Decentralized asymmetric algorithm consensus:

One-time Ring Signature Peer-to-Peer Untraceable Electronic Cash System and MimbleWimble Consensus Algorithm

Lu Wang

COO@Niangniao.com

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.

While several consensus algorithms exist for the Byzantine Generals Problem, specifically as it pertains to distributed payment systems, many suffer from high latency induced by the requirement that all nodes within the network communicate synchronously. In this work, we present a novel consensus algorithm that circumvents this requirement by utilizing collectively-trusted subnetworks within the larger network. We show that the "trust" required of these subnetworks is in fact minimal and can be further reduced with principled choice of the member nodes. In addition, we show that minimal connectivity is required to maintain agreement throughout the whole network. The result is a low-latency consensus algorithm which still maintains robustness in the face of Byzantine failures. We present this algorithm in its embodiment in the Decentralized asymmetric algorithm consensus Protocol.

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.

Interest and research in distributed consensus systems has increased markedly in recent years, with a central focus being on distributed payment networks. Such networks allow for fast, low-cost transactions which are not controlled by a centralized source. While the economic benefits and drawbacks of such a system are worthy of much research in and of themselves, this work focuses on some of the technical challenges that all distributed payment systems must face. While these problems are varied, we group them into three main categories: correctness, agreement, and utility.

By correctness, we mean that it is necessary for a distributed system to be able to discern the difference between a correct and fraudulent transaction. In traditional fiduciary settings, this is done through trust between institutions and cryptographic signatures that guarantee a transaction is indeed coming from the institution that it claims to be coming from. In distributed systems, however, there is no such trust, as the identity of any and all members in the network may not even be known. Therefore, alternative methods for correctness must be utilized.

Agreement refers to the problem of maintaining a single global truth in the face of a decentralized accounting system. While similar to the correctness problem, the difference lies in the fact that while a malicious user of the network may be unable to create a fraudulent transaction (defying correctness), it may be able to create multiple correct transactions that are somehow unaware of each other, and thus combine to create a fraudulent act. For example, a malicious user may make two simultaneous purchases, with only enough funds in their account to cover each purchase individually, but not both together. Thus each transaction by itself is correct, but if executed simultaneously in such a way that the distributed network as a whole is unaware of both, a clear problem arises, commonly referred to as the "Double-Spend Problem." Thus the agreement problem can be summarized as the requirement that only one set of globally recognized transactions exist in the network.

Utility is a slightly more abstract problem, which we define generally as the "usefulness" of a distributed payment system, but which in practice most often simplifies to the latency of the system. A distributed system that is both correct and in agreement but which requires one year to process a transaction, for example, is obviously an inviable payment system. Additional aspects of utility may include the level of computing power required to participate in the correctness and agreement processes or the technical proficiency required of an end user to avoid being defrauded in the network.

Many of these issues have been explored long before the advent of modern distributed computer systems, via a problem known as the "Byzantine Generals Problem." In this problem, a group of generals each control a portion of an army and must coordinate an attack by sending messengers to each other. Because the generals are in unfamiliar and hostile territory, messengers may fail to reach their destination (just as nodes in a distributed network may fail, or send corrupted data instead of the intended message). An additional aspect of

the problem is that some of the generals may be traitors, either individually, or conspiring together, and so messages may arrive which are intended to create a false plan that is doomed to failure for the loyal generals (just as malicious members of a distributed system may attempt to convince the system to accept fraudulent transactions, or multiple versions of the same truthful transaction that would result in a double-spend). Thus a distributed payment system must be robust both in the face of standard failures, and so-called "Byzantine" failures, which may be coordinated and originate from multiple sources in the network.

In this work, we analyze one particular implementation of a distributed payment system: the Decentralized asymmetric algorithm consensus Protocol. We focus on the algorithms utilized to achieve the above goals of correctness, agreement, and utility, and show that all are met (within necessary and predetermined tolerance thresholds, which are well-understood). In addition, we provide code that simulates the consensus process with parameterizable network size, number of malicious users, and message-sending latencies.

"Bitcoin" has been a successful implementation of the concept of p2p electronic cash. Both professionals and the general public have come to appreciate the convenient combination of public transactions and proof-of-work as a trust model. Today, the user base of electronic cash is growing at a steady pace; customers are attracted to low fees and the anonymity provided by electronic cash and merchants value its predicted and decentralized emission. Bitcoin has effectively proved that electronic cash can be as simple as paper money and as convenient as credit cards.

Unfortunately, Bitcoin suffers from several deficiencies. For example, the system's distributed nature is inflexible, preventing the implementation of new features until almost all of the network users update their clients. Some critical flaws that cannot be fixed rapidly deter Bitcoin's widespread propagation. In such inflexible models, it is more efficient to roll-out a new project rather than perpetually fix the original project.

In this paper, we study and propose solutions to the main deficiencies of Bitcoin. We believe that a system taking into account the solutions we propose will lead to a healthy competition among different electronic cash systems. We also propose our own electronic cash, "Decentralized asymmetric algorithm consensus", a name emphasizing the next breakthrough in electronic cash.

Bitcoin is the first widely used financial system for which all the necessary data to validate the system status can be cryptographically verified by anyone. However, it accomplishes this feat by storing all transactions in a public database called "the blockchain" and someone who genuinely wishes to check this state must download the whole thing and basically replay each transaction, check each one as they go. Meanwhile, most of these transactions have not affected the actual final state (they create outputs that are destroyed a transaction later).

At the time of this writing, there were nearly 150 million transactions committed in the blockchain, which must be replayed to produce a set of only 4 million unspent outputs.

It would be better if an auditor needed only to check data on the outputs themselves, but this is impossible because they are valid if and only if the output is at the end of a chain of previous outputs, each signs the next. In other words, the whole blockchain must be validated to confirm the final state.

Add to this that these transactions are cryptographically atomic, it is clear what outputs go into every transaction and what emerges. The "transaction graph" resulting reveals a lot of information and is subjected to analysis by many companies whose business model is to monitor and control the lower classes. This makes it very non-private and even dangerous for people to use.

Some solutions to this have been proposed. Greg Maxwell discovered to encrypt the amounts, so that the graph of the transaction is faceless but still allow validation that the sums are correct. Dr Maxwell also produced CoinJoin, a system for Bitcoin users to combine interactively transactions, confusing the transaction graph. Nicolas van Saberhagen has developed a system to blind the transaction entries, goes much further to

cloud the transaction graph (as well as not needed the user interaction). Later, Shen Noether combined the two approaches to obtain "confidential transactions" of Maxwell AND the darkening of van Saberhagen.
These solutions are very good and would make Bitcoin very safe to use. But the problem of too much data is made even worse. Confidential transactions require multi-kilobyte proofs on every output, and van Saberhagen signatures require every output to be stored for ever, since it is not possible to tell when they are truly spent.
Dr. Maxwell's CoinJoin has the problem of needing interactivity. Dr. Yuan Horas Mouton fixed this by making transactions freely mergeable, but he needed to use pairing-based cryptography, which is potentially slower and more difficult to trust. He called this "one-way aggregate signatures" (OWAS).
OWAS had the good idea to combine the transactions in blocks. Imagine that we can combine across blocks (perhaps with some glue data) so that when the outputs are created and destroyed, it is the same as if they never existed. Then, to validate the entire chain, users only need to know when money is entered into the system (new money in each block as in Bitcoin or Monero or peg-ins for sidechains) and final unspent outputs, the rest can be removed and forgotten.
Then we can have Confidential Transactions to hide the amounts and OWAS to blur the transaction graph, and use LESS space than Bitcoin to allow users to fully verify the blockchain. And also imagine that we must not pairing-based cryptography or new hypotheses, just regular discrete logarithms signatures like Bitcoin. Here is what I propose.
I call my creation Decentralized asymmetric algorithm consensus because it is used to prevent the blockchain from talking about all user's information.

Definitions, Formalization and Previous Work

We begin by defining the components of the Decentralized asymmetric algorithm consensus Protocol. In order to prove correctness, agreement, and utility properties, we first formalize those properties into axioms. These properties, when grouped together, form the notion of consensus: the state in which nodes in the network reach correct agreement. We then highlight some previous results relating to consensus algorithms, and finally state the goals of consensus for the Decentralized asymmetric algorithm consensus Protocol within our formalization framework.

Decentralized asymmetric algorithm consensus Protocol Components

We begin our description of the Decentralized asymmetric algorithm consensus network by defining the following terms:

```
Server:
A server is any entity running the Decentralized asymmetric algorithm
consensus Server software (as opposed to the Decentralized asymmetric
algorithm consensus Client software which only lets a user send and
receive funds), which participates in the consensus process.
```

```
Ledger:
The ledger is a record of the amount of currency in each userâ€™s account
and represents the "ground truth" of the network. The ledger is repeatedly
```

```
updated with transactions that successfully pass through the consensus
process.
```

```
Last-Closed Ledger:
The last-closed ledger is the most recent ledger that has been ratified by
the consensus process and thus represents the current state of the
network.
```

```
Open Ledger:
The open ledger is the current operating status of a node (each node
maintains its own open ledger). Transactions initiated by end users of a
given server are applied to the open ledger of that server, but
transactions are not considered final until they have passed through the
consensus process, at which point the open ledger becomes the last-closed
ledger.
```

```
Unique Node List (UNL):
Each server, s, main- tains a unique node list, which is a set of other
servers that s queries when determining consensus. Only the votes of the
other members of the UNL of s are considered when determining consensus
(as opposed to every node on the network). Thus the UNL represents a
subset of the network which when taken collectively, is "trusted" by s to
not collude in an attempt to defraud the network. Note that this
definition of "trust" does not require that each individual member of the
UNL be trusted.
```

```
Proposer:
Any server can broadcast transactions to be included in the consensus
process, and every server attempts to include every valid transaction when
a new consensus round starts. During the consensus process, however, only
proposals from servers on the UNL of a server s are considered by s.
```

Formalization

We use the term nonfaulty to refer to nodes in the net- work that behave honestly and without error. Conversely, a faulty node is one which experiences errors which may be honest (due to data corruption, implementation errors, etc.), or malicious (Byzantine errors). We reduce the notion of validating a transaction to a simple binary decision problem: each node must decide from the information it has been given on the value 0 or 1.

As in Attiya, Dolev, and Gill, 1984, we define consensus according to the following three axioms:

```
1. (C1): Every nonfaulty node makes a decision infinite time
```

```
1. (C2): All nonfaulty nodes reach the same decision value
```

```
3. (C3): 0 and 1 are both possible values for all non-faulty nodes. (This removes the trivial solution in which all nodes decide 0 or 1 regardless of the information they have been presented).
```

Existing Consensus Algorithms

There has been much research done on algorithms that achieve consensus in the face of Byzantine errors. This previous work has included extensions to cases where all participants in the network are not known ahead of time, where the messages are sent asynchronously (there is no bound on the amount of time an individual node will take to reach a decision), and where there is a delineation between the notion of strong and weak consensus.

One pertinent result of previous work on consensus algorithms is that of Fischer, Lynch, and Patterson, 1985, which proves that in the asynchronous case, non-termination is always a possibility for a consensus algorithm, even with just one faulty process. This introduces the necessity for time-based heuristics, to ensure convergence (or at least repeated iterations of non-convergence). We shall describe these heuristics for the Decentralized asymmetric algorithm consensus Protocol later.

The strength of a consensus algorithm is usually measured in terms of the fraction of faulty processes it can tolerate. It is provable that no solution to the Byzantine Generals problem (which already assumes synchronicity, and known participants) can tolerate more than (n - 1)/3 byzantine faults, or 33% of the network acting maliciously. This solution does not, however, require verifiable authenticity of the messages delivered between nodes (digital signatures). If a guarantee on the unforgeability of messages is possible, algorithms exist with much higher fault tolerance in the synchronous case.

Several algorithms with greater complexity have been proposed for Byzantine consensus in the asynchronous case. FaB Paxos will tolerate (n - 1)/5 Byzantine failures in a network of n nodes, amounting to a tolerance of up to 20% of nodes in the network colluding maliciously. Attiya, Doyev, and Gill introduce a phase algorithm for the asynchronous case, which can tolerate (n - 1)/4 failures, or up to 25% of the network. Lastly, Alchieri et al., 2008 present BFT-CUP, which achieves Byzantine consensus in the asynchronous case even with unknown participants, with the maximal bound of a tolerance of (n - 1)/3 failures, but with additional restrictions on the connectivity of the underlying network.

Formal Consensus Goals

Our goal in this work is to show that the consensus algorithm utilized by the Decentralized asymmetric algorithm consensus Protocol will achieve consensus at each ledger-close (even if consensus is the trivial consensus of all transactions being rejected), and that the trivial consensus will only be reached with a known probability, even in the face of Byzantine failures. Our goal in this work is to show that the consensus algorithm utilized by the Decentralized asymmetric algorithm consensus Protocol will achieve consensus at each ledger-close (even if consensus is the trivial consensus of all transactions being rejected), and that the trivial consensus will only be reached with a known probability, even in the face of Byzantine failures.

Lastly we will show that the Decentralized asymmetric algorithm consensus Protocol can achieve these goals in the face of (n-1)/5 failures, which is not the strongest result in the literature, but we will also show that the Decentralized asymmetric algorithm consensus Protocol possesses several other desirable features that greatly enhance its utility.

Irregular emission

Bitcoin has a predetermined emission rate: each solved block produces a fixed amount of coins. Approximately every four years this reward is halved. The original intention was to create a limited smooth emission with exponential decay, but in fact we have a piecewise linear emission function whose breakpoints may cause problems to the Bitcoin infrastructure.

When the breakpoint occurs, miners start to receive only half of the value of their previous reward. The absolute difference between 12.5 and 6.25 BTC (projected for the year 2020) may seem tolerable. However, when examining the 50 to 25 BTC drop that took place on November 28 2012, felt inappropriate for a significant number of members of the mining community. Figure 1 shows a dramatic decrease in the network's hashrate in the end of November, exactly when the halving took place. This event could have been the perfect moment for the malevolent individual described in the proof-of-work function section to carry-out a double spending attack.

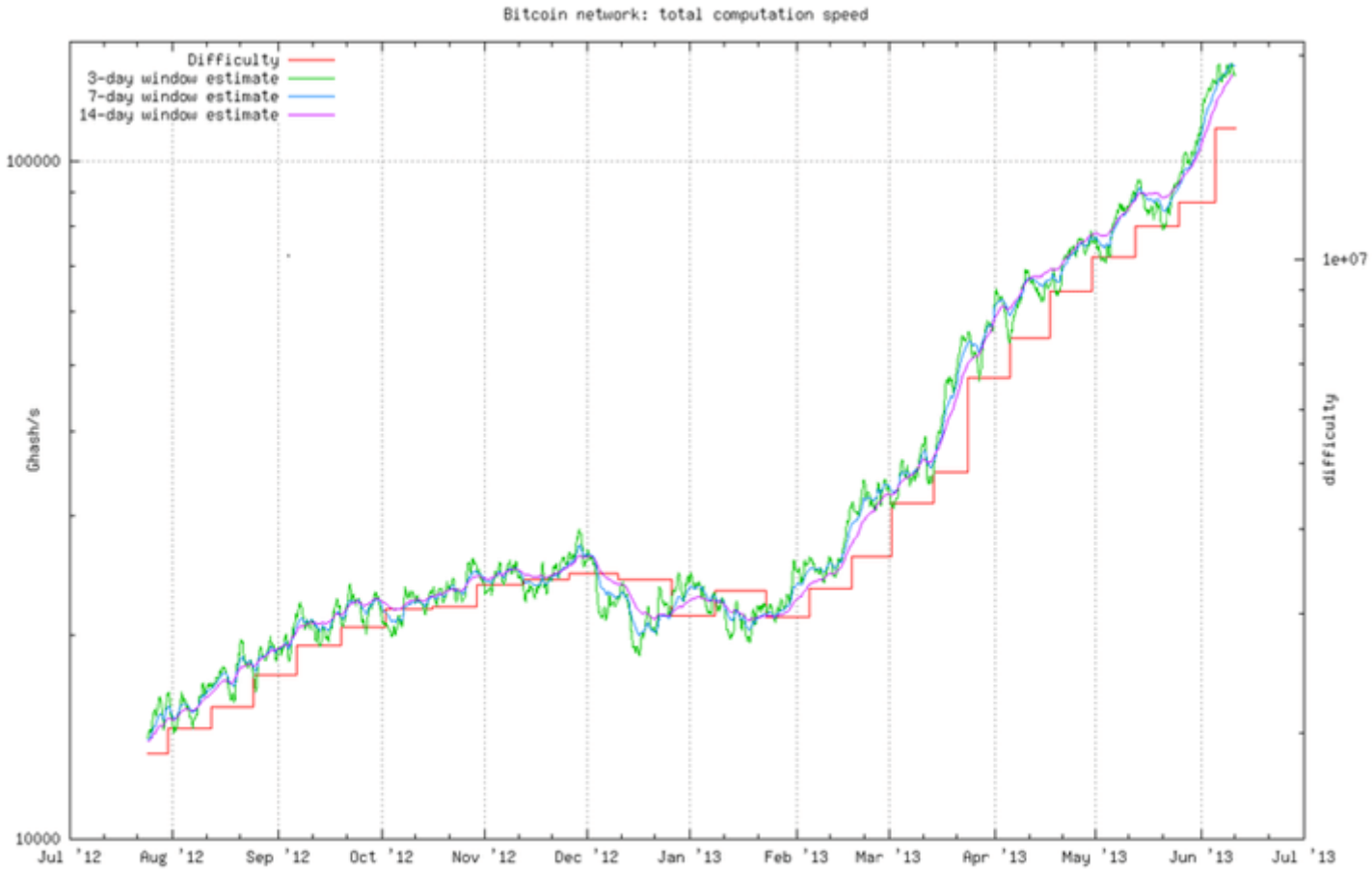


Fig. 1. Bitcoin hashrate chart
(source: http://bitcoin.sipa.be)

Hardcoded constants

Bitcoin has many hard-coded limits, where some are natural elements of the original design (e.g. block frequency, maximum amount of money supply, number of confirmations) whereas other seem to be artificial constraints. It is not so much the limits, as the inability of quickly changing them if necessary that causes the main drawbacks. Unfortunately, it is hard to predict when the constants may need to be changed and replacing them may lead to terrible consequences.

A good example of a hardcoded limit change leading to disastrous consequences is the block size limit set to 250kb1. This limit was sufficient to hold about 10000 standard transactions. In early 2013, this limit had almost been reached and an agreement was reached to increase the limit. The change was implemented in wallet version 0.8 and ended with a 24-blocks chain split and a successful double-spend attack. While the bug was not in the Bitcoin protocol, but rather in the database engine it could have been easily caught by a simple stress test if there was no artificially introduced block size limit.

Constants also act as a form of centralization point. Despite the peer-to-peer nature of Bitcoin, an overwhelming majority of nodes use the official reference client developed by a small group of people. This group makes the decision to implement changes to the protocol and most people accept these changes irrespective of their "correctness". Some decisions caused heated discussions and even calls for boycott, which indicates that the community and the developers may disagree on some important points. It therefore seems logical to have a protocol with user-configurable and self-adjusting variables as a possible way to avoid these problems.

Bulky scripts

The scripting system in Bitcoin is a heavy and complex feature. It potentially allows one to create sophisticated transactions, but some of its features are disabled due to security concerns and some have never even been used. The script (including both senders' and receivers' parts) for the most popular transaction in Bitcoin looks like this:

```
<sig> <pubKey> OP DUP OP HASH160 <pubKeyHash> OP EQUALVERIFY OP CHECKSIG.
```

The script is 164 bytes long whereas its only purpose is to check if the receiver possess the secret key required to verify his signature.

The Decentralized asymmetric algorithm consensus Technology

Now that we have covered the limitations of the Bitcoin technology, we will concentrate on presenting the features of Decentralized asymmetric algorithm consensus.

Decentralized asymmetric algorithm consensus Consensus Algorithm

The Decentralized asymmetric algorithm consensus Protocol consensus algorithm, is applied every few seconds by all nodes, in order to maintain the correctness and agreement of the network. Once consensus is reached, the current ledger is considered "closed" and becomes the last-closed ledger. Assuming that the consensus algorithm is successful, and that there is no fork in the network, the last-closed ledger maintained by all nodes in the network will be identical.

Elliptic curve parameters

As our base signature algorithm we chose to use the fast scheme EdDSA, which is developed and implemented by D.J. Bernstein et al. Like Bitcoin's ECDSA it is based on the elliptic curve discrete logarithm

problem, so our scheme could also be applied to Bitcoin in future. Common parameters are:

q: a prime number; $q = 2^{255} - 19$;

d: an element of $\mathbb{F}_q$; $d = -121665/121666$;

E: an elliptic curve equation; $-x^2 + y^2 = 1 + dx^2y^2$;

G: a base point; $G = (x, -4/5)$;

l: a prime order of the base point; $l = 2^{252} + 27742317777372353535851937790883648493$;

$\mathcal{H}_s$: a cryptographic hash function $\{0,1\}^* \to \mathbb{F}_q$;

$\mathcal{H}_p$: a deterministic hash function $E(\mathbb{F}_q) \to E(\mathbb{F}_q)$.

Terminology

Enhanced privacy requires a new terminology which should not be confused with Bitcoin entities.

private ec-key is a standard elliptic curve private key: a number $a \in [1, l-1]$;

public ec-key is a standard elliptic curve public key: a point $A = aG$;

one-time keypair is a pair of private and public ec-keys;

private user key is a pair (a, b) of two different private ec-keys;

tracking key is a pair (a, B) of private and public ec-key (where $B = bG$ and $a \neq b$);

public user key is a pair (A, B) of two public ec-keys derived from (a, b);

standard address is a representation of a public user key given into human friendly string with error correction;

truncated address is a representation of the second half (point B) of a public user key given into human friendly string with error correction.

The transaction structure remains similar to the structure in Bitcoin: every user can choose several independent incoming payments (transactions outputs), sign them with the corresponding private keys and send them to different destinations.

Contrary to Bitcoin's model, where a user possesses unique private and public key, in the proposed model a sender generates a one-time public key based on the recipient's address and some random data. In this sense, an incoming transaction for the same recipient is sent to a one-time public key (not directly to a unique address) and only the recipient can recover the corresponding private part to redeem his funds (using his unique private key). The recipient can spend the funds using a ring signature, keeping his ownership and actual spending anonymous. The details of the protocol are explained in the next subsections.

Unlinkable payments

Classic Bitcoin addresses, once being published, become unambiguous identifier for incoming payments, linking them together and tying to the recipient's pseudonyms. If someone wants to receive an "untied" transaction, he should convey his address to the sender by a private channel. If he wants to receive different transactions which cannot be proven to belong to the same owner he should generate all the different addresses and never publish them in his own pseudonym.

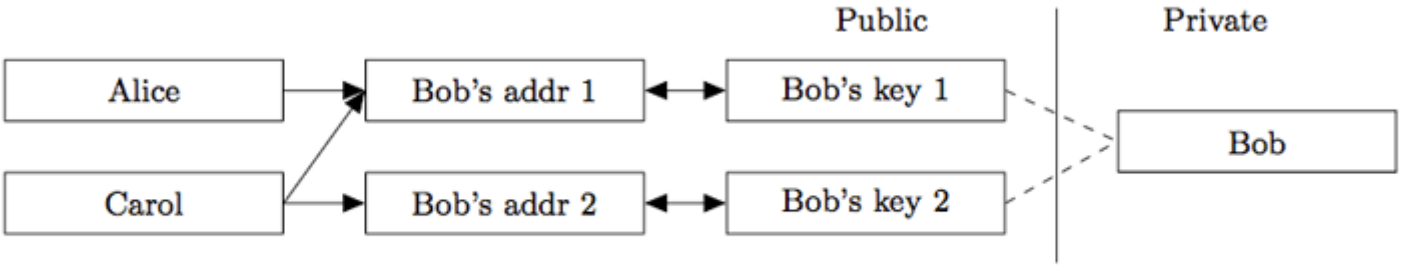


Fig. 2. Traditional Bitcoin keys/transactions model.

We propose a solution allowing a user to publish a single address and receive unconditional unlinkable payments. The destination of each Decentralized asymmetric algorithm consensus output (by default) is a public key, derived from recipient's address and sender's random data. The main advantage against Bitcoin is that every destination key is unique by default (unless the sender uses the same data for each of his transactions to the same recipient). Hence, there is no such issue as "address reuse" by design and no observer can determine if any transactions were sent to a specific address or link two addresses together.

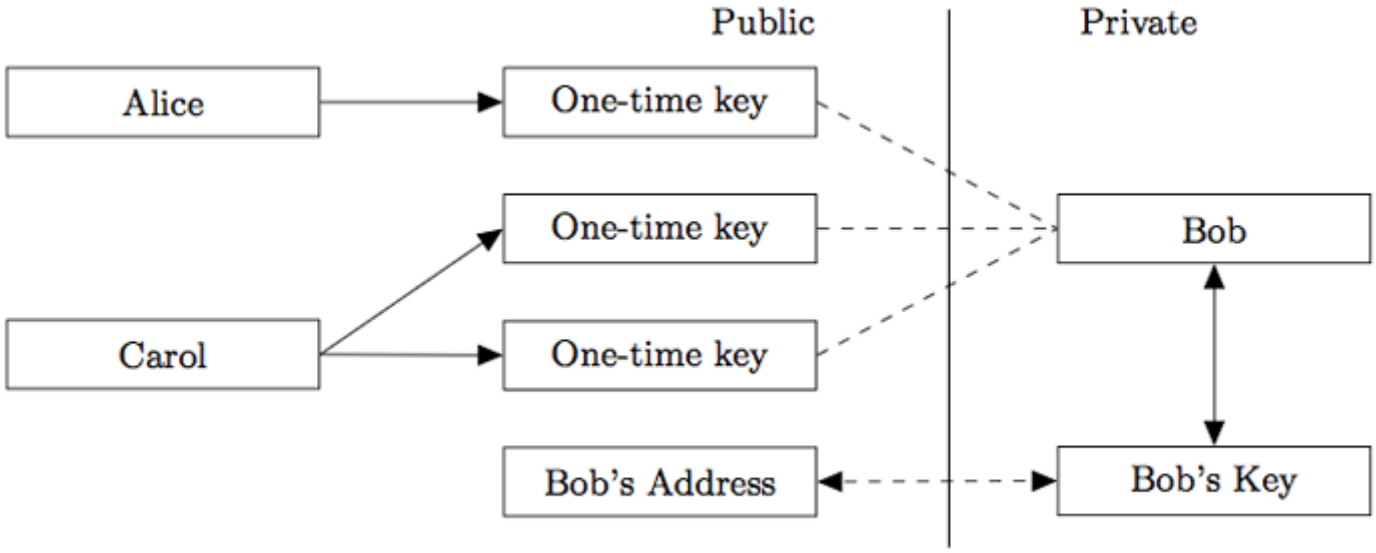


Fig. 3. CryptoNote keys/transactions model.

First, the sender performs a Diffie-Hellman exchange to get a shared secret from his data and half of the recipient's address. Then he computes a one-time destination key, using the shared secret and the second half of the address. Two different ec-keys are required from the recipient for these two steps, so a standard

Decentralized asymmetric algorithm consensus address is nearly twice as large as a Bitcoin wallet address. The receiver also performs a Diffie-Hellman exchange to recover the corresponding secret key.
A standard transaction sequence goes as follows:

1. Alice wants to send a payment to Bob, who has published his standard address. She unpacks the address and gets Bob's public key (A, B).
2. Alice generates a random $r \in [1, l-1]$ and computes a one-time public key $P = \mathcal{H}_s(rA)G + B$.
3. Alice uses P as a destination key for the output and also packs value $R = rG$ (as a part of the Diffie-Hellman exchange) somewhere into the transaction. Note that she can create other outputs with unique public keys: different recipients' keys (A_i, B_i) imply different P_i even with the same r.

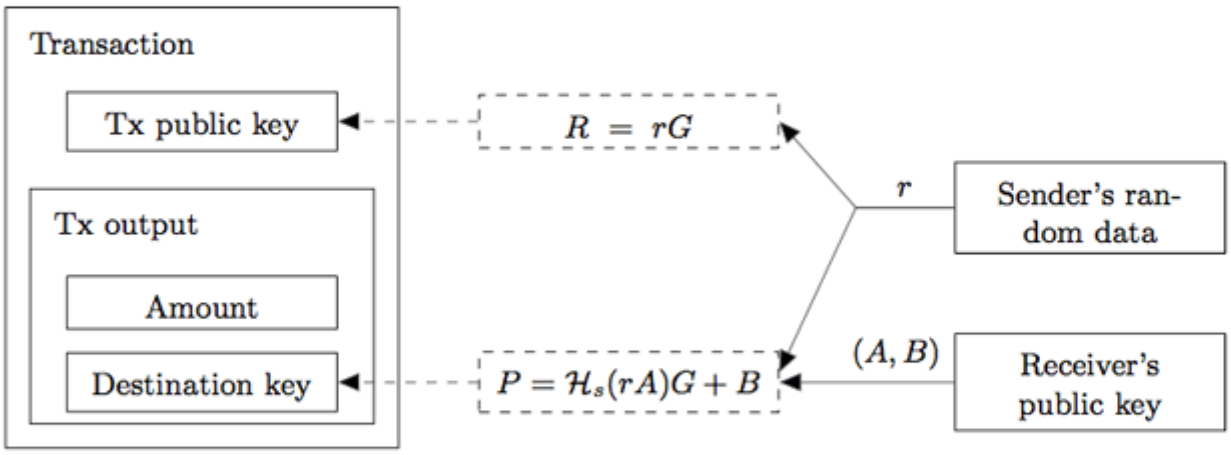


Fig. 4. Standard transaction structure.

4. Alice sends the transaction.
5. Bob checks every passing transaction with his private key (a, b), and computes $P' = \mathcal{H}_s(aR)G + B$. If Alice's transaction for with Bob as the recipient was among them, then $aR = arG = rA$ and $P' = P$.

6. Bob can recover the corresponding one-time private key: $x = \mathcal{H}_s(aR) + b$, so as $P = xG$. He can spend this output at any time by signing a transaction with x.

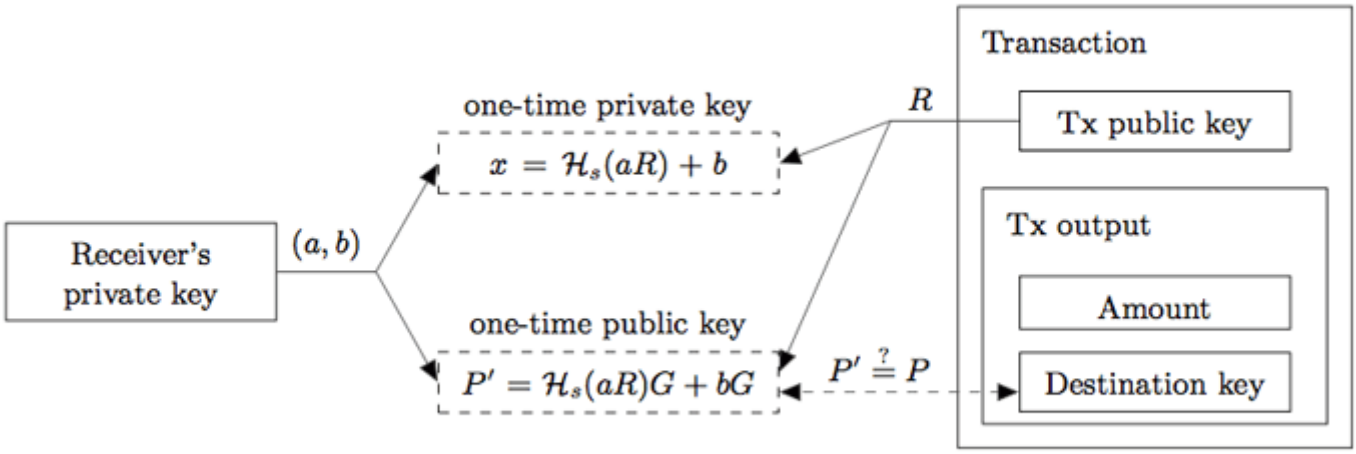


Fig. 5. Incoming transaction check.

As a result Bob gets incoming payments, associated with one-time public keys which are unlinkable for a spectator. Some additional notes:

```
When Bob "recognizes" his transactions (see step 5) he practically uses only half of his private information: (a,B). This pair, also known as the tracking key, can be passed to a third party (Carol). Bob can delegate her the processing of new transactions. Bob doesn't need to explicitly trust Carol, because she can't recover the one-time secret key p without Bob's full private key (a, b). This approach is useful when Bob lacks bandwidth or computation power (smartphones, hardware wallets etc.).
```

```
In case Alice wants to prove she sent a transaction to Bob's address she can either disclose r or use any kind of zero-knowledge protocol to prove she knows r (for example by signing the transaction with r).
```

```
If Bob wants to have an audit compatible address where all incoming transaction are linkable, he can either publish his tracking key or use a truncated address. That address represent only one public ec-key B, and the remaining part required by the protocol is derived from it as follows: a = Hs(B) and A = Hs(B)G. In both cases every person is able to "recognize" all of Bob's incoming transaction, but, of course, none can spend the funds enclosed within them without the secret key b.
```

One-time ring signatures

A protocol based on one-time ring signatures allows users to achieve unconditional unlinkability. Unfortunately, ordinary types of cryptographic signatures permit to trace transactions to their respective

senders and receivers. Our solution to this deficiency lies in using a different signature type than those currently used in electronic cash systems.
We will first provide a general description of our algorithm with no explicit reference to electronic cash.
A one-time ring signature contains four algorithms: (GEN, SIG, VER, LNK):

GEN: takes public parameters and outputs an ec-pair (P, x) and a public key I.

SIG: takes a message m, a set $\mathcal{S}'$ of public keys $\{P_i\}_{i \neq s}$, a pair (P_s, x_s) and outputs a signature σ and a set $\mathcal{S} = S' \cup \{P_s\}$.

VER: takes a message m, a set $\mathcal{S}$, a signature σ and outputs "true" or "false".

LNK: takes a set $\mathcal{I} = \{I_i\}$, a signature σ and outputs "linked" or "indep".

The idea behind the protocol is fairly simple: a user produces a signature which can be checked by a set of public keys rather than a unique public key. The identity of the signer is indistinguishable from the other users whose public keys are in the set until the owner produces a second signature using the same keypair.

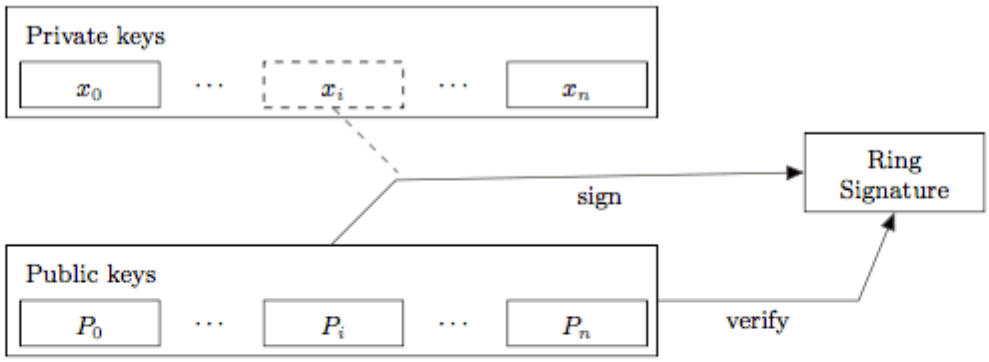


Fig. 6. Ring signature anonymity.

GEN: The signer picks a random secret key $x \in [1, l-1]$ and computes the corresponding public key $P = xG$. Additionally he computes another public key $I = x\mathcal{H}_p(P)$ which we will call the "key image".

SIG: The signer generates a one-time ring signature with a non-interactive zero-knowledge proof using the techniques from [21]. He selects a random subset $\mathcal{S}'$ of n from the other users' public keys P_i, his own keypair (x, P) and key image I. Let $0 \le s \le n$ be signer's secret index in $\mathcal{S}$ (so that his public key is P_s).

He picks a random $\{q_i \mid i = 0 \dots n\}$ and $\{w_i \mid i = 0 \dots n, i \neq s\}$ from $(1 \dots l)$ and applies the following *transformations*:

$$L_i = \begin{cases} q_i G, & \text{if } i = s \\ q_i G + w_i P_i, & \text{if } i \neq s \end{cases}$$

$$R_i = \begin{cases} q_i \mathcal{H}_p(P_i), & \text{if } i = s \\ q_i \mathcal{H}_p(P_i) + w_i I, & \text{if } i \neq s \end{cases}$$

The next step is getting the non-interactive *challenge*:

$$c = \mathcal{H}_s(m, L_1, \dots, L_n, R_1, \dots, R_n)$$

Finally the signer computes the *response*:

$$c_i = \begin{cases} w_i, & \text{if } i \neq s \\ c - \sum_{i=0}^{n} c_i \mod l, & \text{if } i = s \end{cases}$$

$$r_i = \begin{cases} q_i, & \text{if } i \neq s \\ q_s - c_s x \mod l, & \text{if } i = s \end{cases}$$

The resulting signature is $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$.

VER: The verifier checks the signature by applying the inverse transformations:

$$\begin{cases} L'_i = r_i G + c_i P_i \\ R'_i = r_i \mathcal{H}_p(P_i) + c_i I \end{cases}$$

Finally, the verifier checks if $\sum_{i=0}^{n} c_i \stackrel{?}{=} \mathcal{H}_s(m, L'_0, \ldots, L'_n, R'_0, \ldots, R'_n) \mod l$

If this equality is correct, the verifier runs the algorithm **LNK**. Otherwise the verifier rejects the signature.

LNK: The verifier checks if I has been used in past signatures (these values are stored in the set $\mathcal{I}$). Multiple uses imply that two signatures were produced under the same secret key.

The meaning of the protocol: by applying L-transformations the signer proves that he knows such x that at least one $P_i = xG$. To make this proof non-repeatable we introduce the key image as $I = x\mathcal{H}_p(P)$. The signer uses the same coefficients (r_i, c_i) to prove almost the same statement: he knows such x that at least one $\mathcal{H}_p(P_i) = I \cdot x^{-1}$.

If the mapping $x \to I$ is an injection:

1. Nobody can recover the public key from the key image and identify the signer;
2. The signer cannot make two signatures with different I's and the same x.

Standard Decentralized asymmetric algorithm consensus transaction

By combining both methods (unlinkable public keys and untraceable ring signature) Bob achieves new level of privacy in comparison with the original Bitcoin scheme. It requires him to store only one private key (a, b) and publish (A, B) to start receiving and sending anonymous transactions.

While validating each transaction Bob additionally performs only two elliptic curve multiplications and one addition per output to check if a transaction belongs to him. For his every output Bob recovers a one-time keypair (pi,Pi) and stores it in his wallet. Any inputs can be circumstantially proved to have the same owner only if they appear in a single transaction. In fact this relationship is much harder to establish due to the one-time ring signature.

With a ring signature Bob can effectively hide every input among somebody else's; all possible spenders will be equiprobable, even the previous owner (Alice) has no more information than any observer.

When signing his transaction Bob specifies n foreign outputs with the same amount as his output, mixing all of them without the participation of other users. Bob himself (as well as anybody else) does not know if any of these payments have been spent: an output can be used in thousands of signatures as an ambiguity factor and never as a target of hiding. The double spend check occurs in the LNK phase when checking against the used key images set.

Bob can choose the ambiguity degree on his own: n = 1 means that the probability he has spent the output is 50% probability, n = 99 gives 1%. The size of the resulting signature increases linearly as O(n + 1), so the improved anonymity costs to Bob extra transaction fees. He also can set n = 0 and make his ring signature to consist of only one element, however this will instantly reveal him as a spender.

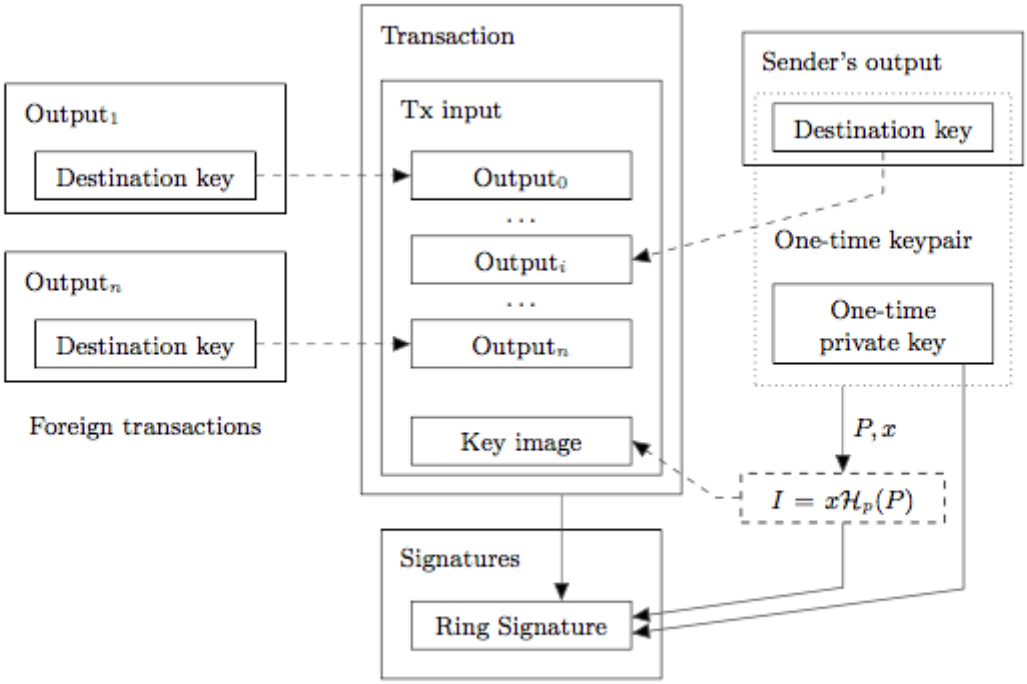


Fig. 7. Ring signature generation in a standard transaction.

Egalitarian Proof-of-work

In this section we propose and ground the new proof-of-work algorithm. Our primary goal is to close the gap between CPU (majority) and GPU/FPGA/ASIC (minority) miners. It is appropriate that some users can have a certain advantage over others, but their investments should grow at least linearly with the power. More generally, producing special-purpose devices has to be as less profitable as possible.

Definition

The RPCA proceeds in rounds. In each round:

```
Initially, each server takes all valid transactions it has seen prior to
the beginning of the consensus round that have not already been applied
(these may include new transactions initiated by endusers of the server,
transactions held over from a previous consensus process, etc.), and makes
them public in the form of a list known as the "candidate set".
```

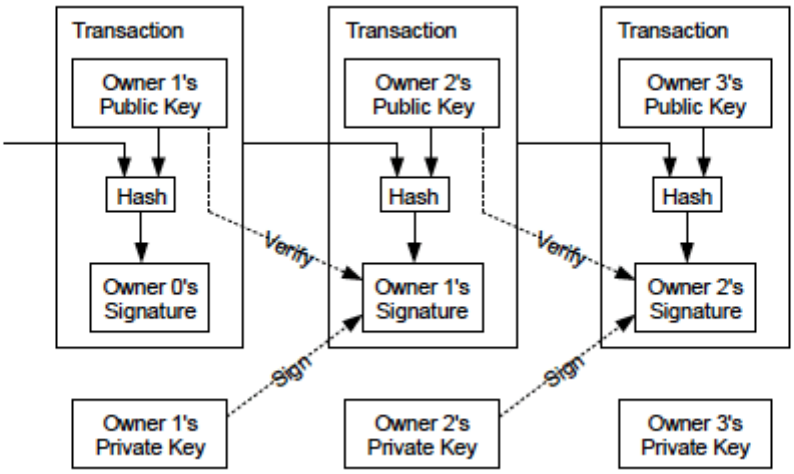
```
Each server then amalgamates the candidate sets of all servers on its UNL,
and votes on the veracity of all transactions.
```

```
Transactions that receive more than a minimum percentage of "yes" votes
are passed on to the next round, if there is one, while transactions that
do not receive enough votes will either be discarded, or included in the
candidate set for the beginning of the consensus process on the next
ledger.
```

```
The final round of consensus requires a minimum percentage of 80% of a
server's UNL agreeing on a transaction. All transactions that meet this
requirement are applied to the ledger, and that ledger is closed, becoming
the new last-closed ledger.
```

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, 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.

Confidential Transactions and OWAS

The first thing we need to do is remove Bitcoin Script. This is sad, but it is too powerful so it is impossible to merge transactions using general scripts. We will demonstrate that confidential transactions of Dr. Maxwell are enough (after some small modification) to authorize spending of outputs and also allows to make combined transactions without interaction. This is in fact identical to OWAS, and allows relaying nodes take some transaction fee or the recipient to change the transaction fees. These additional things Bitcoin can not do, we get for free.

We start by reminding the reader how confidential transactions work. First, the amounts are coded by the following equation:

```
C = r*G + v*H
```

where C is a Pedersen commitment, G and H are fixed nothing-up-my-sleeve elliptic curve group generators, v is the amount, and r is a secret random blinding key.
Attached to this output is a rangeproof which proves that v is in [0, 2^64], so that user cannot exploit the blinding to produce overflow attacks, etc.
To validate a transaction, the verifier will add commitments for all outputs, plus f*H (f here is the transaction fee which is given explicitly) and subtracts all input commitments. The result must be 0, which proves that no amount was created or destroyed overall.
We note that to create such a transaction, the user must know the sum of all the values of r for commitments entries. Therefore, the r-values (and their sums) act as secret keys. If we can make the r output values known only to the recipient, then we have an authentication system! Unfortunately, if we keep the rule that commits all add to 0, this is impossible, because the sender knows the sum of all _his_ r values, and therefore knows the receipient's r values sum to the negative of that. So instead, we allow the transaction to sum to a nonzero value k*G, and require a signature of an empty string with this as key, to prove its amount component is zero.
We let transactions have as many k*G values as they want, each with a signature, and sum them during verification.

To create transactions sender and recipient do following ritual:

```
1. Sender and recipient agree on amount to be sent. Call this b.

2. Sender creates transaction with all inputs and change output(s), and
gives recipient the total blinding factor (r-value of change minus
r-values of inputs) along with this transaction. So the commitments sum to
r*G - b*H.

3. Recipient chooses random r-values for his outputs, and values that sum
to b minus fee, and adds these to transaction (including range proof).
Now the commitments sum to k*G - fee*H for some k that only recipient
knows.

4. Recipient attaches signature with k to the transaction, and the
explicit fee. It has done.
```

Now, creating transactions in this manner supports OWAS already. To show this, suppose we have two transactions that have a surplus k1*G and k2*G, and the attached signatures with these. Then you can combine the lists of inputs and outputs of the two transactions, with both k1*G and k2*G to the mix, and voilÃ¡! is again a valid transaction. From the combination, it is impossible to say which outputs or inputs are from which original transaction.

Because of this, we change our block format from Bitcoin to this information:

```
1. Explicit amounts for new money (block subsidy or sidechain peg-ins)
with whatever else data this needs. For a sidechain peg-in maybe it
references a Bitcoin transaction that commits to a specific excess k*G
value?

2. Inputs of all transactions

3. Outputs of all transactions

4. Excess k*G values for all transactions
```

Each of these are grouped together because it do not matter what the transaction boundaries are originally. In addition, Lists 2 3 and 4 should be required to be coded in alphabetical order, since it is quick to check and prevents the block creator of leaking any information about the original transactions.
Note that the outputs are now identified by their hash, and not by their position in a transaction that could easily change. Therefore, it should be banned to have two unspent outputs are equal at the same time, to avoid confusion.

Traceability of Transactions

Privacy and anonymity are the most important aspects of electronic cash. Peer-to-peer payments seek to be concealed from third party's view, a distinct difference when compared with traditional banking. In particular, T. Okamoto and K. Ohta described six criteria of ideal electronic cash, which included "privacy: relationship between the user and his purchases must be untraceable by anyone." From their description, we derived two properties which a fully anonymous electronic cash model must satisfy in order to comply with the requirements outlined by Okamoto and Ohta:

```
Untraceability: for each incoming transaction all possible senders are
equiprobable.
```

```
Unlinkability: for any two outgoing transactions it is impossible to prove
they were sent to the same person.
```

Unfortunately, Bitcoin does not satisfy the untraceability requirement. Since all the transactions that take place between the network's participants are public, any transaction can be unambiguously traced to a unique origin and final recipient. Even if two participants exchange funds in an indirect way, a properly engineered path-finding method will reveal the origin and final recipient.
It is also suspected that Bitcoin does not satisfy the second property. Some researchers stated that a careful blockchain analysis may reveal a connection between the users of the Bitcoin network and their transactions. Although a number of methods are disputed, it is suspected that a lot of hidden personal information can be extracted from the public database.
Bitcoin's failure to satisfy the two properties outlined above leads us to conclude that it is not an anonymous

but a pseudo-anonymous electronic cash system. Users were quick to develop solutions to circumvent this shortcoming. Two direct solutions were "laundering services" and the development of distributed methods. Both solutions are based on the idea of mixing several public transactions and sending them through some intermediary address; which in turn suffers the drawback of requiring a trusted third party.
Recently, a more creative scheme was proposed by I. Miers et al.: "Zerocoin". Zerocoin utilizes a cryptographic one-way accumulators and zero-knoweldge proofs which permit users to "convert" bitcoins to zerocoins and spend them using anonymous proof of ownership instead of explicit public-key based digital signatures. However, such knowledge proofs have a constant but inconvenient size - about 30kb (based on today's Bitcoin limits), which makes the proposal impractical. Authors admit that the protocol is unlikely to ever be accepted by the majority of Bitcoin users.

Untraceable Transactions

In this section we propose a scheme of fully anonymous transactions satisfying both untraceability and unlinkability conditions. An important feature of our solution is its autonomy: the sender is not required to cooperate with other users or a trusted third party to make his transactions; hence each participant produces a cover traffic independently.

Merging Transactions Across Blocks

Now, we have used Dr. Maxwell's Confidential Transactions to create a noninteractive version of Dr. Maxwell's CoinJoin, but we have not seen the last of marvelous Dr. Maxwell! We need another idea, transaction cut-through, he described in. Again, we create a noninteractive version of this, and to show how it is used with several blocks.
We can imagine now each block as one large transaction. To validate it, we add all the output commitments together, then subtracts all input commitments, k*G values, and all explicit input amounts times H. We find that we could combine transactions from two blocks, as we combined transactions to form a single block, and the result is again a valid transaction. Except now, some output commitments have an input commitment exactly equal to it, where the first block's output was spent in the second block. We could remove both commitments and still have a valid transaction. In fact, there is not even need to check the rangeproof of the deleted output.
The extension of this idea all the way from the genesis block to the latest block, we see that EVERY nonexplicit input is deleted along with its referenced output. What remains are only the unspent outputs, explicit input amounts and every k*G value. And this whole mess can be validated as if it were one transaction: add all unspent commitments output, subtract the values k*G, validate explicit input amounts (if there is anything to validate) then subtract them times H. If the sum is 0, the entire chain is good.

What is this mean? When a user starts up and downloads the chain he needs the following
data from each block:

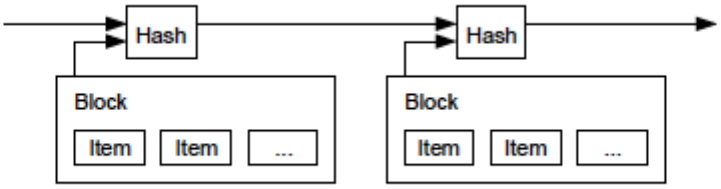
```
1. Explicit amounts for new money (block subsidy or sidechain peg-ins)
with whatever else data this needs.

2. Unspent outputs of all transactions, along with a merkle proof that
each output appeared in the original block.
```

3. Excess k*G values for all transactions.

Bitcoin today there are about 423000 blocks, totaling 80GB or so of data on the hard drive to validate everything. These data are about 150 million transactions and 5 million unspent nonconfidential outputs. Estimate how much space the number of transactions take on a Decentralized asymmetric algorithm consensus chain. Each unspent output is around 3Kb for rangeproof and Merkle proof. Each transaction also adds about 100 bytes: a k*G value and a signature. The block headers and explicit amounts are negligible. Add this together and get 30Gb -- with a confidential transaction and obscured transaction graph!

Timestamp Server

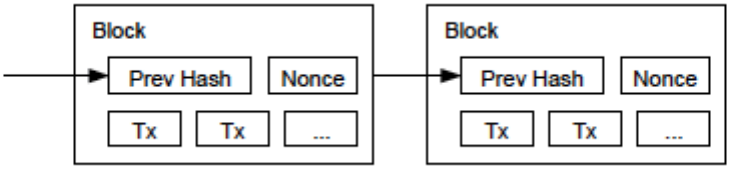
The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



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, 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.

Bitcoin creator Satoshi Nakamoto described the majority decision making algorithm as "one-CPU-one-vote" and used a CPU-bound pricing function (double SHA-256) for his proof-of-work scheme. Since users vote for the single history of transactions order, the reasonableness and consistency of this process are critical conditions for the whole system.

The security of this model suffers from two drawbacks. First, it requires 51% of the network's mining power to be under the control of honest users. Secondly, the system's progress (bug fixes, security fixes, etc...) require the overwhelming majority of users to support and agree to the changes (this occurs when the users update their wallet software).Finally this same voting mechanism is also used for collective polls about implementation of some features.

This permits us to conjecture the properties that must be satisfied by the proof-of-work pricing function. Such function must not enable a network participant to have a significant advantage over another participant; it requires a parity between common hardware and high cost of custom devices. From recent examples, we can see that the SHA-256 function used in the Bitcoin architecture does not posses this property as mining becomes more efficient on GPUs and ASIC devices when compared to high-end CPUs.

Therefore, Bitcoin creates favourable conditions for a large gap between the voting power of participants as it violates the "one-CPU-one-vote" principle since GPU and ASIC owners posses a much larger voting power when compared with CPU owners. It is a classical example of the Pareto principle where 20% of a system's participants control more than 80% of the votes.

One could argue that such inequality is not relevant to the network's security since it is not the small number of participants controlling the majority of the votes but the honesty of these participants that matters. However, such argument is somewhat flawed since it is rather the possibility of cheap specialized hardware appearing rather than the participants' honesty which poses a threat. To demonstrate this, let us take the following example. Suppose a malevolent individual gains significant mining power by creating his own mining farm through the cheap hardware described previously. Suppose that the global hashrate decreases significantly, even for a moment, he can now use his mining power to fork the chain and double-spend. As we shall see later in this article, it is not unlikely for the previously described event to take place.

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.

Incentive

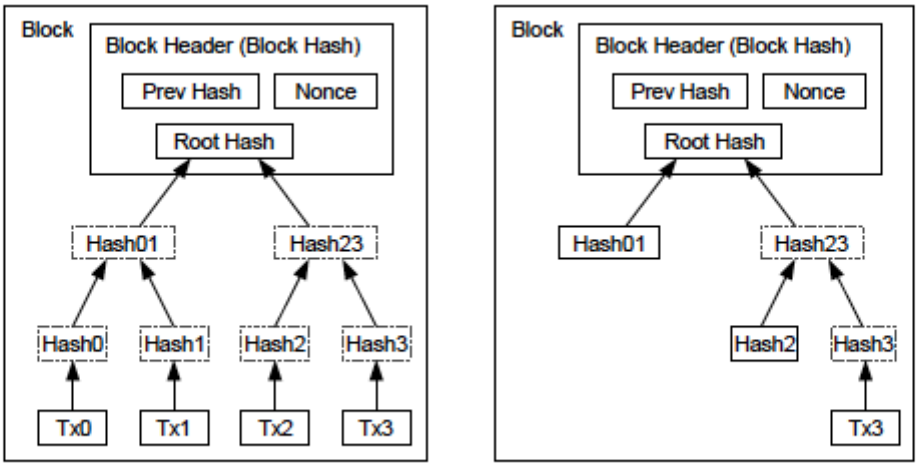
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of 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.

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.

Reclaiming Disk Space

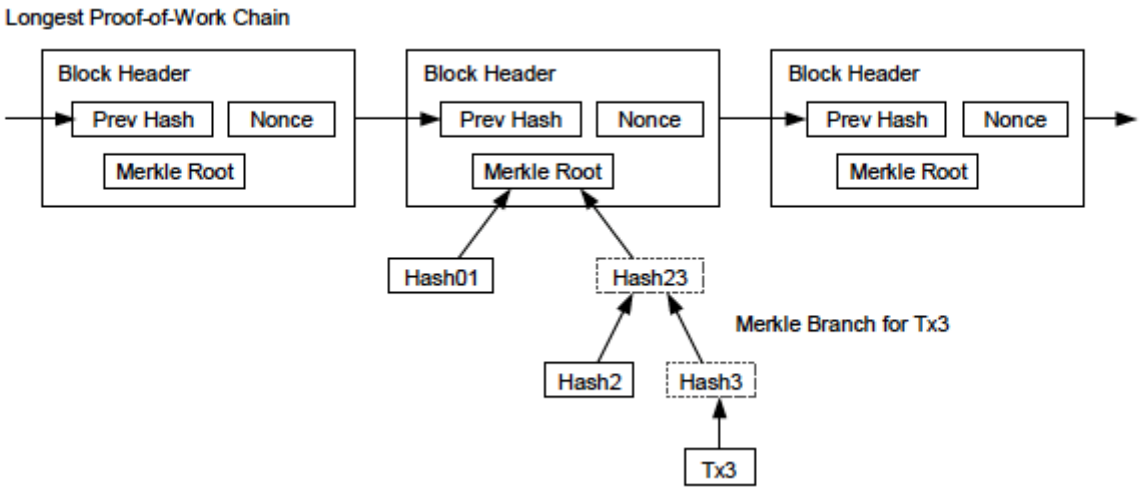
Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree, with only the root included in the block's hash. Old blocks can then 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 bytes * 6 * 24 * 365 = 4.2MB 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.

Simplified Payment Verification

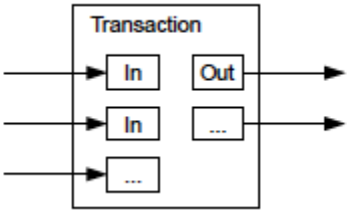
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction 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.

Combining and Splitting Value

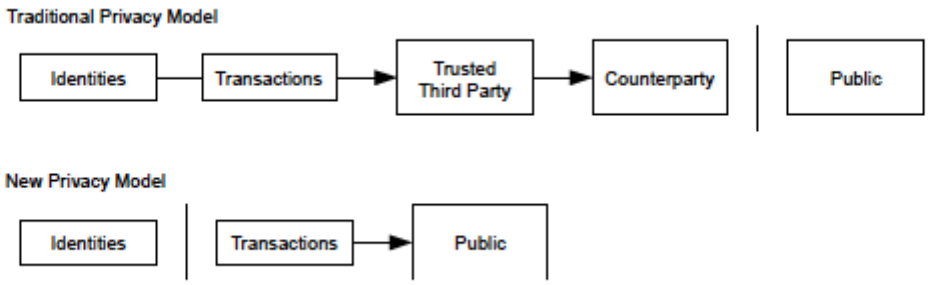
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous 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.

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.

The proposed algorithm

We propose a new memory-bound algorithm for the proof-of-work pricing function. It relies on random access to a slow memory and emphasizes latency dependence. As opposed to scrypt every new block (64 bytes in length) depends on all the previous blocks. As a result a hypothetical "memory-saver" should increase his calculation speed exponentially.

Our algorithm requires about 2 Mb per instance for the following reasons:

```
1. It fits in the L3 cache (per core) of modern processors, which should become mainstream in a few years;
```

```
2. A megabyte of internal memory is an almost unacceptable size for a modern ASIC pipeline;
```

```
3. GPUs may run hundreds of concurrent instances, but they are limited in other ways: GDDR5 memory is slower than the CPU L3 cache and remarkable for its bandwidth, not random access speed.
```

```
4. Significant expansion of the scratchpad would require an increase in iterations, which in turn implies an overall time increase. "Heavy" calls in a trust-less p2p network may lead to serious vulnerabilities, because nodes are obliged to check every new block's proof-of-work. If a node spends a considerable amount of time on each hash evaluation, it can be easily DDoSed by a flood of fake objects with arbitrary work data (nonce values).
```

Smooth emission

The upper bound for the overall amount of Decentralized asymmetric algorithm consensus digital coins is: MSupply = 2^64 - 1 atomic units. This is a natural restriction based only on implementation limits, not on intuition such as "N coins ought to be enough for anybody". To ensure the smoothness of the emission process we use the following formula for block rewards:BaseReward = (MSupply - A) >> 18, where A is amount of previously generated coins.

Difficulty

Decentralized asymmetric algorithm consensus contains a targeting algorithm which changes the difficulty of every block. This decreases the system's reaction time when the network hashrate is intensely growing or shrinking, preserving a constant block rate. The original Bitcoin method calculates the relation of actual and target time-span between the last 2016 blocks and uses it as the multiplier for the current difficulty. Obviously this is unsuitable for rapid recalculations (because of large inertia) and results in oscillations.

The general idea behind our algorithm is to sum all the work completed by the nodes and divide it by the time they have spent. The measure of work is the corresponding difficulty values in each block. But due to

inaccurate and untrusted timestamps we cannot determine the exact time interval between blocks. A user can shift his timestamp into the future and the next time intervals might be improbably small or even negative. Presumably there will be few incidents of this kind, so we can just sort the timestamps and cut-off the outliers (i.e. 20%). The range of the rest values is the time which was spent for 80% of the corresponding blocks.

Size limits

Users pay for storing the blockchain and shall be entitled to vote for its size. Every miner deals with the trade-off between balancing the costs and profit from the fees and sets his own "soft-limit" for creating blocks. Also the core rule for the maximum block size is necessary for preventing the blockchain from being flooded with bogus transaction, however this value should not be hard-coded.

Let MN be the median value of the last N blocks sizes. Then the "hard-limit" for the size of accepting blocks is 2 * MN . It averts the blockchain from bloating but still allows the limit to slowly grow with time if necessary.

Transaction size does not need to be limited explicitly. It is bounded by the size of a block; and if somebody wants to create a huge transaction with hundreds of inputs/outputs (or with the high ambiguity degree in ring signatures), he can do so by paying sufficient fee.

Excess size penalty

A miner still has the ability to stuff a block full of his own zero-fee transactions up to its maximum size 2 * Mb. Even though only the majority of miners can shift the median value, there is still a possibility to bloat the blockchain and produce an additional load on the nodes. To discourage malevolent participants from creating large blocks we introduce a penalty function:

$$NewReward = BaseReward \cdot \left(\frac{BlkSize}{M_N} - 1 \right)^2$$

This rule is applied only when BlkSize is greater than minimal free block size which should be close to max(10kb, MN Â· 110%). Miners are permitted to create blocks of "usual size" and even exceed it with profit when the overall fees surpass the penalty. But fees are unlikely to grow quadratically unlike the penalty value so there will be an equilibrium.

Correctness

In order to achieve correctness, given a maximal amount of Byzantine failures, it must be shown that it is impossible for a fraudulent transaction to be confirmed during consensus, unless the number of faulty nodes exceeds that tolerance. The proof of the correctness of the RPCA then follows directly: since a transaction is only approved if 80% of the UNL of a server agrees with it, as long as 80% of the UNL is honest, no fraudulent transactions will be approved. Thus for a UNL of n nodes in the network, the consensus protocol will maintain correctness so long as:

$$f \leq (n-1)/5$$

where f is the number Byzantine failures. In fact, even in the face of (n - 1)/5 + 1 Byzantine failures, correctness is still technically maintained. The consensus process will fail, but it will still not be possible to confirm a fraudulent transaction. Indeed it would take (4n + 1)/5 Byzantine failures for an incorrect transaction to be confirmed. We call this second bound the bound for weak correctness, and the former the bound for strong correctness.

It should also be noted that not all "fraudulent" trans- actions pose a threat, even if confirmed during consen- sus. Should a user attempt to double-spend funds in two transactions, for example, even if both transactions are confirmed during the consensus process, after the first transaction is applied, the second will fail, as the funds are no longer available. This robustness is due to the fact that transactions are applied deterministically, and that consensus ensures that all nodes in the network are applying the deterministic rules to the same set of transactions.

For a slightly different analysis, let us assume that the probability that any node will decide to collude and join a nefarious cartel is pc. Then the probability of correctness is given by p*, where:

$$p^* = \sum_{i=0}^{\lceil (\frac{n-1}{5}) \rceil} \binom{n}{i} p_c^i (1-p_c)^{n-i}$$

This probability represents the likelihood that the size of the nefarious cartel will remain below the maximal threshold of Byzantine failures, given pc. Since this likelihood is a binomial distribution, values of pc greater than 20% will result in expected cartels of size greater than 20% of the network, thwarting the consensus process. In practice, a UNL is not chosen randomly, but rather with the intent to minimize pc. Since nodes are not anonymous but rather cryptographically identifiable, selecting a UNL of nodes from a mixture of continents, nations, industries, ideologies, etc. will produce values of pc much lower than 20%. As an example, the probability of the Anti-Defamation League and the Westboro Baptist Church colluding to defraud the network, is certainly much, much smaller than 20%. Even if the UNL has a relatively large pc, say 15%, the probability of correctness is extremely high even with only 200 nodes in the UNL: 97.8%.

A graphical representation of how the probability of incorrectness scales as a function of UNL size for differing values of pc is depicted in Figure 1. Note that here the vertical axis represents the probability of a nefarious cartel thwarting consensus, and thus lower values indi- cate greater probability of consensus success. As can be seen in the figure, even with a pc as high as 10%, the probability of consensus being thwarted very quickly becomes negligible as the UNL grows past 100 nodes.

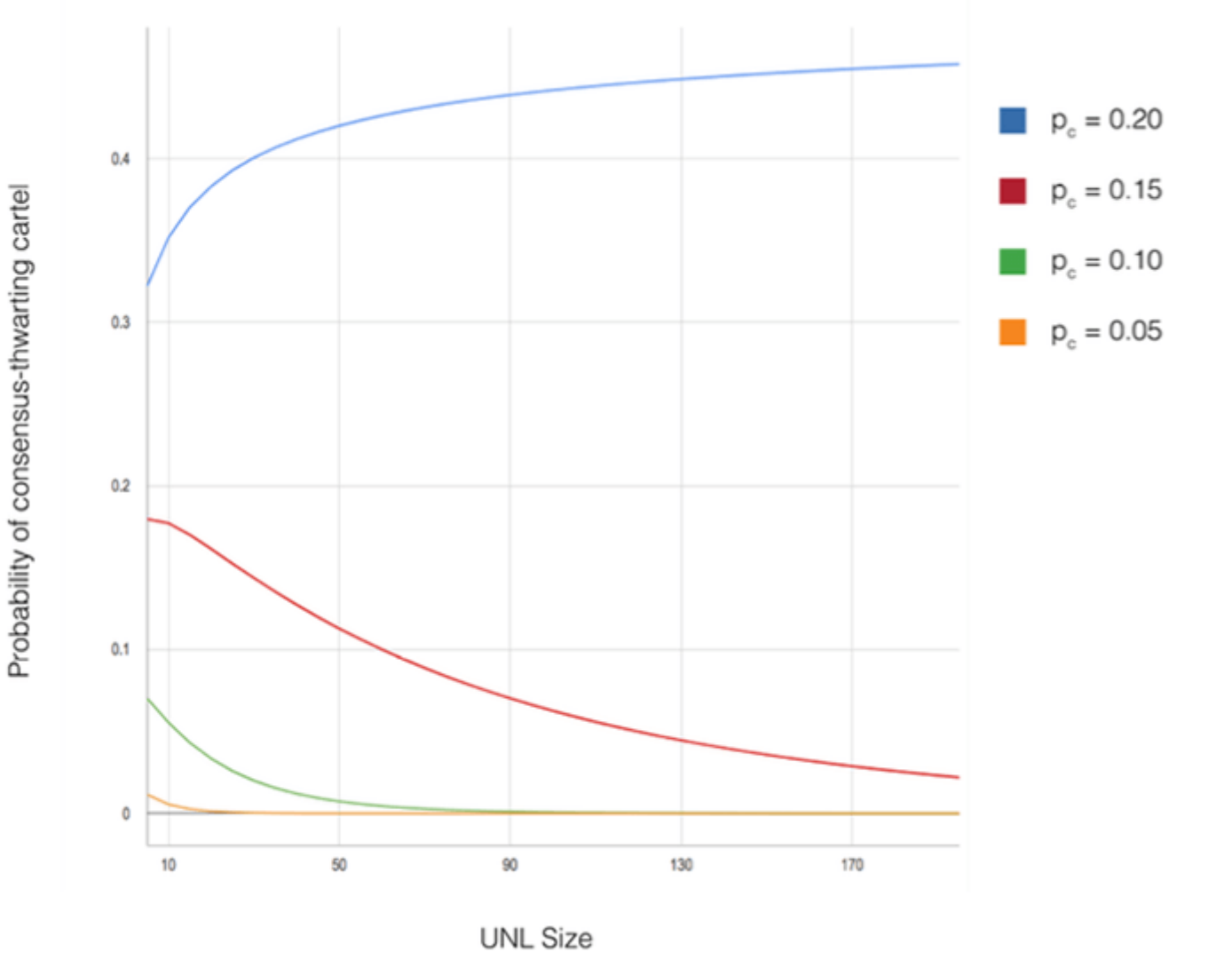


Figure 1. Probability of a nefarious cartel being able to thwart consensus as a function of the size of the UNL, for different values of pc, the probability that any member of the UNL will decide to collude with others. Here, lower values indicate a higher probability of consensus success.

Agreement

To satisfy the agreement requirement, it must be shown that all nonfaulty nodes reach consensus on the same set of transactions, regardless of their UNLs. Since the UNLs for each server can be different, agreement is not inherently guaranteed by the correctness proof. For example, if there are no restrictions on the membership of the UNL, and the size of the UNL is not larger than 0.2 * ntotal where ntotal is the number of nodes in the entire network, then a fork is possible. This is illustrated by a simple example (depicted in figure 2): imagine two cliques within the UNL graph, each larger than 0.2 * ntotal. By cliques, we mean a set of nodes where each nodeâ€™s UNL is the selfsame set of nodes. Because these two cliques do not share any members, it is possible for each to achieve a correct consensus independently of each other, violating agreement. If the connectivity of the two cliques surpasses 0.2 * ntotal, then a fork is no longer possible, as disagreement between the cliques would prevent consensus from being reached at the 80% agreement threshold that is required.

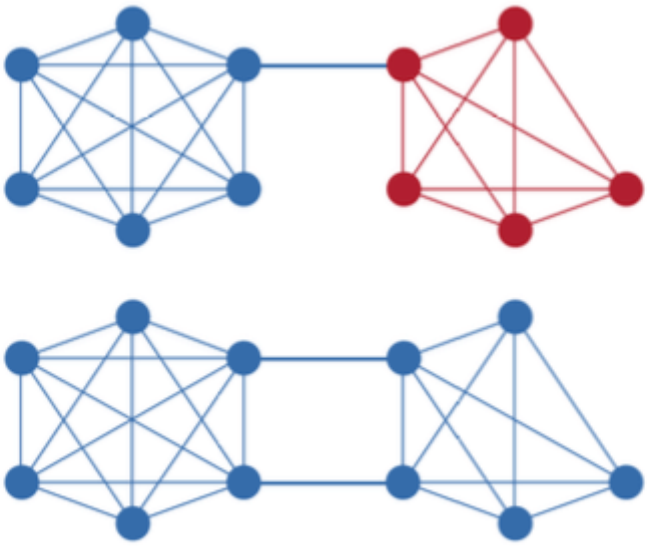


Figure 2. An example of the connectivity required to prevent a fork between two UNL cliques.

An upper bound on the connectivity required to prove agreement is given by:

$$|UNL_i \cap UNL_j| \geq \frac{1}{5}\max(|UNL_i|, |UNL_j|) \forall i, j$$

This upper bound assumes a clique-like structure of UNLs, i.e. nodes form sets whose UNLs contain other nodes in those sets. This upper bound guarantees that no two cliques can reach consensus on conflicting transactions, since it becomes impossible to reach the 80% threshold required for consensus. A tighter bound is possible when indirect edges between UNLs are taken into account as well. For example, if the structure of the network is not clique-like, a fork becomes much more difficult to achieve, due to the greater entanglement of the UNLs of all nodes.

It is interesting to note that no assumptions are made about the nature of the intersecting nodes. The intersection of two UNLs may include faulty nodes, but so long as the size of the intersection is larger than the bound required to guarantee agreement, and the total number of faulty nodes is less than the bound required to satisfy strong correctness, then both correctness and agreement will be achieved. That is to say, agreement is dependent solely on the size of the intersection of nodes, not on the size of the intersection of nonfaulty nodes.

Utility

While many components of utility are subjective, one that is indeed provable is convergence: that the consensus process will terminate in finite time.

We define convergence as the point in which the RPCA reaches consensus with strong correctness on the ledger, and that ledger then becomes the last-closed ledger. Note that while technically weak correctness still represents convergence of the algorithm, it is only convergence in the trivial case, as proposition C3 is violated, and no transactions will ever be confirmed. From the results above, we know that strong correctness is always achievable in the face of up to (n âˆ’ 1)/5 Byzantine failures, and that only one consensus will be achieved in the entire network so long as the UNL-connectedness con- dition is met (Equation 3). All that

remains is to show that when both of these conditions are met, consensus is reached in finite time.
Since the consensus algorithm itself is deterministic, and has a preset number of rounds, t, before consensus is terminated, and the current set of transactions are declared approved or not-approved (even if at this point no transactions have more than the 80% required agreement, and the consensus is only the trivial consensus), the limiting factor for the termination of the algorithm is the communication latency between nodes. In order to bound this quantity, the response-time of nodes is monitored, and nodes whoâ€™s latency grows larger than a preset bound b are removed from all UNLs. While this guarantees that consensus will terminate with an upper bound of tb, it is important to note that the bounds described for correctness and agreement above must be met by the final UNL, after all nodes that will be dropped have been dropped. If the conditions hold for the initial UNLs for all nodes, but then some nodes are dropped from the network due to latency, the correctness and agreement guarantees do not automatically hold but must be satisfied by the new set of UNLs.
As mentioned above, a latency bound heuristic is en- forced on all nodes in the Decentralized asymmetric algorithm consensus Network to guarantee that the consensus algorithm will converge. In addi- tion, there are a few other heuristics and procedures that provide utility to the RPCA.

```
There is a mandatory 2 second window for all nodes to propose their
initial candidate sets in each round of consensus. While this does
introduce a lower bound of 2 seconds to each consensus round, it also
guarantees that all nodes with reasonable latency will have the ability to
participate in the consensus process.
```

```
As the votes are recorded in the ledger for each round of consensus, nodes
can be flagged and removed from the network for some common,
easily-identifiable malicious behaviors. These include nodes that vote
"No" on every transaction, and nodes that consistently propose
transactions which are not validated by consensus.
```

```
A curated default UNL is provided to all users, which is chosen to
minimize pc, described in section 3.2. While users can and should select
their own UNLs, this default list of nodes guarantees that even naive
users will participate in a consensus process that achieves correctness
and agreement with extremely high probability.
```

```
A network split detection algorithm is also employed to avoid a fork in
the network. While the consensus algorithm certifies that the transactions
on the last-closed ledger are correct, it does not prohibit the
possibility of more than one last-closed ledger existing on different
subsections of the network with poor connectivity. To try and identify if
such a split has occurred, each node monitors the size of the active
members of its UNL. If this size suddenly drops below a preset threshold,
it is possible that a split has occurred. In order to prevent a false
```

```
positive in the case where a large section of a UNL has temporary latency,
nodes are allowed to publish a "partial validation", in which they do not
process or vote on transactions, but declare that are still participating
in the consensus process, as opposed to a different consensus process on a
disconnected subnetwork.
```

```
While it would be possible to apply the RPCA in just one round of
consensus, utility can be gained through multiple rounds, each with an
increasing minimum-required percentage of agreement, before the final
round with an 80% requirement. These rounds allow for detection of latent
nodes in the case that a few such nodes are creating a bottleneck in the
transaction rate of the network. These nodes will be able to initially
keep up during the lower-requirement rounds but fall behind and be
identified as the threshold increases. In the case of one round of
consensus, it may be the case that so few transactions pass the 80%
threshold, that even slow nodes can keep up, lowering the transaction rate
of the entire network.
```

Simulation Code

The provided simulation code demonstrates a round of RPCA, with parameterizable features (the number of nodes in the network, the number of malicious nodes, latency of messages, etc.). The simulator begins in perfect disagreement (half of the nodes in the network initially propose "yes", while the other half propose "no"), then proceeds with the consensus process, showing at each stage the number of yes/no votes in the network as nodes adjust their proposals based upon the proposals of their UNL members. Once the 80% threshold is reached, consensus is achieved. We encourage the reader to experiment with different values of the constants defined at the beginning of "Sim.cpp", in order to become familiar with the consensus process under different conditions.

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:

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:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \leq z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1-\sum_{k=0}^{z}\frac{\lambda^k e^{-\lambda}}{k!}\left(1-(q/p)^{(z-k)}\right)$$

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
```

Related works

The original Bitcoin proof-of-work protocol uses the CPU-intensive pricing function SHA-256. It mainly consists of basic logical operators and relies solely on the computational speed of processor, therefore is perfectly suitable for multicore/conveyer implementation.

However, modern computers are not limited by the number of operations per second alone, but also by memory size. While some processors can be substantially faster than others, memory sizes are less likely to vary between machines.

Memory-bound price functions were first introduced by Abadi et al and were defined as "functions whose computation time is dominated by the time spent accessing memory." The main idea is to construct an algorithm allocating a large block of data ("scratchpad") within memory that can be accessed relatively slowly (for example, RAM) and "accessing an unpredictable sequence of locations" within it. A block should be large enough to make preserving the data more advantageous than recomputing it for each access. The algorithm also should prevent internal parallelism, hence N simultaneous threads should require N times more memory at once.

Dwork et al investigated and formalized this approach leading them to suggest another variant of the pricing function: "Mbound". One more work belongs to F. Coelho, who proposed the most effective solution: "Hokkaido".

To our knowledge the last work based on the idea of pseudo-random searches in a big array is the algorithm known as "scrypt" by C. Percival. Unlike the previous functions it focuses on key derivation, and not proof-of-work systems. Despite this fact scrypt can serve our purpose: it works well as a pricing function in the partial hash conversion problem such as SHA-256 in Bitcoin.

By now scrypt has already been applied in Litecoin and some other Bitcoin forks. However, its implementation is not really memory-bound: the ratio "memory access time / overall time" is not large enough because each instance uses only 128 KB. This permits GPU miners to be roughly 10 times more effective and continues to leave the possibility of creating relatively cheap but highly-efficient mining devices.

Moreover, the scrypt construction itself allows a linear trade-off between memory size and CPU speed due to the fact that every block in the scratchpad is derived only from the previous. For example, you can store every second block and recalculate the others in a lazy way, i.e. only when it becomes necessary.

Literature review

Our scheme relies on the cryptographic primitive called a group signature. First presented by D. Chaum and E. van Heyst, it allows a user to sign his message on behalf of the group. After signing the message the user provides (for verification purposes) not his own single public key, but the keys of all the users of his group. A verifier is convinced that the real signer is a member of the group, but cannot exclusively identify the signer.

The original protocol required a trusted third party (called the Group Manager), and he was the only one who could trace the signer. The next version called a ring signature, introduced by Rivest et al. in, was an

autonomous scheme without Group Manager and anonymity revocation. Various modifications of this scheme appeared later: linkable ring signature allowed to determine if two signatures were produced by the same group member, traceable ring signature limited excessive anonymity by providing possibility to trace the signer of two messages with respect to the same metainformation (or "tag").
A similar cryptographic construction is also known as a ad-hoc group signature. It emphasizes the arbitrary group formation, whereas group/ring signature schemes rather imply a fixed set of members.
For the most part, our solution is based on the work "Traceable ring signature" by E. Fujisaki and K. Suzuki. In order to distinguish the original algorithm and our modification we will call the latter a one-time ring signature, stressing the user's capability to produce only one valid signature under his private key. We weakened the traceability property and kept the linkability only to provide one-timeness: the public key may appear in many foreign verifying sets and the private key can be used for generating a unique anonymous signature. In case of a double spend attempt these two signatures will be linked together, but revealing the signer is not necessary for our purposes.

Discussion

We have described the RPCA, which satisfies the conditions of correctness, agreement, and utility which we have outlined above. The result is that the Decentralized asymmetric algorithm consensus Protocol is able to process secure and reliable transactions in a matter of seconds: the length of time required for one round of consensus to complete. These transactions are provably secure up to the bounds outlined in section 3, which, while not the strongest available in the literature for Asynchronous Byzantine consensus, do allow for rapid convergence and flexibility in network membership. When taken together, these qualities allow the Decentralized asymmetric algorithm consensus Network to function as a fast and low-cost global payment network with well-understood security and reliability properties.
While we have shown that the Decentralized asymmetric algorithm consensus Protocol is provably secure so long as the bounds described in equations 1 and 3 are met, it is worth noting that these are maximal bounds, and in practice the network may be secure under significantly less stringent conditions. It is also important to recognize, however, that satisfying these bounds is not inherent to the RPCA itself, but rather requires management of the UNLs of all users. The default UNL provided to all users is already sufficient, but should a user make changes to the UNL, it must be done with knowledge of the above bounds. In addition, some monitoring of the global network structure is required in order to ensure that the bound in equation 3 is met, and that agreement will always be satisfied.
We believe the RPCA represents a significant step forward for distributed payment systems, as the low-latency allows for many types of financial transactions previously made difficult or even impossible with other, higher latency consensus methods.

Questions and Intuition

Here are some questions that since these weeks, dreams asked me and I woke up sweating. But in fact it is OK.

```
Q. If you delete the transaction outputs, user cannot verify the
rangeproof and maybe a negative amount is created.

A. This is OK. For the entire transaction to validate all negative amounts
must have been destroyed. User have SPV security only that no illegal
```

```
inflation happened in the past, but the user knows that _at this time_ no
inflation occurred.
```

```
Q. If you delete the inputs, double spending can happen.

A. In fact, this means: maybe someone claims that some unspent output was
spent in the old days. But this is impossible, otherwise the sum of the
combined transaction could not be zero.

An exception is that if the outputs are amount zero, it is possible to
make two that are negatives of each other, and the pair can be revived
without anything breaks. So to prevent consensus problems, outputs
0-amount should be banned. Just add H at each output, now they all amount
to at least 1.
```

Future Research

Here are some questions I can not answer at the time of this writing.

```
1. What script support is possible? We would need to translate script
operations into some sort of discrete logarithm information.
```

```
2. We require user to check all k*G values, when in fact all that is
needed is that their sum is of the form k*G. Instead of using signatures
is there another proof of discrete logarithm that could be combined?
```

```
3. There is a denial-of-service option when a user downloads the chain,
the peer can give gigabytes of data and list the wrong unspent outputs.
The user will see that the result do not add up to 0, but cannot tell
where the problem is.

For now maybe the user should just download the blockchain from a Torrent
or something where the data is shared between many users and is reasonably
likely to be correct.
```

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.

We have investigated the major flaws in Bitcoin and proposed some possible solutions. These ad- vantageous features and our ongoing development make new electronic cash system Decentralized asymmetric algorithm consensus a serious rival to Bitcoin, outclassing all its forks.

Nobel prize laureate Friedrich Hayek in his famous work proves that the existence of con- current independent currencies has a huge positive effect. Each currency issuer (or developer in our case) is trying to attract users by improving his product. Currency is like a commodity: it can have unique benefits and shortcomings and the most convenient and trusted currency has the greatest demand. Suppose we had a currency excelling Bitcoin: it means that Bitcoin would develop faster and become better. The biggest support as an open source project would come from its own users, who are interested in it.

We do not consider Decentralized asymmetric algorithm consensus as a full replacement to Bitcoin. On the contrary, having two (or more) strong and convenient currencies is better than having just one. Running two and more different projects in parallel is the natural flow of electronic cash economics.

References

W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

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.

S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, pages 99-111, 1991.

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.

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.

A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

R.C. Merkle, "Protocols for public key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980.

W. Feller, "An introduction to probability theory and its applications," 1957.

Koinster, "White Paper Generator," http://whitepaper.koinster.com, 2017.

Nakamoto, Satoshi. "Bitcoin: A peer-to-peer elec- tronic cash system." Consulted 1.2012 (2008): 28.

Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine generals problem." ACM Transactions on Programming Languages and Systems (TOPLAS) 4.3 (1982): 382-401.

Attiya, C., D. Dolev, and J. Gill. "Asynchronous Byzantine Agreement." Proc. 3rd. Annual ACM Symposium on Principles of Distributed Computing. 1984.

Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." Journal of the ACM (JACM) 32.2 (1985): 374-382.

Martin, J-P., and Lorenzo Alvisi. "Fast byzantine consensus." Dependable and Secure Computing, IEEE Transactions on 3.3 (2006): 202-215.

Alchieri, Eduardo AP, et al. "Byzantine consensus with unknown participants." Principles of Distributed Systems. Springer Berlin Heidelberg, 2008. 22-40.

http://bitcoin.org.

https://en.bitcoin.it/wiki/Category:Mixing Services.

http://blog.ezyang.com/2012/07/secure-multiparty-bitcoin-anonymization.

https://bitcointalk.org/index.php?topic=279249.0.

http://msrvideo.vo.msecnd.net/rmcvideos/192058/dl/192058.pdf.

https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki#Specification.

https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki#Backwards Compatibility.

https://en.bitcoin.it/wiki/Mining hardware comparison.

https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki.

http://luke.dashjr.org/programs/bitcoin/files/charts/branches.html.

https://bitcointalk.org/index.php?topic=196259.0.

https://en.bitcoin.it/wiki/Contracts.

https://en.bitcoin.it/wiki/Script.

http://litecoin.org.

Martin Abadi, Michael Burrows, and Ted Wobber. Moderately hard, memory-bound functions. In NDSS, 2003.

Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Ad-hoc-group signatures from hijacked keypairs. In in DIMACS Workshop on Theft in E-Commerce, 2005.

Man Ho Au, Sherman S. M. Chow, Willy Susilo, and Patrick P. Tsang. Short linkable ring signatures revisited. In EuroPKI, pages 101-115, 2006.

Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. J. Cryptographic Engineering, 2(2):77-89, 2012.

David Chaum and Eug`ene van Heyst. Group signatures. In EUROCRYPT, pages 257-265, 1991.

Fabien Coelho. Exponential memory-bound functions for proof of work protocols. IACR Cryptology ePrint Archive, 2005:356, 2005.

Ronald Cramer, Ivan Damg ÌŠard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In CRYPTO, pages 174-187, 1994.

Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In CRYPTO, pages 426-444, 2003.

Eiichiro Fujisaki. Sub-linear size traceable ring signatures without random oracles. In CT- RSA, pages 393-415, 2011.

Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Public Key Cryptogra- phy, pages 181-200, 2007.

Jezz Garzik. Peer review of "quantitative analysis of the full bitcoin transaction graph". https://gist.github.com/3901921, 2012.

Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In ACISP, pages 325-335, 2004.

Joseph K. Liu and Duncan S. Wong. Linkable ring signatures: Security models and new schemes. In ICCSA (2), pages 614-623, 2005.

Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In IEEE Symposium on Security and Privacy, pages 397- 411, 2013.

Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the bitcoin transaction

graph. Future internet, 5(2):237-250, 2013.

Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In CRYPTO, pages 324-337, 1991.

Marc Santamaria Ortega. The bitcoin transaction graph â€" anonymity. Masterâ€™s thesis, Universitat Oberta de Catalunya, June 2013.

Colin Percival. Stronger key derivation via sequential memory-hard functions. Presented at BSDCanâ€™09, May 2009.

Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. CoRR, abs/1107.4524, 2011.

Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In ASIACRYPT, pages 552-565, 2001.

Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. IACR Cryptology ePrint Archive, 2012:584, 2012.

Meni Rosenfeld. Analysis of hashrate-based double-spending. 2012.

Maciej Ulas. Rational points on certain hyperelliptic curves over finite fields. Bulletin of the Polish Academy of Sciences. Mathematics, 55(2):97-104, 2007.

Qianhong Wu, Willy Susilo, Yi Mu, and Fangguo Zhang. Ad hoc group signatures. In IWSEC, pages 120-135, 2006.

https://people.xiph.org/~greg/confidential_values.txt

https://bitcointalk.org/index.php?topic=279249.0

https://cryptonote.org/whitepaper.pdf

https://eprint.iacr.org/2015/1098.pdf

https://download.wpsoftware.net/bitcoin/wizardry/horasyuanmouton-owas.pdf

http://blockstream.com/sidechains.pdf

http://fr.harrypotter.wikia.com/wiki/Sortilege_de_Langue_de_Plomb

https://bitcointalk.org/index.php?topic=281848.0

Donate

Feel free to support this project by donating Ethereum here:

```
0x3C4B3bAaBc372bA3Fc6a2bbdB4A962f05De9415B
```