity proof of each one, before a client can be fully satisfied that a transaction has a certain color. After over a year of investigation, including help from ourselves, no solution has been found to this problem.
2. **Incompatibility with scripting** - as mentioned above, Bitcoin does have a moderately flexible scripting system, for example allowing users to sign transactions of the form "I release this transaction output to anyone who is willing to pay to me 1 BTC". Other examples include [assurance contracts](#), [efficient micropayments](#) and on-blockchain auctions. However, this system is inherently not color-aware; that is to say, one cannot make a transaction of the form "I release this transaction output to anyone who is willing to pay me one gold coin defined by the genesis $H:i$ ", because the scripting language has no idea that there even are different colors. One major consequence of this is that, while trust-free swapping of two different colored coins is possible, a full decentralized exchange is not since there is no way to place an order to buy or sell that is enforceable.
3. **Same limitations as Bitcoin** - theoretically, on-blockchain protocols can support advanced derivatives, bets and various kinds of conditional transfers that will be described in more detail later in this paper. Colored coins inherits the limitations of Bitcoin in terms of the impossibility of many such arrangements.



Simplified payment verification in Bitcoin. Any attempt to change any part of the Merkle tree will eventually lead to an inconsistency somewhere up the chain.

Metacoins

Another concept, once again in the spirit of sitting on top of Bitcoin much like HTTP over TCP, is that of "metacoins". The concept of a metacoin is simple: the metacoin protocol

provides for a way of encoding metacoin transaction data into the outputs of a Bitcoin transaction, and a metacoin node works by processing all Bitcoin transactions and evaluating Bitcoin transactions that are valid metacoin transactions in order to determine the current balance sheet at any given time. For example, a simple metacoin protocol might require a transaction to have four outputs: MARKER, FROM, TO and VALUE. MARKER would be a specific marker address to identify a transaction as a metacoin transaction, FROM would be the address that coins are sent from, TO would be the address that coins are sent to and VALUE would be an address encoding the amount sent. Because the Bitcoin protocol is not metacoin-aware, and thus will not reject invalid metacoin transactions, the metacoin protocol must treat all transactions with the first output going to MARKER as valid and react accordingly. For example, an implementation of the relevant part of this metacoin protocol might look like this:

```
if tx.output[0] != MARKER:
    break
else if balance[tx.output[1]] < decode_value(tx.output[3]):
    break
else if not tx.hasSignature(tx.output[1]):
    break
else:
    balance[tx.output[1]] -= decode_value(tx.output[3]);
    balance[tx.output[2]] += decode_value(tx.output[3]);
```

The advantage of a metacoin protocol is that it can allow for more advanced transaction types, including custom currencies, decentralized exchange, derivatives, etc, that are impossible on top of Bitcoin itself. However, metacoins on top of Bitcoin have one major flaw: simplified payment verification, already difficult with colored coins, is outright impossible on a metacoin. The reason is that while one can use SPV to determine that there is a transaction sending 30 metacoins to address X, that by itself does not mean that address X has 30 metacoins; what if the sender of the transaction did not have 30 metacoins to start with and so the transaction is invalid? Finding out any part of the current state essentially requires scanning through all transactions going back to the metacoin's original launch to figure out which transactions are valid and which ones are not. This makes it impossible to have a truly secure client without downloading the entire 12 GB Bitcoin blockchain.

Ethereum solves both of these issues by being hosted on its own blockchain, and by storing a distinct "state tree" representing the current balance of each address and a "transaction list" representing the transactions between the current block and the previous block in each block. Ethereum contracts are allowed to have an arbitrarily large memory, and the Turing-complete scripting language makes it possible to encode an entire currency inside of a single contract. The intention of Ethereum is not to replace colored coins and metacoins by offering superior features; rather, Ethereum intends to serve as a superior foundational layer offering a uniquely powerful scripting system on top of which arbitrarily advanced contracts, currencies and other decentralized applications can be built. If existing colored coins and metacoin projects were to move onto Ethereum, they would gain the benefits of Ethereum's simplified payment verification, compatibility with financial derivatives and decentralized exchange, and the ability to work together on a single network. With Ethereum, someone with an idea for a new contract or transaction type that might drastically improve the state of what can be done with cryptocurrency would not need to start their own coin; they could simply implement their idea in Ethereum script code.

Philosophy

The design behind Ethereum is intended to follow the following principles:

1. **Simplicity** - the Ethereum protocol should be as simple as possible, even at the cost of some data storage inefficiency or time inefficiency. An average programmer should ideally be able to follow and implement the entire specification, so as to eventually help minimize the influence that any specific individual or elite group can have on the protocol and furthering the vision of Ethereum as a protocol that is open to all. Optimizations which add complexity should not be included unless they provide very substantial benefit.
2. **Universality** - it is a fundamental part of Ethereum's design philosophy that Ethereum does not have "features". Instead, Ethereum provides an internal Turing-complete scripting language which you can use to construct any smart contract or transaction type that can be mathematically defined. Want to invent your own financial derivative? With Ethereum, you can. Want to make your own currency? Set it up as an Ethereum contract. Want to set up a full-scale Daemon or Skynet? You may need to have a few thousand interlocking contracts, and be sure to feed them generously, to do that, but nothing is stopping you.
3. **Modularity** - different parts of the Ethereum protocol should be designed to be as modular and separable as possible. Over the course of development, it should be easy to make a small protocol modification in one place and have the application stack continue to function without any further modification. Innovations such as [Dagger](#), [Patricia trees](#) and [RLP](#) should be implemented as separate libraries and made to be feature-complete even if Ethereum does not require certain features so as to make them usable in other protocols as well. Ethereum development should be maximally done so as to benefit the entire cryptocurrency ecosystem, not just itself.
4. **Non-discrimination** - the protocol should not attempt to actively restrict or prevent specific categories of usage, and all regulatory mechanisms in the protocol should be designed to directly regulate the harm, not attempt to oppose specific undesirable applications. You can even run an infinite loop script on top of Ethereum for as long as you are willing to keep paying the per-computational-step transaction fee for it.

Basic Building Blocks

At its core, Ethereum starts off as a fairly regular proof-of-work mined cryptocurrency without many extra complications; in fact, Ethereum is in many ways simpler than the Bitcoin-based cryptocurrencies that we use today. The concept of a transaction having multiple inputs and outputs, for example, is gone, replaced by a more intuitive balance-based model (to prevent transaction replay attacks, as part of each account balance we also store an incrementing nonce). Sequence numbers and lock times are also removed, and all transaction and block data is encoded in a single format. Instead of addresses being the RIPEMD160 hash of the SHA256 hash of the public key prefixed with 04, addresses are simply the last 20 bytes of the SHA3 hash of the public key. Unlike other cryptocurrencies, which aim to have "features", Ethereum intends to take features away, and instead provide its users with near-infinite power through an all-encompassing mechanism called "contracts".

Ethereum Client P2P Protocol

The Ethereum client P2P protocol is a fairly standard cryptocurrency protocol, and can just as easily be used for any other cryptocurrency; the only modification is the introduction of the "Greedy Heaviest Observed Subtree" (GHOST) protocol first introduced by Yonatan Sompolinsky and Aviv Zohar [in December 2013](#); the motivation for and implementation of GHOST will be described in more detail below. The Ethereum client will be entirely reactive; it will not do anything by itself (except for the networking daemon maintaining connections) unless provoked. However, the client will also be more powerful; unlike bitcoind, which only stores a limited amount of data about the blockchain, the Ethereum client will also act as a fully functional backend for a block explorer.

When the client reads a message, it will perform the following steps:

1. Hash the data, and check if the data with that hash has already been received. If so, exit. Otherwise, pass it along to the data parser.
2. If the data parser says that a data item is a valid block, go to step 3. If the data parser says that a data item is a transaction, see if there are enough funds in the sending address for the transaction to go through, and if there are add it to the local transaction list and publish it to the network. If the data parser says that a data item is a message, send the message to the message responder and return the response.
3. Check if the "parent" parameter in the block is already stored in the database. If it is not, exit
4. Check if every block header in the "uncles" parameter in the block has the block's parent as its own parent. If any is not, exit. Note that uncle blocks do not need to be in the database; they just need to have valid proof of work.
5. Call the state updater with arguments (1) the parent of the block, (2) the transaction list of the block, (3) the timestamp of the block and (4) the coinbase of the block and see if the block header outputted by the state updater is exactly the same. If not, exit. If yes, add the block to the database and publish it to the network.
6. Determine $TD(block)$ ("total difficulty") for the new block. TD is defined recursively by $TD(genesis_block) = 0$ and $TD(B) = TD(B.parent) + \sum(u.difficulty \text{ for } u \text{ in } B.uncles) + B.difficulty$. If the new block has higher TD than the current block, set the current block to the new block.

If the node is mining, the node performs the following additional steps upon receiving a block:

1. Determine if the block's parent is in the database. If not, discard it.
2. Determine $\tau D(\text{block})$ ("total difficulty"). If $\tau D(\text{block})$ is higher than of any existing block in the database, start mining on that block and clear all uncles. Also, remove all transactions in the new block from the transaction pool if present, and immediately apply the remaining transactions to the new block.
3. If the block's parent is the parent of the highest block, add the block header to the set of uncles and restart the proof of work.

Upon receiving a transaction, the miner should apply the transaction to the current block and then start mining the new block header after the current mining round finishes.

The motivation behind GHOST is that blockchains with fast confirmation times currently suffer from reduced security due to a high stale rate; because blocks take a certain time to propagate through the network, call it t , if miner A mines a block and then miner B happens to mine another block before miner A's block propagates to B, miner B's block will end up wasted and will not contribute to network security. Furthermore, there is a centralization issue: if miner A is a mining pool with 30% hashpower and B has 10% hashpower, A will have a risk of producing stale blocks 70% of the time whereas B will have a risk of producing stale blocks 90% of the time. Thus, if the stale rate is high, A will be substantially more efficient simply by virtue of its size. With these two effects combined, blockchains which produce blocks quickly are very likely to lead to one mining pool having enough percentage of the network to have de facto control over the mining process. GHOST solves the first issue of network security loss by including stale blocks in the calculation of which chain is the "longest". Ethereum only does GHOST down one level for simplicity (ie. stale blocks must be included as uncles in the next block in order to count), but this should have 90%+ of the benefit of GHOST. Also, in Ethereum we give stales 3/4 of their block reward (and the nephew that includes them 1/8 as a reward); this modification is intended to solve the second issue of centralization bias.

Currency and Issuance

The Ethereum network includes its own built-in currency, ether. The main reason for including a currency in the network is twofold. First, like Bitcoin, ether is rewarded to miners to as to incentivize network security. Second, it serves as a mechanism for paying transaction fees for anti-spam purposes. Of the two main alternatives to fees, per-transaction proof of work similar to [Hashcash](#) and feeless laissez-faire, the former is wasteful of resources and unfairly punitive against weak computers and smartphones and the latter would lead to the network being almost immediately overwhelmed by an infinitely looping "logic bomb" contract. Ether will have a theoretical hard cap of 2^{128} units (compare $2^{50.9}$ in BTC), although not more than 2^{100} units will be released in the foreseeable future. For convenience and to avoid future argument (see the current mBTC/uBTC/satoshi debate), the denominations will be pre-labelled:

- 1: wei
- 10^3 :
- 10^6 :
- 10^9 : koblitz
- 10^{12} : szabo
- 10^{15} : finney
- 10^{18} : ether

This should be taken as an expanded version of the concept of "dollars" and "cents" or "BTC" and "satoshi" that is intended to be future proof; only szabo, finney and ether will likely be used in the foreseeable future. The right to name the 10^3 and 10^6 units will be left as a high-level secondary reward for the fundraiser subject to pre-approval from ourselves.

The issuance model will be as follows:

- Ether will be sold in a Mastercoin-style fundraiser at the price of 1 ether for 0.0001 BTC. Suppose that X ether gets collected in this way.
- 0.25X ether will be given to the founders.
- 0.25X ether will be given to the Ethereum organization as a reserve pool to pay expenses in ETH such as ETH salaries or bounties for those developers who want part or all of their compensation to be in this form
- 0.5X ether will be mined per year forever after that point (ie. permanent linear inflation)

For example, after five years and assuming no transactions, 25% of the ether will be in the hands of the fundraiser participants, 6.25% in the founder pool, 6.25% paid to the reserve pool, and 62.5% will belong to miners. The linear model reduces the risk of what some see as excessive wealth concentration in Bitcoin, and gives individuals living in present and future eras a fair chance to acquire currency units, while at the same time retaining a strong incentive to invest because the inflation "rate" still tends to zero over time. We also theorize that because coins are always lost over time, and coin loss can be modeled as a percentage of the total supply per year, that the total currency supply in circulation will in fact eventually stabilize at a value equal to the annual issuance divided by the loss rate (eg. 200x annual issuance at a loss rate of 0.5%).

	After 1 year	After 5 years
Currency units	2X	4X
Fundraiser	50%	25%
Founders	12.5%	6.25%
Reserve	12.5%	6.25%
Miners	25%	62.5%

Data Format

All data in Ethereum will be stored in [recursive length prefix encoding](#), which serializes arrays of strings of arbitrary length and dimension into strings. For example, ['dog', 'cat'] is serialized (in byte array format) as [130, 67, 100, 111, 103, 67, 99, 97, 116]; the general idea is to encode the data type and length in a single byte followed by the actual data (eg. converted into a byte array, 'dog' becomes [100, 111, 103], so its serialization is [67, 100, 111, 103]). Note that RLP encoding is, as suggested by the name, recursive; when RLP encoding an array, one is really encoding a string which is the concatenation of the RLP encodings of each of the elements. In the event that storing a number is required, the number will be stored in big-endian base 256 format (eg. 32767 in byte array format as [127, 255]).

A full block is stored as:

```
[ block_header, transaction_list, uncle_list ]
```

The block header is:

```
[ parent hash, sha3(rlp_encode(uncle_list)), coinbase address, state_root, sha3(rlp_encode(transaction_list)), difficulty, timestamp, nonce ]
```

Where the data for the proof of work is the RLP encoding of the block WITHOUT the nonce. `uncle_list` and `transaction_list` are the lists of the uncle block headers and transactions in the block, respectively.

The `state_root` is the root of a [Merkle Patricia tree](#) containing (key, value) pairs for all addresses where each address is represented as a 20-byte binary string. At each address, the value stored in the Merkle Patricia tree is a string which is the RLP-serialized form of an object of one of the following two forms:

```
[ balance, nonce ]
```

```
[ balance, nonce, contract_root ]
```

The `nonce` is the number of transactions made from the address, and is incremented every time a transaction is made. The purpose of this is to (1) make each transaction valid only once to prevent replay attacks, and (2) to make it impossible (more precisely, cryptographically infeasible) to construct a contract with the same hash as a pre-existing contract.

balance refers to the contract or address's balance, denominated in wei. `contract_root` is the root of yet another Patricia tree, containing the contract's memory, if that address is controlled by a contract. Note that in the main Patricia tree all addresses have a length of 20, even if they start with one or more zero bytes, and in the contract subtrees all indices have a length of 32, prepending with zero bytes if necessary.

Mining algorithm

Over the past five years of experience with Bitcoin and alternative cryptocurrencies, one important property for proof of work functions that has been discovered is that of "memory-hardness" - computing a valid proof of work should require not only a large number of computations, but also a large amount of memory. Currently, two major categories of memory-hard functions, script and Primecoin mining, exist, but both are imperfect; neither require nearly as much memory as an ideal memory-hard function could require, and both suffer from time-memory tradeoff attacks, where the function can be computed with significantly less memory than intended at the cost of sacrificing some computational efficiency. Ethereum instead uses an algorithm called Dagger, a memory-hard proof of work based on moderately connected directed acyclic graphs (DAGs, hence the name), which, while far from optimal, has much stronger memory-hardness properties than anything else in use today: an estimated 50-500 MB of RAM will be required per thread depending on the choice of parameters.

Dagger is more fully specified here: <http://wiki.ethereum.org/index.php/Dagger>

Transactions

A transaction is stored as:

```
[ nonce, receiving_address, value, [ data item 0, data item 1 ... data item n ], v, r, s ]
```

`nonce` is the number of transactions already sent by that address, or an empty string if this is the first transaction. (v, r, s) is the raw Electrum-style signature of the transaction without the signature made with the private key corresponding to the sending address, with $27 \leq v \leq 30$. From an Electrum-style signature (65 bytes) it is possible to extract the public key, and thereby the address, directly. Note that all transactions in Ethereum are valid; invalid transactions or transactions with insufficient balance simply have no effect. Transaction fees will be included automatically. If one wishes to voluntarily pay a higher fee, one is always free to do so by constructing a contract which forwards transactions but automatically sends a certain amount or percentage to the miner of the current block.

Difficulty adjustment

Difficulty is adjusted by the formula:

```
D(genesis_block) = 2^36
D(block) =
  if block.timestamp >= block.parent.timestamp + 42: D(block.parent) - floor(D(block.parent) / 1024)
  else: D(block.parent) + floor(D(block.parent) / 1024)
```

This stabilizes around a block time of 60 seconds automatically (note that at a block time of 60 seconds, 50% of blocks come within 42 seconds of the previous; $1 - 1/e = 63.22\%$ come within 60 seconds), and adjusts at a maximum rate of about 0.1% per block (~100% per 12 hours).

Transactions sent to the empty string as an address are a special type of transaction, creating a "contract".

Contracts

Here is where we get to the actually interesting part of the Ethereum protocol. In Ethereum, there are actually two types of entities that can generate and receive transactions: actual people (or bots, as cryptographic protocols cannot distinguish between the two) and contracts. A contract is essentially an automated agent that lives on the Ethereum network, has an Ethereum address and balance, and can send and receive transactions. A contract is "activated" every time someone sends a transaction to it, at which point it runs its code, perhaps modifying its internal state or even sending some transactions, and then shuts down. The "code" for a contract is written in a special-purpose low-level language consisting of a stack, which is not persistent, and 2^{256} memory entries, which constitute the contract's permanent state. Note that Ethereum users will not need to code in this low-level stack language; we will provide a simple C-like language with variables, expressions, conditionals and while loops, and provide a compiler down to Ethereum code.

Applications

Here are some examples of what can be done with Ethereum contracts, with all code examples written in a language that will likely be very similar to the C-like high-level language that we will release. The variables `tx.sender`, `tx.value`, `tx.fee`, `tx.data` and `tx.data_n` are properties of the incoming transaction, `contract.memory`, `contract.address` and `contract.creator` of the contract itself, and `block.contract_memory`, `block.addressbalance`, `block.number`, `block.difficulty`, `block.basefee` and `block.timestamp` properties of the block. `block.basefee` is the "base fee" which all transaction fees in Ethereum are calculated as a multiple of; for more info see the "fees" section below. All variables expressed as capital letters (eg. A) are constants, to be set by the contract creator when actually releasing the contract. The keywords `wei`, `szabo`, `finney` and `ether`, representing the denominations of ether, simply act as multipliers of 1, 1 million, etc, respectively.

Sub-currencies

Sub-currencies have many applications, ranging from currencies representing assets such as USD or gold to company stocks and even currencies with only one unit issued to represent collectibles or smart property. Advanced special-purpose financial protocols sitting on top of Ethereum may also wish to organize themselves with an internal currency. Sub-currencies are surprisingly easy to implement in Ethereum; this section describes a fairly simple contract for doing so.

The idea is that if someone wants to send X currency units to address A in currency contract C, they will need to make a transaction of the form $(C, 100 * \text{block.basefee}, [A, X])$, and the contract parses the transaction and adjusts balances accordingly. For a transaction to be valid, it must send 100 times the base fee worth of ether to the contract in order to "feed" the contract (as each computational step after the first 16 for any contract costs the contract a small fee and the contract will stop working if its balance drains to zero).

```
if tx.value < 100 * block.basefee:
    stop
if contract.memory[1000]:
    from = tx.sender
    to = tx.data[0]
    value = tx.data[1]
    if to <= 1000:
        stop
    if contract.memory[from] < value:
        stop
    contract.memory[from] = contract.memory[from] - value
    contract.memory[to] = contract.memory[to] + value
else:
    contract.memory[mycreator] = 1000000000000000000
    contract.memory[1000] = 1
```

Ethereum sub-currency developers may also wish to add some other more advanced features:

- Include a mechanism by which people can buy currency units in exchange for ether, perhaps auctioning off a set number of units every day.
- Allow transaction fees to be paid in the internal currency, and then refund the ether transaction fee to the sender. This solves one major problem that all other "sub-currency" protocols have had to date: the fact that sub-currency users need to maintain a balance of sub-currency units to use and units in the main currency to pay transaction fees in. Here, a new account would need to be "activated" once with ether, but from that point on it would not need to be recharged.
- Allow for a trust-free decentralized exchange between the currency and ether. Note that trust-free decentralized exchange between any two contracts is theoretically possible in Ethereum even without special support, but special support will allow the process to be done about ten times more cheaply.

Financial derivatives

The underlying key ingredient of a financial derivative is a data feed to provide the price of a particular asset as expressed in another asset (in Ethereum's case, the second asset will usually be ether). There are many ways to implement a data feed; one method, pioneered by the developers of [Mastercoin](#), is to include the data feed in the blockchain. Here is the code:

```
if tx.value < block.basefee:
    stop
if tx.sender != contract.creator:
    stop
memory[data[0]] = data[1]
```

Any other contract will then be able to query index `I` of data store `D` by using `block.contract_memory(D,I)`. A more advanced way to implement a data feed may be to do it off-chain - have the data feed provider sign all values, require anyone attempting to trigger the contract to include the latest signed data, and then use Ethereum's internal scripting functionality to verify the signature. Pretty much any derivative can be made from this, including leveraged trading, options, and even more advanced constructions like collateralized debt obligations (no bailouts here though, so be mindful of black swan risks).

To show an example, let's make a hedging contract. The basic idea is that the contract is created by party A, who puts up 4000 ether as a deposit. The contract then lies open for any party to accept it by putting in 1000 ether. Say that 1000 ether is worth \$25 at the time the contract is made, according to index `I` of data store `D`. If party B accepts it, then after 30 days anyone can send a transaction to make the contract process, sending the same dollar value worth of ether (in our example, \$25) back to B and the rest to A. B gains the benefit of being completely insulated against currency volatility risk without having to rely on any issuers. The only risk to B is if the value of ether falls by over 80% in 30 days - and even then, if B is online B can simply quickly hop onto another hedging contract. The benefit to A is the implicit 0.2% fee in the contract, and A can hedge against losses by separately holding USD in another location (or, alternatively, A can be an individual who is optimistic about the future of Ethereum and wants to hold ether at 1.25x leverage, in which case the fee may even be in B's favor).

```
state = contract.memory[1000]
if state == 0:
    if tx.value < 1000 ether:
        stop
    contract.memory[1001] = 998 * block.contract_memory(D,I)
    contract.memory[1002] = block.timestamp
    contract.memory[1003] = tx.sender
else:
    if tx.value < 200 finney:
        stop
    ethervalue = contract.memory[1000] / block.contract_memory(D,I)
    if ethervalue >= 5000 ether:
        send(contract.memory[1003],2000 ether,[])
    else if block.timestamp < contract.memory[1002]:
        send(contract.memory[1003],ethervalue,[])
        send(A,5000 - ethervalue,[])
```

More advanced financial contracts are also possible; complex multi-clause options (eg. "Anyone, hereinafter referred to as X, can claim this contract by putting in 2 USD before Dec 1. X will have a choice on Dec 4 between receiving 1.95 USD on Dec 29 and the right to choose on Dec 11 between 2.20 EUR on Dec 28 and the right to choose on Dec 18 between 1.20 GBP on Dec 30 and paying 1 EUR and getting 3.20 EUR on Dec 29") can be defined simply by storing a state variable just like the contract above but having more clauses in the code, one clause for each possible state. Note that financial contracts of any form do need to be fully collateralized; the Ethereum network controls no enforcement agency and cannot collect debt.

Identity and Reputation Systems

The earliest alternative cryptocurrency of all, [Namecoin](#), attempted to use a Bitcoin-like blockchain to provide a name registration system, where users can register their names in a public database alongside other data. The major cited use case is for a [DNS](#) system, mapping domain names like "bitcoin.org" (or, in Namecoin's case, "bitcoin.bit") to an IP address. Other use cases include email authentication and potentially more advanced reputation systems. Here is a simple contract to provide a Namecoin-like name registration system on Ethereum:

```
if tx.value < 25 ether:
    stop
if contract.memory[tx.data[0]]:
    stop
contract.memory[tx.data[0]] = contract.memory[tx.data[1]]
```

One can easily add more complexity to allow users to change mappings, automatically send transactions to the contract and have them forwarded, and even add reputation and web-of-trust systems.

Decentralized Autonomous Organizations

The general concept of a "decentralized autonomous corporation" is that of an entity that has a certain set of shareholders which collect dividends and which, perhaps with a 67% majority, have the right to modify its code. The shareholders would collectively decide on how the corporation should allocate its funds, using either bounties, salaries or even more exotic mechanisms such as an internal currency to reward work, and the funds would be distributed automatically. This essentially replicates the legal trappings of a traditional company but using only cryptographic blockchain technology for enforcement. This is the "corporate" model of a decentralized organization; an alternative, perhaps described as a "decentralized autonomous community", would have all members have an equal share in the decision making and require 67% of existing members to agree to add or remove a member. The requirement that one person can only have one membership would then need to be enforced collectively by the group.

Some "skeleton code" for a DAO might look as follows.

There are three transaction types:

- `[0,k]` to register a vote in favor of a code change
- `[1,k,L,v0,v1...vn]` to register a code change at code `k` in favor of setting memory starting from location `L` to `v0, v1 ... vn`

- [2,k] to finalize a given code change

Note that the design relies on the randomness of addresses and hashes for data integrity; the contract will likely get corrupted in some fashion after about 2^{128} uses, but that is acceptable since nothing close to that volume of usage will exist in the foreseeable future. 2^{255} is used as a magic number to store the total number of members, and a membership is stored with a 1 at the member's address. The last three lines of the contract are there to add C as the first member; from there, it will be C's responsibility to use the democratic code change protocol to add a few other members and code to bootstrap the organization.

```
k = sha3(tx.data[1])
if tx.data[0] == 0:
    if contract.memory[tx.sender] == 0:
        stop
    if contract.memory[k + tx.sender] == 0:
        contract.memory[k + tx.sender] = 1
        contract.memory[k] += 1
else if tx.data[0] == 1:
    if tx.value <= tx.datan * block.basefee * 200:
        stop
    if contract.memory[k] > 0:
        stop
    i = 3
    while i < tx.datan:
        contract.memory[k + i] = tx.data[i]
        i = i + 1
    contract.memory[k] = 1
    contract.memory[k+1] = tx.datan
    contract.memory[k+2] = tx.data[2]
else if tx.data[0] == 2:
    if contract.memory[k] >= contract.memory[2 ^ 255] * 2 / 3:
        if tx.value <= tx.datan * block.basefee * 200:
            stop
        i = 3
        L = contract.memory[k+1]
        loc = contract.memory[k+2]
        while i < L:
            contract.memory[loc+i-3] = tx.data[i]
            i = i + 1
if contract.memory[2 ^ 255 + 1] == 0:
    contract.memory[2 ^ 255 + 1] = 1
    contract.memory[C] = 1
```

This implements the "egalitarian" DAO model; one can easily extend it to a shareholder model by also storing how many shares each owner holds and providing a simple way to transfer shares.

Further Applications

- 1) **Crop insurance.** One can easily make a financial derivatives contract but using a data feed of the weather instead of any price index. If a farmer in Iowa purchases a derivative that pays out inversely based on the precipitation in Iowa, then if there is a drought the farmer will automatically receive money and if there is enough rain the farmer will be happy because their crops would do well.
- 2) **Generic insurance,** relying on one of a specified set of third-party judges to adjudicate claims. The contract can even include a complex appeals process if so desired.
- 3) A **decentrally managed data feed**, using proof-of-stake voting to give an average (or more likely, median) of everyone's opinion on the price of a commodity, the weather or any other relevant data
- 4) An offer to participate in a cryptographically secure, trust-free **peer-to-peer bet**
- 5) **SatoshiDice.** The entire gambling site can essentially be replicated on the blockchain using either peer-to-peer bet functionality or a centralized model.
- 6) A full-scale **on-chain stock market.** Prediction markets are also easy to implement as a trivial consequence.
- 7) An **on-chain decentralized marketplace**, using the identity and reputation system as a base.

How do contracts work?

A contract making transaction is encoded as follows:

```
[
  nonce,
  '',
  value,
  [
    data item 0,
    data item 1,
    ...
  ],
  v,
  r,
  s
]
```

The data items will, in most cases, be script codes (more on this below). Contract creation transaction validation happens as follows:

1. Deserialize the transaction, and extract its sending address from its signature.
2. Calculate the transaction's fee. Check that the balance of the creator is at least the endowment plus the fee. If not, exit, and if yes pay the fee.
3. Take the last 20 bytes of the hash of the transaction making the contract. If a contract with that address already exists, exit. Otherwise, create the contract at that address
4. Copy data item i to memory slot i for all i in [0 ... n-1] in the contract.

Whenever a transaction is sent to a contract, the contract executes its scripting code. The precise steps that happen when a contract receives a transaction are as follows:

1. The contract's ether balance increases by the amount sent
2. The index pointer is set to zero, and STEPCOUNT = 0
3. Repeat forever:

Contract Script Interpretation



- ◊ if the command at the index pointer is STOP, invalid or greater than 255, exit from the loop
- ◊ set MINERFEE = 0, VOIDFEE = 0
- ◊ set STEPCOUNT <- STEPCOUNT + 1
- ◊ if STEPCOUNT > 16, set MINERFEE <- MINERFEE + STEPFEE
- ◊ see if the command is LOAD or STORE. If so, set MINERFEE <- MINERFEE + DATAFEE
- ◊ see if the command will fill up a previously zero memory field. If so, set VOIDFEE <- VOIDFEE + MEMORYFEE
- ◊ see if the command will zero a previously used memory field. If so, set VOIDFEE <- VOIDFEE - MEMORYFEE
- ◊ see if the command is EXTRO or BALANCE. If so, set MINERFEE <- MINERFEE + EXTROFEE
- ◊ see if the command is a crypto operation. If so, set MINERFEE <- MINERFEE + CRYPTOFEES
- ◊ if MINERFEE + VOIDFEE > CONTRACT.BALANCE, HALT and exit from the loop
- ◊ else, subtract MINERFEE + VOIDFEE from the contract's balance and add MINERFEE to a running counter that will be added to the miner's balance once all transactions are parsed. Note that MINERFEE may be negative in some cases, in which case the contract balance would actually increase
- ◊ run the command
- ◊ if the command did not exit with an error, update the index pointer and return to the start of the loop. If the contract did exit with an error, break out of the loop.

The language maintains a stack which is initialized as empty, and all operations either manipulate the stack, perform special operations such as sending transactions and setting memory, or do both. In the following descriptions, $S[-1]$, $S[-2]$, etc represent the topmost, second topmost, etc items on the stack. The individual operations are defined as follows:

- (00) STOP - halts execution
- (01) ADD - pops two items and pushes $S[-2] + S[-1] \bmod 2^{256}$.
- (02) MUL - pops two items and pushes $S[-2] * S[-1] \bmod 2^{256}$
- (03) SUB - pops two items and pushes $S[-2] - S[-1] \bmod 2^{256}$
- (04) DIV - pops two items and pushes $\text{floor}(S[-2] / S[-1])$. If $S[-1] = 0$, halts execution.
- (05) SDIV - pops two items and pushes $\text{floor}(S[-2] / S[-1])$, but treating values above 2^{255} as negative (ie. $x \rightarrow 2^{256} - x$). If $S[-1] = 0$, halts execution.
- (06) MOD - pops two items and pushes $S[-2] \bmod S[-1]$. If $S[-1] = 0$, halts execution.
- (07) SMOD - pops two items and pushes $S[-2] \bmod S[-1]$, but treating values above 2^{255} as negative (ie. $x \rightarrow 2^{256} - x$). If $S[-1] = 0$, halts execution.
- (08) EXP - pops two items and pushes $S[-2] ^ S[-1] \bmod 2^{256}$
- (09) NEG - pops one item and pushes $2^{256} - S[-1]$
- (0a) LT - pops two items and pushes 1 if $S[-2] < S[-1]$ else 0
- (0b) LE - pops two items and pushes 1 if $S[-2] \leq S[-1]$ else 0
- (0c) GT - pops two items and pushes 1 if $S[-2] > S[-1]$ else 0
- (0d) GE - pops two items and pushes 1 if $S[-2] \geq S[-1]$ else 0
- (0e) EQ - pops two items and pushes 1 if $S[-2] == S[-1]$ else 0
- (0f) NOT - pops one item and pushes 1 if $S[-1] = 0$ else 0
- (10) MYADDRESS - pushes the contract's address as a number
- (11) TXSENDER - pushes the transaction sender's address as a number
- (12) TXVALUE - pushes the transaction value
- (13) TXDATAN - pushes the number of data items
- (14) TXDATA - pops one item and pushes data item $S[-1]$, or zero if index out of range
- (15) BLK_PREVHASH - pushes the hash of the previous block (NOT the current one since that's impossible!)
- (16) BLK_COINBASE - pushes the coinbase of the current block
- (17) BLK_TIMESTAMP - pushes the timestamp of the current block
- (18) BLK_NUMBER - pushes the current block number
- (19) BLK_DIFFICULTY - pushes the difficulty of the current block
- (1a) BASEFEE - pushes the base fee (x as defined in the fee section below)
- (20) SHA256 - pops one item and then $\text{ceil}(S[-1] / 32)$ further items. Then takes those items as 32-byte strings, concatenates them in top-to-bottom order, taking out the low-order bytes of the bottommost if necessary, and pushes the SHA256 of the resulting string.
- (21) RIPEMD160 - works just like SHA256 but with the RIPEMD-160 hash
- (22) ECMUL - pops three items. If $(S[-2], S[-1])$ are a valid point in secp256k1, including both coordinates being less than P , pushes $(S[-2], S[-1]) * S[-3]$, using $(0,0)$ as the point at infinity. Otherwise, pushes $(2^{256} - 1, 2^{256} - 1)$. Note that there are no restrictions on $S[-3]$.
- (23) ECADD - pops four items and pushes $(S[-4], S[-3]) + (S[-2], S[-1])$ if both points are valid, otherwise $(2^{256} - 1, 2^{256} - 1)$.
- (24) ECSTGN - pops two items and pushes (v, r, s) as the Electrum-style RFC6979 deterministic signature of message hash $S[-1]$ with private key $S[-2] \bmod N$.
- (25) ECRECOVER - pops four items and pushes (x, y) as the public key from the signature $(S[-3], S[-2], S[-1])$ of message hash $S[-4]$. If the signature has invalid v, r, s values (ie. v not in $[27, 28]$, r not in $[0, P]$, s not in $[0, N]$), return $(2^{256} - 1, 2^{256} - 1)$.
- (26) ECVALID - pops two items and pushes 1 if $(S[-2], S[-1])$ is a valid secp256k1 point (including $(0,0)$) else 0
- (27) SHA3 - works just like SHA256 but with the SHA3 hash, 256 bit version.
- (30) PUSH - pushes the item in memory at the index pointer + 1, and advances the index pointer by 2.
- (31) POP - pops one item.
- (32) DUP - pushes $S[-1]$ to the stack.
- (33) DUPN - reads item in memory at the index pointer + 1, say N , and pushes $S[-N]$ to the stack. Advances the index pointer by 2. If $N = 0$, exits with an error instead.
- (34) SWAP - pops two items and pushes $S[-1]$ then $S[-2]$
- (35) SWAPN - reads item in memory at the index pointer + 1, say N , and swaps $S[-1]$ and $S[-N]$ to the stack. Advances the index pointer by 2. If $N = 0$, exits with an error instead.
- (36) LOAD - pops one item and pushes the item in memory at index $S[-1]$
- (37) STORE - pops two items and sets the item in memory at index $S[-1]$ to $S[-2]$
- (40) JMP - pops one item and sets the index pointer to $S[-1]$
- (41) JMPi - pops two items and sets the index pointer to $S[-2]$ only if $S[-1]$ is nonzero
- (42) IND - pushes the index pointer
- (50) EXTRO - pops two items and pushes memory index $S[-2]$ of contract $S[-1]$
- (51) BALANCE - pops one item and pushes balance of that address, or zero if the address is invalid
- (60) MKTX - pops three items and initializes a transaction to send $S[-2]$ ether to $S[-1]$ with $S[-3]$ data items. Pops that many data items and adds them in order popped as data items to the transaction. Then sends the transaction. Note that if $S[-1] = 0$ then this creates a new contract.
- (ff) SUICIDE - pops one item, destroys the contract and clears all memory, sending the entire balance plus the negative fee from clearing memory to the address at $S[-1]$

Fees

Ethereum will have seven primary fees, of which one applies to transaction senders and six apply to contracts. The fees are:

- TXFEE (100x) - fee for sending a transaction
- NEWCONTRACTFEE (100x) - fee for creating a new contract, not including the memory fee for each item in script code
- STEPFEE (x) - fee for every computational step after than first sixteen in contract execution
- MEMORYFEE (100x) - fee for adding a new item to a contract's memory, including when first creating a contract. The memory fee is the only fee that is not paid to a miner, and is refunded when memory from a contract is removed.
- DATAFEE (20x) - fee for accessing or setting a contract's memory from inside that contract
- EXTROFEE (40x) - fee for accessing memory from another contract inside a contract
- CRYPTOFEE (20x) - fee for using any of the cryptographic operations

One novel innovation in Ethereum is that the fees will be inversely proportional to the square root of the difficulty; that is, $x = \text{floor}(10^{21} / \text{floor}(\text{difficulty} ^ 0.5))$. The reason why this is done is twofold:

1. It creates a fully decentralized mechanism for setting transaction fees that ensures that fees do not grow too quickly with the growth of the network. For example, if ether becomes 4x more valuable, then it is expected that network mining power and therefore difficulty will increase by 4x, meaning that ether-denominated transaction fees will decrease by 2x and therefore real-value transaction fees will only increase 2x.
2. If computers get 4x more powerful due to Moore's Law, difficulty will increase 4x and so fees will decrease 2x, reflecting computers' increased ability to handle higher load.

The reason the exponent should be kept below 1 is that (1) Moore's Law may affect computing power faster than the slowest component of the transaction processing pipeline, and (2) if the Ethereum network grows very quickly due to an increased number of computers and not any increase in power to each individual computer, an exponent of 1 may push per-computer load too high.

Conclusion

The Ethereum protocol's design philosophy is in many ways the opposite from that taken by many other cryptocurrencies today. Other cryptocurrencies aim to add complexity and increase the number of "features"; Ethereum, on the other hand, takes features away. The protocol does not "support" multisignature transactions, multiple inputs and outputs, hash codes, lock times or many other features that even Bitcoin provides. Instead, all complexity comes from an all-powerful, Turing-complete assembly language, which can be used to build up literally any feature that is mathematically describable. As a result, we have a cryptocurrency protocol whose codebase is very small, and yet which can do anything that any cryptocurrency will ever be able to do.

References and Further Reading

1. Colored coins whitepaper: https://docs.google.com/a/buterin.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0llzrTLuoWu2z1BE/edit
2. Mastercoin whitepaper: <https://github.com/mastercoin-MSC/spec>
3. Decentralized autonomous corporations, Bitcoin Magazine: <http://bitcoinmagazine.com/7050/bootstrapping-a-decentralized-autonomous-corporation-part-i/>
4. Smart property: https://en.bitcoin.it/wiki/Smart_Property
5. Smart contracts: <https://en.bitcoin.it/wiki/Contracts>
6. Simplified payment verification: <https://en.bitcoin.it/wiki/Scalability#Simplifiedpaymentverification>
7. Merkle trees: http://en.wikipedia.org/wiki/Merkle_tree
8. Patricia trees: http://en.wikipedia.org/wiki/Patricia_tree
9. Bitcoin whitepaper: <http://bitcoin.org/bitcoin.pdf>
10. GHOST: <http://www.cs.huji.ac.il/~avivz/pubs/13/btcscaleabilityfull.pdf>
11. StorJ and Autonomous Agents, Jeff Garzik: <http://garzikrants.blogspot.ca/2013/01/storj-and-bitcoin-autonomous-agents.html>
12. Ethereum RLP: <http://wiki.ethereum.org/index.php/RLP>
13. Ethereum Merkle Patricia trees: http://wiki.ethereum.org/index.php/Patricia_Tree
14. Ethereum Dagger: <http://wiki.ethereum.org/index.php/Dagger>

[news](#) [what](#) [why](#) [philosophy](#) [how](#) [who](#) [meetup](#) [community](#)



ethereum

a revolutionary new platform for applications



Enter your email

Subscribe to our newsletter

news

[introduction](#)

[status](#)

[twitter](#)

[videos](#)

[links](#)



download

Proof of Concept client

windows client



mac client

linux source



build from source

what

Ethereum is a platform and a programming language that makes it possible for any developer to build and publish next-generation distributed applications.

Ether, Ethereum's internal currency, powers its applications and acts as a 'token of exchange' on the decentralized network.

Ethereum can be used to codify, decentralize, secure and trade just about anything: voting, domain names, financial exchanges, crowdfunding, company governance, contracts and agreements of most kind, intellectual property, and even smart property thanks to hardware integration.

Ethereum borrows the concept of decentralized consensus that makes bitcoin so resilient, yet makes it trivial to build on its foundation. To find out more about how Ethereum works, consult the [white](#) and [yellow](#) papers.

...will you build with ethereum?



name registration



```
if tx.value < block.basefee * 200:
    stop
if contract.storage[tx.data[0]] or tx.data[0] < 100:
    stop
contract.storage[tx.data[0]] = tx.data[1]
```

why?

Most of the services you use today have something in common: they are centralized. For example, when you deposit money at your bank, you trust them to be honest, be secure and be independently audited. The same is true when you post pictures on Facebook, or an important document on Dropbox. History has proven time and time again that this model is flawed, but necessary, as 'trustless' operations have been so far both unprofitable and too complex to implement.

Enter Ethereum

Applications built on Ethereum do not require their users to trust the developers with their personal information or funds. Ethereum not only addresses the issues described above, it also opens the door to whole new kinds of applications that have never been seen before.

philosophy

simplicity

Ethereum is a protocol that is open to all. Any programmer should be able to write smart contracts and Ethereum apps in languages they already are familiar with.

Programmers should also be able to implement the entire specification with relative ease, in order to minimize the influence that any specific individual or group could have on the protocol.

The Ethereum protocol should be as simple as possible, and optimizations which add complexity should not be included unless they provide a very substantial benefit.



Simplicity



Agility



Modularity



Universality



Non-Discrimination

what can you make out of ether?



Create a Currency
and issue Shares

Contracts are the main building blocks of Ethereum. A contract is a computer program that lives inside the distributed Ethereum network and has its own ether balance, memory and code. Every time you send a transaction to a contract, it executes its code, which can store data, send transactions and interact with other contracts.

Contracts are maintained by the network, without any central ownership or control. Contracts are written in languages instantly familiar to any programmer and powered by Ether, Ethereum's cryptofuel. You can find out more in our [wiki](#).

open source, one click app install.



— or —



make your own



Mobile & desktop reference clients.

Turing complete contracts on a blockchain..



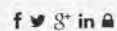
flexible implementations.

who?

Vitalik Buterin
Founder



Charles Hoskinson
Founder



Anthony Di Iorio
Founder



Mihai Alisie
Founder



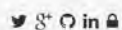
Gavin Wood
Founder



Joseph Lubin
Founder



Jeffrey Wilcke
Founder



Amir Chetrit
Founder

Eric Lombrozo
Development

Neal Koblitz
Advisor



Stanley Tuck

Taylor Colton

Jeremy Wood

Gregory Bortone

Mathias Grosche

Stephan Tual
Communications



Taylor Gehring
Development



Jeremy Wood
Administrative



Lorenzo Patuzzo
Creative



Matthias Grimmhuber
Development



Ryan Taylor
Development

Anthony D'Onofrio
Creative



Tim Hughes
Development



Richard Stott
Creative



Ian Meikle
Creative

Nicolas Fierro
Development



Aeron Buchanan
Development

Jason Colby
Administrative



Roxana Sureanu
Administrative



Marko Simovic
Development

Maran Hidskes
Development



Jonathan Mohan
Communications

Paul Snow
Development

Jerry Liu
Chinese Partner

Bin Lu
Chinese Partner

Houwu Chen
Development/Partner

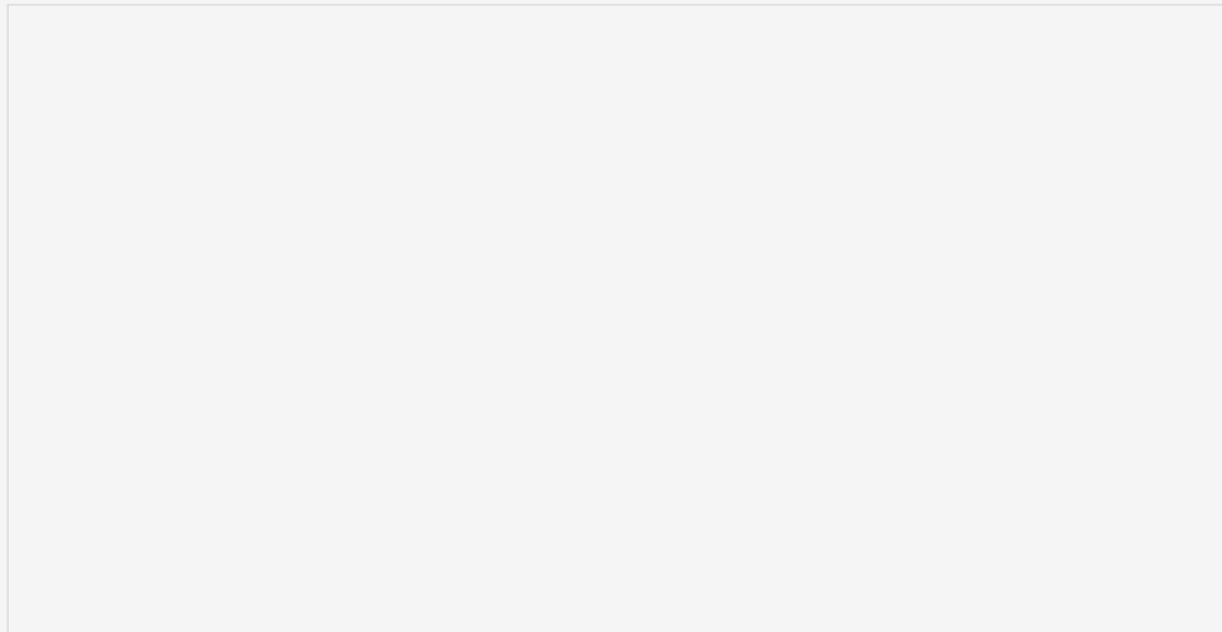


Hai Nguyen
Accounting

Steve Dakh
Development/Kryptokit

Ethan Wilding
Administrative

ethereum meetups



Meet developers and entrepreneurs at one of our [meetups](#)
You can also find answers to your questions on our [forums](#), [wiki](#), or [blog](#)



https://web.archive.org/web/20130823110620/https://www.reddit.com/r/Bitcoin/comments/1j45q5/oreilly_book_on_bitcoin_its_official_i_will_be/

189  O'Reilly book on bitcoin - It's official, I will be writing "Mastering Bitcoin" for publication by O'Reilly Media. Check out bitcoinbook.info and help shape the book!

(bitcoinbook.info)
submitted 27 days ago by andreasma
40 comments share

all 40 comments - sorted by: best

[-] [thewanderer5](#) 32 points 27 days ago

Since everyone else seems to be too stupid to say it... good fucking job man!

O'Reilly has always been my go to, about as legit as it gets. Can't wait to see what animal it's going to be!

permalink save

[-] [andreasma](#) [S] 13 points 27 days ago

Thank you.

There are competing proposals and in the end it is not my decision. The mockup I did has the leafcutter ant (vast, de-centralized hive that stores long term wealth)

Many have suggested the Honey Badger.

I may hold a vote and raffle. With competing videos.

permalink save parent

[-] [zagaberoo](#) 4 points 26 days ago

I'd definitely prefer an elegant animal metaphor over a meme.

Go team ant!

permalink save parent

[-] [fuyuasha](#) 24 points 27 days ago

Honey Badger - must use Honey Badger for the cover !!!!!

permalink save

[-] [syriven](#) 3 points 27 days ago

Yes! Honey Badger!

permalink save parent

[-] [manny1020](#) 5 points 27 days ago

Please tell me you'll accept bitcoin as payment.

permalink save

[-] [andreasma](#) [S] 12 points 27 days ago

I very much hope I will be able to persuade O'Reilly to adopt. But it will take a long time and a lot of customer support. I have requested it, explicitly, from the proposal stage.

Stay tuned, or better yet, sign up for updates.

permalink save parent

[-] [neuronstorm](#) 5 points 27 days ago

+/u/bitcointip 0.01 BTC

Signed up for updates. I'll buy that book and others if they accept Bitcoin!

permalink save parent

[-] [manny1020](#) 2 points 27 days ago

Oh, I thought it had been a done deal. Definitely supporting this.

permalink save parent

[-] [fuyuasha](#) 1 point 27 days ago

Totally supporting this!

permalink save parent

[-] [notreefitty](#) 1 point 26 days ago

You've got my coin.

permalink save parent

[-] [TIGGERNITS](#) 3 points 27 days ago

I saw "O'Reilly book on bitcoin" and all I could think was "Oh Christ, that asshole is going to write *Killing Bitcoin* next.. I'm very glad it was a misinterpretation :)

EDIT: Thanks for being a positive force in the community! Education is the key to acceptance!

permalink save

[-] [syriven](#) 2 points 27 days ago

The page you linked to provides a link to "shape the book". Is this still open to input, or are you nearing the end of the development cycle?

permalink save

[-] [andreasma](#) [S] 1 point 27 days ago

Just opened, just started. Input is very welcome!

permalink save parent

[-] [murclock](#) 2 points 27 days ago

search reddit

this post was submitted on 26 Jul 2013

189 points (85% like it)

226 upvotes 37 downvotes

shortlink: <http://redd.it/1j45q5>

username

password

☐

remember me

reset password

login

Submit a new link

Submit a new text post

Bitcoin

subscribe 48,691 readers

226 users here now

Bitcoin is *the currency of the Internet*: a distributed, worldwide, decentralized digital money. Unlike traditional currencies such as dollars, bitcoins are issued and managed without any central authority whatsoever: there is no government, company, or bank in charge of Bitcoin. As such, it is more resistant to wild inflation and corrupt banks. With Bitcoin, you can be *your own bank*.

If you are new to Bitcoin, check out [We Use Coins](#) and [Bitcoin.org](#). You can also explore the [Bitcoin Wiki](#):

- Getting Started
- FAQ
- Common Myths
- Bitcoin Merchants

How to buy bitcoins

Buy Reddit Gold using bitcoins!

Will I earn money by mining?

Community rules

- Do not use URL shortening services: always submit the real link.
- Begging/asking for bitcoins is absolutely not allowed, no matter how badly you need the bitcoins. Only requests for donations to large, recognized charities are allowed, and only if there is good reason to believe that the person accepting bitcoins on behalf of the charity is trustworthy.
- News articles that do not contain the word "Bitcoin" are usually off-topic. This subreddit is not about general financial news.
- Submissions that are mostly about some other cryptocurrency belong elsewhere.
- No referral links in submissions.
- No compilations of free Bitcoin sites.
- Trades should usually not be advertised here. For example, submissions like "Buying 100 BTC" or "Selling my computer for bitcoins" do not belong here. [r/Bitcoin](#) is primarily for news and discussion.
- Please avoid repetition — [r/bitcoin](#) is a subreddit devoted to new information and discussion about Bitcoin and its ecosystem. New merchants are welcome to announce their services for Bitcoin, but after those have been announced they are no longer news and should not be re-posted. Aside from *new merchant* announcements, those interested in advertising to our audience should consider Reddit's self-serve advertising system.

Related communities

Awesome!

permalink save

[-] [Thistlemanizzle](#) 1 point 27 days ago

Lol, thought this was Bill O'reilly for a second. Flat currency is for pinheads!

permalink save

[-] [SlowMotionOcean](#) 1 point 26 days ago

Just a small correction: Bill O'Reilly is a melon head, not a pin head!

permalink save parent

[-] [Thistlemanizzle](#) 1 point 26 days ago

"Pinhead" is Bill O'Reilly's favorite insult.

permalink save parent

[-] [SlowMotionOcean](#) 1 point 26 days ago

Ahh. I don't watch him.

permalink save parent

[-] [deleted] 27 days ago

[deleted]

permalink save

[-] [andreasma](#) [S] 7 points 27 days ago

Not that O'Reilly.

O'Reilly Media, best technical publishers in the world. Make a series of books with a different animal on each cover?

permalink save parent

[-] [apteryx_274](#) 1 point 27 days ago

oh my bad, I'll delete it. Thanks!

permalink save parent

[-] [FootofGod](#) 1 point 26 days ago

...but if he dd. Could you imagine?!

permalink save parent

[-] [Aahzmundus](#) 1 point 26 days ago

From listening to you on lets talk bitcoin, I have faith you will do well... but I have some concern.

Often when something about bitcoin bothers you, you compare bitcoin to the ideal, instead of bitcoin to the current system. Remember when writing this book that while bitcoin may be far from the ideal in some respects, we are far better in nearly every way from the current system.

I will take your survey, and participate in any way I can. But please, please... do not fuck this up XD

permalink save

[-] [andreasma](#) [S] 2 points 26 days ago*

There's a vast difference between a talk show and a book, don't worry, I am aware ;-)

This book is intended to be a technical reference and guide. You can see my style of writing on this topic in the O'Reilly Radar blog, though even those articles are not the same style as a book:

<http://radar.oreilly.com/amantonopoulos>

Yes, I will try hard not to fuck this up. I aim for this to be the definitive technical guide.

I very much appreciate any and all help I can get.

Start with the *outline*. After reviewing that, add comments, then fill in the *survey* so that I know what's important to you. Both are on bitcoinbook.info

Thanks!

permalink save parent

[-] [bitcoinbash](#) 1 point 25 days ago

did the survey. The outline looks amazing. I would love to buy some of these books just to give away. You should try to get o'reilly to sell the book in packs of 5 and 10 for a discount.

permalink save

[-] [andreasma](#) [S] 1 point 25 days ago

Thank you!

permalink save parent

[-] [sirfaxalot](#) 0 points 27 days ago

| This site is not endorsed or operated by O'Reilly Media.

permalink save

[-] [andreasma](#) [S] 7 points 27 days ago

That is correct, I am clear about it, the site is not endorsed or run by ORA. I run the site, representing my own views.

However, I am most definitely writing a bitcoin book, under contract with ORA. It's the culmination of several months of work on the proposal and associated articles I wrote (see O'Reilly Radar blog).

permalink save parent

[+] [jvjks4](#) comment score below threshold (5 children)

[-] [reafdfdc](#) 0 points 27 days ago

I found this so hard to believe that I went to O'Reilly's website and found no mention of you or anything bitcoin related in the past year, and only a few blog posts from 2012 and 2011. If you were being published by O'Reilly, why are they so embarrassed to even mention your existence?

A quick google search only produced this thread.

Kind of weird, but I guess I can see why you would want a 'how to' bitcoin book with a familiar and trusted face attached to it.

Also, I quickly glanced through your book. I found it to be poorly written.

permalink save

(Sorted roughly by decreasing popularity.)

Bitcoin communities

- Local Bitcoin communities
- Girls Gone Bitcoin (NSFW)
- BitMarket
- BitcoinMining
- BitcoinMarkets
- Chateau deCrypto (NSFW)
- MtGox
- Bitcoinip
- Jobs4Bitcoin
- Bitcoin Beginners
- bitcoinwallet
- [More]

Non-Bitcoin communities

- Technology
- Economics
- Libertarian
- Anarcho-Capitalism
- Crypto
- Austrian economics
- CryptoAnarchy
- Agorism
- [More]

Join us on IRC

[#bitcoin](http://irc.freenode.net) and [#bitcoin-dev](http://irc.freenode.net)

Other Bitcoin sites

[Bitcoin Forum](#)

[Bitcoin Stack Exchange](#)

Ad campaign

Bitcoins sent here will eventually be used for a Reddit advertising campaign with the goal of promoting Bitcoin and /r/Bitcoin:
[16KaCJB7fVuT6hvA7wzgzVjAnHz28bNvvh](https://blockchain.info/address/16KaCJB7fVuT6hvA7wzgzVjAnHz28bNvvh)
(1.878 BTC spent so far)

MODERATORS

[message the moderators](#)

theymos
hugolp
jesset77
ThePiachu
rbitcoin-bot
MiningBuddy
JordanTheBrobot

[-] [andreasma](#) [5] 2 points 26 days ago

Nothing weird to it. This book deal was just completed, the book is in the earliest planning stage. O'Reilly Media will move at their pace, they don't start marketing until a lot later. However, just to be clear, *I have signed contracts to publish this book with O'Reilly Media of Sebastopol, CA.* Title, content, outline may change, of course, but I have committed to delivering it to the publisher.

I am seeking input on the outline and the audience, as I hope to engage the entire bitcoin community in producing the definitive bitcoin book. Several prominent community members were supportive and contributed ideas for the outline during the proposal stage that started many months ago.

You can see my recent articles on O'Reilly's Radar blog: <http://radar.oreilly.com/amantonopoulos>

I am sorry you did not like the first draft outline. Feel free to offer specific comments, and I will gladly listen.

Andreas

[permalink](#) [save](#) [parent](#)

[+] [champbronc2](#) -1 points 26 days ago

Bitcoin actually aligns quite well with the republican/libertarian viewpoint. Anti government appeal.

[permalink](#) [save](#)

[+] [andreasma](#) [5] 2 points 26 days ago

This book is non-partisan. I am personally neither republican nor libertarian.

Bitcoin is a technocratic, scientific and neutral de-centralized currency/protocol/network based on simple mathematical rules that are clear and visible to all in the code.

[permalink](#) [save](#) [parent](#)

[+] [champbronc2](#) 1 point 26 days ago

But would you not agree the nature of Bitcoin is of itself appealing to libertarians?

[permalink](#) [save](#) [parent](#)

[+] [Reallybruh](#) comment score below threshold (1 child)

about

[blog](#)
[about](#)
[team](#)
[source code](#)
[advertise](#)

help

[wiki](#)
[FAQ](#)
[reddiquette](#)
[rules](#)
[contact us](#)

tools

[mobile](#)
[firefox extension](#)
[chrome extension](#)
[buttons](#)
[widget](#)

<3

[reddit gold](#)
[store](#)
[redditgifts](#)
[reddit.tv](#)
[radio reddit](#)

Use of this site constitutes acceptance of our [User Agreement](#) and [Privacy Policy](#). © 2013 reddit inc. All rights reserved.
REDDIT and the ALIEN Logo are registered trademarks of reddit inc.

GitHub

This repository Search

Explore Features Enterprise Blog

Sign up

Sign in

aantonop / bitcoinbook

★ Star

599

🍴 Fork

145

Mastering Bitcoin - Unlocking digital currencies - Early Release Draft

734 commits

2 branches

0 releases

40 contributors

<> Code

🔔 Issues

🔗 Pull Requests

📊 Pulse

📈 Graphs

branch: develop bitcoinbook / +

HTTPS clone URL

https://github.com/i

You can clone with HTTPS or Subversion.

📄 Download ZIP

Merge pull request #154 from kaibakker/patch-1

enderminh authored 20 days ago

latest commit 76b2143a50

code	code examples	2 months ago
images	replaced SVG with PNG	2 months ago
selected BIPs	Added appendix with selected BIPs. Converted from mediawiki using pan...	6 months ago
.gitignore	ch4 fixes	6 months ago
README.md	README added	3 months ago
appdx-bips.asciidoc	BIP list	2 months ago
appdx-scriptops.asciidoc	move OP appendix	3 months ago
appdx-sx.asciidoc	formatting fixes	2 months ago
appdx01.asciidoc	test formatting	a year ago
book.asciidoc	Updated book.asciidoc	2 months ago
ch01.asciidoc	Changed Mobile Bitcoin -> Mobile Client	26 days ago
ch02.asciidoc	review comments from Allyson MacDonald	2 months ago
ch03.asciidoc	Fix typo	29 days ago
ch04.asciidoc	Standardizing the use of bitcoin-cli over bitcoind in a few other pla...	2 months ago
ch05.asciidoc	Typo fix, s/public/public/	a month ago
ch06.asciidoc	review comments from Allyson MacDonald	2 months ago
ch07.asciidoc	Changing the sentence to 'The merkle tree is a binary tree with an ev...	a month ago
ch08.asciidoc	0 BTC is correct value to send	29 days ago
ch09.asciidoc	fixes	2 months ago
ch10.asciidoc	Reverting to some of the original language which I think was pretty g...	a month ago
index.asciidoc	fixes and index	a year ago
planned_toc.asciidoc	new TOC for next few chapter planned in early release	7 months ago
preface.asciidoc	revised acknowledgments	2 months ago
pycoin.asciidoc	First check-in of pycoin.asciidoc.	6 months ago

README.md

Mastering Bitcoin

Mastering Bitcoin is a book for developers, but the first two chapters cover bitcoin at a level that is approachable to non-programmers. Anyone with a basic understanding of technology can read the first

approachable to non-programmers. Anyone with a basic understanding of technology can read the first two chapters and get a great understanding of bitcoin.

Contributing

To contribute to this book, please fork and make changes to your forked copy, then submit a pull request. Or alternatively, if you cannot use the github.com pull request system, submit an Issue from the menu on the right. Major contributions will receive an acknowledgment in the preface of the book.

Purchasing

"Mastering Bitcoin" will be available in print and e-book format in Fall 2014. To pre-order your copy, please visit:

<http://bitcoinbook.info>



EXHIBIT B

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

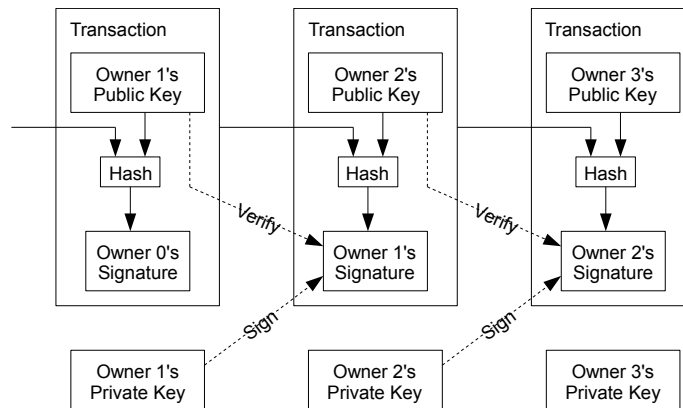
1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

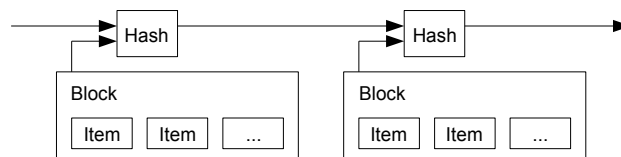


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

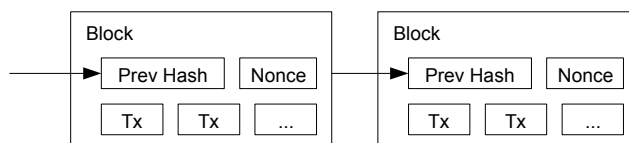
The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

- 1) New transactions are broadcast to all nodes.
- 2) Each node collects new transactions into a block.
- 3) Each node works on finding a difficult proof-of-work for its block.
- 4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
- 5) Nodes accept the block only if all transactions in it are valid and not already spent.
- 6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

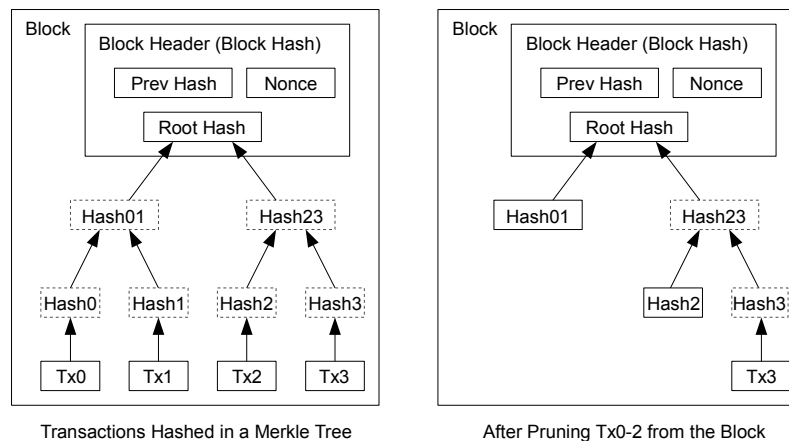
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

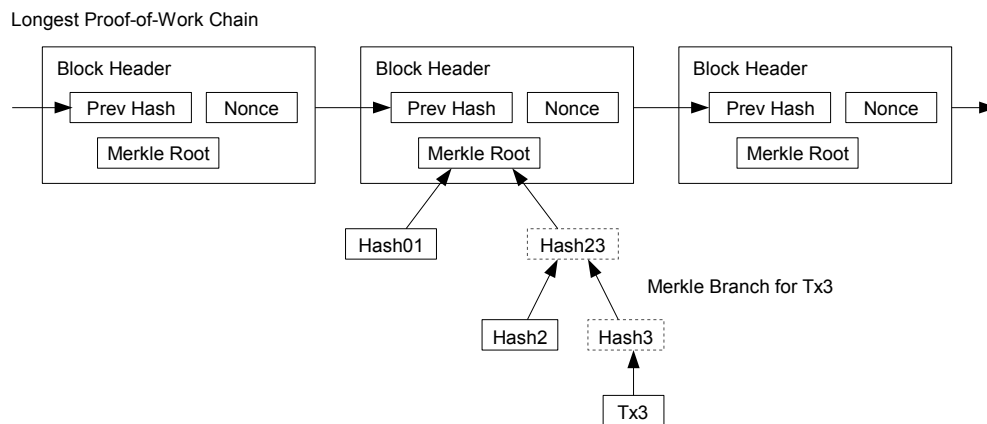
Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, $80 \text{ bytes} * 6 * 24 * 365 = 4.2\text{MB}$ per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

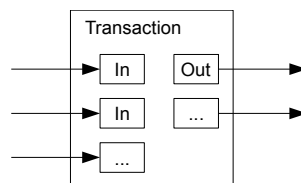
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He 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.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

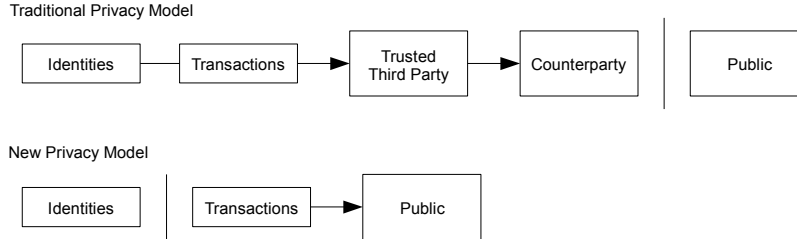
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
 q = probability the attacker finds the next block
 q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z \frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \begin{cases} (q/p)^{(z-k)} & \text{if } k \leq z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^z \frac{\lambda^k e^{-\lambda}}{k!} (1 - (q/p)^{(z-k)})$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0    P=1.0000000
z=1    P=0.2045873
z=2    P=0.0509779
z=3    P=0.0131722
z=4    P=0.0034552
z=5    P=0.0009137
z=6    P=0.0002428
z=7    P=0.0000647
z=8    P=0.0000173
z=9    P=0.0000046
z=10   P=0.0000012
```

```
q=0.3
z=0    P=1.0000000
z=5    P=0.1773523
z=10   P=0.0416605
z=15   P=0.0101008
z=20   P=0.0024804
z=25   P=0.0006132
z=30   P=0.0001522
z=35   P=0.0000379
z=40   P=0.0000095
z=45   P=0.0000024
z=50   P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10  z=5
q=0.15  z=8
q=0.20  z=11
q=0.25  z=15
q=0.30  z=24
q=0.35  z=41
q=0.40  z=89
q=0.45  z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

- [1] W. Dai, "b-money," <http://www.weidai.com/bmoney.txt>, 1998.
- [2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.
- [3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.
- [4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.
- [5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.
- [6] A. Back, "Hashcash - a denial of service counter-measure," <http://www.hashcash.org/papers/hashcash.pdf>, 2002.
- [7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.
- [8] W. Feller, "An introduction to probability theory and its applications," 1957.

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

DR. GAVIN WOOD
CTO, ETHEREUM PROJECT
GAVIN@ETHEREUM.ORG

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, not least Bitcoin. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. INTRODUCTION

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technological-rooted movements like Bitcoin have demonstrated through the power of the default, consensus mechanisms and voluntary respect of the social contract that it is possible to use the internet to make a decentralised value-transfer system, shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographically secure, transaction-based state machine. Follow-up systems such as Namecoin adapted this original “currency application” of the technology into other applications albeit rather simplistic ones.

Ethereum is a project that attempts to build the generalised technology; technology on which all transaction-based state machine concepts may be built. Moreover it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto unexplored compute paradigm in the mainstream: a trustful object messaging compute framework.

1.1. Driving Factors. There are many goals of this project; one key goal is to facilitate transactions between consenting individuals who would otherwise have no means to trust one another. This may be due to geographical separation, interfacing difficulty, or perhaps the incompatibility, incompetence, unwillingness, expense, uncertainty, inconvenience or corruption of existing legal systems. By specifying a state-change system through a rich and unambiguous language, and furthermore architecting a system such that we can reasonably expect that an agreement will be thus enforced autonomously, we can provide a means to this end.

Dealings in this proposed system would have several attributes not often found in the real world. The incorruptibility of judgement, often difficult to find, comes naturally from a disinterested algorithmic interpreter. Transparency, through being about to see exactly how a state or judgement came about through the transaction log and rules or instructional codes never happens perfectly in human-based systems.

Overall, we wish to provide a system such that the user can be guaranteed that any other individuals, systems or organisations that they interact with, they can do so with

absolute confidence in the possible outcomes and how they might come about.

2. THE BLOCKCHAIN PARADIGM

Ethereum, taken as a whole, can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to morph it into some final state. It is this final state which we accept as the canonical “version” of the world of Ethereum. The state can include such information as account balances, reputations, trust arrangements, data pertaining to information of the physical world; in short, anything that can currently be represented by a computer is admissible. Transactions thus represent a valid arc between two states; the ‘valid’ part is important—there exist far more invalid state changes than valid state changes. Invalid state changes might, e.g. be things such as reducing an account balance without an equal and opposite increase elsewhere. We define a valid state transitions as one which comes about through a transaction. Formally:

$$(1) \quad S_{t+1} \equiv \Upsilon(S_t, T)$$

where Υ is the Ethereum state transition function. In Ethereum, Υ , together with S are considerably more powerful than any existing comparable system; Υ allows components to carry out arbitrary computation, while S allows components to store arbitrary state between transactions.

Transactions are collated into blocks; we then chain blocks together using a cryptographic hash as a means of reference. Blocks function as a journal, recording a series of transactions together with the previous block and an identifier for the final state (though do not store the final state itself—that would be far too big). They also punctuate the transaction series with incentives for nodes to *mine*. This incentivisation takes places as a state-transition function, adding value to a nominated account.

Mining is the process of dedicating effort (working) to bolster one series of transactions (a block) over any other potential competitor block. It is achieved thanks to a cryptographically secure proof. This scheme is known as a proof-of-work and is discussed in detail in section 10.4.

Formally, we expand to:

$$\begin{aligned}
(2) \quad \mathbb{S}_{t+1} &\equiv \Pi(\mathbb{S}_t, B) \\
(3) \quad B &\equiv (\dots, [T_0, T_1, \dots]) \\
(4) \quad \Pi(\mathbb{S}, B) &\equiv \Omega(B, \Upsilon(\Upsilon(\mathbb{S}, T_0), T_1) \dots)
\end{aligned}$$

Where Ω is the block-finalisation state transition function (a function that rewards a nominated party); B is this block, which includes a series of transactions amongst some other components; and Π is the block-level state-transition function.

This is the basis of the blockchain paradigm, a model that forms the backbone of not only Ethereum, but all decentralised consensus-based transaction systems to date.

2.1. Value. In order to incentivise computation within the network, there needs to be an agreed method for transmitting value. To address this issue, Ethereum has an intrinsic currency, Ether, known also as ETH. The smallest subdenomination of Ether, and thus the one in which all integer values of the currency are counted, is the Wei. One Ether is defined as being 10^{18} Wei. There exist other subdenominations of Ether:

Multiplier	Name
10^0	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

Throughout the present work, any reference to value, in the context of Ether, currency, a balance or a payment, should be assumed to be counted in Wei.

2.2. Which History? Since the system is decentralised and all parties have an opportunity to create a new block on some older pre-existing block, the resultant structure is necessarily a tree of blocks. In order to form a consensus as to which path, from root (the genesis block) to leaf (the block containing the most recent transactions) through this tree structure, known as the blockchain, there must be an agreed-upon scheme. If there is ever a disagreement between nodes as to which root-to-leaf path down the block tree is the ‘best’ blockchain, then a *fork* occurs.

This would mean that past a given point in time (block), multiple states of the system may coexist: some nodes believing one block to contain the canonical transactions, other nodes believing some other block to be canonical, potentially containing radically different or incompatible transactions. This is to be avoided at all costs as the uncertainty that would ensue would likely kill all confidence in the entire system.

The scheme we use in order to generate consensus is a simplified version of the GHOST algorithm[?]. This process is described in detail in section 9.

3. BLOCKS, STATE AND TRANSACTIONS

Having introduced the basic concepts behind Ethereum, we will discuss the meaning of a transaction, a block and the state in more detail.

3.1. World State. The world state (*state*, see Appendix G), is a mapping between addresses (160-bit identifiers) and account states (a data structure serialised as RLP, see Appendix C). Though not stored on the blockchain, it is assumed that the implementation will maintain this mapping in a modified Merkle Patricia tree (*trie*, see Appendix D). The trie requires a simple database backend that maintains a mapping of bytearrays to bytearrays; we name this underlying database the state database. This has a number of benefits; firstly the root node of this structure is cryptographically dependent on all internal data and as such its hash can be used as a secure identity for the entire system state. Secondly, being an immutable data structure, it allows any previous state (whose root hash is known) to be recalled by simply altering the root hash accordingly. Since we store all such root hashes in the blockchain, we are able to trivially revert to old states.

The account state comprises the first two, and potentially the last two, of the following fields:

nonce: A scalar value equal to the number of transactions sent from this address, or, in the case of contract accounts, the number of contract-creations made by this account.

balance: A scalar value equal to the number of wei owned by this address.

stateRoot: A 256-bit hash equal to the root node of a further trie structure that encodes the storage contents of the contract. Though a separate data structure, this trie is still stored using the same underlying state database. This trie takes the form as a simple mapping between domains of 256-bit values.

codeHash: The hash of the EVM code of this contract—this is the code that gets executed should this address receive a call; it is immutable and thus, unlike all other fields, cannot be changed after construction. All such code fragments are contained in the state database under their corresponding hashes for later retrieval.

If the latter two fields are missing, the node represents a simple (non-contract) account: it maintains no state and executes no code on receipt of a transaction.

3.2. The Transaction. A transaction (formally, T) is a single cryptographically signed instruction sent by an actor external to Ethereum. An external actor can be a person (via a mobile device or desktop computer) or could be from a piece of automated software running on a server. There are two types of transactions: those which result in message calls and those which result in the creation of new contracts. Both types specify a number of common fields:

nonce: A scalar value equal to the number of transactions sent by the sender; formally T_n .

value: A scalar value equal to the number of wei to be transferred to the message call’s recipient or, in the case of contract creation, as an endowment to the newly created contract’s account; formally T_v .

gasPrice: A scalar value equal to the number of wei to be paid per unit of gas for all computation costs incurred as a result of the execution of this transaction; formally T_p .

gasLimit: A scalar value equal to the maximum amount of gas that should be used in executing this transaction. This is paid up-front, before any computation is done and may not be topped-up later; formally T_g .

to: The 160-bit address of the message call's recipient or the zero address for a contract-creation transaction; formally T_r .

v, r, s: Three scalar values corresponding to the signature of the transaction and used to determine the sender of the transaction.

Additionally, a contract-creation transaction contains:

body: An unlimited size byte array specifying the EVM-code for the contract, formally T_b .

init: An unlimited size byte array specifying the EVM-code for the contract initialisation procedure, formally T_i .

Both **body** and **init** are EVM-code fragments; **body** is executed each time the contract account receives a message call (either through a transaction or due to the internal execution of code). **init** is a piece of code that is executed only once at contract creation and that gets discarded immediately thereafter.

Whereas a message call transaction contains:

data: An unlimited size byte array specifying the input data of the message call, formally T_d .

Appendix H specifies the function mapping transactions to the sender S , which happens through the ECDMA of the SECP-256k1 curve, using the hash of the transaction as the datum to sign. For the present we simply assert that the sender of a given transaction T can be represented with $S(T)$.

3.3. The Block. The block in Ethereum is the collection of relevant pieces of information (known as the block *header*), H , together with a set of transactions, $\mathbf{T}$, and a set of other block headers $\mathbf{U}$ that are known to have a parent equal to the present block's parent's parent (such blocks are known as *uncles*). The block header contains several pieces of information:

parentHash: The SHA3 256-bit hash of the parent block, in its entirety; formally H_p .

unclesHash: The SHA3 256-bit hash of the uncles list portion of this block; formally H_u .

coinbase: The 160-bit address to which all fees collected from the successful mining of this block be transferred; formally H_b .

stateRoot: The SHA3 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied; formally H_r .

transactionsHash: The SHA3 256-bit hash of the transactions list portion of this block; formally H_t .

difficulty: A scalar value corresponding to the difficulty level of this block. This can be calculated from the previous block's difficulty level and the timestamp; formally H_d .

timestamp: A scalar value equal to the reasonable output of Unix's time() at this block's inception; formally H_s .

extraData: An arbitrary byte array containing data relevant to this block. Generally 256-bit or

less, but could be much larger in the genesis block. Not formally used.

nonce: A 256-bit hash which proves that a sufficient amount of computation has been carried out on this block; formally H_n .

The other two components in the block are simply a list of uncle block headers (of the same format as above) and a list of the transactions (as defined previously). Formally, we can refer to a block B :

$$(5) \quad B \equiv (H, \mathbf{U}, \mathbf{T})$$

We can assert its validity if and only if it satisfies several conditions: It must be internally consistent with the uncle and transaction block hashes; the given transactions $\mathbf{T}$, when executed in order on the base state $\mathbb{S}$ result in a new state of the identity H_r :

$$(6) \quad \text{SHA3}(\text{RLP}(\mathbf{U})) \equiv H_u$$

$$(7) \quad \wedge \quad \text{SHA3}(\text{RLP}(\mathbf{T})) \equiv H_t$$

$$(8) \quad \wedge \quad r(\Pi(\mathbb{S}, B)) \equiv H_r$$

where $r(\mathbb{S})$ is the root node hash of the Merkle Patricia tree structure containing the key-value pairs of $\mathbb{S}$ with values encoded using RLP.

3.3.1. Block Header Validity. We define $P(B_H)$ to be the parent block of B , formally:

$$(9) \quad P(H) \equiv B' : \text{SHA3}(\text{RLP}(B')) = H_p$$

The canonical difficulty of a block of header H , is defined as $D(H)$:

$$(10) \quad D(H) \equiv \begin{cases} 2^{22} & \text{if genesis block} \\ P(H)_{H_d} + \frac{P(H)_{H_d} - 1024}{1024} & \text{if } H_t < P(H)_{H_t} + 42 \\ P(H)_{H_d} - \frac{P(H)_{H_d} - 1024}{1024} & \text{otherwise} \end{cases}$$

H_t is the timestamp of block H , must fulfill the relation:

$$(11) \quad H_t > P(H)_{H_t}$$

This mechanism enforces a homeostasis in terms of the time between blocks; a smaller period between the last two blocks results in an increase in the difficulty level and thus additional computation required, lengthening the likely next period. Conversely, if the period is too large, the difficulty, and expected time to the next block, is reduced.

The nonce, H_n , must satisfy the relation:

$$(12) \quad \text{PoW}(H, H_n) \leq \frac{2^{256}}{H_d}$$

Where **PoW** is the proof-of-work function (see section 10.4): this evaluates to an pseudo-random number cryptographically dependent on the parameters H and H_n . Given an approximately uniform distribution in the range $[0, 2^{256})$, the expected time to find a solution is proportional to the difficulty, H_d .

This is the foundation of the security of the blockchain and is the fundamental reason why a malicious node cannot propagate newly created blocks that would otherwise overwrite ("rewrite") history. Because the nonce must satisfy this requirement, and because satisfaction of this requirement depends on the contents of the block and in turn its composed transactions, creating new, valid, blocks is difficult, and, over time, requires approximately the total compute power of the trustworthy portion of the mining peers.

Thus we are able to define the block header validity function $V(H)$:

$$(13) \quad V(H) \equiv \text{PoW}(H, H_n) \leq \frac{2^{256}}{H_d}$$

$$(14) \quad \wedge \quad H_d = D(H)$$

$$(15) \quad \wedge \quad H_t > P(H)_{H_t}$$

4. GAS AND PAYMENT

In order to avoid issues of network abuse and to side-step the inevitable questions stemming from Turing completeness, all programmable computation in Ethereum is subject to fees. The fee schedule is specified in units of *gas* (see Appendix B for the fees associated with various computation). Thus any given fragment of programmable computation (this includes creating contracts, making message calls, utilising and accessing contract storage and executing operations on the virtual machine) has a universally agreed cost in terms of gas.

Every transaction has a specific amount of gas associated with it: **gasLimit**. This is the amount of gas which is implicitly purchased from the sender's account balance. The purchase happens at the according **gasPrice**, also specified in the transaction. The transaction is considered invalid if the account balance cannot support such a purchase. It is named **gasLimit** since any unused gas at the end of the transaction is refunded (at the same rate of purchase) to the sender's account. Gas does not exist outside of the execution of a transaction. Thus for trusted contracts, a relatively high gas limit may be set and left alone.

Any ether used to purchase gas that is not refunded is delivered to the *coinbase* address, the address of an account typically under the control of the miner. Transactors are free to specify any **gasPrice** that they wish, however miners are free to ignore transactions as they choose. A higher gas price on a transaction will therefore cost the sender more in terms of Ether and deliver a greater value to the miner and thus will more likely be selected for inclusion by more miners. Miners, in general, will choose to advertise the minimum gas price for which they will execute transactions and transactors will be free to canvas these prices in determining what gas price to offer. Since there will be a (weighted) distribution of minimum acceptable gas prices, transactors will necessarily have a trade-off to make between lowering the gas price and maximising the chance that their transaction will be mined in a timely manner.

5. TRANSACTION EXECUTION

The execution of a transaction is the most complex part of the Ethereum protocol: it defines the state transition function Υ . It is assumed that any transactions executed first pass the initial tests of intrinsic validity. These include:

- (1) The transaction signature is valid;
- (2) the transaction nonce is valid (equivalent to the sender account's current nonce);
- (3) the gas limit is no smaller than the intrinsic gas, g_0 , used by the transaction;
- (4) the sender account balance contains at least the cost, v_0 , required in up-front payment;

Formally, we consider the function Υ , with T being a transaction and $\mathbb{S}$ the state:

$$(16) \quad \mathbb{S}' = \Upsilon(\mathbb{S}, T)$$

Thus $\mathbb{S}'$ is the post-transactional state.

We define intrinsic gas g_0 , the amount of gas this transaction requires to be paid prior to execution, as follows:

$$(17) \quad g_0 \equiv \begin{cases} (|T_i| + |T_b|)G_{data} + G_{create} & \text{if } T_r = 0 \\ |T_d|G_{data} + G_{call} & \text{otherwise} \end{cases}$$

where $|T_d|$, $|T_i|$ and $|T_b|$ are the sizes, in bytes, of the transaction's associated data, initialisation and body EVM-code, respectively and G is defined in Appendix B.

The up-front cost v_0 is calculated as:

$$(18) \quad v_0 \equiv T_g T_p + T_v$$

the validity is determined as:

$$(19) \quad S(T) \neq \emptyset$$

$$(20) \quad \wedge \quad \mathbb{S}[S(T)] \neq \emptyset$$

$$(21) \quad \wedge \quad T_n = \mathbb{S}[S(T)]_{nonce}$$

$$(22) \quad \wedge \quad g_0 \leq T_g$$

$$(23) \quad \wedge \quad v_0 \leq \mathbb{S}[S(T)]_{balance}$$

The transaction must not require more gas to be utilised in this block than the global block limit of gas, specified at 10^6 .

The execution of a valid transaction begins with an irrevocable change made to the state: the nonce of the account of the sender ($S(T)$) is incremented by one and the balance is reduced by the up-front cost, v_0 . The gas available for the proceeding computation, g , is defined as $T_g - g_0$. The computation, whether contract creation or a message call, results in an eventual state (which may legally be equivalent to the current state), the change to which is deterministic and never invalid: there can be no invalid transactions from this point.

We define the rollback state $\mathbb{S}_0$:

$$(24) \quad \mathbb{S}_0 \equiv \mathbb{S} \text{ except:}$$

$$(25) \quad \mathbb{S}_0[S(T)]_{balance} \equiv \mathbb{S}[S(T)]_{balance} - c$$

$$(26) \quad \mathbb{S}_0[S(T)]_{nonce} \equiv \mathbb{S}[S(T)]_{nonce} + 1$$

Evaluating $\mathbb{S}_0$ from $\mathbb{S}_0$ depends on the transaction type; either contract creation or message call; we define the pairing of post-execution provisional state ($\mathbb{S}_P$) and remaining gas (g'):

$$(27) \quad (\mathbb{S}_P, g') \equiv \begin{cases} \Lambda(\mathbb{S}_0, S(T), T_o, g, T_p, T_v, T_i, T_b) & \text{if } T_r = 0 \\ \Theta^*(\mathbb{S}_0, S(T), T_o, T_r, g, T_p, T_v, T_d) & \text{otherwise} \end{cases}$$

where

$$(28) \quad g \equiv T_g - g_0$$

Note we use Θ^* to denote the fact that only the first two components of the function's value are taken; the third represents the message-call's output value (a byte array) and is unused in the context of transaction evaluation.

After the message call or contract creation is processed, the state is finalised by refunding g' , the remaining gas, to the sender at the original rate. The Ether for the gas that was actually used is given to the miner, whose address is specified as the coinbase of the present block B . So we

define the final state $\mathbb{S}'$ in terms of the provisional state $\mathbb{S}_P$:

$$(29) \quad \mathbb{S}' \equiv \mathbb{S}_P \text{ except}$$

$$(30) \quad \mathbb{S}'[s]_{balance} \equiv \mathbb{S}_P[s]_{balance} + g'T_p$$

$$(31) \quad \mathbb{S}'[m]_{balance} \equiv \mathbb{S}_P[m]_{balance} + (g - g')T_p$$

$$(32) \quad m \equiv B_{H_b}$$

In the case that $s = m$ then we simply return the Ether back to the sender/miner, collapsing the exception into:

$$(33) \quad \mathbb{S}'[s]_{balance} \equiv \mathbb{S}_P[s]_{balance} + g$$

6. CONTRACT CREATION

There are number of intrinsic parameters used when creating a contract: sender (s), nonce (o), available gas (g), gas price (p), endowment (v) together with the two arbitrary length byte arrays, $\mathbf{i}$ for the initialisation EVM code and $\mathbf{b}$ for the EVM code of the body.

We define the creation function formally as the function Λ , which evaluates from these values, together with the state $\mathbb{S}_0$ to the tuple containing the new state, additional database entries and remaining gas ($\mathbb{S}_P, g'$), as in 5:

$$(34) \quad (\mathbb{S}_P, g') \equiv \Lambda(\mathbb{S}_0, s, o, g, p, v, \mathbf{i}, \mathbf{b})$$

The address of the new contract is defined as being the rightmost 160 bits of the SHA3 hash of RLP encoding of the structure [**sender**, **nonce**]. In the unlikely event that the address is already in use, it is treated as a big-endian integer and incremented by one until an unused address is arrived at. Thus we define the creation address function A :

$$(35) \quad A(s, n) \equiv a \text{ where:}$$

$$(36) \quad a = \arg \min_x : x \geq a' \wedge \mathbb{S}_0[x] = \emptyset$$

$$(37) \quad a' = \mathcal{B}_{96..255}(\text{SHA3}(\text{RLP}(\langle s, n \rangle)))$$

where SHA3 is the SHA3 256-bit hash function, RLP is the RLP encoding function, $\mathcal{B}_{a..b}(X)$ evaluates to binary value containing the bits of indices in the range $[a, b]$ of the binary data X and $\mathbb{S}_0[x]$ is the address state of x or $\emptyset$ if none exists. Note we use one fewer than the sender's nonce value; we assert that we have incremented the sender account's nonce prior to this call, and so the value used is the sender's nonce at the beginning of the responsible transaction or VM operation.

The account's nonce is initially defined as zero, the balance as the value passed, the storage as empty and the code hash as the SHA3 256-bit hash of the code, thus the mutated state becomes $\mathbb{S}^*$:

$$(38) \quad \mathbb{S}^* \equiv \mathbb{S} \text{ except:}$$

$$(39) \quad \mathbb{S}^*[A(s, \mathbb{S}[s]_{nonce})] \equiv (0, v, \emptyset, \text{SHA3}(\mathbf{b}))$$

It is asserted that the state database will also change such that it defines the pair $(\text{SHA3}(\mathbf{b}), \mathbf{b})$.

Finally, the contract is initialised through the execution of the initialising EVM code $\mathbf{i}$ according to the execution model (see section 8). Code execution can effect several events that are not internal to the execution state: the contract's storage can be altered, further contracts can be created and further message calls can be made. As such, the code execution function Ξ evaluates to a tuple of the resultant state $\mathbb{S}'$ and available gas remaining g' .

Code execution depletes gas; thus it may exit before the code has come to a natural halting state. In this exceptional case we say an Out-of-Gas exception has occurred: The evaluated state is defined as being the empty set $\emptyset$ and the entire create operation should have no effect on the state, effectively leaving it as it was immediately prior to attempting the creation. The gas remaining should of course be zero. If the creation was conducted as the manifestation of a transaction, then this doesn't affect payment of the intrinsic cost: it is paid regardless.

If such an exception does not occur, then the remaining gas is refunded to the originator and the now-altered state is allowed to persevere. Thus formally, we may specify the new state and gas as $\mathbb{S}', g'$ where:

$$(40) \quad (\mathbb{S}^{**}, g') \equiv \Xi(\mathbb{S}^*, g, I)$$

$$(41) \quad \mathbb{S}_P \equiv \begin{cases} \mathbb{S}_0 & \text{if } \mathbb{S}^{**} = \emptyset \\ \mathbb{S}^{**} & \text{otherwise} \end{cases}$$

$$(42) \quad I_a \equiv a$$

$$(43) \quad I_o \equiv s$$

$$(44) \quad I_p \equiv p$$

$$(45) \quad I_d \equiv []$$

$$(46) \quad I_s \equiv s$$

$$(47) \quad I_v \equiv v$$

$$(48) \quad I_b \equiv \mathbf{i}$$

I_d evaluates to the empty array. I_H and I_n have no special treatment and are determined from the blockchain.

7. MESSAGE CALL

In the case of executing a message call, several parameters are required: sender (s), transaction originator (o), recipient (r), available gas (g), value (v) and gas price (p) together with an arbitrary length byte array, $\mathbf{d}$, the input data of the call. Aside from evaluating to a new state and additional database entries, message calls also have an extra component—the output data denoted by the byte array $\mathbf{o}$. This is ignored when executing transactions, however message calls can be initiated due to VM-code execution and in this case this information is used.

$$(49) \quad (\mathbb{S}_P, g', \mathbf{o}) \equiv \Theta(\mathbb{S}, s, o, r, g, p, v, \mathbf{d})$$

We define $\mathbb{S}_1$, the rollback state as the original state but with the value transferred to the recipient:

$$(50) \quad \mathbb{S}_1 \equiv \mathbb{S} \text{ except:}$$

$$(51) \quad \mathbb{S}_1[r]_{balance} \equiv \mathbb{S}_1[r]_{balance} + v$$

If the recipient account contains no code to be executed (i.e. it's not a contract), then we define the final state as being equivalent to the rollback state. In the case that the recipient contains code to be executed (i.e. that the account is a contract), then the contract's body code (identified as the fragment whose SHA3 hash is $\mathbb{S}[r]_{code}$) is executed according to the execution model (see section 8). Just as with contract creation, if the execution halts due to an exhausted gas supply, then no gas is refunded

to the caller and the state is reverted to the point immediately prior to code execution.

$$(52) \quad \mathbb{S}' \equiv \begin{cases} \mathbb{S}_1 & \text{if } \mathbb{S}_1[r]_{code} = \emptyset \\ \mathbb{S}_1 & \text{if } \mathbb{S}^{**} = \emptyset \\ \mathbb{S}^{**} & \text{otherwise} \end{cases}$$

$$(53) \quad (\mathbb{S}^{**}, g', \mathbf{o}) \equiv \Xi(\mathbb{S}_1, g, I)$$

$$(54) \quad I_a \equiv a$$

$$(55) \quad I_o \equiv o$$

$$(56) \quad I_p \equiv p$$

$$(57) \quad I_d \equiv \mathbf{d}$$

$$(58) \quad I_s \equiv s$$

$$(59) \quad I_v \equiv v$$

$$(60) \text{ Let } \text{SHA3}(I_b) = \mathbb{S}[r]_{code}$$

We assert that the state database contains the entry $(\text{SHA3}(I_b), I_b)$.

8. EXECUTION MODEL

The execution model specifies how the system state is altered given a series of bytecode instructions and a small tuple of environmental data. This is specified through a formal model of a virtual state machine, known as the Ethereum Virtual Machine (EVM). It is a *quasi*-Turing-complete machine; the *quasi* qualification comes from the fact that the computation is intrinsically bounded through a parameter, *gas*, which limits the total amount of computation done.

8.1. Basics. The EVM is a simple stack-based architecture. The word size of the machine (and thus size of stack item) is 256-bit. This was chosen to facilitate the SHA3-256 hash scheme and elliptic-curve computations. The memory model is a simple word-addressed byte array. The stack has an unlimited size. The machine also has an independent storage model; this is similar in concept to the memory but rather than a byte array, it is a word-addressable word array. Unlike memory, which is volatile, storage is non volatile and is maintained as part of the system state in the state database. All locations in both storage and memory are well-defined initially as zero.

The machine does not follow the standard von Neumann architecture. Rather than storing program code in memory or storage, it is stored separately in a virtual ROM. It cannot be read directly; instead it exists only as a model for determining the next instruction to execute.

The machine can have exceptional execution for several reasons, including stack underflows and invalid instructions. These unambiguously and validly result in immediate halting of the machine with all state changes left intact. The one piece of exceptional execution that does not leave state changes intact is the out-of-gas (OOG) exception. Here, the machine halts immediately and reports the issue to the execution agent (either the transaction processor or, recursively, the spawning execution environment) and which will deal with it separately.

8.2. Fees Overview. Fees (denominated in gas) are charged under three distinct circumstances, all three as prerequisite to the execution of an operation. The first and most common is the fee intrinsic to the computation

of the operation. Most operations require a single gas fee to be paid for their execution; exceptions include SSTORE, SLOAD, CALL, CREATE, BALANCE and SHA3. Secondly, gas may be deducted in order to form the payment for a subordinate message call or contract creation; this forms part of the payment for CREATE and CALL. Finally, gas may be paid due to an increase in the usage of the memory.

Over a contract's execution, the total fee for memory-usage payable is proportional to smallest multiple of 32 bytes that are required such that all memory indices (whether for read or write) are included in the range. This is paid for on a just-in-time basis; as such, referencing an area of memory at least 32 bytes greater than any previously indexed memory will certainly result in an addition memory usage fee. Due to this fee it's highly unlikely addresses will ever go above 32-bit bounds since at the present price of ether and default gas price, that would cost around US\$20M for the memory fee alone.

Storage fees have a slightly nuanced behaviour—to incentivise contracts to minimise storage use (which corresponds directly to a larger state database on all nodes), the execution fee for an operation that clears an entry in the storage is waived; in fact, it is effectively paid up-front since the initial usage of a storage location costs twice as much as the normal usage.

More formally, given an instruction, it is possible to calculate the gas cost of executing it as follows:

- SHA3 costs G_{sha3} gas
- SLOAD costs G_{sload} gas
- BALANCE costs $G_{balance}$ gas
- SSTORE costs $d \cdot G_{sstore}$ gas where:
 - $d = 2$ if the new value of the storage is non-zero and the old is zero;
 - $d = 0$ if the new value of the storage is zero and the old is non-zero;
 - $d = 1$ otherwise.
- CALL costs G_{call} , though additional gas may be taken for the execution of contract body code.
- CREATE costs G_{create} , though additional gas may be taken for the execution of contract initialisation code.
- STOP costs G_{stop} gas
- All other operations cost G_{step} gas.

Additionally, when memory is accessed with MSTORE, MSTORE8, MLOAD, RETURN, SHA3, CREATE or CALL, the memory should be enlarged to the smallest multiple of words such that all addressed bytes now fit in it. See Appendix I for a rigorous definition of the EVM gas cost.

8.3. Execution Environment. In addition to the system state $\mathbb{S}$, and the remaining gas for computation g , there are several pieces of important information used in the execution environment that the execution agent must provide; these are contained in the tuple I :

- I_a , the address of the account which owns the code that is executing.
- I_o , the sender address of the transaction that originated this execution.
- I_p , the price of gas in the transaction that originated this execution.
- I_d , the byte array that is the input data to this execution; if the execution agent is a transaction, this would be the transaction data.

- I_s , the address of the account which caused the code to be executing; if the execution agent is a transaction, this would be the transaction sender.
- I_v , the value passed to this account as part of the same procedure as execution; if the execution agent is a transaction, this would be the transaction value.
- I_b , the byte array that is the machine code to be executed.
- I_H , the block header of the previous block.
- I_n , the block number of the current block.

The execution model defines the function Ξ , which can compute the resultant state $\mathbb{S}'$ and the remaining gas g' , given these definitions:

$$(61) \quad (\mathbb{S}', g') \equiv \Xi(\mathbb{S}, g, I)$$

8.4. Execution Overview. We must now define the Ξ function. In most practical implementations this will be modelled as an iterative progression of the pair comprising the full system state, $\mathbb{S}$ and the machine state, $\mathbb{M}$. Formally, we define it recursively with a function X . This uses an iterator function O (which defines the result of a single cycle of the state machine) together with functions Z which determines if the present state is an exceptional halting state of the machine and H , specifying the output data of the instruction if and only if the present state is a normal halting state of the machine. We consider the empty sequence (denoted $\square$ to be non-equal to the empty set, denoted $\emptyset$).

$$(62) \quad \Xi(\mathbb{S}, g, I) \equiv X(\mathbb{S}, \mathbb{M}, I)$$

$$(63) \quad \mathbb{M}_g \equiv g$$

$$(64) \quad \mathbb{M}_{pc} \equiv 0$$

$$(65) \quad \mathbb{M}_M \equiv [0, 0, \dots]$$

$$(66) \quad \mathbb{M}_i \equiv 0$$

$$(67) \quad \mathbb{M}_S \equiv \square$$

$$(68) \quad X(\mathbb{S}, \mathbb{M}, I) \equiv \begin{cases} (\mathbb{S}, \mathbb{M}, I, \square) & \text{if } Z(\mathbb{S}, \mathbb{M}, I) \\ (O(\mathbb{S}, \mathbb{M}, I), \mathbf{o}) & \text{if } \mathbf{o} \neq \emptyset \\ X(O(\mathbb{S}, \mathbb{M}, I)) & \text{otherwise} \end{cases}$$

where

$$(69) \quad \mathbf{o} \equiv H(\mathbb{S}, \mathbb{M}, I)$$

The machine state $\mathbb{M}$ is defined as the tuple (g, pc, M, i, T) which are the gas available, the program counter, the memory contents, the active number of words in memory (counting continuously from position 0), and the stack contents. The memory contents $\mathbb{M}_M$ are a series of zeroes of size 2^{256} .

For the ease of reading, the instruction mnemonics, written in smallcaps (e.g. ADD), should be interpreted as their numeric equivalents; the full table of instructions and their specifics is given in Appendix I.

For the purposes of defining Z , H and O , we define w as the current operation to be executed:

$$(70) \quad w \equiv \begin{cases} I_b[\mathbb{M}_{pc}] & \text{if } \mathbb{M}_{pc} < \|I_b\| \\ \text{STOP} & \text{otherwise} \end{cases}$$

We also assume the fixed amounts of δ and α , specifying the stack movement items removed and added, both subscriptable on the instruction and an instruction cost function C evaluating to the full cost, in gas, of executing the given instruction.

8.4.1. Exceptional Halting. The exceptional halting function Z is defined as:

$$(71) \quad Z(\mathbb{S}, \mathbb{M}, I) \equiv \mathbb{M}_g < C(\mathbb{S}, \mathbb{M}, I) \vee \delta_w = \emptyset \vee \|\mathbb{M}_S\| < \delta_w$$

This states that the execution is in an exceptional halting state if there is insufficient gas, if the instruction is invalid (and therefore its δ subscript is undefined) or if there are insufficient stack items. The astute reader will realise that this implies that no instruction can, through its execution, cause an exceptional halt.

8.4.2. Normal Halting. We define the normal halting function H :

$$(72) \quad H(\mathbb{S}, \mathbb{M}, I) \equiv \begin{cases} H_{\text{RETURN}}(\mathbb{M}) & \text{if } w = \text{RETURN} \\ \square & \text{if } w \in \{\text{STOP}, \text{SUICIDE}\} \\ \emptyset & \text{otherwise} \end{cases}$$

The data-returning halt operation, RETURN, has a special function H_{RETURN} , defined in Appendix I.

8.5. The Execution Cycle. Stack items are added or removed from the left-most, lower-indexed portion of the series; all other items remain unchanged:

$$(73) \quad O(\mathbb{S}, \mathbb{M}, I) \equiv (\mathbb{S}', \mathbb{M}', I)$$

$$(74) \quad \Delta \equiv \alpha_w - \delta_w$$

$$(75) \quad \|\mathbb{M}'_T\| \equiv \|\mathbb{M}_T\| + \Delta$$

$$(76) \quad \forall x \in [\alpha_w, \|\mathbb{M}'_T\|) : \mathbb{M}'_{Tx} \equiv \mathbb{M}_{Tx+\Delta}$$

The gas is reduced by the instruction's gas cost and for most instructions, the program counter increments on each cycle, for the three exceptions, we assume a function J , subscripted by one of two instructions, which evaluates to the according value:

$$(77) \quad \mathbb{M}'_g \equiv \mathbb{M}_g - C(\mathbb{S}, \mathbb{M}, I)$$

$$(78) \quad \mathbb{M}'_{pc} \equiv \begin{cases} J_o(\mathbb{M}) & \text{if } w \in \{\text{JUMP}, \text{JUMPI}\} \\ \mathbb{M}_{pc} + p & \text{if } w \in \{\text{PUSH1}, \text{PUSH32}\} \\ \mathbb{M}_{pc} + 1 & \text{otherwise} \end{cases}$$

where p is the byte size of the push instruction, defined as:

$$(79) \quad p \equiv w - \text{PUSH1} + 2$$

In general, we assume the memory and system state don't change:

$$(80) \quad \mathbb{M}'_M \equiv \mathbb{M}_M$$

$$(81) \quad \mathbb{M}'_i \equiv \mathbb{M}_i$$

$$(82) \quad \mathbb{S}' \equiv \mathbb{S}$$

However, instructions do typically alter one or several components of these values. Altered components listed by instruction are noted in Appendix I, alongside values for α and δ and formal a description of the gas requirements.

9. BLOCKTREE TO BLOCKCHAIN

The canonical blockchain is a path from root to leaf through the entire block tree. In order to have consensus over which path it is, conceptually we identify the path that has had the most computation done upon it, or, the *heaviest* path. Clearly one factor that helps determine the heaviest path is the block number of the leaf, equivalent to the number of blocks, not counting the unmined genesis block, in the path. The longer the path, the greater the total mining effort that must have been done in order to arrive at the leaf. This is akin to existing schemes, such as that employed in Bitcoin-derived protocols.

This scheme notably ignores so-called *stale* blocks: valid, mined blocks, which were propagated too late into the network and thus were beaten to network consensus by a sibling block (one with the same parent). Such blocks become more common as the network propagation time approaches the ideal inter-block time. However, by counting the computation work of stale block headers, we are able to do better: we can utilise this otherwise wasted computation and put it to use in helping to buttress the more popular blockchain making it a stronger choice over less popular (though potentially longer) competitors.

This increases overall network security by making it much harder for an adversary to silently mine a canonical blockchain (which, it is assumed, would contain different transactions to the current consensus) and dump it on the network with the effect of reversing existing blocks and the transactions within.

In order to validate the extra computation, a given block B may include the block headers from any known uncle blocks (i.e. blocks whose parent is equivalent to the grandparent of B). Since a block header includes the nonce, a proof-of-work, then the header alone is enough to validate the computation done. Any such blocks contribute toward the total computation or *total difficulty* of a chain that includes them. To incentivise computation and inclusion, a reward is given both to the miner of the stale block and the miner of the block that references it.

Thus we define the total difficulty of block B recursively as:

$$(83) \quad B_t \equiv B'_t + B_d + \sum_{U \in B_U} U_d$$

$$(84) \quad B' \equiv P(H)$$

$$(85) \quad H \equiv B_H$$

As such given a block B , B_t is its total difficulty, B' is its parent block, B_d is its difficulty and B_U is its set of uncle blocks.

10. BLOCK FINALISATION

The process of finalising a block involves three stages:

- (1) Validate (or, if mining, determine) uncles.
- (2) Apply rewards.
- (3) Verify (or, if mining, compute a valid) state and nonce.

10.1. Uncle Validation. The validation of uncle headers means nothing more than verifying that each uncle header is both a valid header and satisfies the relation of uncle to

the present block. Formally:

$$(86) \quad \bigwedge_{U \in B_U} V(U) \wedge P(U) = P(P(B_H)) \wedge P(B_H) \neq B$$

10.2. Reward Application. The application of rewards to a block involves raising the balance of the accounts of the coinbase address of the block and each uncle by a certain amount. We raise the block's coinbase account by R_b , the block reward, and the coinbase of each uncle by $\frac{7}{8}$ of that. Formally we define the function Ω :

$$(87) \quad \Omega(B, \mathbb{S}) \equiv \mathbb{S}' : \mathbb{S}' = \mathbb{S} \text{ except:}$$

$$(88) \quad \mathbb{S}'[B_{Hb}] = \mathbb{S}[B_{Hb}] + R_b$$

$$(89) \quad \forall U \in B_U : \mathbb{S}'[U_b] = \mathbb{S}[U_b] + \frac{7}{8} R_b$$

We define the block reward as 1500 Finney:

$$(90) \quad \text{Let } R_b = 1.5 * 10^{16}$$

10.3. State & Nonce Validation. We may now define the function, Γ , that maps a block B to its initiation state:

$$(91) \quad \Gamma(B) \equiv \begin{cases} \mathbb{S}_0 & \text{if } P(B_H) = \emptyset \\ \mathbb{S}_i : r(\mathbb{S}_i) = P(B_H)_{H_r} & \text{otherwise} \end{cases}$$

Here, $r(\mathbb{S}_i)$ means the hash of the root node of a trie of state $\mathbb{S}_i$; this is stored in the state database since the trie is a mutable data structure.

And finally define Φ , the block transition function, which maps an incomplete block B to a complete block B' :

$$(92) \quad \Phi(B) \equiv B' : B' = B^* \text{ except:}$$

$$(93) \quad B'_n = n : \text{PoW}(B^*, n) < \frac{2^{256}}{H_d}$$

$$(94) \quad B^* \equiv B \text{ except: } B'_r = r(\Pi(\Gamma(B), B))$$

As specified at the beginning of the present work, Π is the state-transition function, which is defined in terms of Ω , the block finalisation function and Υ , the transaction-evaluation function, both now well-defined:

$$(95) \quad \Pi(\mathbb{S}, B) \equiv \Omega(B, \Upsilon(\Upsilon(\mathbb{S}, T_0), T_1) \dots)$$

$$(96) \quad B \equiv (H, [T_0, T_1, \dots], \mathbf{U})$$

Thus the complete block-transition mechanism, less PoW , the proof-of-work function is defined.

10.4. Mining Proof-of-Work. The mining proof-of-work (PoW) exists as a cryptographically secure nonce that proves beyond reasonable doubt that a particular amount of computation has been expended in the determination of some token value n . It is utilised to enforce the blockchain security by giving meaning and credence to the notion of difficulty (and, by extension, total difficulty). However, since mining new blocks comes with an attached reward, the proof-of-work not only functions as a method of securing confidence that the blockchain will remain canonical into the future, but also as a wealth distribution mechanism.

For both reasons, there are two important goals of the proof-of-work function; firstly, it should be as accessible as possible to as many people as possible. The requirement of or reward from specialised and uncommon hardware should be minimised. This makes the distribution model as open as possible, and, ideally, makes the act of mining a simple swap from electricity to ether at roughly the same rate for anyone around the world.

Secondly, it should not be possible to make super-linear profits, and especially not so with a high initial barrier. Such a mechanism allows a well-funded adversary to gain a troublesome amount of the network's total mining power and as such gives them a super-linear reward (thus skewing distribution in their favour) as well as reducing the network security.

One plague of the Bitcoin world is ASICs. These are specialised pieces of compute hardware that exist only to do a single task. In Bitcoin's case the task is the SHA256 hash function. While ASICs exist for a proof-of-work function, both goals are placed in jeopardy. Because of this, a proof-of-work function that is ASIC-resistant (i.e. difficult or economically inefficient to implement in specialised compute hardware) has been identified as the proverbial silver bullet.

Two directions exist for ASIC resistance; firstly make it sequential memory-hard, i.e. engineer the function such that the determination of the nonce requires a lot of memory and that the memory cannot be used in parallel to discover multiple nonces simultaneously. The second is to make the type of computation it would need to do general-purpose; the meaning of "specialised hardware" for a general-purpose task set is, naturally, general purpose hardware and as such commodity desktop computers are likely to be pretty close to "specialised hardware" for the task.

More formally, the proof-of-work function takes the form of PoW:

$$(97) \quad \text{PoW}(H_{\mathcal{H}}, n) < \frac{2^{256}}{H_d}$$

Where $H_{\mathcal{H}}$ is the new block's header H , but *without* the nonce component; H_d is the new block's difficulty value (i.e. the block difficulty from section 9).

As of the proof-of-concept (PoC) series of the Ethereum software, the proof-of-work function is simplistic and does not attempt to secure these goals. It will be described here for completeness.

10.4.1. *PoC Series.* For the PoC series, we use a simplified proof-of-work. This is not ASIC resistant and is meant merely as a placeholder. It utilises the bare SHA3 hash function to secure the block chain by requiring the SHA3 hash of the concatenation of the nonce and the header's SHA3 hash to be sufficiently low.

It is formally defined as PoW:

$$(98) \quad \text{PoW}(H, n) \equiv \text{BE}(\text{SHA3}(\text{SHA3}(\text{RLP}(H_{\mathcal{H}})) \circ n))$$

where: $\text{RLP}(H_{\mathcal{H}})$ is the RLP encoding of the block header H , not including the final nonce component; SHA3 is the SHA3 hash function accepting an arbitrary length series of bytes and evaluating to a series of 32 bytes (i.e. 256-bit); n is the nonce, a series of 32 bytes; $\circ$ is the series concatenation operator; $\text{BE}(X)$ evaluates to the value equal to X when interpreted as a big-endian-encoded integer.

10.4.2. *Release Series.* For the release series, we use a more complex proof-of-work. This has yet to be formally

defined, but involves two components; firstly that it concerns contract processing, and by extension the evaluation of programs on the EVM. Secondly that it concerns the utilisation of either the blockchain or the full state trie.

As an overview, the output of the function is based upon the system state, defined as the hash of the root node of the state trie. A set of transactions, pseudo-randomly determined from the nonce value and selected from the last N blocks is taken. N is large enough and the selection criteria are such that execution of the transactions requires some non-negligible amount of processing by the EVM. Whenever code is executed on the EVM, it is pseudo-randomly (seeded again by the nonce) corrupted before alteration. Corruption could involve switching addresses with other transactions or rotating them through in the state trie (perhaps to the next address with the same order of magnitude of funds), rotating through instructions that have equivalent stack behaviour (e.g. swapping ADD for SUB or GT for EQ), or more destructive techniques such as randomly changing opcodes. This results in a problem that both require generalised computation hardware and is sequentially memory (and perhaps even disk) hard.

Any specialised hardware to perform this task could also be leveraged to speed up (and thus drive down costs) of general Ethereum transaction processing.

11. IMPLEMENTING CONTRACTS

11.1. **Data Feeds.** External server runs node. Creates & signs a transaction every minute containing new data. Sends to contract which knows to accept data only from this address. Allows polling of data from all other contracts (possibly for a microfee).

11.2. **Random Numbers.** Hash the block timestamp; for a series of pseudorandoms, take the previous, add some constant and hash the result.

For a more secure pseudo-random offering both parties agree on a number of random data feed contracts; these are concatenated along with the block timestamp and hashed to produce the first number in the series.

12. FUTURE DIRECTIONS

Rather than being hard-specified in the protocol, the block limit will become based upon some fixed ratio about a long-term moving average of actual gas usage in blocks, allowing it to slowly increase according to the network's ability to manage the transaction volume.

State database won't keep everything; maintains an age and eventually throws out nodes that are not recent enough or checkpoints.

Nodes in state trie that haven't sent/received a transaction in over X blocks could be thrown out, reducing Ether-leakage and reducing the growth of the state database.

13. CONCLUSION

This paper has introduced, discussed and formally defined the protocol of Ethereum.

APPENDIX A. TERMINOLOGY

External Actor: A person or other entity able to interface to an Ethereum node, but external to the world of Ethereum. It can interact with Ethereum through depositing signed Transactions and inspecting the blockchain and associated state. Has one (or more) intrinsic Accounts.

Address: A 160-bit code used for identifying Accounts.

Account: Accounts have an intrinsic balance and transaction count maintained as part of the Ethereum state. They are owned either by External Actors or intrinsically (as an identity) an Autonomous Object within Ethereum. If an Account identifies an Autonomous Object, then Ethereum will also maintain a Storage State particular to that Account. Each Account has a single Address that identifies it.

Transaction: A piece of data, signed by an External Actor. It represents either a Message or a new Autonomous Object. Transactions are recorded into each block of the blockchain.

Autonomous Object: A virtual object existant only within the hypothetical state of Ethereum. Has an intrinsic address. Incorporated only as the state of the storage component of the VM.

Storage State: The information particular to a given Autonomous Object that is maintained between the times that it runs.

Message: Data (as a set of bytes) and Value (specified as Ether) that is passed between two Accounts in a perfectly trusted way, either through the deterministic operation of an Autonomous Object or the cryptographically secure signature of the Transaction.

Message Call: The act of passing a message from one Account to another. If the destination account is an Autonomous Object, then the VM will be started with the state of said Object and the Message acted upon. If the message sender is an Autonomous Object, then the Call passes any data returned from the VM operation.

Gas: The fundamental network cost unit. Paid for exclusively by Ether (as of PoC-4), which is converted freely to and from Gas as required. Gas does not exist outside of the internal Ethereum computation engine; its price is set by the Transaction and miners are free to ignore Transactions whose Gas price is too low.

Contract: Synonym for Autonomous Object used for non-technical audiences.

Object: Synonym for Autonomous Object.

App: An end-user-visible application hosted in the Ethereum Browser.

Ethereum Browser: (aka Ethereum Reference Client) A cross-platform GUI of an interface similar to a simplified browser (ala Chrome) that is able to host sandboxed applications whose backend is purely on the Ethereum protocol.

Ethereum Virtual Machine: (aka EVM) The virtual machine that forms the key part of the execution model for a contract's program code.

EVM Code: The bytecode that the EVM can natively execute.

EVM Assembly: The human-readable form of EVM-code.

LLL: The Lisp-like Low-level Language, a human-writable language used for authoring simple contracts.

APPENDIX B. FEE SCHEDULE

Name	Value	Description*
G_{step}	1	Default amount of gas to pay for execution cycle.
G_{stop}	0	Nothing paid for the STOP operation.
G_{sha3}	20	Paid for a SHA3 operation.
G_{sload}	20	Paid for a SLOAD operation.
G_{sstore}	100	Paid for a normal SSTORE operation (doubled of waived sometimes).
$G_{balance}$	20	Paid for a BALANCE operation.
G_{create}	100	Paid for a CREATE operation and for a contract creation transaction.
G_{call}	20	Paid for a CALL operation and for a message call transaction.
G_{memory}	1	Paid for every additional word when expanding memory.
G_{txdata}	5	Paid for every byte of data or code for a transaction.

APPENDIX C. RECURSIVE LENGTH PREFIX

This is a method for encoding arbitrarily structured binary data (byte arrays).

APPENDIX D. MODIFIED MERKLE PATRICIA TREE

The modified Merkle Patricia tree (trie) provides a persistent data structure to map between arbitrary-length binary data (byte arrays). It is defined in terms of a mutable data structure to map between 256-bit binary fragments and arbitrary-length binary data, typically implemented as a database.

APPENDIX E. HEX-PREFIX ENCODING

Hex-prefix encoding is an efficient method of encoding a arbitrary number of nibbles as a byte array. It is able to store an additional flag which, when used in the context of the trie (the only context in which it is used), disambiguates between node types.

APPENDIX F. FORMAL SPECIFICATION OF STRUCTURES

Though RLP is data-agnostic, it does specify a canonical representation for integer quantities. It is big-endian with no leading zero-bytes. Thus for elements than feasibly can be stored as integers, it becomes important to specify whether they should adhere to this canonical representation or be left in some other (perhaps more 'native') format.

In the case of counts, balances, fees and amounts of wei, the canon-integer form must be used when storing in RLP. We call these INTs.

In the case of hashes (256-bit or 160-bit), user-visible strings and specialised byte-arrays (e.g. hex-prefix notation from the trie), they should be stored as unformatted byte-array data and not altered into some other form. We call these BINs.

When interpreting RLP data, clients are required to consider non-canonical INT fields in the same way as otherwise invalid RLP data and dismiss it completely.

Specifically:

for the Block header:

```
[
  parentHash: BIN (256-bit),
  unclesHash: BIN (256-bit),
  coinbase: BIN (160-bit),
  stateRoot: BIN (256-bit),
  transactionsHash: BIN (256-bit),
  difficulty: INT,
  timestamp: INT,
  extraData: BIN (typically 256-bit),
  nonce: BIN (256-bit)
]

(note: 'nonce', the last element, refers to a hash here and so is binary)
for entries in the State trie for normal addresses:
```

```
[
  balance: INT,
  nonce: INT
]
```

and for contract addresses:

```
[
  balance: INT,
  nonce: INT,
  storageRoot: BIN (256-bit),
  codeHash: BIN (256-bit)
]

(note: 'nonce', the second element, refers to a tx-count here and so is integer)
for message call transactions:
```

```
[
  nonce: INT,
  value: INT,
  gasPrice: INT,
  gas: INT,
  recvAddr: BIN (160-bit),
  data: BIN (arbitrary length),
  v: INT,
  r: INT,
  s: INT
]
```

or, for contract creation transactions,

```
[
  nonce: INT,
  value: INT,
  gasPrice: INT,
  gas: INT,
  0: BIN (160-bit),
  code: BIN (arbitrary length),
  init: BIN (arbitrary length),
  v: INT,
  r: INT,
```

```
s: INT
]
```

(note: 'nonce', the first element, refers to a tx-count here and so is integer)

The nonce in the transaction refers to the total amount of transactions sent from the address up until that moment in time.

for blocks, there are no immediate data field, but lists:

```
[
  blockHeader: [...]
  uncleList: [ [...], [...], ... ]
  txList: [ [...], [...], ... ]
]
```

Uncle-blocks contain only the uncle's header.

APPENDIX G. THE STATE DB

The state database is a mapping between SHA3-256 hashes (i.e. 256-bit binary fragments) and arbitrary byte data.

APPENDIX H. SIGNING TRANSACTIONS

The method of signing transactions is similar to the 'Electrum style signatures'; it utilises the SECP-256k1 curve.

APPENDIX I. VIRTUAL MACHINE SPECIFICATION

As previously specified in section 8, these definitions take place in the final context there. In particular we assume O is the EVM state-progression function and define the terms pertaining to the next cycle's state $\mathbb{S}'$ $\mathbb{M}'$ such that:

$$(99) \quad O(\mathbb{S}, \mathbb{M}, I) \equiv (\mathbb{S}', \mathbb{M}', I)$$

When interpreting 256-bit binary values as integers the representation is big-endian.

When a 32-byte machine datum is converted to and from as a 160-bit address or hash, the leftwards (high-order for BE) 20 bytes are used and the right most 12 are discarded or filled with zeroes.

The general gas cost function is defined as:

$$(100) \quad C(\mathbb{S}, \mathbb{M}) \equiv G_{memory}(\mathbb{M}'_i - \mathbb{M}_i) + \begin{cases} c_{SSTORE} & \text{if } o = SSTORE \\ G_{call} & \text{if } o = CALL \\ G_{create} & \text{if } o = CREATE \\ G_{sha3} & \text{if } o = SHA3 \\ G_{sload} & \text{if } o = SLOAD \\ G_{balance} & \text{if } o = BALANCE \\ G_{stop} & \text{if } o = STOP \\ G_{step} & \text{otherwise} \end{cases}$$

where c_o is specified in the appropriate section below and C_M is the memory cost function, given as the product of G_{memory} and the maximum of 0 and the ceiling of the number of words in size that the memory must be over the current number of words, $\mathbb{M}_i$ in order that all accesses reference valid memory whether for read or write; $\mathbb{M}'_i$ is defined as this new maximum number of words of active memory; special-cases are given where these two are not equal.

0s: Stop and Arithmetic Operations

All arithmetic is modulo 2^{256} .

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution. $\mathbb{M}'_R = []$
0x01	ADD	2	1	Addition operation. $\mathbb{M}'_T[0] \equiv \mathbb{M}_S[0] + \mathbb{M}_S[1]$
0x02	MUL	2	1	Multiplication operation. $\mathbb{M}'_T[0] \equiv \mathbb{M}_S[0] \times \mathbb{M}_S[1]$
0x03	SUB	2	1	Subtraction operation. $\mathbb{M}'_T[0] \equiv \mathbb{M}_S[0] - \mathbb{M}_S[1]$
0x04	DIV	2	1	Integer division operation. $\mathbb{M}'_T[0] \equiv \begin{cases} 0 & \text{if } \mathbb{M}_S[1] = 0 \\ \lfloor \mathbb{M}_S[0] \div \mathbb{M}_S[1] \rfloor & \text{otherwise} \end{cases}$
0x05	SDIV	2	1	Signed integer division operation. $\mathbb{M}'_T[0] \equiv \begin{cases} 0 & \text{if } \mathbb{M}_S[1] = 0 \\ \lfloor \mathbb{M}_S[0] \div \mathbb{M}_S[1] \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as signed 256-bit integers for the purposes of this operation.
0x06	MOD	2	1	Modulo remainder operation. $\mathbb{M}'_T[0] \equiv \begin{cases} 0 & \text{if } \mathbb{M}_S[1] = 0 \\ \mathbb{M}_S[0] \bmod \mathbb{M}_S[1] & \text{otherwise} \end{cases}$
0x07	SMOD	2	1	Modulo remainder operation. $\mathbb{M}'_T[0] \equiv \begin{cases} 0 & \text{if } \mathbb{M}_S[1] = 0 \\ \mathbb{M}_S[0] \bmod \mathbb{M}_S[1] & \text{otherwise} \end{cases}$ Where all values are treated as signed 256-bit integers for the purposes of this operation.
0x08	EXP	2	1	Exponential operation. $\mathbb{M}'_T[0] \equiv \mathbb{M}_S[0]^{\mathbb{M}_S[1]}$
0x09	NEG	1	1	Negation operation. $\mathbb{M}'_T[0] \equiv -\mathbb{M}_S[0]$ Where all values are treated as signed 256-bit integers for the purposes of this operation.
0x0a	LT	2	1	Less-than comparison. $\mathbb{M}'_T[0] \equiv \begin{cases} 1 & \text{if } \mathbb{M}_S[0] < \mathbb{M}_S[1] \\ 0 & \text{otherwise} \end{cases}$
0x0b	GT	2	1	Greater-than comparison. $\mathbb{M}'_T[0] \equiv \begin{cases} 1 & \text{if } \mathbb{M}_S[0] > \mathbb{M}_S[1] \\ 0 & \text{otherwise} \end{cases}$
0x0c	EQ	2	1	Equality comparison. $\mathbb{M}'_T[0] \equiv \begin{cases} 1 & \text{if } \mathbb{M}_S[0] = \mathbb{M}_S[1] \\ 0 & \text{otherwise} \end{cases}$
0x0d	NOT	1	1	Simple not operator. $\mathbb{M}'_T[0] \equiv \begin{cases} 1 & \text{if } \mathbb{M}_S[0] = 0 \\ 0 & \text{otherwise} \end{cases}$

10s: Bitwise Logic Operations

$\mathbb{M}_S[0]_i$ gives the i th bit (counting from zero) of $\mathbb{M}_S[0]$

Value	Mnemonic	δ	α	Description
-------	----------	----------	----------	-------------

0x10	AND	2	1	Bitwise AND operation. $\forall i \in [0..255] : \mathbb{M}'_T[0]_i \equiv \mathbb{M}_S[0]_i \wedge \mathbb{M}_S[1]_i$
------	-----	---	---	---

0x11	OR	2	1	Bitwise OR operation. $\forall i \in [0..255] : \mathbb{M}'_T[0]_i \equiv \mathbb{M}_S[0]_i \vee \mathbb{M}_S[1]_i$
------	----	---	---	--

0x12	XOR	2	1	Bitwise XOR operation. $\forall i \in [0..255] : \mathbb{M}'_T[0]_i \equiv \mathbb{M}_S[0]_i \oplus \mathbb{M}_S[1]_i$
------	-----	---	---	---

0x13	BYTE	2	1	Retrieve single byte from word. $\forall i \in [0..7] : \mathbb{M}'_T[0]_i \equiv \begin{cases} \mathbb{M}_S[1]_{(i+8\mathbb{M}_S[0])} & \text{if } \mathbb{M}_S[0] < 32 \\ 0 & \text{otherwise} \end{cases}$
------	------	---	---	--

For Nth byte, we count from the left (i.e. N=0 would be the most significant in big endian).

20s: SHA3

Value	Mnemonic	δ	α	Description
-------	----------	----------	----------	-------------

0x20	SHA3	2	1	Compute SHA3-256 hash. $\mathbb{M}'_T[0] \equiv \text{SHA3}(\mathbb{M}_M[\mathbb{M}_S[0] \dots (\mathbb{M}_S[0] + \mathbb{M}_S[1] - 1)])$ $\mathbb{M}'_i \equiv \max(\mathbb{M}_i, \lceil (\mathbb{M}_S[0] + \mathbb{M}_S[1]) \div 32 \rceil)$
------	------	---	---	--

30s: Environmental Information

Value	Mnemonic	δ	α	Description
-------	----------	----------	----------	-------------

0x30	ADDRESS	0	1	Get address of currently executing account. $\mathbb{M}'_T[0] \equiv I_a$
------	---------	---	---	--

0x31	BALANCE	0	1	Get balance of currently executing account. $\mathbb{M}'_T[0] \equiv \mathbb{S}[I_a]_{balance}$
------	---------	---	---	--

0x32	ORIGIN	0	1	Get execution origination address. $\mathbb{M}'_T[0] \equiv I_o$ This is the sender of original transaction; it's never a contract.
------	--------	---	---	---

0x33	CALLER	0	1	Get caller address. $\mathbb{M}'_T[0] \equiv I_c$ This is the address of the account that is directly responsible for this execution.
------	--------	---	---	---

0x34	CALLVALUE	0	1	Get deposited value by the instruction/transaction responsible for this execution. $\mathbb{M}'_T[0] \equiv I_v$
------	-----------	---	---	---

0x35	CALLDATALOAD	1	1	Get input data of current environment. $\mathbb{M}'_T[0] \equiv I_d[\mathbb{M}_S[0] \dots (\mathbb{M}_S[0] + 32)]$ This pertains to the input data passed with the message call instruction or transaction.
------	--------------	---	---	---

0x36	CALLDATASIZE	0	1	Get size of input data in current environment. $\mathbb{M}'_T[0] \equiv \ I_d\ $ This pertains to the input data passed with the message call instruction or transaction.
------	--------------	---	---	---

0x37	GASPRICE	0	1	Get price of gas in current environment. $\mathbb{M}'_T[0] \equiv I_p$ This is gas price specified by the originating transaction.
------	----------	---	---	--

40s: Block Information				
Value	Mnemonic	δ	α	Description
0x40	PREVHASH	0	1	Get hash of most recent complete block. $\mathbb{M}'_T[0] \equiv I_{H_p}$ I_{H_p} is the previous block's hash.
0x41	COINBASE	0	1	Get the block's coinbase address. $\mathbb{M}'_T[0] \equiv I_{H_b}$
0x42	TIMESTAMP	0	1	Get the block's timestamp. $\mathbb{M}'_T[0] \equiv I_{H_t}$
0x43	NUMBER	0	1	Get the block's number. $\mathbb{M}'_T[0] \equiv n(I_H)$ $n(I_H)$ is the total number of ancestor blocks of the current block.
0x44	DIFFICULTY	0	1	Get the block's difficulty. $\mathbb{M}'_T[0] \equiv I_{H_d}$
0x45	GASLIMIT	0	1	Get the block's gas limit. $\mathbb{M}'_T[0] \equiv 10^6$ Presently, the gas limit is hard-coded in the protocol.

50s: Stack, Memory, Storage and Flow Operations

Value	Mnemonic	δ	α	Description
0x50	POP	1	0	Remove item from stack. $\mathbb{M}'_T[0] \equiv \mathbb{M}_S[0]$
0x51	DUP	1	2	Duplicate stack item. $\mathbb{M}'_T[0] \equiv \mathbb{M}_S[0]$
0x52	SWAP	2	2	Exchange stack items. $\mathbb{M}'_T[0] \equiv \mathbb{M}_S[1]$ $\mathbb{M}'_T[1] \equiv \mathbb{M}_S[0]$
0x53	MLOAD	1	1	Load word from memory. $\mathbb{M}'_T[0] \equiv \mathbb{M}_M[\mathbb{M}_S[0] \dots (\mathbb{M}_S[0] + 32)]$ $\mathbb{M}'_i \equiv \max(\mathbb{M}_i, \lceil (\mathbb{M}_S[0] + 32) \div 32 \rceil)$
0x54	MSTORE	2	0	Save word to memory. $\mathbb{M}'_M[\mathbb{M}_S[0] \dots (\mathbb{M}_S[0] + 32)] \equiv \mathbb{M}_S[1]$ $\mathbb{M}'_i \equiv \max(\mathbb{M}_i, \lceil (\mathbb{M}_S[0] + 32) \div 32 \rceil)$
0x55	MSTORE8	2	0	Save byte to memory. $\mathbb{M}'_M[\mathbb{M}_S[0]] \equiv (\mathbb{M}_S[1] \bmod 256)$ $\mathbb{M}'_i \equiv \max(\mathbb{M}_i, \lceil (\mathbb{M}_S[0] + 1) \div 32 \rceil)$
0x56	SLOAD	1	1	Load word from storage. $\mathbb{M}'_T[0] \equiv \mathbb{S}'[I_a]_{storage}[\mathbb{M}_S[0]]$
0x57	SSTORE	2	0	Save word to storage. $\mathbb{S}'[I_a]_{storage}[\mathbb{M}_S[0]] \equiv \mathbb{M}_S[1]$ $c_{SSTORE} \equiv \begin{cases} 2G_{sstore} & \text{if } \mathbb{M}_S[1] \neq 0 \wedge \mathbb{S}[I_a]_{storage}[\mathbb{M}_S[0]] = 0 \\ 0 & \text{if } \mathbb{M}_S[1] = 0 \wedge \mathbb{S}[I_a]_{storage}[\mathbb{M}_S[0]] \neq 0 \\ G_{sstore} & \text{otherwise} \end{cases}$
0x58	JUMP	1	0	Alter the program counter. $J_{JUMP}(\mathbb{M}) \equiv \mathbb{M}_S[0]$ This has the effect of writing said value to $\mathbb{M}_{pc}$. See section 8.
0x59	JUMPI	2	0	Conditionally alter the program counter. $J_{JUMPI}(\mathbb{M}) \equiv \begin{cases} \mathbb{M}_S[0] & \text{if } \mathbb{M}_S[0] = 0 \\ \mathbb{M}_{pc} + 1 & \text{otherwise} \end{cases}$ This has the effect of writing said value to $\mathbb{M}_{pc}$. See section 8.
0x5a	PC	0	1	Get the program counter. $\mathbb{M}'_T[0] \equiv \mathbb{M}_{pc}$
0x5b	MSIZE	0	1	Get the size of active memory. $\mathbb{M}'_T[0] \equiv \mathbb{M}_i$
0x5c	GAS	0	1	Get the amount of available gas. $\mathbb{M}'_T[0] \equiv \mathbb{M}_g$

60s & 70s: Push Operations

Value	Mnemonic	δ	α	Description
0x60	PUSH1	0	1	Place 1 byte item on stack. $\mathbb{M}'_T[0] \equiv I_b[\mathbb{M}_{pc} + 1]$ The byte is right-aligned (takes the lowest significant place in big endian).
0x61	PUSH2	0	1	Place 2-byte item on stack. $\mathbb{M}'_T[0] \equiv I_b[(\mathbb{M}_{pc} + 1) \dots (\mathbb{M}_{pc} + 2)]$ The bytes are right-aligned (takes the lowest significant place in big endian).
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0x7f	PUSH32	0	1	Place 32-byte (full word) item on stack. $\mathbb{M}'_T[0] \equiv I_b[(\mathbb{M}_{pc} + 1) \dots (\mathbb{M}_{pc} + 32)]$

f0s: System operations				
Value	Mnemonic	δ	α	Description
0xf0	CREATE	5	1	Create a new contract. $\mathbf{b} \equiv \mathbb{M}_M[\mathbb{M}_S[1] \dots (\mathbb{M}_S[1] + \mathbb{M}_S[2] - 1)]$ $\mathbf{i} \equiv \mathbb{M}_M[\mathbb{M}_S[3] \dots (\mathbb{M}_S[3] + \mathbb{M}_S[4] - 1)]$ $(\mathbb{S}', \mathbb{M}'_g) \equiv \Lambda(\mathbb{S}^*, I_a, I_o, \mathbb{M}_g, I_p, \mathbb{M}_S[0], \mathbf{i}, \mathbf{b})$ $\mathbb{S}^* \equiv \mathbb{S} \quad \text{except} \quad \mathbb{S}^*[I_a]_{nonce} = \mathbb{S}[I_a]_{nonce} + 1$ $\mathbb{M}'_T[0] \equiv x$ where $x = 0$ if the code execution for this operation failed due to lack of gas; $x = A(I_a, \mathbb{S}[I_a]_{nonce})$, the address of the newly created contract, otherwise. $\mathbb{M}'_i \equiv \max(\mathbb{M}_i, \lceil \max(\mathbb{M}_S[1] + \mathbb{M}_S[2], \mathbb{M}_S[3] + \mathbb{M}_S[4]) \div 32 \rceil)$
0xf1	CALL	7	1	Message-call into contract. $\mathbf{i} \equiv \mathbb{M}_M[\mathbb{M}_S[3] \dots (\mathbb{M}_S[3] + \mathbb{M}_S[4] - 1)]$ $\mathbf{o} \equiv \mathbb{M}_M[\mathbb{M}_S[5] \dots (\mathbb{M}_S[5] + \mathbb{M}_S[6] - 1)]$ Let $g_k = \begin{cases} 0 & \text{if } \mathbb{M}_S[2] = 0 \\ \mathbb{M}_g - \min(\mathbb{M}_g, \mathbb{M}_S[2]) & \text{otherwise} \end{cases}$ $(\mathbb{S}', g', \mathbf{o}) \equiv \Theta(\mathbb{S}, I_a, I_o, \mathbb{M}_S[0], g, I_p, \mathbb{M}_g - g_k, \mathbb{M}_S[1], \mathbf{i})$ $\mathbb{M}'_g \equiv g' + g_k$ $\mathbb{M}'_T[0] \equiv x$ where $x = 0$ if the code execution for this operation failed due to lack of gas; $x = 1$ otherwise. $\mathbb{M}'_i \equiv \max(\mathbb{M}_i, \lceil \max(\mathbb{M}_S[3] + \mathbb{M}_S[4], \mathbb{M}_S[5] + \mathbb{M}_S[6]) \div 32 \rceil)$
0xf2	RETURN	2	0	Halt execution returning output data. $H_{\text{RETURN}}(\mathbb{M}) \equiv \mathbb{M}_M[\mathbb{M}_S[0] \dots (\mathbb{M}_S[0] + \mathbb{M}_S[1] - 1)]$ This has the effect of halting the execution at this point with output defined. See section 8. $\mathbb{M}'_i \equiv \max(\mathbb{M}_i, \lceil (\mathbb{M}_S[0] + \mathbb{M}_S[1]) \div 32 \rceil)$
0xff	SUICIDE	1	0	Halt execution and obliterate account. $\mathbb{S}'[\mathbb{M}_S[0]]_{balance} \equiv \mathbb{S}[\mathbb{M}_S[0]]_{balance} + \mathbb{S}[I_a]_{balance}$ $\mathbb{S}'[I_a] \equiv \emptyset$

APPENDIX J. WIRE PROTOCOL

The wire-protocol specifies a network-level protocol for how two peers can communicate. It includes handshake procedures and the means for transferring information such as blocks & transactions.

APPENDIX K. GENESIS BLOCK

The header of the genesis block is 9 items, and is specified thus:

$\langle 0_{256}, \text{SHA3}(\text{RLP}(\square)), 0_{160}, 0_{256}, \text{SHA3}(\text{RLP}(\square)), 2^{22}, 0, \square, 42 \rangle$

Where:

0_{256} refers to the parent hash, a 256-bit hash which is all zeroes.

0_{160} refers to the coinbase address, a 160-bit hash which is all zeroes.

2^{22} refers to the difficulty.

0 refers to the timestamp (the Unix epoch).

$\square$ refers to the extradata, an empty byte array.

$\text{SHA3}(\text{RLP}(\square))$ values refer to the hashes of the transaction and uncle lists in RLP, both empty.