Bitter to Better — How to Make Bitcoin a Better Currency

Simon Barber [1], Xavier Boyen [1], Elaine Shi [2⋆], and Ersin Uzun [1]

[1] Palo Alto Research Center
[2] University of California, Berkeley

Abstract. Bitcoin is a distributed digital currency which has attracted a substantial number of users. We perform an in-depth investigation to understand what made Bitcoin so successful, while decades of research on cryptographic e-cash has not lead to a large-scale deployment. We ask also how Bitcoin could become a good candidate for a long-lived stable currency. In doing so, we identify several issues and attacks of Bitcoin, and propose suitable techniques to address them.

1 Introduction

Bitcoin is a decentralized electronic cash system initially designed and developed by Satoshi Nakamoto (whose name is conjectured to be fake by some, and who has not been heard from since April 2011). The design of Bitcoin was first described in a self-published paper by Nakamoto [14] in October 2008, after which an open-source project was registered on sourceforge. The genesis block was established on January 3rd 2009, and the project was announced on the Cryptography mailing list on January 11th 2009.

Since its invention, Bitcoin has gained amazing popularity and much attention from the press. At the time of the writing, approximately 7M Bitcoins are in circulation; approximately USD $2M to $5M worth of transactions take place each day in Bitcoin; and about eighteen Bitcoin exchanges exist offering exchange services with many real world currencies, (e.g., EUR, USD, CAD, GBP, PLN, JPY, HKD, SEK, AUD, CHF, and so on). Bitcoin's exchange rate has varied widely, reaching as high as USD $30 per Bitcoin although at the time of writing is around USD $5 per Bitcoin.

Despite some pessimists' critiques and disbelief, Bitcoin has admittedly witnessed enormous success since its invention. To the security and cryptographic community, the idea of digital currency or electronic cash is by no means new. As early as 1982, Chaum has outlined his blueprint of an anonymous e-cash scheme in his pioneering paper [10]. Ever since then, hundreds of academic papers have been published to improve the efficiency and security of e-cash constructions — to name a few, see [15, 8, 9].

Naturally, an interesting question arises: *Despite three decades' research on e-cash, why have e-cash schemes not taken off, while Bitcoin — a system designed and initially implemented possibly single-handedly by someone previously unknown, a system that uses no fancy cryptography, and is by no means perfect — has enjoyed a swift rise to success*? Looking forward, one also wonders: *Does Bitcoin have what it takes to become a serious candidate for a long-lived stable currency, or is it yet another transient fad*?

Intrigued by these questions, we investigated Bitcoin's design and history, and came to many interesting realizations. We therefore present this paper to the (financial) cryptography research community, with the following goals and expectations:

⋆ Work done while author was affiliated with PARC.

1. To investigate the Bitcoin phenomenon, and achieve a deeper understanding of the crucial factors underlying its success, especially compared to other e-cash schemes.
2. To scrutinize the design of Bitcoin and analyze its strengths and weakness, in order to expose and anticipate potential attacks, and focus investigation on key issues;
3. To suggest redesigns, improvements, or extensions, such as, e.g., our fail-safe mixer protocol that requires no third-party and no system modification (Section 7).
4. To pose open research problems stemming from our broad reflections on Bitcoin;
5. Last but not least, to bring Bitcoin to the attention of the cryptography research community, to encourage it to reflect on its success, and draw lessons therein.

2 The Intriguing Success of Bitcoin: A Comparative Study

As mentioned earlier, despite three decades' research on e-cash by the cryptographic community [10, 15, 8, 9], all these efforts seem to have been dwindled by the swift success of Bitcoin. Has Nakamoto, a single individual whose name previously unheard of, outsmarted the ingenuity of all the cryptographers combined? Bitcoin is by no means perfect and some well-known problems are discussed later on. So what is it in Bitcoin that has ensured its success?

After an in-depth investigation of Bitcoin, we found that although Bitcoin uses no fancy cryptography, its design actually reflects a suprising amount of ingenuity and sophistication. Most importantly, it addresses the *incentive* problems most expeditiously.

No central point of trust. Bitcoin has a completely distributed architecture, without any single trusted entity. Bitcoin assumes that the majority of nodes in its network are honest, and resorts to a majority vote mechanism for double spending avoidance, and dispute resolution. In contrast, most e-cash schemes require a centralized bank who is trusted for purposes of e-cash issuance, and double-spending detection. This greatly appeals to individuals who wish for a freely-traded currency not in control by any governments, banks, or authorities — from libertarians to drug-dealers and other underground economy proponents (note that apart from the aforementioned illegal usages, there are numerous legitimate uses as well, which will be mentioned later). In a spirit similar to the original motivation for a distributed Internet, such a purely decentralized system guarantees that no single entity, no matter how initially benevolent, can succumb to the temptation or be coerced by a government into subverting it for its own benefit.

Incentives and economic system. Bitcoin's *eco-system* is ingeniously designed, and ensures that users have economic incentives to participate. First, the generation of new bitcoins happens in a distributed fashion at a predictable rate: "bitcoin *miners*" solve computational puzzles to generate new bitcoins, and this process is closely coupled with the verification of previous transactions. At the same time, miners also get to collect optional transaction fees for their effort of vetting said transactions. This gives users clear economic incentives to invest spare computing cycles in the verification of Bitcoin transactions and the generation of new Bitcoins. At the time of writing the investment of a GPU to accelerate Bitcoin puzzle solution can pay for itself in $\sim$6 months.

Predictable money supply. Bitcoin makes sure that new coins will be minted at a fixed rate, that is, the larger the Bitcoin community and the total computational resource devoted to coin generation, the more difficult the computational puzzle becomes. This

provides strong incentives for early adopters — the earlier in the game, the cheaper the coins minted. (In a later section we discuss negative consequences that the adopted money supply schedule will have, in the long term, on value, incentives, and security.)

Divisibility and fungibility. One practical appeal of Bitcoin is the ease with which coins can be both divided and recombined to create essentially any denomination possible. This is an Achilles' heel of (strongly anonymous) e-cash systems, because denominations had to be standardized to be unlinkable, which incidentally makes the computational cost of e-cash transactions linear in the amount. In Bitcoin, linkage is inherent, as it is what prevents double spending; but it is the identities that are "anonymous".

Versatility, openness, and vibrancy. Bitcoin is remarkably flexible partly due to its completely distributed design. The open-source nature of the project entices the creation of new applications and spurs new businesses. Because of its flexibility and openness, a rich extended ecosystem surrounding Bitcoin is flourishing. For example, *mixer* services have spawned to cater to users who need better anonymity guarantees (see Section 7 for details). There are payment processor services that offer gadgets venders can embed in their webpages to receive Bitcoin payments alongside regular currency.

Scripting. Another salient and very innovative feature is allowing users (payers and payees) to embed scripts in their Bitcoin transactions. Although today's reference implementations have not fully utilized the power of this feature, in theory, one can realize rich transactional semantics and contracts through scripts [2], such as deposits, escrow and dispute mediation, assurance contracts, including the use of external states, and so on. It is conceivable that in the future, richer forms of financial contracts and mechanisms are going to be built around Bitcoin using this feature.

Transaction irreversibility. Bitcoin transactions quickly become irreversible. This attracts a niche market where vendors are concerned about credit-card fraud and chargebacks. Through personal communication with a vendor selling specialty magazines, he mentioned that before, he could not conduct business with customers in certain countries where credit-card fraud prevails. With Bitcoin, he is able to extend his business to these countries due to the protection he obtains from the irreversibility of transactions.

Low fees and friction. The Bitcoin verifiers' market currently bears very low transaction fees (which are optional and chosen by the payer); this can be attractive in micropayments where fees can dominate. Bitcoin is also appealing for its lack of additional costs traditionally tacked upon international money transfers, due to disintermediation.

Readily available implementations. Last but not the least, in comparison with other e-cash schemes, Bitcoin has provided readily available implementations, not only for the desktop computer, but also for mobile phones. The open-source project is maintained by a vibrant community, and has had healthy developments.

3 Under the Hood of the Bitcoin System

Bitcoin is based on a peer-to-peer network layer that broadcasts data to all nodes on the network. There are two types of object that are broadcast: transactions and blocks. Both object types are addressed by a hash of the object data, and are broadcast through the network to all nodes. Transactions are the operations whereby money is combined, divided, and remitted. Blocks record the transactions vetted as valid.

Spending. Suppose that Alice wishes to remit 1 bitcoin to Bob and 2 to Carol. Alice's coins "reside" in prior transactions that designate her public key as beneficiary. To spend coins, Alice creates a new transaction that endorses any such coins she has not spent yet, e.g., she can endorse, using a digital signature, 4 coins each received from Diane and Edgar as the *inputs* of her new transaction. As *outputs* she specifies 1 coin for Bob, 2 for Carol, and 4.99 of "change" back to herself. In this example, Alice chose to leave a residual of 0.01 coin, which can be claimed as a *fee* by whoever vets it first.

Vetting. In order for a transaction to be confirmed, its various components must be validated and checked against double spending. Once verified, transactions are incorporated in frequently issued official records called *blocks*. Anyone is allowed to create such blocks, and indeed two sorts of incentives are offered to attract verifiers to compete for block creation: (1) the collection of fees; and (2) the minting of new coins.

Minting. The bitcoin money supply expands as each block created may contain a special *generation transaction* (with no explicit input) that pays the block creator a time-dependent amount for the effort (50 coins today, rapidly decreasing). The rate of block, hence money, creation is limited by a *proof of work* of adaptive difficulty, that strives to maintain a creation rate of one block every 10 minutes across the whole network. Bitcoin transaction verification is thus a lucrative race open to all, but a computationally expensive one. Note: "bad" blocks will be rejected by peers, invalidating their rewards.

3.1 Transactions and Scripting: the Tools for Spending

One of the main powers of the Bitcoin system is that the input and output of transactions need not have a fixed format, but rather are constructed using a Forth-like stack-based flexible scripting language. We remark that transaction principals are not named users but anonymous public keys, which users may freely create in any number they wish.

Transactions. Transaction encapsulate the movement of bitcoins by transfering the value received from its *inputs* to its *outputs* (exception: *generation transactions* have no explicit input at all). An input identifies a previous transaction output (as the hash of the earlier transaction and an index to an output within it), and claims its full value. An output specifies an amount; the outputs' total must not exceed the inputs'. Both also contain fragments of executable script, on the input side for redeeming inflows, and on the output side for designating payees.

Script fragments. The scripting language is a Forth-like stack-based language. Operators include cryptographic operations like SHA1 (which replaces the top item on the stack with its hash), and CHECKSIG (which pops an ECDSA public key and signature from the stack, verifies the signature for a "message" implicitly defined from the transaction data, and leaves the result as a true or false on the stack). For a transaction to be valid, its outputs must not exceed its inputs, and its issuer must show title to each input claimed. Title is tested by evaluating the input script fragment concatenated with the script fragment from the output (of an earlier transaction) that the input references.

Standard transfer. To illustrate how the stack-based scripting language can be used, among other things, to designate and enforce the recipient of a transfer, we study the example of the standard Bitcoin transaction used for transfer. To send coins to an address stated as the hash of a public key, the payer, Alice, creates a transaction output with

the following associated script fragment (recall that since the amount is specified in a special record associated with the output; the script only needs to enforce the recipient):

```
DUP HASH160 <recipient-address> EQUALVERIFY CHECKSIG      (⋆)
```

The recipient, Bob, will notice the remittance (since it is broadcast to all), and mark it for spending. Later on, to spend those received coins, he creates a transaction with an input that redeems them, and an output that spends them. The redeeming input script is:

```
<signature> <public-key>                                  (⋆⋆)
```

Bob will have managed to spend coins received from Alice if his redemption is valid. This is checked by executing the concatenated script $(\star, \star\star)$: the input fragment $(\star)$ pushes a signature and a key on the stack; the output fragment $(\star\star)$ checks that the key hash matches the recipient, and checks the signature against transaction and key.

3.2 Blocks and Coin Creation: the Process of Verifying

Transactions become effective after they have been referenced in a *block*, which serve as the official record of executed transactions. Transactions may only be listed in a block if they satisfy such conditions as valid timestamping and absence of double spending.

Blocks. A block consists of one "*coinbase*" minting transaction, zero or more regular spending transactions, a computational proof of work, and a reference to the chronologically prior block. Thus the blocks form a singly linked *blockchain*, rooted in Nakamoto's genesis block whose hash is hardcoded in the software. The regular creation of new blocks serves the dual purpose of ensuring the timely vetting of new transactions, and the creation of new coins, all in a decentralized process driven by economic incentives (the minting of new coins and the collection of fees) balanced by computational costs. The *difficulty* of the required proof of work is adjusted by a feedback mechanism that ensures an average block creation interval of 10 minutes across the entire network.

Coinbase. Currently, each new block may contain a coinbase transaction with an implicit input value of 50 coins, with about $7M$ already minted as of this writing. The minting rate is slated to decrease shortly, eventually to reach zero when the total supply reaches about $21M$ bitcoins. The coinbase transaction also serves to claim all the *fees* in the transactions collected in the block. Both minting and fees motivate people to create blocks and hence keep the system alive.

3.3 Forking and Conflict Resolution

If two blocks are published nearly simultaneously, a fork in the chain can occur. Nodes are programmed to follow the blockchain whose total proof-of-work *difficulty* is the largest and discard blocks from other forks. Transactions on the discarded branch will eventually be collected into blocks on the prevailing branch. This mechanism ensures that one single ordering of transactions becomes apparent and accepted by all (although it may take a few blocks' time to become clear), and hence this solves the double-spending problem.

4 Structural Problems and Potential Solutions

Whether by accident or by design, the Bitcoin system as presently parameterized defines a currency with *extreme deflationary* characteristics built into it. Currently coins are

minted by verifiers (i.e., block creators, or "*miners*") as an incentive to keep the Bitcoin ecosystem running, but minting is poised to expire gradually, and rather soon, resulting in a hard cap on the coins total. Moreover, coins whose private key has been forgotten or destroyed — let us call them *zombie coins* — can never be replaced, resulting in further shrinkage of the money base. For perspective, of the 21M coins maximum, 7M have already been minted; and of those, tens of thousands have reportedly become zombies.

Aside from economic considerations that have been discussed at length [4], The potential deflationary spiral in a decentralized system like Bitcoin has security implications that should not be neglected.

4.1 Deflationary Spiral

In capped supply, bitcoins have no alternative but to appreciate tremendously should the system ever gain more than marginal acceptance. Even in a "mature market" scenario with, say, a stable 1% of the US GDP transacted in BitCoins and 99% in dollars, the real purchasing power of coins would still increase over time, as each coin would capture a correspondingly constant fraction of the country's growing wealth. Put in another way, while the Federal Reserve can increase the number of dollars in circulation to accommodate economic growth, in a Bitcoin economy the only outlet for growth would be appreciation of the currency. While it has been observed that the money supply cap could lead to a severe deflationary spiral [4], it is quite a paradox that the intrinsic strength of the Bitcoin currency could be its greatest weakness, causing an even more catastrophic unraveling than through "mere" deflation.

Hoarding: a moral hazard? Bitcoins much more than any other currency in existence derive their value from the presence of a live, dynamic infrastructure loosely constituted by the network of verifiers participating in block creation. Because of their appreciation potential, bitcoins will tend to be saved rather than spent. As hoarded bitcoins vanish from circulation, transaction volume will dwindle and block creation will become less profitable (fewer fees to collect). If circulation drops too much, it can precipitate a loss of interest in the system, resulting in "bit rot" and verifier dearth, until such point that the system has become too weak to heal and defend itself. Of particular concern is an unavoidable large-scale fraud that we describe in the next section, and whose aftermath includes sudden loss of confidence, collapse of value, and repudiation.

Towards decentralized organic inflation. An antidote to the preceding predicament could take the form of a Bitcoin-like electronic currency with a decentralized inflationary feedback built-in, that could control the global minting rate based, e.g., on transaction volume statistics. While we leave the devising of monetary parameters for such an "organically inflationary" currency as an open problem, we show next how deflationary expectations negatively impact the long-term structural security of the Bitcoin system.

4.2 Doomsday, or the "History-Revision" Attack

In the Bitcoin world, transactions are irrevocably valid once they are incorporated into the ever growing Block Chain, *insofar as they do not end up in the discarded branch of a fork*. As previously described, short-lived forks may arise, but tend to be quickly resolved per the rule that the chain whose "total difficulty" is the greatest, prevails.

Most forks are benign, causing the few transactions on the wrong side of the fork to be delayed — merely a temporary rejection, unless double spending was attempted.

This approach works well, under the crucial assumption that no attacker should ever be able to muster so much computational power that it is able to fake and publish an "alternative history", created *ex post facto*, that has greater total difficulty and hence is more authoritative than the actual history. In such event, the forking rules would cause the actual history to be discarded in favor of the alternative history, from the forking point onwards. We designate this as the *history-revision* attack. In the extreme case where the fork is made near time zero, a history-revision attacker would cause the entire coin base ever created to be replaced with a figment of its forgery.

One may take solace in the ludicrous amount of computing power that, one might hope, such a history-revision attack would require. Alas, the threat is very real — owing both to technical and monetary characteristics of Bitcoin.

Technical vulnerability. The attack's feasibility stems from Moore's law, which empirically posits that computation power per unit cost is doubling every year or so. Assuming a stable population of verifiers, the *block difficulty* parameter (set by the system to maintain a block creation mean interval of 10 minutes) is thus an exponential function of time, $f(t) = \alpha\, e^{t/\tau}$. The *total difficulty* of the block chain at any point in time is thus approximated by the integral $F(t) = \int_{t_0}^{t} f(t')dt' \propto f(t)$. It follows that, regardless of the block chain's length, an attacker that can muster a small multiple (say $2\times$) of the computation power of the legitimate verifiers together, and starting an attack at time $t = t_1$, will be able to create an entire alternative history *forked at the origin time* t_0, whose total difficulty $F'(t)$ overtakes $F(t)$ at some future time $t = t_2$, where the attack length $\Delta t = t_2 - t_1$ is bounded by a constant (about 1–2 years for a $2\times$ multiple). [3]

Economic motivation. The strong deflationary characteristic of Bitcoin further compounds the problem. On the one hand, Bitcoins are a currency poised to explode in value, *ceteris paribus*, as already discussed; and hence so will the incentive for theft. On the other hand, the way deflation comes into play, driven by a hard cap on the money supply, will all but eliminate the money-minting incentive that currently draws in the many verifiers that by their competition contribute to make block creation a difficult problem. With this incentive dwindling, laws of economics dictate that the competitive effort devoted to verifying transactions and creating blocks will diminish. In other words, while block difficulty may continue to increase for some time into the future, it will eventually start to *decrease* relatively to the power of the day's typical PC. History revision attacks will thence become *easier* not harder.

4.3 Countering "Revisionism" by Checkpointing the Past

We outline a distributed strategy to tackle the history-revision attack threat in a simple and elegant way. Its principle is rooted in the commonsense notion that one ought to be suspicious of tales that conflict with one's own first-hand recollection of events.

[3] To underscore the seriousness of the threat, we note that it is common nowadays for *miners* to pool their resources and, by some estimates, one such *mining pool*, `deepbit`, contributes 40% of the total computation power devoted to mining in the entire system. Merely doubling its "market share" would make it able to revise the entire Bitcoin history in a year's time, owing to Moore's law. Botnets and governments may be there already.

Translated in the Bitcoin world, we propose that a Verifier that has been running without interruption for a long time, should be "highly skeptical" of any long-range fork resolution that would drastically change its own private view of the transaction history acquired from *first-hand* data collection and block creation.

Private checkpointing. Verifiers should thus timestamp published transactions as they see them, and privately take regular snapshots of their own view of the transaction history (such snapshots should be made tamper-proof, e.g., with a cryptographic forward-secure signature). If in the future a drastic fork is put forth that is inconsistent with many of the various snapshots taken, the verifier should demand an increasingly high burden of proof before accepting the "new" branch as correct. E.g., the verifier should not merely accept an alternative branch whose total difficulty exceeds that of the privately checkpointed history, but demand an increasingly high margin of excess, the longer and the more improbable the alternative branch is deemed w.r.t. the verifier's private knowledge.

Implicit voting and phase transition. Verifiers ought to make such determination independently, based on *their own* remembered history. That is to say that "young" verifiers that recently came online, and acquired their history by downloading the transaction log *ex post facto*, would have little first-hand checkpointing to rely upon, and would thus behave as in the current system (merely favoring the most difficult branch in a fork). "Seasoned" verifiers that have seen and checkpointed ancient transactions first-hand, would on the contrary oppose a resisting force of skepticism against what they perceive could be an attempt to revise history. As a result, the network would partition into two camps, but only briefly, as certain verifiers that are on the fence "flip" one way or the other based on observing their peers' endorsement of either branch of the fork. Eventually, as more and more verifiers endorse one position over the other, and the corresponding branch of the fork grows faster, the whole network will "phase-transition" back to a single unified view.

Comparative behavior. Our strategy is a strict improvement over the current Bitcoin handling of history-revision attacks, for in all cases where a history-revision attack would *fail* in the current system, our system would behave identically (and exhibit no partition, and no subsequent phase transition). It is only in cases where a history-revision attack would have *succeeded* in the current system, that a partition could occur in the new system. A partition could remain meta-stable for a certain time, but eventually ought to resolve itself by taking the bulk of the network to one side or the other.

Checkpointing today. We remark that the current protocol already does what we would call "fiat checkpointing", where authoritative checkpoints (in the form of hardcoded hashes of certain blocks) are pushed out with software updates [12]. Alas, there is no reason to trust a download of the software any more than one of the transaction history itself. This is unlke our private checkpointing proposal which emphatically prescribes first-hand checkpoints, independently made by each verifier in a privately tamper-proof decentralized way.

We leave as an open problem the formal design and analysis of "anti-revisionism profiles" that offer marked security against vastly powerful history-revision attacks, while guaranteeing that partitions caused by accidental forks get quickly resolved.

5 Theft or Loss of Bitcoins

As all bitcoins are public knowledge (in the form of unredeemed transaction outputs), what enables a user to spend a coin is possession of the associated private key. Theft or loss of private keys, or signature forgeries, thus equate to loss of money in this world.

5.1 Malware Attacks

Reported malware attacks on Bitcoin are on the rise [16, 1], resulting in the theft of private keys. The online wallet service `mybitcoin.com` recently lost $\$1.3$ million worth of users' coins due to malware [1]. Several solutions can be envisaged; we mention:

Threshold cryptography. A natural countermeasure to malware is to split private keys into random shares, using standard threshold cryptography techniques [11, 13], and distribute them onto multiple locations, e.g., a user's desktop computer, her smart phone, and an online service provider. In this way, only when a threshold number of these devices collaborate, can a user spend her coins. Of course, doing so can harm the usability of the system, since coins can no longer be spent without operating multiple devices (even though not all the devices but only a chosen number of them are needed at once).

Super-wallets. To address the usability concern, we propose the simple idea of *super-wallet*, i.e., a user's "personal bank" where most of her coins are stored. The super-wallet is split across multiple computing devices, using threshold techniques as above. In addition, the user carries a small *sub-wallet* with her on her smartphone. Pre-approved transactions are setup so that the user can withdraw money from her super-wallet onto her sub-wallet, periodically in small amounts (similar to how real banks let people withdraw cash from ATMs today). The user now only needs her smartphone to spend money in her wallet, and in case her smartphone is captured by an adversary, the user only loses the small amount of money that she has in her wallet, but not that in her personal bank. Large amounts can always be spent from the super-wallet using a threshold of devices.

Both approaches can be implemented as backward-compatible and incrementally deployable wrappers, requiring changes in the signature generation but not verification.

5.2 Accidental Loss of Bitcoins

Apart from malware, system failures or human errors can cause the accidental loss of the wallet file (which stores the private keys needed to spend coins), which in turn leads to the loss of coins (turning them into *zombies*). For example, `bitomat`, the third largest bitcoin exchange, recently lost about $\$200K$ worth of bitcoins (at the exchange rate at the time) due to the loss of its private wallet file — the cause was later identified to be human error, as the developer hosted the wallet on non-persistent cloud storage [3].

Backups. Naturally, the universal answer against accidental loss or adversarial destruction of data, is to follow best-practice backup procedures. For backup purposes, the wallet file should be treated like any other private cryptographic asset — meaning that backups are a non-trivial proposition, not because of volume, but because of secrecy. With Bitcoin, things are complicated by the incessant creation of keys.

Pseudo-random keys. To avoid having to back up a constantly growing wallet file, a trivial solution is to generate all of one's private keys not at random, but pseudo-randomly from a master secret that never changes, using a standard PRG. The problem then reduces to that of backing up the short and static PRG seed, e.g., in a bank vault.

Encryption. A natural idea is to encrypt the wallet using a password sufficiently strong that the resulting ciphertext can be widely replicated without fear of cryptanalysis. This approach is especially useful in conjunction with pseudo-random keys, as then coins can be spent and received without requiring the ciphertext to be updated. The main problem, of course, is that strong passwords are prone to memory loss and palimpsest.

Offline (single-)password-based encryption. One solution relies on the "optimal" password-based encryption system of [7], which offers optimal trade-offs between password strength (how tough it is to guess) and "snappiness" (how quickly it can be used, which is also kept a secret). Users can even set multiple passwords with varying trade-offs for a common security goal: e.g., an everyday password, complex but snappy; and a backup password, simple but just as secure by virtue of being made "sluggish". A pseudo-random wallet seed, encrypted à la [7], would combine static portability with usable protection against both loss and theft, and is probably the best approach for an isolated user who trusts his mental possessions more than his physical ones.

Online (multi-)password-based encryption. Another approach is to combine the power of several memorable secrets into a single high-security "vault", using the protocols of [5]. Each member in some circle of friends holds a short totally private and long-term memorable phrase. One member is a distinguished *leader*. Without revealing their secrets, the members can perform private operations such as signing or decrypting a message on behalf of the leader. With this protocol, a group of users can cooperate to let the leader spend the coins from his wallet (kept as a public, static, accessible, encrypted file), by issuing signatures on messages created by the leader. This approach provides strong safety against loss, plus security against compromise of a subset of the group.

Trusted paths. Any of the above approaches can be combined with *trusted-path* devices, which are dedicated hardware devices that let humans input and read out (tiny amounts of) cryptographic data out of the reach of any malware. European banks use the DigiPass, for example. Alas, while trusted-path protocols are well known and very safe when it can be assumed that the remote server is uncorrupted (e.g., when talking to a bank), in the Bitcoin case the server is the user's own PC, possibly infected. It is an interesting open problem to devise trusted-path protocols that are secure in this model, *when the trusted-path data is too tiny* to provide cryptographic strength by itself.

6 Scalability

Bitcoin suffers from several scalability issues, among which we note the following.

6.1 Data Retention and Communication Failures

The smooth operation of Bitcoin relies on the timely broadcast of transactions and blocks. A preprint [6] suggests that verifiers competing for the same reward have an incentive to withhold the information needed to do so. However, since *transactors* have an incentive to disseminate their data as quickly and widely as possible, not only is retention futile, but economic forces will counter it by fostering circumvention services.

6.2 Linear Transaction History

As discussed, the Bitcoin *wallet* software fetches the entire Bitcoin blockchain at installation, and all new transactions and blocks are (supposedly) broadcast to all nodes.

The Bitcoin nodes cryptographically verify the authenticity of all blocks and transactions as they receive them. Clearly, this approach introduces a scalability issue in the longer term, in terms of both network bandwidth, and computational overhead associated with cryptographic transaction verification. The scalability issue can be worrying for smart phones with limited bandwidth, computational power, and battery supply.

The scalability issue can be addressed with a subscription-based filtering service. Recall that Bitcoin nodes can be divided into broadly two classes, *verifiers* and *clients*. Verifiers create new blocks and hence mint new coins. Verifiers are mostly nodes with ample computational and bandwidth resources, typically desktop computers. By contrast, clients are Bitcoin nodes that are not actively minting new coins, such as smart phones. While verifiers have incentives to receive all transactions (to earn transaction fees), clients may not care. In particular, all that is needed for clients to spend their coins is that they receive transactions payable to their public key(s).

Bitcoin filtering service. Our filtering service is a third-party cloud service provider which filters Bitcoin transactions, and sends only relevant transactions to nodes that have registered for the service. A Bitcoin client (e.g., a user's smartphone) can send a cryptographic capability to the filtering service, which allows the filtering service to determine whether a transaction is payable to one or more of its public keys.
We identify the following desirable security and usability requirements.

- *Unlinkability without the capability.* While a user may allow the filtering service to determine which transactions are payable to itself, no other party should be able to link a user's multiple public keys better than they can today (i.e., without the filtering service).
- *Forward security.* The filtering service should be able to update its capability periodically, such that in the case of compromise or a subpoena, the revealed capability can allow one to identify new transactions targeted to a specific user, but cannot be used to link the users' transactions in the past.
- *Reasonable false positives and low false negatives.* A false positive is when the filtering service mistakenly sends a user a non-relevant transaction. False positives wastes a user's bandwidth and computational power, but a user can locally detect such false positives after receiving the transactions. A false negative is when the filtering service fails to send a user a relevant transaction. The false negative rate should ideally be 0.

Constructing a filtering service. We now propose a potential approach to build such a filtering service, in a way that is backward compatible with today's Bitcoin. Assume that the user and the filtering service are weakly time synchronized. A user can generate a random Message Authentication Code (MAC) key K and share it with the filtering service. This MAC key K will be used as the initial MAC key. For forward security, in every time epoch (e.g., every day), both the user and the filtering service will update their MAC key by applying a Pseudo-Random Generator (PRG): $K \leftarrow \mathsf{PRG}(K)$. The MAC key K will then be used as below. When the user needs to pick a public key to receive money, it will pick a public key PK whose hash $H(\mathsf{PK})$ satisfies the following condition: $\mathsf{MAC}_K(H(\mathsf{PK})) \bmod 2^\ell = 0$. In particular, when ℓ is not too large, the user can find such a public key by randomly generating public-private key pairs until a public-key satisfying the above condition is found. ℓ is a parameter used to engineer

the tradeoff between the false positive rate and the computation cost needed to generate public keys. Since a transaction payable to user A includes user A's public key hashes in one or more outputs, the filtering service can now identify transactions possibly targeted for user A by checking the above condition.

6.3 Delayed Transaction Confirmation

Another related scalability issue is delayed transaction confirmation. In the current implementation, a new block is generated about every 10 minutes, so it takes at least 10 minutes or so to get a transaction confirmed. This can be problematic in certain application scenarios, e.g., on-demand video playback Worse still, after a single confirmation it is still possible the transaction is a double spend, and the blockchain has forked.

One approach, already seen in the Bitcoin ecosystem, uses intermediate "semi-trusted" third parties acting as short-term banks, issuing the Bitcoin equivalents to cashiers' checks (essentially, a transaction signed by the bank's key). Banks would have no incentive to double-spend, as their fraud would immediately become apparent to all.

Another approach is to fundamentally reduce the transaction confirmation delay by re-parameterizing the computational puzzles to reduce the average block creation interval from 10 minutes to 10 seconds. However, this would increase the forking propensity on slow communication networks, which could become a concern.

6.4 Dynamically Growing Private Key Storage

To achieve better anonymity, users are urged to use a different public key for each transaction. However, this means that the user has to store the corresponding private keys for all previously generated public keys — a private key should only be deleted if one is certain that no payment to its public key will ever be made (lest zombie coins result). Aside from size, the dynamic nature of the private key storage is another difficulty.

Pseudo-random generation of private keys. An easy answer to both concerns, already mentioned, is to generate all of one's private keys pseudo-randomly from a static secret.

Explicit expiration of public keys. Another way to address this problem is to introduce explicit expiration dates for public keys, and ensure that no money can be sent to expired keys, or that such money can be reclaimed somehow. In any case, it is good practice that keys be made to expire, and this should be encouraged. In view of this, it seems desirable to give the scripting language facilities for reading and comparing timestamps.

7 Improving Anonymity with Reduced Trust

Bitcoin partially addresses the anonymity and unlinkability issue, by allowing users to use different addresses and public keys in every transaction. However, Bitcoin still exposes their users to a weak form of linkability. Specifically, multiple public keys of the same user can potentially be linked when the user pays change to herself, in which case two or more of a single user's public keys will appear in the same transaction [17].

To improve users' anonymity, third-party services called *mixers* have emerged, that take multiple users' coins, mix them, and issue back coins in equal denominations. Today, the mixers are trusted entities, in the sense that users send money to the mixer, trusting that it will issue back the money later. As a malicious mixer can cheat and

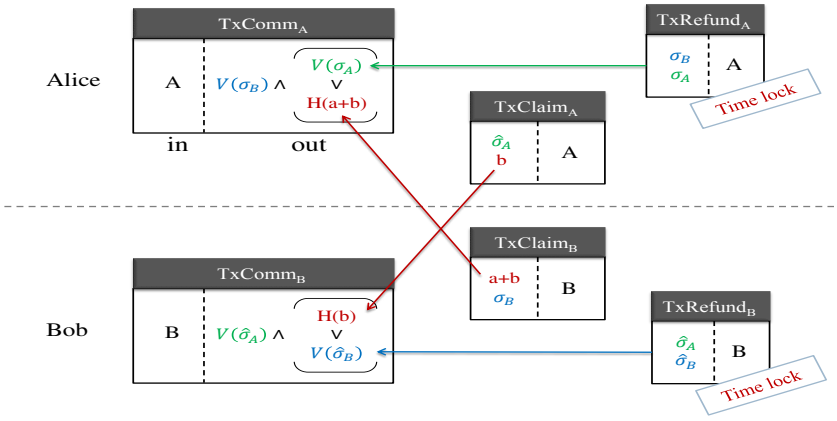

Fig. 1: A fair exchange protocol: mixing Bitcoins with an untrusted mixer.

not pay the money back, a cautious user could send the money to the mixer in small amounts, and only continue sending when the mixer has paid back. However, this approach is unscalable, especially as each transaction can take 10 minutes to confirm.

An alternative and better approach is to implement a *fair exchange* protocol. One contribution of this paper is to demonstrate how to implement such a fair exchange protocol in Bitcoin in a backward compatible manner.

7.1 A Fair Exchange Protocol

A fair exchange protocol consists of three types of transactions:

- A *commitment transaction*, denoted TxComm_A or TxComm_B, commits a party to the money exchange;
- A *refund transaction*, denoted $\mathsf{TxRefund}_A$ or $\mathsf{TxRefund}_B$, refunds a party's committed money at a future date, in case the exchange protocol aborts.
- A *claim transaction* denoted $\mathsf{TxClaim}_A$ or $\mathsf{TxClaim}_B$, allows a party to claim the other party's committed money. To ensure fairness one party conducts the first claim transaction in which it must publish a secret which enables the second claim.

Secrets setup phase. As depicted in Figure 2, Alice and Bob perform key generation, and exchange public keys. The reasons for each party to generate two key pairs is to later use different keys for different transactions to ensure unlinkability. Alice and Bob then engage in a cut-and-choose protocol. At the end of the protocol, the remaining set indexed by $\overline{I} := [n] \backslash I$ will later be used in the fair exchange. Specifically, the hash values $\{H(a_i + b_i) : \forall i \in \overline{I}\}$ will later be included in the output script of TxComm_A, and the hash values $\{H(b_i) : \forall i \in \overline{I}\}$ will be later included in the output script of TxComm_B. For Bob to claim Alice's committed money TxComm_A, it has to reveal all of the correct $\{a_i + b_i : i \in \overline{I}\}$, such that their hash values match those in TxComm_A. For Alice to later claim Bob's committed money TxComm_B, it only has to reveal one b_i for some $i \in \overline{I}$, such that it matches one hash value in TxComm_B. The key to ensuring fairness is that when Bob claims TxComm_A, it has to reveal the secrets $\{a_i + b_i : i \in \overline{I}\}$, allowing Alice to learn the values of $\{b_i : i \in \overline{I}\}$, enabling Alice to claim TxComm_B.

A cheating Bob can potentially supply Alice with the wrong $H(b_i)$ values, in an attempt to prevent Alice from claiming TxComm_B, while retaining its own ability to

1. A: Select random key pairs $(\mathsf{PK}_{A1}, \mathsf{SK}_{A1})$ and $(\mathsf{PK}_{A2}, \mathsf{SK}_{A2})$
   B: Select random key pairs $(\mathsf{PK}_{B1}, \mathsf{SK}_{B1})$ and $(\mathsf{PK}_{B2}, \mathsf{SK}_{B2})$
   A $\Leftrightarrow$ B : Exchange public keys
2. A: Select random secrets $\Omega_A = \{a_1, a_2, \ldots, a_n\}$
   B: Select random secrets $\Omega_B = \{b_1, b_2, \ldots, b_n\}$
   A $\rightarrow B$: Ω_A
   B $\rightarrow A$: $\{H(a_i + b_i), H(b_i) : \forall i \in [n]\}$
3. A $\rightarrow B$: random index set $I \subseteq [n]$, s.t. $|I| = n - k$
   B $\rightarrow A$: $\{b_i | i \in I\}$
   A: $\forall i \in I$: verify correctness of previously received $H(a_i + b_i), H(b_i)$

Fig. 2: Secrets setup phase. A and B exchange keys then engage in cut-and-choose. At the end of the protocol, the remaining set of size k indexed by $[n] \backslash I$ will later be used in the fair exchange.

```
<PK A2> CHECKSIGVERIFY
IF       // refund case
  <PK B2> CHECKSIG
ELSE       // claim case
  HASH
  DUP <H(b1)> EQUAL SWAP
  DUP <H(b2)> EQUAL SWAP
  <H(b3)> EQUAL
  BOOLOR BOOLOR
ENDIF
```

```
<PK B1> CHECKSIGVERIFY
IF       // refund case
  <PK A1> CHECKSIG
ELSE       // claim case
  HASH <H(a1 + b1)> EQUALVERIFY
  HASH <H(a2 + b2)> EQUALVERIFY
  HASH <H(a3 + b3)> EQUAL
ENDIF
```

Fig. 3: *On left:* Output script of TxComm_B. *On right:* Output script of TxComm_A.

claim TxComm_A. Suppose that $\overline{I}$ has size k. Through elementary probability analysis, we can show that a cheating B can succeed only with very small probability: $\Pr[B \text{ succeeds in cheating}] = 1/\binom{n}{k} \simeq 1/n^k$.

Transaction setup phase — Bob. Bob generates TxComm_B, using an output script that will allow 2 forms of redemption. The redemption can either be the refund transaction (dated in the future), with an input script signed by SK_{A2} and SK_{B2}, or the claim transaction with an input script signed by SK_{A2} and supplying any one of Bob's secrets from the set $\{b_i : i \in \overline{I}\}$. Figure 3 shows an example of TxComm_B's output script for set $\overline{I}$ of size $k = 3$.

Bob then generates a partial $\mathsf{TxRefund}_B$ releasing his money, with the locktime set to t+3 (timing in units of 'certain transaction confirmation time', an agreed number of block times, plus a buffer), with this incomplete input script: `<sig B2> 1`. Bob sends the partial $\mathsf{TxRefund}_B$ to party A, who verifies the locktime, and adds his signature to the input script `<sig B2> 1 <sig A2>`, and returns the completed refund transaction to Bob. Bob verifies the signed $\mathsf{TxRefund}_B$ and publishes TxComm_B and $\mathsf{TxRefund}_B$, and is now committed to the exchange.

Transaction setup phase — Alice. Alice waits until TxComm_B confirms, verifies $\mathsf{TxRefund}_B$ and checks the value. Alice generates TxComm_A, again using an output script that allows 2 forms of redemption. The first form enables the refund transaction, requiring signature by $\mathsf{SK}_{B1}$and SK_{A1}. The second form allows the claim transaction requiring signature by SK_{B1} and all of $\{a_i + b_i : i \in \overline{I}\}$. Figure 3 shows an example output script of TxComm_A, for a set $\overline{I}$ of size $k = 3$.

Then, Alice generates $\mathsf{TxRefund}_A$, with the locktime set to t+1, with the incomplete input script `<sig A1> 1` and with a standard output addressed to PK_{A1} (returning the money to herself). Alice sends this incomplete $\mathsf{TxRefund}_A$ to Bob. Bob verifies the locktime and adds his signature to the input script: `<sig A1> 1 <sig B1>`, and returns the now complete transaction to Alice. Alice verifies the returned $\mathsf{TxRefund}_A$

is unmodified and correctly signed. Alice now broadcasts TxComm_A and $\mathsf{TxRefund}_A$, and is now committed to the exchange.

Money claim phase. Bob waits for TxComm_A to confirm, and checks the amount is sufficient. Bob also needs to ensure he has enough time for his claim of Alice's money to confirm before $\mathsf{TxRefund}_A$'s time lock (hence the requirements on time locks). Now Bob claims Alice's money; he does this by taking $\mathsf{TxRefund}_A$ and modifying the time lock to "now" and the output to PK_{B1}. He also updates the input script to become (modified to include $\{a_i + b_i : i \in \overline{I}\}$), `<a3+b3> <a2+b2> <a1+b1> 0 <sig B1>`, thus creating $\mathsf{TxClaim}_B$. Bob now publishes $\mathsf{TxClaim}_B$. This claims Alice's money, while also revealing Alice's b_is for $i \in \overline{I}$. Now Alice can claim Bob's money by taking $\mathsf{TxRefund}_B$, removing the locktime, changing the output to PK_{A2}, and updating the input script to this form (modified to include one b_i from $\overline{I}$): `<b> 0 <sig A2>`. Alice earlier made sure that the locktime on $\mathsf{TxRefund}_B$ would give her sufficient time for this claim to confirm before the Bob's refund.

8 Conclusion

We have provided a preliminary but broad study of the crypto-monetary phenomenon Bitcoin, whose popularity has far overtaken the e-cash systems based on decades of research. Bitcoin's appeal lies in its simplicity, flexibility, and decentralization, making it easy to grasp but hard to subvert. We studied this curious contraption with a critical eye, trying to gauge its strengths and expose its flaws, suggesting solutions and research directions. Our conclusion is nuanced: while the instantiation is impaired by its poor parameters, the core design could support a robust decentralized currency if done right.

References

1. Bitcoin ewallet vanishes from internet. `www.tribbleagency.com/?p=8133`.
2. Bitcoin wiki: Contracts. `en.bitcoin.it/wiki/Contracts`.
3. Bitomat loses data and mybitcoin shuts down. `www.launch.is/blog`.
4. Deflationary spiral. `en.bitcoin.it/wiki/Deflationary_spiral`.
5. M. Abdalla, X. Boyen, C. Chevalier, and D. Pointcheval. Distributed public-key cryptography from weak secrets. *Proc. PKC*, 2009.
6. M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar. On bitcoin and red balloons. `research.microsoft.com/pubs/156072/bitcoin.pdf`, 2011.
7. X. Boyen. Halting password puzzles. *Proc. Usenix Security*, 2007.
8. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. *Proc. Eurocrypt*, 2005.
9. S. Canard and A. Gouget. Divisible e-cash systems can be truly anonymous. *Eurocrypt '07*.
10. D. Chaum. Blind signatures for untraceable payments. *Proc. Crypto*, 1982.
11. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 2007.
12. B. Laurie. Decentralised currencies are probably impossible but let's at least make them efficient. `www.links.org/files/decentralised-currencies.pdf`.
13. P. MacKenzie and M. Reiter. Two-party generation of DSA signatures. *Proc. Crypto*, 2001.
14. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `www.bitcoin.org`.
15. T. Okamoto. An efficient divisible electronic cash scheme. *Proc. Crypto*, 1995.
16. K. Poulsen. New malware steals your bitcoin. `wired.com/threatlevel/2011/06`.
17. F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. Arxiv:1107.4524.