CryptoNote v 2.0

Nicolas van Saberhagen

October 17, 2013

1 Introduction

"Bitcoin" [1] 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, "CryptoNote", a name emphasizing the next breakthrough in electronic cash.

2 Bitcoin drawbacks and some possible solutions

2.1 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" [30]. 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

The general public has never heard of proof-of-work. However, the author is correct in saying usage is growing at a "steady pace." On a log scale, anyhow.

The low fees attract merchants as much as customers. Shop and restaurant owners dropping from 3.5% credit card fees to essentially 0% bitcoin fees gain an edge in the market.

Many folks, myself included, feel that bitcoin is more complicated than physical cash. Even obtaining bitcoin is a pain in the butt, finding a place to use it in real life is all but impossible, and don't even try to explain the cryptography to your grandmother.

Having said that, using any technology in it's infant stages is notoriously a pain in the butt. The convenience of Bitcoin (again, in it's current state) depends entirely upon whether merchants accept it. The idealized convenience and simplicity of Bitcoin is quite a bit different from the every-day convenience and simplicity of Bitcoin.

The author's proposed solution to this inflexibility opens up a new can of worms (which we'll talk about eventually here): "What if our adaptive system, not requiring a hard implementation of new features, is ill-posed and grows wildly out of control?" This swings the pendulum to the other end of the flexibility spectrum, an area where we're constantly over-correcting for a wildly fish-tailing system.

This "over-sensitivity" problem may not be an issue, but there may be other issues. If it is an issue, it may be resolved fairly easily. We'll see how all this goes down.

I disagree that privacy/anonymity is the most important trait, and so should you. Imagine that you are 11 years old, and you and a friend want to use files on a computer as money, because you're a silly 11 year old. One way you could trick your friend is to hit "CTRL-C, CTRL-V" the file a bunch of times to print more "money files," presuming you were a dishonest 11 year old. You could then simply give yourself as much money as you wanted. If this is the case, then your version of money can't really be used as money at all. Hence, some property other than privacy is more important. Such as "unforgeability", a finite supply, "uniqueness" (or semi-uniqueness), or at least some inability or computational infeasibility to duplicate.

Further, anonymity should be optional or at least scalable. For example, what if you want to prove you paid your water bill by pointing to the blockchain? Or what if you want to prove, by pointing to the blockchain, and without revealing any private information, the following statement: At least one address controlled by my company paid at least X Bitcoin in payroll across the following database of addresses, with a particular distribution of salary sizes.

Luckily, Cryptonote seems to implement this latter property.

CryptoNote v 2.0

Nicolas van Saberhagen

October 17, 2013

1 Introduction

"Bitcoin" [1] 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, "CryptoNote", a name emphasizing the next breakthrough in electronic cash.

2 Bitcoin drawbacks and some possible solutions

2.1 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" [30]. 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

In traditional banking, verification of transactions uses a "trusted" third party, and violation of anonymity can be limited to that third party alone. However, this is, of course, a centralized structure and any corruption causes massive problems. My point is this: payments in meatspace are still often sought to be kept secret. See, e.g., so-called "secret" Swiss banking. Banking, as far as I know, was largely been an anonymous endeavor before the First World War. Laissez-faire economics, and all that; perhaps this was just in America? History reading required.

On the other hand, why is anonymity considered an ideal feature of electronic cash? Simply general privacy principles, or to help assist in fungibility? Is there something fundamental about *money* that requires anonymity? I don't think so. We just have a tendency to think about how physical coins work.

Perhaps a barebones, minimalistic system doesn't need this privacy principle. Of course, we all want privacy.

http://link.springer.com/chapter/10.1007/3-540-46766-1_27

The reason untraceability is phrased this way is due to the usage of ring signatures as a core piece of the technology behind cryptonote. We can interpret Okamoto and Ohta's privacy notion to be restated in the following cryptography context:

Alex wants to send money to Brenda by transmitting a message to the network, but if Eva overhears the message, then Eva can't determine if it was Alex who sent the message, or Brenda, or Charlie, or whoever. Furthermore, if Eva intercepts two different messages, and even if Eva can ascertain somehow that one of them has gone to Brenda, it will be impossible for her to determine if the other one did as well. Kind of an uncertainty principle thing?

Example 1, Bitcoin and it's variants: Alex wants to send some money to Brenda. A Alex broadcasts a message like "I, Alex, send 0.112 Bitcoin to Brenda" and signs it with her private key and the previous UTXOs (except with bitcoin addresses instead of names). If Eva overhears this broadcast, she'll see those addresses. So bitcoin fails untraceability.

In Cryptonote, Alex chooses a group of people to include in her message, say Chris, Duke, Edward, and Frank. Alex broadcasts a message like: "One of the folks in the group Alex, Chris, Duke, Edward, and Frank, want to send 0.112 Cryptonote to Brenda," and she signs it with her private key and A, C, D, E, and F's public keys. When she broadcasts her transaction, due to the choice of a ring signature scheme, if Eva overhears her transaction, she'll have no idea *which* member of the group actually sent that money. So cryptonote implements untraceability.

Okamoto and Ohta's privacy notion: "The privacy of the user should be protected. That is, the relationship between the user and his purchases must be untraceable by anyone."

van Saberhagen's notion of untraceability: "Given a transaction, all senders [included in the ring signature] are equiprobable."

van Saberhagen's notion of unlinkability: "Given two transactions with receivers X and Y, it's impossible [or at least computationally infeasible] to determine if X=Y."

Comments on page 1

CryptoNote v 2.0

Nicolas van Saberhagen

October 17, 2013

1 Introduction

"Bitcoin" [1] 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, "CryptoNote", a name emphasizing the next breakthrough in electronic cash.

2 Bitcoin drawbacks and some possible solutions

2.1 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" [30]. 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

Bitcoin definitely fails "untraceability." When I send you BTC, the wallet from which it is sent is irrevocably stamped on the blockchain. There is no question about who sent those funds, because only the knower of the private keys can send them.

Comments on page 1

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 ([33, 35, 29, 31]) 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 [25], 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" [2] and the development of distributed methods [3, 4]. 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. [28]: "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 [5].

2.2 The proof-of-work function

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 [1], 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) [6].Finally this same voting mechanism is also used for collective polls about implementation of some features [7].

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 [8], 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

Presumably, if every user helps their own anonymity out by always generating a new address for EVERY received payment (which is absurd but technically the "correct" way to do it), and if every user helped out everyone else's anonymity by insisting that they never send funds to the same BTC address twice, then Bitcoin would still only *circumstantially* pass the unlinkability test.

Why? Consumer data can be used to figure an astonishing amount about people all the time. See, for example http://www.applieddatalabs.com/content/target-knows-it-shows

Now, imagine this is 20 years in the future and further imagine that Target didn't just know about your purchase habits at Target, but they had been mining the blockchain for ALL OF YOUR PERSONAL PURCHASES WITH YOUR COINBASE WALLET FOR THE PAST TWELVE YEARS.

They'll be like "hey buddy you might want to pick up some cough medicine tonight, you won't feel well tomorrow."

This may not be the case if multi-party sorting is exploited correctly. See, for example, this blog post: http://blog.ezyang.com/2012/07/secure-multiparty-bitcoin-anonymization/

I'm not totally convinced of the math on that, but ... one paper at a time, right?

Citation needed. Whereas the Zerocoin protocol (standalone) may be insufficient, the Zerocash protocol seems to have implemented a 1kb sized transactions. That project is supported by the US and Israeli militaries, of course, so who knows about it's robustness. On the other hand, no one wants to be able to spend funds without oversight more than the military. http://zerocash-project.org/

I'm not convinced... see, for example, http://fc14.ifca.ai/bitcoin/papers/bitcoin14_submission_12.pdf

Quoting a Cryptonote developer Maurice Planck (presumably a pseudonym) from the cryptonote fora:

"Zerocoin, Zerocash. This is the most advanced technology, I must admit. Yes, the quote above is from the analysis of the previous version of the protocol. To my knowledge, it's not 288, but 384 bytes, but anyway this is good news.

**They used a brand new technic called SNARK, which has certain downsides: for example, large initial database of public parameters required to create a signature (more than 1 GB) and significant time required to create a transaction (more than a minute). Finally, they're using a young crypto, which I've mentioned to be an arguable idea: https://forum.cryptonote.org/viewtopic.php?f=
" - Maurice P. Thu Apr 03, 2014 7:56 pm**

A function that is performed in the CPU and is not suitable for GPU, FPGA, or ASIC computation.

The "puzzle" used in proof-of-work is referred to as the pricing function, cost function, or puzzle function.

Comments on page 2

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 ([33, 35, 29, 31]) 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 [25], 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" [2] and the development of distributed methods [3, 4]. 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. [28]: "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 [5].

2.2 The proof-of-work function

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 [1], 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) [6].Finally this same voting mechanism is also used for collective polls about implementation of some features [7].

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 [8], 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

Not that it matters too much when a billion people in the world live on less than a dollar a day and have no hope in ever participating in any sort of mining network... but an economic world driven by a p2p currency system with one-cpu-one-vote would be, presumably, more fair than a system driven by fractional reserve banking.

But Cryptonote's protocol still requires 51% honest users... see, for example, the Cryptonote forums where one of the developers, Pliskov, says that a traditional replace-the-data-on-the-blockchain 51% attack can still work. https://forum.cryptonote.org/viewtopic.php?f=2&t=198

Note that you don't really need 51% honest users. You just really need "no single dishonest faction with more than 51% of the hashing power of the network."

Let's call this so-called problem of bitcoin "adaptive rigidity." Cryptonote's solution to adaptive rigidity is adaptive flexibility in the protocol parameter values. If you need block sizes bigger, no problem, the network will have been gently adjusting the whole time. That is to say, the way that Bitcoin adjusts difficulty over time can be replicated across all of our protocol parameters so that network consensus need not be obtained for updating the protocol.

On the surface this seems like a good idea, but without careful forethought, a self-adjusting system can become quite unpredictable and chaotic. We'll look further into this later as the opportunities arise. "Good" systems are somewhere between adaptively rigid and adaptively flexible, and perhaps even the rigidity itself are adaptive.

If we truly had "one-CPU-one-vote," then collaborating and developing pools to get to 51% would be more difficult. We would expect every CPU in the world to be mining, from phones to the on-board CPU in your Tesla while it's charging.

http://en.wikipedia.org/wiki/Pareto_principle

I claim that the Pareto equilibrium is somewhat unavoidable. Either 20% of the system will own 80% of the CPUs, or 20% of the system will own 80% of the ASICs. I hypothesize this because the underlying distribution of wealth in society already exhibits the Pareto distribution, and as new miners join up, they are drawn from that underlying distribution.

However, I argue that protocols with one-cpu-one-vote will see ROI on hardware. Block reward per node will be more closely proportional to number of nodes in the network because distribution of performance across the nodes will be much more tight. Bitcoin, on the other hand, sees a block reward (per node) more proportional to the computational capacity of that node. That is to say, only the "big boys" are still in the mining game. On the other hand, even though the Pareto principle will still be in play, in a one-cpu-one-vote world, *everyone* participates in network security and gains a bit of mining income.

In an ASIC world, it's not sensible to rig every XBox and cell phone to mine. In a one-cpu-one-vote world, it's very sensible in terms of mining reward. As a delightful consequence, gaining 51% of the vote is more difficult when there are more and more votes, yielding a lovely benefit to network security..

Comments on page 2

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.

2.3 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 [36].

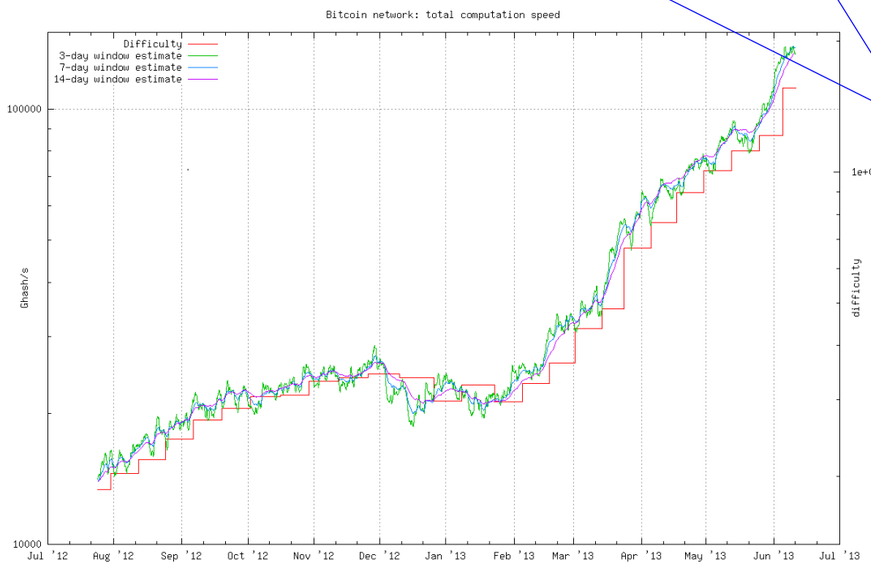

Fig. 1. Bitcoin hashrate chart
(source: http://bitcoin.sipa.be)

2.4 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

Let's call this what it is, a zombie attack. Let's discuss how continuous emission may be related to one-cpu-one-vote in a zombie attack scenario.

In a one-cpu-one-vote world, every cell phone and car, whenever idle, would be mining. Collecting heaps of cheap hardware to create a mining farm would be very very easy, because just about everything has a CPU in it. On the other hand, at that point, the number of CPUs required to launch a 51% attack would be quite astonishing, I would think. Furthermore, precisely *because* it would be easy to collect cheap hardware, we can reasonably expect a lot of folks to start hoarding anything with a CPU. The arms race in a one-cpu-one-vote world is necessarily more egalitarian than in an ASIC world. Hence, a discontinuity in network security due to emission rates should be LESS of a problem in a one-cpu-one-vote world.

However, two facts remain: 1) discontinuity in emission rate can lead to a stuttering effect in the economy and in network security both, which is bad, and 2) even though a 51% attack performed by someone collecting cheap hardware can still occur in a one-cpu-one-vote world, it seems like it should be harder.

Presumably, the safeguard against this is that *all* the dishonest actors will be trying this simultaneously, and we fall back to Bitcoin's previous security notion: "we require no dishonest faction to control more than 51% of the network."

The author is claiming here that one problem with bitcoin is that discontinuity in coin emission rate could lead to sudden drops in network participation, and hence network security. Thus, a continuous, differentiable, smooth coin emission rate is preferable.

The author ain't wrong, necessarily. Any sort of sudden decrease in network participation can lead to such a problem, and if we can remove one source of it, we should. Having said that, it's possible that long periods of "relatively constant" coin emission punctuated by sudden changes is the ideal way to go from an economics point of view. I'm not an economist. So, perhaps we must decide if we are going to trade network security for economic something-whatsit here?

http://arxiv.org/abs/1402.2009

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 250kb[1]. 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 [9]. 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 [10] 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 [11], 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.

2.5 Bulky scripts

The scripting system in Bitcoin is a heavy and complex feature. It *potentially* allows one to create sophisticated transactions [12], but some of its features are disabled due to security concerns and some have never even been used [13]. 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.

3 The CryptoNote Technology

Now that we have covered the limitations of the Bitcoin technology, we will concentrate on presenting the features of CryptoNote.

4 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.

4.1 Literature review

Our scheme relies on the cryptographic primitive called a *group signature*. First presented by D. Chaum and E. van Heyst [19], 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

[1] This is so-called "soft limit" — the reference client restriction for creating new blocks. Hard maximum of *possible* blocksize was 1 MB

In retrospect, it seems to have been a big mistake to make block size a fixed limit in the code. Visa and Mastercard can process thousands, if not hundreds of thousands, of transactions per second. However, transactions come in a stochastic process, sometimes in massive bursts, sometimes being quiet for hours. Think of the volume of bitcoin exchange.

Seems like a grand idea to design a system that increases block size dynamically when necessary to accommodate increased transaction traffic, and decrease it dynamically when necessary to increase bandwidth efficiency.

Now, apply that notion to all system parameters. And as long as we're careful to keep the system from fishtailing out of control, this should work great.

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

As previously mentioned, if variables self-adjust, some controls must be imposed in order to keep the system from proceeding wildly out of control. We'll get to that.

If this were a wikipedia article, it'd be labeled "STUB." Although we are certainly in the section introducing the "Problems of Bitcoin," I would like some elaboration here. Why is 164 bytes unacceptable for a simple "check for secret key" task? How small can they get for a reasonable scripting language? I'm not a computer scientist, though.

http://download.springer.com/static/pdf/412/chp%253A10.1007%252F3-540-46416-6_22.pdf?auth66=140

Group signatures, as described, require a group manager. The group manager is capable of revoking the anonymity of any signer. Hence, there is in-built centralization in a group signature scheme.

Comments on page 4

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 [34], was an autonomous scheme without Group Manager and anonymity revocation. Various modifications of this scheme appeared later: *linkable ring signature* [26, 27, 17] allowed to determine if two signatures were produced by the same group member, *traceable ring signature* [24, 23] limited excessive anonymity by providing possibility to trace the signer of two messages with respect to the same metainformation (or "tag" in terms of [24]).

A similar cryptographic construction is also known as a *ad-hoc group signature* [16, 38]. 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 [24]. 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.

4.2 Definitions

4.2.1 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. [18]. 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 $E(\mathbb{F}_q) \to E(\mathbb{F}_q)$.

4.2.2 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;

A ring signature works like this: Alex wants to leak a message to WikiLeaks about her employer. Every employee in her Company has a private/public key pair (Ri, Ui). She composes her signature with input set as her message, m, her private key, Ri, and EVERYBODY's public keys, (Ui;i=1...n). Anyone (without knowing any private keys) can verify easily that *some* pair (Rj, Uj) must have been used to construct the signature... someone who works for Alex's employer... but it's essentially a random guess to figure out which one it could be.

http://en.wikipedia.org/wiki/Ring_signature#Crypto-currencies

http://link.springer.com/chapter/10.1007/3-540-45682-1_32#page-1

http://link.springer.com/chapter/10.1007/11424826_65

http://link.springer.com/chapter/10.1007/978-3-540-27800-9_28

http://link.springer.com/chapter/10.1007%2F11774716_9

Notice that a linkable ring signature described here is kind of the opposite of "unlinkable" described above. Here, we intercept two messages, and we can determine whether the same party sent them, although we should still be unable to determine who that party is. The definition of "unlinkable" used to construct Cryptonote means we cannot determine whether the same party is receiving them.

Hence, what we really have here is FOUR things going on. A system can be linkable or non-linkable, depending on whether or not it's possible to determine whether the sender of two messages are the same (regardless of whether this requires revoking anonymity). And a system can be unlinkable or non-unlinkable, depending on whether or not it's possible to determine whether the receiver of two messages are the same (regardless of whether or not this requires revoking anonymity).

Please don't blame me for this terrible terminology. Graph theorists should probably be pleased. Some of you may be more comfortable with "receiver linkable" versus "sender linkable."

http://link.springer.com/chapter/10.1007/978-3-540-71677-8_13

When I read this, this seemed like a silly feature. Then I read that it may be a feature for electronic voting, and that seemed to make sense. Kinda cool, from that perspective. But I'm not totally sure about purposely implementing traceable ring signatures.

http://search.ieice.org/bin/summary.php?id=e95-a_1_151

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 [34], was an autonomous scheme without Group Manager and anonymity revocation. Various modifications of this scheme appeared later: *linkable ring signature* [26, 27, 17] allowed to determine if two signatures were produced by the same group member, *traceable ring signature* [24, 23] limited excessive anonymity by providing possibility to trace the signer of two messages with respect to the same metainformation (or "tag" in terms of [24]).

A similar cryptographic construction is also known as a *ad-hoc group signature* [16, 38]. 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 [24]. 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.

4.2 Definitions

4.2.1 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. [18]. 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 $E(\mathbb{F}_q) \to E(\mathbb{F}_q)$.

4.2.2 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;

Gosh, the author of this whitepaper sure could have worded this better! Let's say that an employee-owned company wants to take a vote on whether or not to acquire certain new assets, and Alex and Brenda are both employees. The Company provides each employee a message like "I vote yes on Proposition A!" which has the metainformation "issue" [PROP A] and asks them to sign it with a traceable ring signature if they support the proposition.

Using a traditional ring signature, a dishonest employee can sign the message multiple times, presumably with different nonces, in order to vote as many times as they like. On the other hand, in a traceable ring signature scheme, Alex will go to vote, and her private key will have been used on the issue [PROP A]. If Alex tries to sign a message like "I, Brenda, approve of proposition A!" to "frame" Brenda and double vote, this new message will also have the issue [PROP A]. Since Alex's private key has already tripped the [PROP A] issue, Alex's identity will be immediately revealed as a fraud.

Which, face it, is pretty cool! Cryptography enforced voting equality.

http://link.springer.com/chapter/10.1007/978-3-540-71677-8_13

This paper is interesting, essentially creating an ad-hoc ring signature but without any of the other participant's consent. The structure of the signature may be different; I haven't dug deep, and I haven't seen whether it's secure.

https://people.csail.mit.edu/rivest/AdidaHohenbergerRivest-AdHocGroupSignaturesFromHijackedKeypai

Ad-hoc group signatures are: ring signatures, which are group signatures with no group managers, no centralization, but allows a member in an ad-hoc group to provably claim that it has (not) issued the anonymous signature on behalf of the group.

http://link.springer.com/chapter/10.1007/11908739_9

This isn't quite correct, from my understanding. And my understanding will likely change as I get deeper into this project. But from my understanding, the hierarchy looks like this.

Group sigs: group managers control traceability and the ability of adding or removing members from being signers.

Ring sigs: Arbitrary group formation without a group manager. No anonymity revocation. No way to repudiate oneself from a particular signature. With traceable and linkable ring signatures, anonymity is somewhat scaleable.

Ad-hoc group signatures: like ring signatures, but members can prove that they did not create a particular signature. This is important when anyone in a group can produce a signature.

http://link.springer.com/chapter/10.1007/978-3-540-71677-8_13

Fujisaki and Suzuki's algorithm is tweaked later by the author to provide one-time-ness. So we will analyze Fujisaki and Suzuki's algorithm concurrently with the new algorithm rather than going over it here.

Comments on page 5

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 [34], was an autonomous scheme without Group Manager and anonymity revocation. Various modifications of this scheme appeared later: *linkable ring signature* [26, 27, 17] allowed to determine if two signatures were produced by the same group member, *traceable ring signature* [24, 23] limited excessive anonymity by providing possibility to trace the signer of two messages with respect to the same metainformation (or "tag" in terms of [24]).

A similar cryptographic construction is also known as a *ad-hoc group signature* [16, 38]. 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 [24]. 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.

4.2 Definitions

4.2.1 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. [18]. 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 $E(\mathbb{F}_q) \to E(\mathbb{F}_q)$.

4.2.2 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;

Linkability in the sense of "linkable ring signatures" means we can tell if two outgoing transactions came from the same source without revealing who the source is. The authors weakened linkability so as to (a) preserve privacy, but still (b) spot any transaction using a private key a second time as invalid.

Okay, so this is an order-of-events question. Consider the following scenario. My mining computer will have the current blockchain, it will have it's own block of transactions it calls legitimate, it will be working on that block in a proof-of-work puzzle, and it will have a list of pending transactions to be added to the next block. It will also be sending any new transactions into that pool of pending transactions. If I do not solve the next block, but someone else does, I get an updated copy of the blockchain. The block I was working on and my list of pending transactions both may have some transactions that are now incorporated into the blockchain.

Unravel my pending block, combine that with my list of pending transactions, and call that my pool of pending transactions. Remove any that are now officially in the blockchain.

Now, what do I do? Should I first go through and "remove all double-spends"? On the other hand, should I search through the list and make sure that each private key has not yet been used, and if it has been used already in my list, then I received the first copy first, and hence any further copy is illegitimate. Thus I proceed to simply delete all instances after the first of the the same private key.

Algebraic geometry has never been my strong suit.

http://en.wikipedia.org/wiki/EdDSA

Such speed, much wow. THIS is algebraic geometry for the win. Not that I'd know anything about that.

Problematically, or not, discrete logs are getting very fast. And quantum computers eat them for breakfast.

http://link.springer.com/article/10.1007/s13389-012-0027-1

This becomes a really important number, but there is no explanation or citation to how it was chosen. Simply choosing a single known large prime would be fine, but if there are known facts about this large prime, that could influence our choice. Different variants of cryptonote could choose different values of ell, but there is no discussion in this paper about how that choice will affect our choices of other global parameters listed on page 5.

This paper needs a section on choosing parameter values.

Comments on page 5

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.

4.3 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.

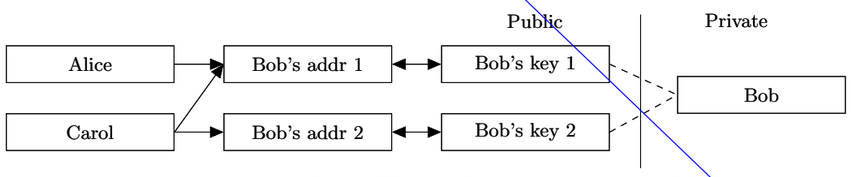

Fig. 2. Traditional Bitcoin keys/transactions model.

We propose a solution allowing a user to publish **a single address** and receive unconditional unlinkable payments. The destination of each CryptoNote 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.

Comments on page 6

So this is like Bitcoin, but with infinite, anonymous PO Boxes, redeemable only by the receiver generating a private key that is as anonymous as a ring signature can be.

Bitcoin works this way.

If Alex has 0.112 Bitcoin in her wallet she just received from Frank, she really has a signed message "I, [FRANK], send 0.112 Bitcoin to [alex] + H0 + N0" where 1) Frank has signed the message with his private key [FRANK], 2) Frank has signed the message with Alex's public key, [alex], 3) Frank has included some form of the history of the bitcoin, H0, and 4) Frank includes a random bit of data called the nonce, N0.

If Alex then wants to send 0.011 Bitcoin to Charlene, she will take Frank's message, and she will set that to H1, and sign two messages: one for her transaction, and one for the change.

H1= "I, [FRANK], send 0.112 Bitcoin to [alex] + H0 + N" "I, [ALEX], send 0.011 Bitcoin to [charlene] + H1 + N1" "I, [ALEX], send 0.101 Bitcoin as change to [alex] + H1 + N2."

where Alex signs both messages with her private key [ALEX], the first message with Charlene's public key [charlene], the second message with Alex's public key [alex], and including the histories and some randomly generated nonces N1 and N2 appropriately.

Cryptonote works this way:

If Alex has 0.112 Cryptonote in her wallet she just received from Frank, she really has a signed message "I, [someone in an ad-hoc group], send 0.112 Cryptonote to [a one-time address] + H0 + N0." Alex discovered that this was her money by checking her private key [ALEX] against [a one-time address] for every passing message, and if she wishes to spend it she does so in the following way.

She chooses a recipient of the money, perhaps Charlene has started voting for drone-strikes so Alex wants to send money to Brenda instead. So Alex looks up Brenda's public key, [brenda], and uses her own private key, [ALEX], to generate a one-time address [ALEX+brenda]. She then chooses an arbitrary collection C from the network of cryptonote users and she constructs a ring signature from this ad-hoc group. We set our history as the previous message, add nonces, and proceed as usual?

H1 = "I, [someone in an ad-hoc group], send 0.112 Cryptonote to [a one-time address] + H0 + N0."

"I, [someone in the collection C], send 0.011 Cryptonote to [one-time-address-made-from-ALEX+brenda] + H1 + N1"

"I, [someone in the collection C], send 0.101 Cryptonote as change to [one-time-address-made-from-ALEX+alex] + H1 + N2"

Now, Alex and Brenda both scan all incoming messages for any one-time-addresses that were created using their key. If they find any, then that message is their very own brand new cryptonote!

And even then, the transaction will still hit the blockchain. If the coins entering that address are known to be sent from criminals, political contributors, or from committees and accounts with strict budgets (i.e. embezzling), or if the new owner of these coins ever makes a mistake and sends these coins to a common address with coins he is known to own, the anonymity jig is up in bitcoin.

Comments on page 6

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.

4.3 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.

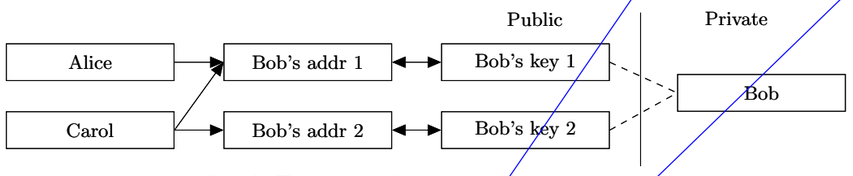

Fig. 2. Traditional Bitcoin keys/transactions model.

We propose a solution allowing a user to publish **a single address** and receive unconditional unlinkable payments. The destination of each CryptoNote 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.

Hence, rather than users sending coins from address (which is really a public key) to address (another public key) using their private keys, users are sending coins from one-time PO-box (which is generating using your friends public key) to one-time PO-box (similarly) using your own private keys.

In a sense, we're saying "Okay, everyone take your hands off the money while it's being transferred around! It's simply enough to know that our keys *can* open that box and that we know how much money is in the box. Never put your fingerprints on the PO Box or actually use it, just trade the box filled with cash itself. That way we don't know who sent what, but the contents of these public addresses are still frictionless, fungible, divisible, and still possess all the other nice qualities of money we desire like bitcoin."

An infinite set of PO boxes.

You publish an address, I have a private key. I use my private key and your address, and some random data, to generate a public key. The algorithm is designed such that, since your address was used to generate the public key, only YOUR private key works to unlock the message.

An observer, Eve, sees you publish your address, and sees the public key I announce. However, she doesn't know if I announced my public key based on your address or hers, or Brenda's or Charlene's, or whoever's. She checks her private key against the public key I announced and sees it doesn't work; it isn't her money. She doens't know anyone else's private key, and only the recipient of the message has the private key that can unlock the message. So no one listening in can determine who received the money much less take the money.

Comments on page 6

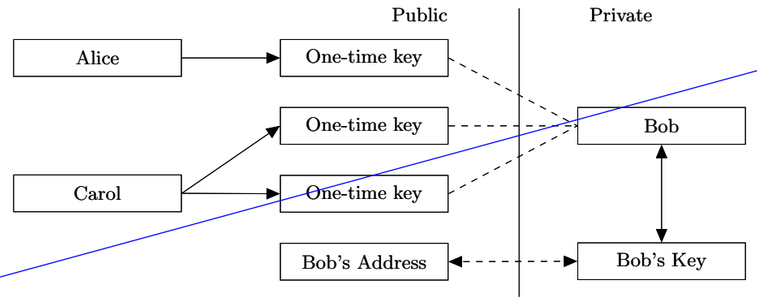

Fig. 3. CryptoNote keys/transactions model.

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 CryptoNote 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.

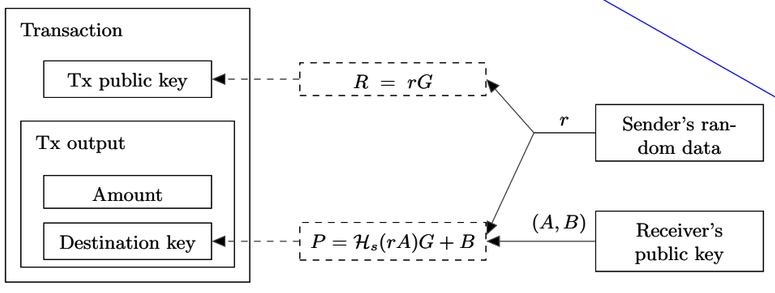

Fig. 4. Standard transaction structure.

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 $P' = P$.

I wonder how much of a pain in the neck it would be to implement a *choice* of cryptography scheme. Elliptic or otherwise. So if some scheme is broken in the future, the currency switches without concern. Probably a big pain in the ass.

Okay, this is exactly what I just explained in my previous comment. The Diffie-Hellman-type exchanges are neato. Say Alex and Brenda each have a secret number, A and B, and a number they don't care about keeping secret, a and b. They wish to generate a shared secret without Eva discovering it. Diffie and Hellman came up with a way for Alex and Brenda to share the public numbers a and b, but not the private numbers A and B, and generate a shared secret, K. Using this shared secret, K, without any Eva listening in being able to generate the same K, Alex and Brenda can now use K as a secret encryption key and pass secret messages back and forth.

Here's how it *CAN* work, although it should work with much larger numbers than 100. We'll use 100 because working over the integers modulo 100 is equivalent to "throwing out all but the last two digit of a number."

Alex and Brenda each choose A, a, B, and b. They keep A and B secret.

Alex tells Brenda her value of a modulo 100 (just the last two digits) and Brenda tells Alex her value of b modulo 100. Now Eva knows (a,b) modulo 100. But Alex knows (a,b,A) so she can compute x=a*b*A modulo 100. Alex chops off all but the last digit because we're working under the integers modulo 100 again. Similarly, Brenda knows (a,b,B) so she can compute y=a*b*B modulo 100. Alex can now publish x and Brenda can publish y.

But now Alex can compute y*A = a*b*B*A modulo 100, and Brenda can compute x*B = a*b*B*A modulo 100. They both know the same number! But all Eva has heard is (a,b,a*b*A,a*b*B). She has no easy way of computing a*b*A*B.

Now, this is the easiest and least secure way of thinking about the Diffie-Hellman exchange. More secure versions exist. But most versions work because integer factorization and discrete logarithms are difficult, and both of those problems are easily solved by quantum computers. I will look into whether any versions that are resistant to quantum exist.

http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

The "standard txn sequence" listed here is missing a whole bunch of steps, like SIGNATURES. They're just taken for granted in here. Which is really bad, because the order in which we sign stuff, the information included in the signed message, and so on... all of this is extremely important to the protocol.

Getting one or two of the steps wrong, even slightly out of order, while implementing "the standard transaction sequence" could throw the security of the whole system into question. Furthermore, the proofs presented later in the paper may not be sufficiently rigorous if the framework under which they work is as loosely defined as in this section.

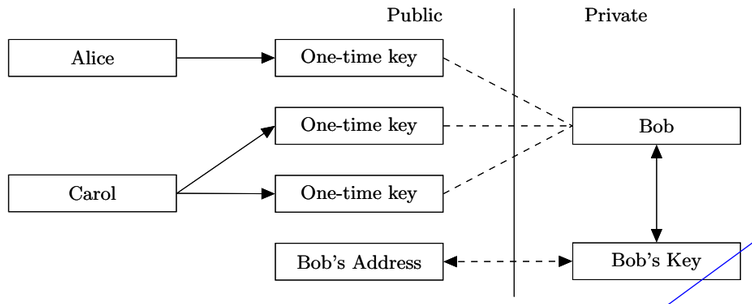

Fig. 3. CryptoNote keys/transactions model.

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 CryptoNote 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.

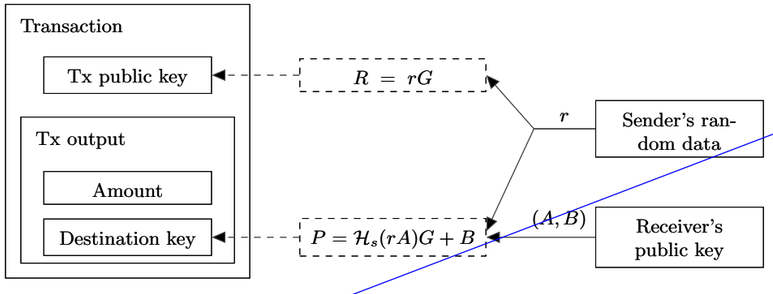

Fig. 4. Standard transaction structure.

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 $P' = P$.

Note that the author(s?) do a terrible job of keeping their terminology straight throughout the text, but especially in this next bit. Next incarnation of this paper will necessarily be much more rigorous.

In the text they refer to P as their one-time public key. In the diagram, they refer to R as their "Tx public key" and P as their "Destination key." If I were going to re-write this, I'd very specifically lay out some terminology before discussing these sections.

This ell is massive. See page 5.

Who chooses ell?

Diagram illustrates that the transaction public key R = rG, which is random and chosen by the sender, is not part of Tx output. This is because it could be the same for multiple transactions to multiple people, and isn't used *LATER* to spend. A fresh R is generated every time you want to broadcast a new CryptoNote transaction. Furthermore, R is only used to check if you are the recipient of the transaction. It's not junk data, but it's junk to anyone without the private keys associated with (A,B).

The Destination key, on the other hand, P = Hs(rA)G + B is part of Tx output. Everyone rifling through every passing transaction's data must check their own generated P* against this P to see if they own this passing transaction. Anyone with an unspent transaction output (UTXO) will have a bunch of these Ps laying around with amounts. In order to spend, they sign some new message including P.

Alice must sign this transaction with one-time private key(s) associated with the unspent transaction output(s) Destination Key(s). Each destination key owned by Alice comes equipped with a one-time private key also owned (presumably) by Alice. Every time Alice wants to send the contents of a destination key to me, or Bob, or Brenda, or Charlie or Charlene, she uses her private key to sign the transaction. Upon receipt of transaction, I will receive a new Tx public key, a new Destination public key, and I will be able to recover a new one-time private key x. Combining my one-time private key, x, with new transaction's public Destination key(s) is how we send a new transaction

Comments on page 7

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.

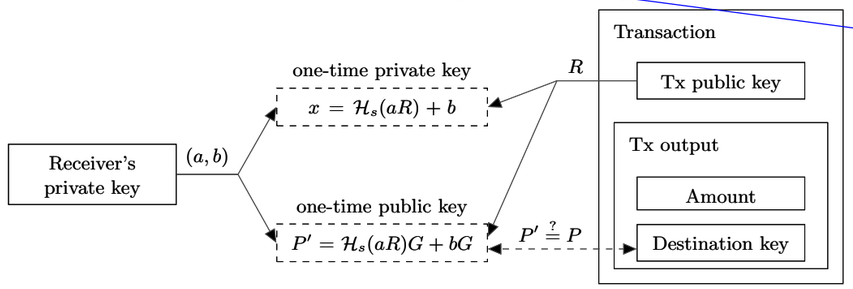

Fig. 5. Incoming transaction check.

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: $a = \mathcal{H}_s(B)$ and $A = \mathcal{H}_s(B)G$. 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.

4.4 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 $\mathcal{S} = S' \cup \{P_s\}$.

What does an unspent transaction output look like here? The diagram suggests that transaction output consists only of two data points: amount and destination key. But this isn't sufficient because when I try to spend this "output" I will still need to know R=r*G. Remember, r is chosen by the sender, and R is a) used to recognize incoming cryptonotes as your own and b) used to generate the one-time private key used to "claim" your cryptonote.

The part about this that I don't understand? Taking the theoretical "okay, we have these signatures and transactions, and we pass 'em back and forth" into the world of programming "okay what information *specifically* makes up an individual UTXO?"

Best way to answer that question is to dig into the body of completely uncommented code. Way to go, bytecoin team.

Recall: linkability means "did the same person send?" and unlinkability means "did the same person receive?". So a system can be linkable or non-linkable, unlinkable or non-unlinkable. Annoying, I know.

So when Nic van Saberhagen here says "...incoming payments [are] associated with one-time public keys which are unlinkable for a spectator," let's see what he means.

First, consider a situation in which Alice sends Bob two separate transactions from the same address to the same address. In the Bitcoin universe, Alice has already made the mistake of sending from the same address and so the transaction has failed our desire for limited linkability. Furthermore, since she sent the money to the same address, she's failed our desire for unlinkability. This bitcoin transaction was both (totally) linkable* and non-unlinkable.

On the other hand, in the cryptonote universe, let's say that Alice sends Bob some cryptonote, using Bob's public address. She chooses as her obfuscating set of public keys all known public keys in the Washington DC metro area. Alex generates a one-time public key using her own information and Bob's public information. She sends the money off, and any observer will only be able to glean "Someone from the Washington DC metro area sent 2.3 cryptonotes to the one-time public address XYZ123."

We have a probabilistic control over linkability here, so we'll call this "almost non-linkable." We also only see the one-time public keys money is sent to. Even if we suspected the receiver was Bob, we don't have his private keys and so we can't test whether a passing transaction belongs to Bob let alone generate his one-time private key to redeem his cryptonote. So this is, in fact, totally "unlinkable."

So, this is the neatest trick of all. Who wants to really trust another MtGox? We may be comfortable storing some amount of BTC on Coinbase, but the ultimate in bitcoin security is a physical wallet. Which is inconvenient.

In this case, you can trustlessly give away half of your private key without compromising your own ability to spend money.

When doing this, all you are doing is telling someone how to break unlinkability. The other properties of CN acting like a currency are preserved, like proof against double spending and whatnot.

Comments on page 8

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.

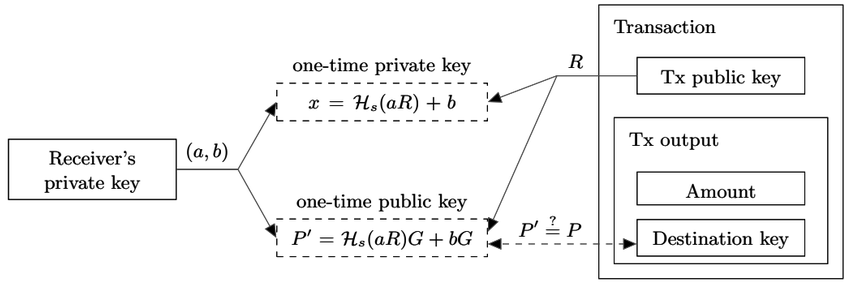

Fig. 5. Incoming transaction check.

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: $a = \mathcal{H}_s(B)$ and $A = \mathcal{H}_s(B)G$. 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.

4.4 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 $\mathcal{S} = S' \cup \{P_s\}$.

Yes, so now we have a) a payment address and b) a payment ID.

A critic could ask "do we really need to do this? After all, if a merchant receives 112.00678952 CN exactly, and that was my order, and I have a screenshot or a receipt or whatever, isn't that insane degree of precision sufficient?" The answer is "maybe, most of the time, in day-to-day, face-to-face transactions."

However, the more common situation (especially in the digial world) is this: a merchant sells a set of objects, each with a fixed price. Say object A is 0.001 CN, object B is 0.01 CN and object C is 0.1 CN. Now, if the merchant receives an order for 1.618 CN, there are many many (many!) ways to arrange an order for a customer. And so without some sort of payment ID, identifying the so-called "unique" order of a customer with the so-called "unique" cost of their order becomes impossible. Even funnier: if everything in my online store costs exactly 1.0 CN, and I get 1000 customers a day? And you want to prove that you bought exactly 3 objects two weeks ago? Without a payment ID? Good luck, buddy.

Long story short: When Bob publishes a payment address, he may end up also publishing a payment ID as well (see, e.g. Poloniex XMR deposits). This is different than what is described in the text here where Alice is the one who generates the payment ID.

There must be some way for Bob to generate a payment ID as well.

(a,B)

Recall that the tracking key (a,B) can be published; losing the secrecy of the value for 'a' will not violate your ability to spend or allow folks to steal from you (I think... that would have to be proven), it will simply allow folks to see all incoming transactions.

A truncated address, as described in this paragraph, simply takes the "private" part of the key and generates it from the "public" part. Revealing the value for 'a' will remove non-linkability but will preserve the rest of the transactions.

The author means "not unlinkable" because unlinkable refers to the receiver and linkable refers to the sender.

It's also clear the author didn't realize that there were two different aspects to linkability. Since, after all, the transaction is a directed object on a graph, there will be two questions: "are these two transactions going to the same person?" and "are these two transactions coming from the same person?"

This is a "no-going-back" policy under which the unlinkability property of CryptoNote is conditional. That is to say, Bob can choose his incoming transactions to be not unlinkable using this policy.

This is a claim they prove under the Random Oracle Model. We'll get to that; the Random Oracle has pros and cons.

Comments on page 8

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.

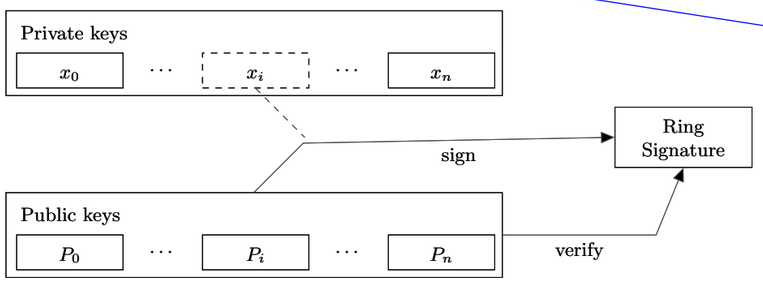

Fig. 6. Ring signature anonymity.

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 $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$.

Perhaps this is stupid, but care must be taken when unioning S and P_s. If you just append the last public key to the end, unlinkability is broken because anyone checking passing transactions can just check the last public key listed in each transaction and boom. That's the public key associated with the sender. So after unioning, a pseudorandom number generator must be used to permute the public keys chosen.

"...until the owner produces a second signature using the same keypair." I wish the author(s?) would elaborate on this.

I believe this means "make sure that every time you choose a set of public keys to obfuscate yourself with, you pick a completely new set with no two keys alike." Which seems like a pretty strong condition to place upon unlinkability. Perhaps "you pick a new random set from all possible keys" with the assumption that, although nontrivial intersections will inevitably happen, they won't happen often.

Either way, I need to dig deeper into this statement.

This is generating the ring signature.

Zero-knowledge proofs are awesome: I challenge you to prove to me that you know a secret without revealing the secret. For example, say we are at the entrance of a donut-shaped cave, and at the back of the cave (beyond sight from the entrance) is a one-way door to which you claim you have the key. If you go one direction, it always lets you through, but if you go the other direction, you need a key. But you don't even want to SHOW me the key, let alone show me that it opens the door. But you want to prove to me that you know how to open the door.

In the interactive setting, I flip a coin. Heads is left, tails is right, and you go down the donut-shaped cave whichever way the coin directs you. At the back, beyond my sight, you open the door to come back around the other side. We repeat the coin-flipping experiment until I'm satisfied that you have the key.

But that's clearly the INTERACTIVE zero-knowledge proof. There are non-interactive versions in which you and I never have to communicate; this way, no eavesdroppers can interfere.

http://en.wikipedia.org/wiki/Zero-knowledge_proof

This is reversed from the previous definition.

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.

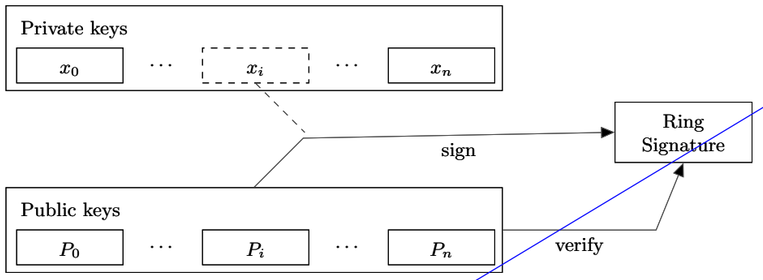

Fig. 6. Ring signature anonymity.

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 $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$.

This whole area is cryptonote agnostic, simply describing the ring signature algorithm without reference to currencies. I suspect some of the notation is consistent with the rest of the paper, though. For example, x is the "random" secret key chosen in GEN, which gives public key P and public key image I. This value of x is the value Bob computes in part 6 page 8. So this is starting to clear up some of the confusion from the previous description.

This is kind of cool; money isn't being transferred from "Alice's public address to Bob's public address." It's being transferred from one-time address to one-time address.

So, in a sense, here's how stuff is working. If Alex has some cryptonotes because someone sent them to her, this means she has the private keys needed to send them to Bob. She uses a Diffie-Hellman exchange using Bob's public information to generate a new one-time address and the cryptonotes are transferred to that address.

Now, since a (presumably secure) DH exchange was used to generate the new one-time address to which Alex sent her CN, Bob is the only one with the private keys needed to repeat the above. So now, Bob is Alex.

http://en.wikipedia.org/wiki/Piecewise#Notation_and_interpretation

Summation should be indexed over j not i. Each c_i is random junk (since w_i is random) except for the c_i associated with the actual key involved in this signature. The value of c is a hash of the previous information.

I think this may contain a typo worse than re-using the index 'i,' though, because c_s seems to be implicitly, not explicitly, defined.

Indeed, if we take this equation on faith, then we determine that c_s = (1/2)*c - (1/2)* sum_i neq s c_i. That is, a hash minus a whole bunch of random numbers.

On the other hand, if this summation is intended to be read "c_s = (c - sum_j neq s c_j) mod l", then we take the hash of our previous information, generate a bunch of random numbers, subtract all of those random numbers off the hash, and that gives us c_s. This seems to be what "should" be happening given my intuition, and matches the verification step on page 10.

But intuition is not mathematics. I'll dig deeper into this.

Same as before; all of these will be random junk except for the one associated with the actual signer's public key x. Except this time, this is more what I would expect from the structure: r_i is random for i!=s and r_s is determined only by the secret x and the s-indexed values of q_i and c_i.

Comments on page 9

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.

A full security analysis is provided in Appendix A.

4.5 Standard CryptoNote 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 (p_i, P_i) 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.

At this point, I'm terribly confused.

Alex receives a message M with signature (I,c__1, ..., c__n, r__1, ..., r__n) and list of public keys S. and she runs VER. This will compute L__i' and R__i'

This verifies that c__s = c - sum__i neq s c__i on the previous page.

At first I was VERy (ha) confused. Anyone can compute L__i' and R__i'. Indeed, each r__i and c__i have been published in the signature sigma together with the value for I. The set S = P__i of all public keys has also been published. So anyone who has seen sigma and the set of keys S = P__i will get the same values for L__i' and R__i' and hence check the signature.

But then I remembered this section is simply describing a signature algorithm, not a "check if signed, check if SENT TO ME, and if so, then go spend the money." This is SIMPLY the signature part of the game.

I'm interested to read Appendix A when I finally get there.

I would like to see a full-scale operation-by-operation comparison of Cryptonote to Bitcoin.

Also, electricity/sustainability.

What pieces of the algorithms constitute "input" here?

Transaction input, I believe, is an Amount and a set of UTXOs that sum to a greater amount than the Amount.

This is unclear.

"Target of hiding?" I have thought about this for a few minutes now and I still haven't the foggiest idea of what it could mean.

A double-spend attack can be executed only by manipulating a node's perceived used-key images set I.

"Ambiguity degree" = n but the total number of public keys included in the transaction is n+1. That is to say, ambiguity degree would be "how many OTHER people do you want in the crowd?"

The answer will probably be, by default "as many as possible."

Comments on page 10

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.

A full security analysis is provided in Appendix A.

4.5 Standard CryptoNote 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 (p_i, P_i) 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.

This is interesting; earlier, we provided a way for a receiver, Bob, to make all INCOMING transactions non-unlinkable either by choosing half of his private keys deterministically or by publishing half his private keys as public. This is a no-going-back sort of policy. Here, we see a way of a sender, Alex, to choose a single outgoing transaction as linkable, but in fact this reveals Alex as the sender to the whole network. This is NOT a no-going-back sort of policy. This is transaction-by-transaction.

Is there a third policy? Can a receiver, Bob, generate a unique payment ID for Alex that never changes, perhaps using a Diffie-Hellman exchange? If anyone includes that payment ID bundled somewhere in her transaction to Bob's address, it must have come from Alex. This way, Alex need not reveal herself to the whole network by choosing to link a particular transaction, but she can still identify herself to the person to whom she sends her money. Isn't this what Poloniex does?

Comments on page 10

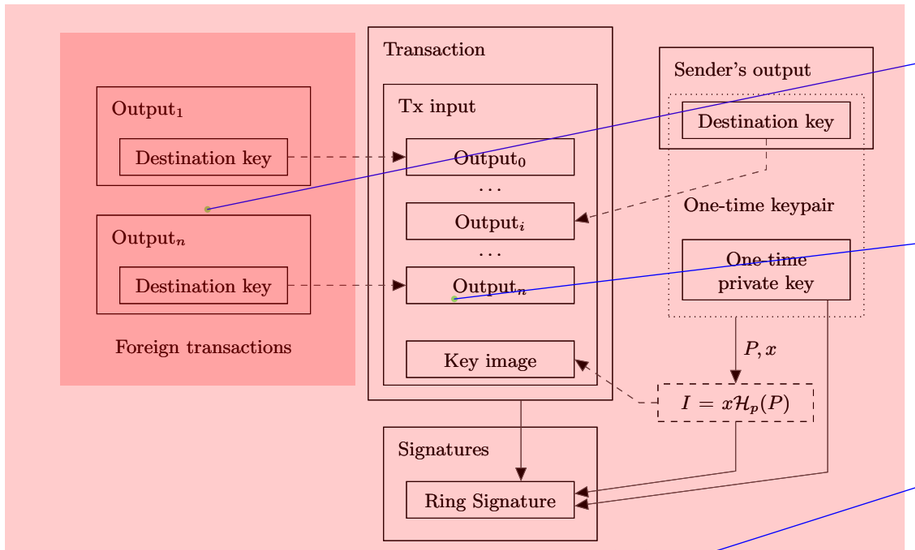

Fig. 7. Ring signature generation in a standard transaction.

These are, ostensibly, our UTXO's: amounts and destination keys. If Alex is the one constructing this standard transaction and is sending to Bob, then Alex also has the private keys to each of these.

I like this diagram a whole lot, because it answers some earlier questions. A Txn input consists of a set of Txn outputs and a key image. It is then signed with a ring signature, including all of the private keys Alex owns to all of the foreign transactions wrapped up in the deal. The Txn output consists of an amount and a destination key. The receiver of the transaction may, at will, generate their one-time private key as described earlier in the paper in order to spend the money.

It will be delightful to find out how much this matches the actual code...

5 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.

No, Nic van Saberhagen describes loosely some properties of a proof of work algorithm, without actually describing that algorithm.

The CryptoNight algorithm itself will REQUIRE a deep analysis.

When I read this, I stuttered. Should investment grow at least linearly with power, or should investment grow *at most* linearly with power?

And then I realized; I, as a miner, or an investor, usually think of "how much power can I get for an investment?" not "how much investment is required for a fixed amount of power?"

Of course, denote investment by I and power by P. If I(P) is investment as a function of power and P(I) is power as a function of investment, they'll be inverses of each other (wherever inverses can exist). And if I(P) is faster-than-linear than P(I) is slower-than-linear. Hence, there will be a reduced rate of returns for investors.

That is to say, what the author is saying here is: "sure, as you invest more, you'll get more power. But we should try to make that a reduced rate of returns thing."

CPU investments will cap out sub-linearly, eventually; the question is whether the authors have designed a POW algorithm that will force the ASICs to also do this.

5.1 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 [8], 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" [15]. 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 [22] investigated and formalized this approach leading them to suggest another variant of the pricing function: "Mbound". One more work belongs to F. Coelho [20], who

Should a hypothetical "future currency" always mine with the slowest/most limited resources?

The paper by Abadi et al, (which has some Google and Microsoft engineers as authors) is, essentially, using the fact that *for the past few years* memory size has had a much smaller variance across machines than processor speed, and with a more-than-linear investment-for-power ratio.

In a few years, this may have to be re-assessed! Everything is an arms race...

Constructing a hash function is difficult; constructing a hash function satisfying these constraints seems to be more difficult. This paper seems to have no explanation of the actual hashing algorithm CryptoNight. I think it's a memory-hard implementation of SHA-3, based on forum posts but I have no idea... and that's the point. It must be explained.

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 [32]. 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 [14] 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. The pseudo-random indexes are assumed to be uniformly distributed, hence the expected value of the *additional* blocks' recalculations is $\frac{1}{2} \cdot N$, where N is the number of iterations. The overall computation time increases less than by half because there are also time independent (constant time) operations such as preparing the scratchpad and hashing on every iteration. Saving 2/3 of the memory costs $\frac{1}{3} \cdot N + \frac{1}{3} \cdot 2 \cdot N = N$ additional recalculations; 9/10 results in $\frac{1}{10} \cdot N + \ldots + \frac{1}{10} \cdot 9 \cdot N = 4.5N$. It is easy to show that storing only $\frac{1}{s}$ of all blocks increases the time less than by a factor of $\frac{s-1}{2}$. This in turn implies that a machine with a CPU 200 times faster than the modern chips can store only 320 bytes of the scratchpad.

5.2 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).

Nevermind, it’s a scrypt coin?

Where is the algorithm? All I see is an advertisement.

This is where Cryptonote, if their PoW algorithm is worthwhile, will really shine. It’s not really SHA-256, it’s not really scrypt. It’s new, memory bound, and non-recursive.

Comments on page 12

6 Further advantages

6.1 Smooth emission

The upper bound for the overall amount of CryptoNote digital coins is: $\texttt{MSupply} = 2^{64} - 1$ 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:

$$BaseReward = (\texttt{MSupply} - A) \gg 18,$$

where A is amount of previously generated coins.

6.2 Adjustable parameters

6.2.1 Difficulty

CryptoNote 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.

6.2.2 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 M_N be the median value of the last N blocks sizes. Then the "hard-limit" for the size of accepting blocks is $2 \cdot M_N$. 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.

6.2.3 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 $2 \cdot M_b$. Even though only the majority of miners can shift the median value, there is still a

Atomic units. I like that. Is this the equivalent of Satoshis?

If so, then that means there are going to be 185 billion cryptonote.

I know this must be, eventually, tweaked in a few pages, or maybe there's a typo?

If the base reward is "all remaining coins" then only one block is sufficient to get all coins. Instamine. On the other hand, if this is supposed to be proportional in some way to the difference in time between now and some coin-production-termination-date? That would make sense.

Also, in my world, two greater than signs like this means "much greater than." Did the author possibly mean something else?

If adjustment to difficulty occurs every block then an attacker could have a very large farm of machines mine on and off in carefully chosen time intervals. This could cause a chaotic explosion (or crash to zero) in difficulty, if difficulty adjustment formulas aren't suitably damped.

No doubt that Bitcoin's method is unsuitable for rapid recalculations, but the idea of inertia in these systems would need to be proven, not taken for granted. Furthermore, oscillations in network difficulty isn't necessarily a problem unless it results in oscillations of ostensible supply of coins - and having a very rapidly changing difficulty could cause "over-correction."

Time spent, especially over a short time span like a few minutes, will be proportional to "total number of blocks created on the network." The constant of proportionality will, itself, grow over time, presumably exponentially if CN takes off.

It may be a better idea to simply adjust the difficulty to keep "total blocks created on the network since the last block was added to the main chain" within some constant value, or with bounded variation or something like that. If an adaptive algorithm that is computationally easy to implement can be determined, this would seem to solve the problem.

But then, if we used that method, someone with a big mining farm could shut their farm down for a few hours, and switch it back on again. For the first few blocks, that farm will make bank.

So, actually, this method would bring up an interesting point: mining becomes (on average) a losing game with no ROI, especially as more folks hop on the network. If the mining difficulty very closely tracked network hashrate, I somehow doubt people would mine as much as they currently do.

Or, on the other hand, instead of keeping their mining farms going 24/7, they may turn them on for 6 hours, off for 2, on for 6, off for 2, or something like that. Just switch to another coin for a few hours, wait for difficulty to drop, then hop back on in order to gain those few extra blocks of profitability as the network adapts. And you know what? This is actually probably one of the better mining scenarios I've put my mind into...

This could be circular, but if block creation time *averages* to about a minute, can we just use the number of blocks as a proxy for "time spent?"

Comments on page 13

6 Further advantages

6.1 Smooth emission

The upper bound for the overall amount of CryptoNote digital coins is: $\texttt{MSupply} = 2^{64} - 1$ 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:

$$BaseReward = (\texttt{MSupply} - A) \gg 18,$$

where A is amount of previously generated coins.

6.2 Adjustable parameters

6.2.1 Difficulty

CryptoNote 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.

6.2.2 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 M_N be the median value of the last N blocks sizes. Then the "hard-limit" for the size of accepting blocks is $2 \cdot M_N$. 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.

6.2.3 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 $2 \cdot M_b$. Even though only the majority of miners can shift the median value, there is still a

Okay, so we have a blockchain, and each block has timestamps IN ADDITION to simply being ordered. This was clearly inserted simply for difficulty adjustment, because timestamps are very unreliable, as mentioned. Are we allowed to have contradicting timestamps in the chain? If Block A comes before Block B in the chain, and everything is consistent in terms of finances, but Block A appears to have been created after Block B? Because, perhaps, someone owned a large part of the network? Is that ok?

Probably, because the finances aren't goofed up.

Okay, so I hate this arbitrary "only 80% of the blocks are legitimate for the main blockchain" approach. It was intended to prevent liars from tweaking their timestamps? But now, it adds incentive for everyone to lie about their timestamps and just pick the median.

Please define. Meaning "for this block, only include transactions that include fees greater than p%, preferentially with fees greater than 2p%" or something like that?

What do they mean by bogus? If the transaction is consistent with past history of the blockchain, and the transaction includes fees that satisfy miners, is that not enough? Well, no, not necessarily. If no maximum block size exists, there is nothing to keep a malicious user from simply uploading a massive block of transactions to himself all at once just to slow down the network.

A core rule for maximum block size prevents people from putting enormous amounts of junk data on the blockchain all at once just to slow things down. But such a rule certianly has to be adaptive - during the christmas season, for example, we could expect traffic to spike, and block size to get very big, and immediately afterward, for the block size to subsequently drop again. So we need either a) some sort of adaptive cap or b) a cap large enough so that 99% of reasonable christmas peaks don't break the cap. Of course, that second one is impossible to estimate - who knows if a currency will catch on? Better to make it adaptive and not worry about it. But then we have a control theory problem: how to make this adaptive without vulnerability to attack or wild & crazy oscillations?

Notice an adaptive method doesn't stop malicious users from accumulating small amounts of junk data over time on the blockchain to cause long-term bloat. This is a different issue altogether and one that cryptonote coins have serious problems with.

Comments on page 13

6 Further advantages

6.1 Smooth emission

The upper bound for the overall amount of CryptoNote digital coins is: $\texttt{MSupply} = 2^{64} - 1$ 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:

$$BaseReward = (\texttt{MSupply} - A) \gg 18,$$

where A is amount of previously generated coins.

6.2 Adjustable parameters

6.2.1 Difficulty

CryptoNote 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.

6.2.2 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 M_N be the median value of the last N blocks sizes. Then the "hard-limit" for the size of accepting blocks is $2 \cdot M_N$. 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.

6.2.3 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 $2 \cdot M_b$. Even though only the majority of miners can shift the median value, there is still a

Rescaling time so that one unit of time is N blocks, the average block size could still, theoretically, grow exponentially proportionally to 2^t. On the other hand, a more general cap on the next block would be M_n*f(M_n) for some function f. What properties of f would we choose in order to guarantee some "reasonable growth" of block size? The progression of block sizes (after rescaling time) would go like this:

M_n f(M_n)*M_n f(f(M_n)*M_n)*f(M_n)*M_n f(f(f(M_n)*M_n)*f(M_n)*M_n)*f(f(M_n)*M_n)*f(

...

And the goal here is to choose f such that this sequence grows no faster than, say, linearly, or perhaps even as Log(t). Of course, if f(M_n) = a for some constant a, this sequence is actually

M_n a*M_n a^2*M_n a^3*M_n ...

And, of course, the only way this can be limited to at-most linear growth is by choosing a=1. This is, of course, infeasible. It does not allow growth at all.

If, on the other hand, f(M_n) is a non-constant function, then the situation is much more complicated and may allow for an elegant solution. I'll think on this for awhile.

This fee will have to be large enough to discount the excess size penalty from the next section.

Why is a general user assumed male, huh? Huh?

Comments on page 13

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 $\texttt{max}(10\text{kb}, M_N \cdot 110\%)$. 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.

6.3 Transaction scripts

CryptoNote has a very minimalistic scripting subsystem. A sender specifies an expression $\Phi = f(x_1, x_2, \ldots, x_n)$, where n is the number of destination public keys $\{P_i\}_{i=1}^n$. Only five binary operators are supported: `min`, `max`, `sum`, `mul` and `cmp`. When the receiver spends this payment, he produces $0 \le k \le n$ signatures and passes them to transaction input. The verification process simply evaluates Φ with $x_i = 1$ to check for a valid signature for the public key P_i, and $x_i = 0$. A verifier accepts the proof iff $\Phi > 0$.

Despite its simplicity this approach covers every possible case:

- **Multi-/Threshold signature.** For the Bitcoin-style "M-out-of-N" multi-signature (i.e. the receiver should provide at least $0 \le M \le N$ valid signatures) $\Phi = x_1 + x_2 + \ldots + x_N \ge M$ (for clarity we are using common algebraic notation). The weighted threshold signature (some keys can be more important than other) could be expressed as $\Phi = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_N \cdot x_N \ge w_M$. And scenario where the master-key corresponds to $\Phi = \texttt{max}(M \cdot x, x_1 + x_2 + \ldots + x_N) \ge M$. It is easy to show that any sophisticated case can be expressed with these operators, i.e. they form basis.
- **Password protection.** Possession of a secret password s is equivalent to the knowledge of a private key, deterministically derived from the password: $k = \texttt{KDF}(s)$. Hence, a receiver can prove that he knows the password by providing another signature under the key k. The sender simply adds the corresponding public key to his own output. Note that this method is much more secure than the "transaction puzzle" used in Bitcoin [13], where the password is explicitly passed in the inputs.
- **Degenerate cases.** $\Phi = 1$ means that anybody can spend the money; $\Phi = 0$ marks the output as not spendable forever.

In the case when the output script combined with public keys is too large for a sender, he can use special output type, which indicates that the recipient will put this data in his input while the sender provides only a hash of it. This approach is similar to Bitcoin's "pay-to-hash" feature, but instead of adding new script commands we handle this case at the data structure level.

7 Conclusion

We have investigated the major flaws in Bitcoin and proposed some possible solutions. These advantageous features and our ongoing development make new electronic cash system CryptoNote a serious rival to Bitcoin, outclassing all its forks.

This may be unnecessary if we can figure out a way to bound block size over time...

This also cannot be correct. They just set "NewReward" to an upward-facing parabola where block size is the independent variable. So new reward blows up to infinity. If, on the other hand, the new reward is Max(0,Base Reward*(1-(BlkSize/Mn - 1)^2)), then the new reward would be a downward facing parabola with peak at blocksize = Mn, and with intercepts at Blocksize = 0 and Blocksize = 2*Mn. And that seems to be what they are trying to describe.

However, this does not

Comments on page 14

References

[1] http://bitcoin.org.

[2] https://en.bitcoin.it/wiki/Category:Mixing_Services.

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

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

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

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

[7] https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki#Backwards_Compatibility.

[8] https://en.bitcoin.it/wiki/Mining_hardware_comparison.

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

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

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

[12] https://en.bitcoin.it/wiki/Contracts.

[13] https://en.bitcoin.it/wiki/Script.

[14] http://litecoin.org.

[15] Martín Abadi, Michael Burrows, and Ted Wobber. Moderately hard, memory-bound functions. In *NDSS*, 2003.

[16] 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.

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

[18] 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.

[19] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.

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

[21] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

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

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

https://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

https://litecoin.org/

http://research.microsoft.com/pubs/54395/memory-longer-acm.pdf

https://people.csail.mit.edu/rivest/AdidaHohenbergerRivest-AdHocGroupSignaturesFromHijackedKeypai

http://link.springer.com/chapter/10.1007/11774716_9

http://link.springer.com/article/10.1007/s13389-012-0027-1

References

[1] http://bitcoin.org.

[2] https://en.bitcoin.it/wiki/Category:Mixing_Services.

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

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

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

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

[7] https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki#Backwards_Compatibility.

[8] https://en.bitcoin.it/wiki/Mining_hardware_comparison.

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

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

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

[12] https://en.bitcoin.it/wiki/Contracts.

[13] https://en.bitcoin.it/wiki/Script.

[14] http://litecoin.org.

[15] Martín Abadi, Michael Burrows, and Ted Wobber. Moderately hard, memory-bound functions. In *NDSS*, 2003.

[16] 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.

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

[18] 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.

[19] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.

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

[21] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

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

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

http://download.springer.com/static/pdf/412/chp%253A10.1007%252F3-540-46416-6_22.pdf?auth66=140

http://www.cri.ensmp.fr/classement/doc/A-370-v1.pdf

http://www.win.tue.nl/ berry/papers/CS-R9413.pdf

http://nozdr.ru/data/media/biblioteka/kolxo3/Cs_Computer_science/CsLn_Lecture%20notes/Advances%20CRYPTO%202003,%2023%20conf.(LNCS2729,%20Springer,%202003)(ISBN%203540406743)(643s).pd

http://labkom.stikom.edu/download/ebook/Topics%20in%20Crytology%20CT_RSA%202011/364219073

Comments on page 19

[24] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *Public Key Cryptography*, pages 181–200, 2007.

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

[26] 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.

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

[28] 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.

[29] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future internet*, 5(2):237–250, 2013.

[30] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *CRYPTO*, pages 324–337, 1991.

[31] Marc Santamaria Ortega. The bitcoin transaction graph — anonymity. Master's thesis, Universitat Oberta de Catalunya, June 2013.

[32] Colin Percival. Stronger key derivation via sequential memory-hard functions. Presented at BSDCan'09, May 2009.

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

[34] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565, 2001.

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

[36] Meni Rosenfeld. Analysis of hashrate-based double-spending. 2012.

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

[38] Qianhong Wu, Willy Susilo, Yi Mu, and Fangguo Zhang. Ad hoc group signatures. In *IWSEC*, pages 120–135, 2006.

http://f3.tiera.ru/2/Cs_Computer%20science/CsLn_Lecture%20notes/Public%20Key%20Cryptography%
%20PKC%202007,%2010%20conf.(LNCS4450,%20Springer,%202007)(ISBN%209783540716761)(501s).pdf

http://f3.tiera.ru/2/Cs_Computer%20science/CsLn_Lecture%20notes/Public%20Key%20Cryptography%
%20PKC%202007,%2010%20conf.(LNCS4450,%20Springer,%202007)(ISBN%209783540716761)(501s).pdf

https://gist.github.com/jgarzik/3901921

http://f3.tiera.ru/2/Cs_Computer%20science/CsLn_Lecture%20notes/Information%20Security%20and%

http://link.springer.com/chapter/10.1007/11424826_65#page-1

http://diyhpl.us/ bryan/papers2/bitcoin/Zerocoin:%20anonymous%20distributed%20e-cash%20from%20b

http://www.mdpi.com/1999-5903/5/2/237/pdf

http://pdf.aminer.org/000/120/358/universal_electronic_cash.pdf

Hard to find Masters theses..

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/23562/9/msantamariaoTFM0613memoria.pdf

And here's a presentation about it

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/23562/10/msantamariaoTFM0613presentation.p

http://prowen.ru/scrypt/scrypt.pdf

http://link.springer.com/chapter/10.1007/978-1-4614-4139-7_10#page-1

https://www.iacr.org/archive/asiacrypt2001/22480554.pdf

http://www.thebitcoin.fr/wp-content/uploads/2014/01/Quantitative-Analysis-of-the-Full-Bitcoin.pdf

http://www.bitcointrading.com/pdf/MeniDoublespend.pdf

http://arxiv.org/abs/0706.1448

http://link.springer.com/chapter/10.1007/11908739_9