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

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

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].

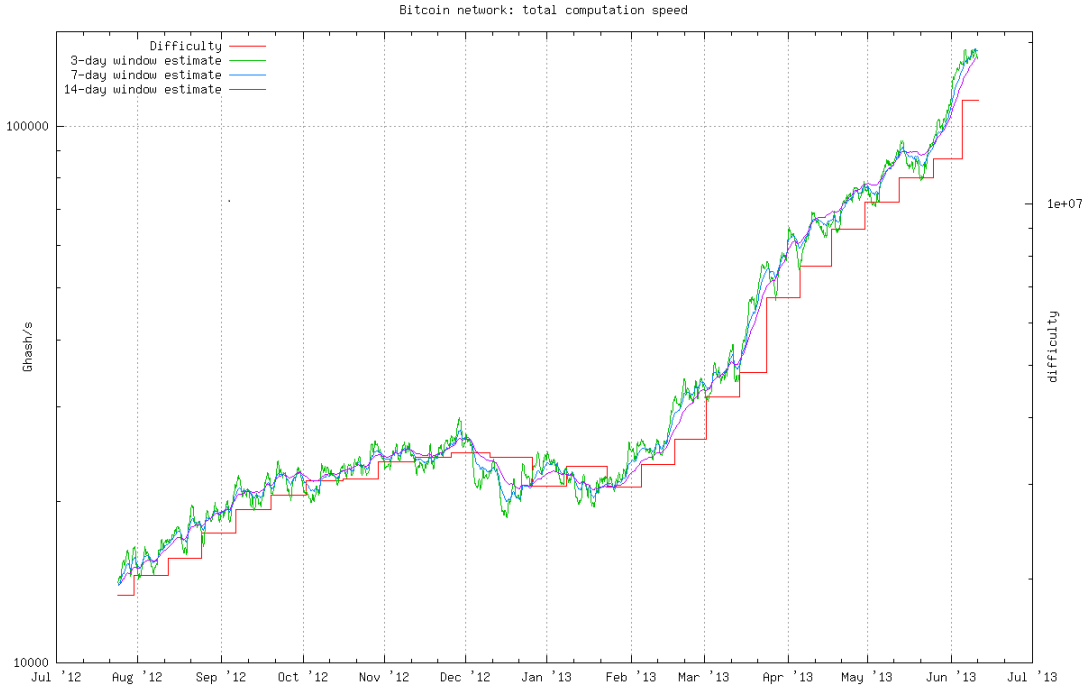

Fig. 1. Bitcoin hashrate chart
(source: http://bitcoin.sipa.be)

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

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

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;

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.

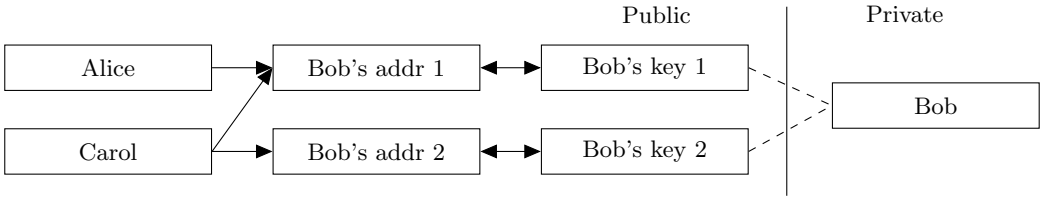

Fig. 2. Traditional Bitcoin keys/transactions model.

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.

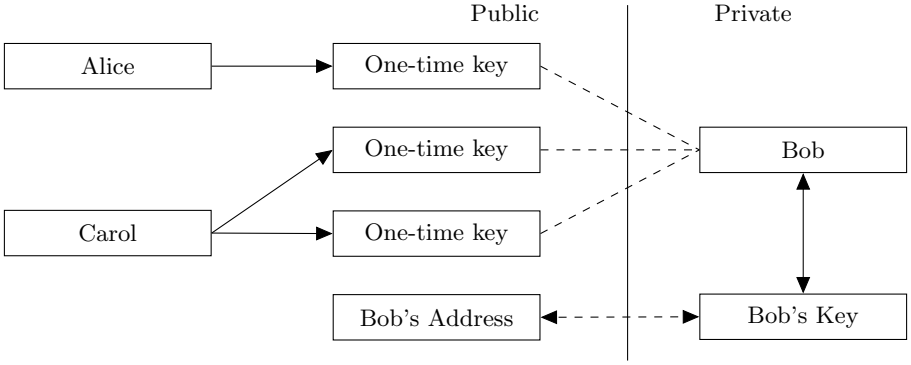

Fig. 3. CryptoNote keys/transactions model.

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.

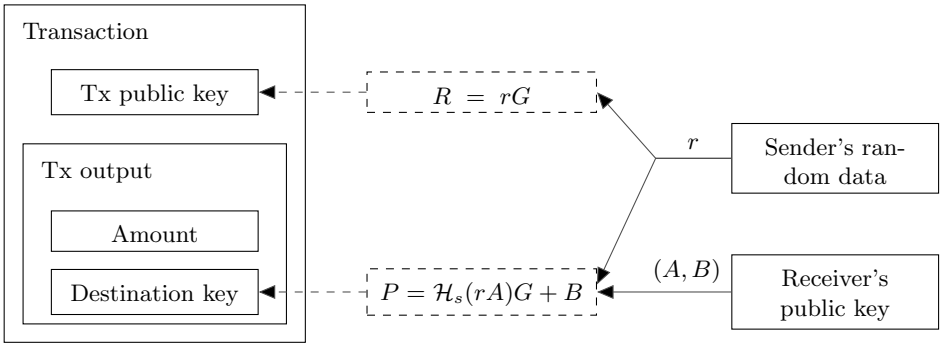

Fig. 4. Standard transaction structure.

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 $P' = P$.

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.

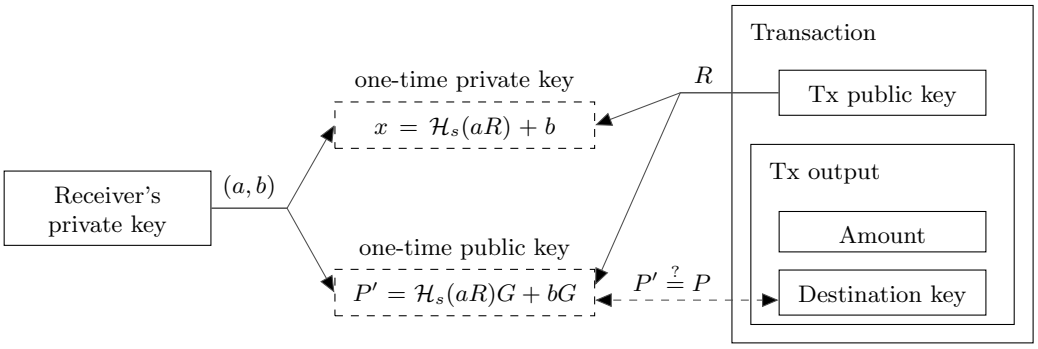

Fig. 5. Incoming transaction check.

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

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

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

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

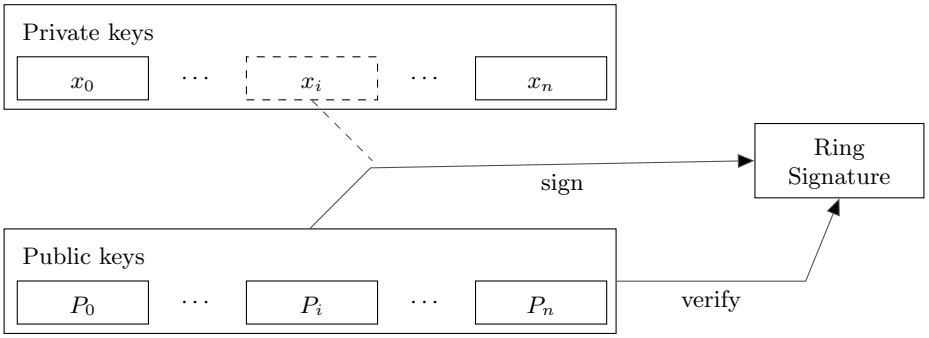

Fig. 6. Ring signature anonymity.

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

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

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

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

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

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

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

Finally the signer computes the *response*:

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

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

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

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

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

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

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

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

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

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

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

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.

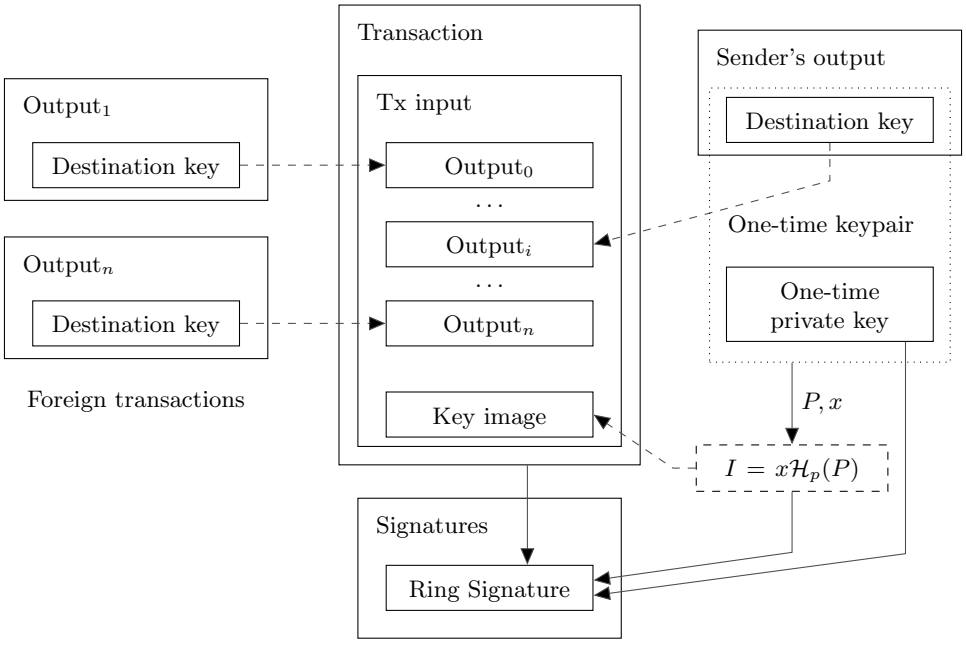

Fig. 7. Ring signature generation in a standard transaction.

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.

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

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).

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

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 $0 \leq k \leq n$ 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 $0 \leq M \leq N$ valid signatures) $\Phi = x_1 + x_2 + \ldots + x_N \geq M$ (for clarity we are using common algebraic notation). The weighted threshold signature (some keys can be more important than other) could be expressed as $\Phi = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_N \cdot x_N \geq w_M$. And scenario where the master-key corresponds to $\Phi = \texttt{max}(M \cdot x, x_1 + x_2 + \ldots + x_N) \geq M$. 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.

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

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

A Security

We shall give a proof for our one-time ring signature scheme. At some point it coincides with the parts of the proof in [24], but we decided to rewrite them with a reference rather than to force a reader to rush about from one paper to another.

These are the properties to be established:

- **Linkability.** Given all the secret keys $\{x_i\}_{i=1}^n$ for a set $\mathcal{S}$ it is impossible to produce $n+1$ valid signatures $\sigma_1, \sigma_2, \ldots, \sigma_{n+1}$, such that all of them pass the **LNK** phase (i.e. with $n+1$ different key images I_i). This property implies the double spending protection in the context of CryptoNote.
- **Exculpability.** Given set $\mathcal{S}$, at most $n-1$ corresponding private keys x_i (excluding $i = j$) and the image I_j of the keys x_j it is impossible to produce a valid signature σ with I_j. This property implies theft protection in the context of CryptoNote.
- **Unforgeability.** Given only a public keys set $\mathcal{S}$ it is impossible to produce a valid signature σ.
- **Anonymity.** Given a signature σ and the corresponding set $\mathcal{S}$ it is impossible to determine the secret index j of the signer with a probability $p > \frac{1}{n}$.

Linkability

Theorem 1. *Our one-time ring signature scheme is linkable under the random oracle model.*

Proof. Suppose an adversary can produce $n+1$ valid signatures σ_i with key images $I_i \neq I_j$ for any $i, j \in [1 \ldots n]$. Since $\#\mathcal{S} = n$, at least one $I_i \neq x_i \mathcal{H}_p(P_i)$ for every i. Consider the corresponding signature $\sigma = (I, c_1, \ldots, c_n, r_1, \ldots, r_n)$. $\mathbf{VER}(\sigma) =$ "true", this means that

$$\begin{cases} L'_i = r_i G + c_i P_i \\ R'_i = r_i \mathcal{H}_p(P_i) + c_i I \\ \sum\limits_{i=1}^{n} c_i = \mathcal{H}_s(m, L'_1, \ldots, L'_n, R'_1, \ldots, R'_n) \mod l \end{cases}$$

The first two equalities imply

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

where $\log_A B$ informally denotes the discrete logarithm of B to the base A.

As in [24] we note that $\nexists i : x_i = \log_{\mathcal{H}_p(P_i)} I$ implies that all c_i's are uniquely determined. The third equality forces the adversary to find a pre-image of $\mathcal{H}_s$ to succeed in the attack, an event whose probability is considered to be negligible. □

Exculpability

Theorem 2. *Our one-time ring signature scheme is exculpable under the discrete logarithm assumption in the random oracle model.*

Proof. Suppose an adversary can produce a valid signature $\sigma = (I, c_1, \ldots, c_n, r_1, \ldots, r_n)$ with $I = x_j \mathcal{H}_P(P_j)$ with given $\{x_i \mid i = 1, \ldots, j-1, j+1, \ldots, n\}$. Then, we can construct an algorithm $\mathcal{A}$ which solves the discrete logarithm problem in $E(\mathbb{F}_q)$.

Suppose inst $= (G, P) \in E(\mathbb{F}_q)$ is a given instance of the DLP and the goal is to get s, such that $P = sG$. Using the standard technique (as in [24]), $\mathcal{A}$ simulates the random and signing oracles and makes the adversary produce two valid signatures with $P_j = P$ in the set $\mathcal{S}$: $\sigma = (I, c_1, \ldots, c_n, r_1, \ldots, r_n)$ and $\sigma' = (I, c'_1, \ldots, c'_n, r'_1, \ldots, r'_n)$.

Since $I = x_j \mathcal{H}_p(P_j)$ in both signatures we compute $x_j = \log_{\mathcal{H}_p(P_j)} I = \frac{r_j - r'_j}{c'_j - c_j} \bmod l$

$\mathcal{A}$ outputs x_j because $L_j = r_j G + c_j P_j = r'_j G + c'_j P_j$ and $P_j = P$. □

Unforgeability

It has been shown in [24] that unforgeability is just an implication of both linkability and exculpability.

Theorem 3. *If a one-time ring signature scheme is linkable and exculpable, then it is unforgeable.*

Proof. Suppose an adversary can forge a signature for a given set $\mathcal{S}$: $\sigma_0 = (I_0, \ldots)$. Consider all valid signatures (produced by the honest signers) for the same message m and the set $\mathcal{S}$: $\sigma_1, \sigma_2, \ldots, \sigma_n$. There are two possible cases:

1. $I_0 \in \{I_i\}_{i=1}^n$. Which contradicts exculpability.
2. $I_0 \notin \{I_i\}_{i=1}^n$. Which contradicts linkability. □

Anonymity

Theorem 4. *Our one-time ring signature scheme is anonymous under the decisional Diffie-Hellman assumption in the random oracle model.*

Proof. Suppose an adversary can determine the secret index j of the Signer with a probability $p = \frac{1}{n} + \epsilon$. Then, we can construct algorithm $\mathcal{A}$ which solves the decisional Diffie-Hellman problem in $E(\mathbb{F}_q)$ with the probability $\frac{1}{2} + \frac{\epsilon}{2}$.

Let inst $= (G_1, G2, Q_1, Q_2) \in E(\mathbb{F}_q)$ be the instance of DDH and the goal to determine if $\log_{G_1} Q_1 = \log_{G_2} Q_2$. $\mathcal{A}$ feeds the adversary with valid signature $\sigma_0 = (I, \ldots)$, where $P_j = x_j G_1 = Q_1$ and $I = Q_2$ and simulates oracle $\mathcal{H}_p$, returning G_2 for query $\mathcal{H}_p(P_j)$.

The adversary returns k as his guess for the index i: $I = x_i \mathcal{H}_P(P_i)$. If $k = j$, then $\mathcal{A}$ returns 1 (for "yes") otherwise a random $r \in \{1, 0\}$. The probability of the right choice is computed as in [24]: $\frac{1}{2} + \Pr(1 \mid \text{inst} \in DDH) - \Pr(1 \mid \text{inst} \notin DDH) = \frac{1}{2} + \Pr(k = j \mid \text{inst} \in DDH) + \Pr(k \neq j \mid \text{inst} \in DDH) \cdot \Pr(r = 1) - \Pr(k = j \mid \text{inst} \notin DDH) - \Pr(k \neq j \mid \text{inst} \notin DDH) \cdot \Pr(r = 0) = \frac{1}{2} + \frac{1}{n} + \epsilon + (\frac{n-1}{n} - \epsilon) \cdot \frac{1}{2} - \frac{1}{n} - \frac{n-1}{n} \cdot \frac{1}{2} = \frac{1}{2} + \frac{\epsilon}{2}$

In fact, the result should be reduced by the probability of collision in $\mathcal{H}_s$, but this value is considered to be negligible. □

Notes on the hash function $\mathcal{H}_p$

We defined $\mathcal{H}_p$ as deterministic hash function $E(\mathbb{F}_q) \to E(\mathbb{F}_q)$. None of the proofs demands $\mathcal{H}_p$ to be an ideal cryptographic hash function. It's main purpose is to get a pseudo-random base for image key $I = x\mathcal{H}_p(xG)$ in some determined way.

With fixed base ($I = xG_2$) the following scenario is possible:

1. Alice sends two standard transactions to Bob, generating one-time tx-keys: $P_2 = \mathcal{H}_s(r_1 A)G + B$ and $P_1 = \mathcal{H}_s(r_2 A)G + B$.

2. Bob recovers corresponding one-time private tx-keys x_1 and x_2 and spends the outputs with valid signatures and images keys $I_1 = x_1 G_2$ and $I_2 = x_2 G_2$.

3. Now Alice can link these signatures, checking the equality $I_1 - I_2 \stackrel{?}{=} (\mathcal{H}_s(r_1 A) - \mathcal{H}_s(r_2 A))G_2$.

The problem is that Alice knows the linear correlation between public keys P_1 and P_2 and in case of fixed base G_2 she also gets the same correlation between key images I_1 and I_2. Replacing G_2 with $\mathcal{H}_p(xG_2)$, which does not preserve linearity, fixes that flaw.

For constructing deterministic $\mathcal{H}_p$ we use algorithm presented in [37].

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.

[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.