Centrally Banked Cryptocurrencies

George Danezis
University College London
g.danezis@ucl.ac.uk

Sarah Meiklejohn
University College London
s.meiklejohn@ucl.ac.uk

May 26, 2015

Abstract

Current cryptocurrencies, starting with Bitcoin, build a decentralized blockchain-based transaction ledger, maintained through proofs-of-work that also generate a monetary supply. Such decentralization has benefits, such as independence from national political control, but also significant limitations in terms of scalability and computational cost. We introduce RSCoin, a cryptocurrency framework in which central banks maintain complete control over the monetary supply, but rely on a distributed set of authorities, or *mintettes*, to prevent double-spending. While monetary policy is centralized, RSCoin still provides strong transparency and auditability guarantees. We demonstrate, both theoretically and experimentally, the benefits of a modest degree of centralization, such as the elimination of wasteful hashing and a scalable system for avoiding double-spending attacks.

1 Introduction

Bitcoin [19], introduced in 2009, and the many alternative cryptocurrencies it inspired (e.g., Litecoin and Ripple), have achieved enormous success: financially, in May 2015, Bitcoin held a market capitalization of 3.4 billion USD and 30 cryptocurrencies held a market capitalization of over 1 million USD. In terms of visibility, cryptocurrencies have been accepted as a form of payment by an increasing number of international merchants, such as the 150,000 merchants using either Coinbase or Bitpay as a payment gateway provider.

Recently, major financial institutions such as JPMorgan Chase [22] and Nasdaq [20] have announced plans to develop blockchain technologies. The potential impacts of cryptocurrencies have now been acknowledged even by government institutions: the European Central Bank anticipates their "impact on monetary policy and price stability" [6], the US Federal Reserve their ability to provide a "faster, more secure and more efficient payment system" [4], and the UK Treasury vowed to "support innovation" [9] in this space. This is unsurprising since financial settlements systems that are currently in use by central banks (e.g., CHAPS, TARGET2, and Fedwire) remain relatively expensive and — at least behind the scenes — have high latency and are stagnant in terms of innovation.

Despite their success, existing cryptocurrencies suffer from a number of limitations. Arguably the most troubling one is their poor scalability: the Bitcoin network (currently by far the most heavily used) can handle at most 10 transactions per second and faces significant challenges in raising this rate much higher, whereas PayPal handles over 100 and Visa handles anywhere from 2,000 to 7,000. This lack of scalability is ultimately due to its reliance on broadcast and the need to expend significant computational energy in proofs-of-work — by one estimate [21], comparable to Ireland's electricity consumption — in order to manage the transaction ledger and make double-spending attacks prohibitively expensive. Alternative cryptocurrencies, such as Litecoin, try to distribute this cost and Permacoin [18] tries to put the computation cycles to good use, but ultimately neither of these solutions removes the costs. A second key limitation of current cryptocurrencies is the loss of control that central banks have over monetary supply, providing little flexibility for macroeconomic policy and extreme volatility in their value as currencies.

Against this backdrop, we present RSCoin, a cryptocurrency framework that decouples the generation of the monetary supply from the maintenance of the transaction ledger. Our design decisions were largely motivated by the desire to create a more scalable cryptocurrency, but were also inspired by the research agenda of the Bank of England [3], and the question of "whether central banks should themselves make use of such technology to issue digital currencies." Indeed, as Bitcoin becomes increasingly widespread, we expect that this is a question of interest to many central banks around the world.

RSCoin's radical shift from traditional cryptocurrencies is to centralize the monetary supply. Every unit of a particular currency is created by a particular central bank, making cryptocurrencies based on RSCoin significantly more palatable to governments. Despite this centralization, RSCoin still provides the benefit over

	CC	e-cash	Bitcoin	RSCoin
Double-spending	online	offline	online	online
Money generation	C	C	D	C
Ledger generation	C	n.a.	D	D*
Transparent	no	no	yes	yes
Pseudonymous	no	yes	yes	yes

Table 1: How existing approaches (credit cards, cryptographic e-cash, and Bitcoin) and how RSCoin compare in terms of the properties they provide. Double-spending refers to the way the system detects double-spending (i.e., as it happens or after the fact), and C stands for centralized and D for decentralized.

existing (non-crypto) currencies of a transparent transaction ledger, a distributed system for maintaining it, and a globally visible monetary supply. This makes monetary policy transparent, allows direct access to payments and value transfers, supports pseudonymity, and benefits from innovative uses of blockchains and digital money.

Centralization of monetary authority allows RSCoin to address some of the scalability issues of fully decentralized cryptocurrencies. In particular, as we describe in Section 3, the central bank delegates the authority of validating transactions to a number of other institutions that we call *mintettes* (following Laurie [15]). Since mintettes are — unlike traditional cryptocurrency miners — known and may ultimately be held accountable for any misbehavior, RSCoin supports a simple and fast mechanism for double-spending detection. As described in Section 4, we adapt a variant of Two-Phase Commit, optimized to ensure the integrity of a transaction ledger. Thus, we achieve a significantly more scalable system: the modest experimental testbed that we describe in Section 4.4.2 (consisting of only 10 mintettes running a basic Python implementation of our consensus mechanism), can process over 700 transactions per second, and performance scales linearly as we increase the number of mintettes. Most transactions take less than one second to clear, as compared to many minutes in traditional cryptocurrency designs. Without earning a mining reward, however, what incentivizes these mintettes to participate in transaction validation? We discuss in Section 4.3 how mintettes may earn fees by collecting transactions fees for good service. In a real deployment of RSCoin, we expect mintettes to be institutions with an existing relationship to the central bank, such as commercial banks, with some existing incentives to perform this service.

The ultimate goal for RSCoin is to achieve not only a scalable cryptocurrency that can be deployed and whose supply can be controlled by one central bank, but a framework that allows *any* central bank to deploy their own cryptocurrency. In fact, there is interest [2] to allow other entities to not only issue instruments that hold value (such as shares and derivative products), but to furthermore allow some visibility into transactions concerning them. With this in mind, we discuss in Section 5.3 what is needed to support some notion of interoperability between different deployments of RSCoin, how different currencies can be exchanged in a transparent and auditable way, and how various considerations — such as a pair of central banks that, for either security or geopolitical reasons, do not support each other — can be resolved without fragmenting the global monetary system. We also discuss other extensions and optimizations in Section 5.

2 Background

In this section, we present a brief background on Bitcoin and traditional cryptocurrencies. Since RSCoin adopts properties of other online payment systems, such as those of credit cards and cryptographic e-cash, we highlight some of the advantages and disadvantages of each of these approaches in Table 1.

Bitcoin is a decentralized cryptocurrency introduced in a whitepaper in 2008 [19] and deployed on 3 January 2009. Since then, Bitcoin has achieved success and has inspired a number of alternative cryptocurrencies (often dubbed "altcoins") that are largely based on the same blockchain technology. The novelty of this blockchain technology is that it fulfills the two key requirements of a currency — the generation of a monetary supply and the establishment of a transaction ledger — in a completely decentralized manner: a global peer-to-peer network serves to both generate new units of currency and to bear witness to the transfer of existing units from one party to another through transaction broadcast and computational proof-of-work protocols.

To highlight the differences between RSCoin and fully decentralized cryptocurrencies such as Bitcoin, we sketch the main operations and entities of these blockchain-based currencies; for a more comprehensive overview, we refer the reader to Bonneau et al. [5]. Briefly, users can generate signing keypairs and use the public key as a *pseudonym* or *address* in which to store some units of the underlying cryptocurrency. To transfer the value stored in this address to the address of another user, he creates a *transaction*, which is cryptographically signed using the secret key associated with this address. (More generally, transactions can transfer value from m input addresses to n output addresses, in which case the transaction must be signed by the secret keys associated with each of the input addresses.)

Once a user has created a transaction, it is broadcast to flood his peers in the network, eventually reaching *miners*. A miner eventually seals the transaction into the global ledger by including it in a pool of transactions, which she then hashes — along with some metadata and, crucially, a nonce — to attempt to produce a hash below a target value (defined by the difficulty of the network). Once a miner is successful in producing such a hash, she broadcasts the pool of transactions and associated hash as a *block*. Among the metadata for a block is a reference to the previously mined block, allowing the acceptance of the miner's block into the *blockchain* to be signaled by the broadcast of another block with a reference to hers (or, in practice, many subsequent blocks). Miners are incentivized by two rewards: the collection of optional fees in individual transactions, and a system-specific mining reward (e.g., as of May 2015, Bitcoin's mining reward of 25 BTC). These rewards are collected in a special *coin generation* transaction that the miner includes in her block's pool of transactions. Crucially, blocks serve to not only generate the monetary supply (via the mining rewards included in each block), but also to provide a global timestamp for the transactions contained within. This timestamping allows all users in the network to eventually impose a global ordering on transactions, and thus thwart double-spending by maintaining a list of *unspent transaction outputs* and validating a transaction only if its input addresses appear in this list.

What we have described above is the typical way of explaining Bitcoin at a high level, but we mention that in reality, bitcoins are not "stored" in an address or "sent"; instead, the sender relinquishes control by broadcasting a transaction that re-assigns to the recipient's address the bitcoins previously associated with that of the sender. An input to a transaction is thus not an address but a (signed) script that specifies an index in a previous transaction in which some bitcoins were received; this *address identifier* uniquely identifies one particular usage of an address, which becomes important as addresses are reused. In what follows, we thus frequently use the notation for an address and for a transaction-index pair interchangeably (and define this formally in Section 3.2).

3 The RSCOIN System

In this section, we describe the structure of RSCoin, focusing on the interaction between the main entities required to produce both the monetary supply and the transaction ledger, and how users can interact with these entities to transfer value amongst themselves.

3.1 Overview

At a high level, RSCoin introduces a degree of centralization into the two typically decentralized components of a blockchain-based ledger: the generation of the monetary supply and the construction of the transaction ledger. In its simplest form, the RSCoin system assumes two structural entities: the *central bank*, a centralized entity that ultimately has complete control over the generation of the monetary supply, and a distributed set of *mintettes* (following Laurie [15]) that are responsible for the maintenance of the transaction ledger. The interplay between these entities — and an overview of RSCoin as a whole — can be seen in Figure 1.

Briefly, mintettes collect transactions from users and collate them into blocks, much as is done with traditional cryptocurrencies. These mintettes differ from traditional cryptocurrency miners, however, in a crucial way: rather than performing some computationally difficult task, each mintette is simply authorized by the central bank to collect transactions. In RSCoin, this authorization is accomplished by a PKI-type functionality, meaning the central bank signs the public key of the mintette, and each lower-level block must contain one of these signatures in order to be considered valid. We refer to the time interval in which blocks are produced by mintettes as an *epoch*, where the length of an epoch varies depending on the mintette. Because these blocks are not ultimately incorporated into the main blockchain, we refer to them as *lower-level blocks*. Mintettes are collectively responsible for producing a consistent ledger, and thus to facilitate this process they communicate internally throughout the course of an epoch — in a manner described in Section 4 — and ultimately reference not only their own previous blocks but also the previous blocks of each other. This means that these lower-level blocks form a (potentially) *cross-referenced* chain.

At the end of some longer pre-defined time interval called a *period*, the mintettes present their blocks to the central bank, which merges these lower-level blocks to form a consistent history in the form of a new block. This *higher-level block* is what is ultimately incorporated into the main blockchain, meaning a user of RSCoin need only keep track of higher-level blocks. (Special users wishing to audit the behavior of mintettes, however, may keep track of lower-level blocks, and we describe in Section 4.3 ways to augment lower-level

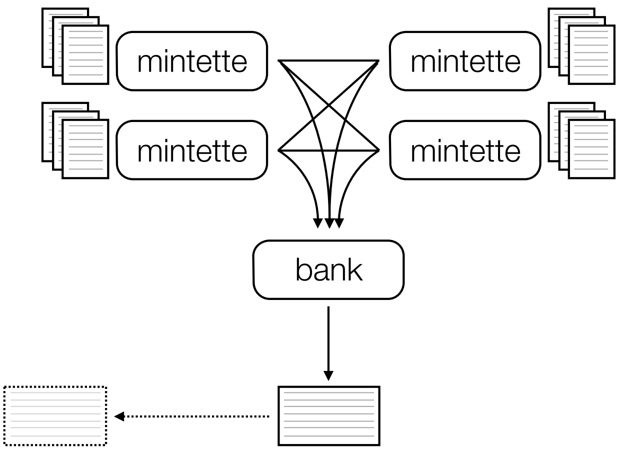


Figure 1: The network structure of RSCoin. Each mintette maintains a set of lower-level blocks, and (possibly) communicates with other mintettes (either directly or indirectly). At some point, the mintettes send these blocks to the central bank, which produces a higher-level block. It is these higher-level blocks that form a chain.

blocks to improve auditability.) Interaction with RSCoin can thus be quite similar to interaction with existing cryptocurrencies, as the structure of its blockchain is nearly identical, and users can create new pseudonyms and transactions in the same ways as before.

Below, we describe in more detail the structure and usage of RSCoin (Sections 3.3 and 3.4) and address considerations that arise in how to overlay RSCoin on top of an existing cryptocurrency like Bitcoin (Section 3.5); incentivize mintettes to present a collectively consistent ledger to the central bank (Section 3.6); and set concrete choices for various system parameters (Section 3.7).

3.2 Notation

We denote a hash function as $H(\cdot)$ and a signature scheme as the tuple $(\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$, where these algorithms behave as follows: via $(pk, sk) \overset{\$}{\leftarrow} \mathsf{Sig.KeyGen}(1^\lambda)$ one generates a signing keypair; via $\sigma \overset{\$}{\leftarrow} \mathsf{Sig.Sign}(sk, m)$ one generates a signature; and via $0/1 \leftarrow \mathsf{Sig.Verify}(pk, m, \sigma)$ one verifies a signature on a message.

We use addr to denote an address; this is identical to a public key pk in terms of the underlying technology[1], but we use the separate term to disambiguate between usage in a transaction (where we use addr) and usage as a signing key (where we use pk). We use $\mathsf{tx}(\{\mathsf{addr}_i\}_i \xrightarrow{n} \{\mathsf{addr}_j\}_j)$ to denote a transaction in which n units of currency are sent from $\{\mathsf{addr}_i\}_i$ to $\{\mathsf{addr}_j\}_j$. Each usage of an address addr can be uniquely identified by the tuple $\mathsf{addrid} = (\mathsf{tx}, \mathsf{index}_{\mathsf{tx}}(\mathsf{addr}), v)$, where tx is the hash of the transaction in which it received some value v, and $\mathsf{index}_{\mathsf{tx}}(\mathsf{addr})$ is the index of addr in the list of outputs. When we use these *address identifiers* later on, we occasionally omit information (e.g., the value v) if it is already implicit or unnecessary. We use b to denote a lower-level block, and $\mathsf{b}_j^{(\mathsf{m})}$ to denote the lower-level block produced by a mintette m within $\mathsf{epoch}_{j,\mathsf{m}}$. Finally, we use B to denote a higher-level block, and $\mathsf{B}_i^{(\mathsf{bank})}$ to denote the higher-level block produced by a central bank bank at the end of the i-th period period_i.

3.3 Higher-level blocks

The higher-level block that marks the end of period_i looks like $\mathsf{B}_{\mathsf{bank}}^{(i)} = (h_i, \mathsf{txset}_i, \sigma_i)$, where h_i is a hash, txset_i is a collection of transactions, and σ_i is a signature from the central bank bank (associated with some public key pk_{bank}) that produced this block. To check that a given block is valid, one checks that (1) $h_i = H(h_{i-1} \| \mathsf{txset}_i)$ (i.e., the block fits into the existing chain for this central bank) and (2) $\mathsf{Sig.Verify}(pk_{\mathsf{bank}}, h_i, \sigma_i) = 1$ (i.e., the block has been produced by bank).

By convention, each higher-level block B_i could contain a special coin generation transaction, in which the central bank generates new units of currency, and — using a mechanism described in Section 4.3.2 — an allocation of fees to the mintettes that earned them. In addition, B_i *must* specify the mintettes authorized for

---

[1] Or, as in the case of Bitcoin, it may be some hashed version of the public key.

the next period period_{i+1}. This involves generating a new period-specific keypair $(pk_{\mathsf{bank}}^{(i+1)}, sk_{\mathsf{bank}}^{(i+1)})$ and using it to sign the public key pk_{m} of each mintette in some set DPK_{i+1}; i.e., the set of mintettes authorized for period_{i+1}. We discuss how to embed these special operations into the blockchain in Section 3.5.

3.4 Lower-level blocks

A lower-level block produced by a mintette m within period_i looks like $\mathsf{b} = (h, \mathsf{txset}, \sigma, \sigma_{\mathsf{bank}}^{(\mathsf{m})}, \mathsf{mset})$, where the first three components are the same as in a higher-level block (a hash, a collection of transactions, and a signature), and the fourth component $\sigma_{\mathsf{bank}}^{(\mathsf{m})}$ is a signature from the bank specifying that m — identified using its public key pk_{m} — has been authorized for period_i. Finally, recall from the overview in Section 3.1 that lower-level blocks from different mintettes can reference each other. The fifth component mset thus specifies this chain property by identifying the hashes of the other previous blocks that are being referenced (in addition to the mintette's own previous block).

Denote by $pk_{\mathsf{bank}}^{(i)}$ the public key and by DPK_i the set of authorized mintettes specified by the bank in $\mathsf{B}_{\mathsf{bank}}^{(i-1)}$ (as described in Section 3.3). Assuming the block b is produced in epoch_j, define $\mathsf{otherblocks} = h_1 \| \ldots \| h_n$ for $\mathsf{mset} = (h_1, \ldots, h_n)$. We then require that

1. $h = H(\mathsf{B}_{\mathsf{bank}}^{(i-1)} \| \mathsf{b}_j^{(\mathsf{m})} \| \mathsf{otherblocks} \| \mathsf{txset})$,
2. $\mathsf{Sig.Verify}(pk_{\mathsf{m}}, h, \sigma) = 1$,
3. $pk_{\mathsf{m}} \in \mathsf{DPK}_i$, and
4. $\mathsf{Sig.Verify}(pk_{\mathsf{bank}}^{(i)}, pk_{\mathsf{m}}, \sigma_{\mathsf{bank}}^{(\mathsf{m})}) = 1$.

As mentioned in the overview, we describe in Section 4.3 ways to augment lower-level blocks with additional information that makes it possible to audit the behavior of mintettes. (In fact, these augmented versions are the only reason we need lower-level blocks at all; otherwise mintettes could simply send their transactions to the central bank at the end of the period.) As the details of how blocks are augmented rely on the details of our consensus mechanism, we defer all further discussion to Section 4.3.

3.5 Embedding special transactions

Recall from Section 3.3 that there are four main operations the central bank needs to embed in the blockchain at the end of a period: (1) the (optional) generation of new units of currency, (2) the indication of the public key $pk_{\mathsf{bank}}^{(i+1)}$ to be used in the next period, (3) the authorization of mintettes DPK_{i+1}, and (4) an allocation of transaction fees to the mintettes that participated in their processing during the period. (As we see in Section 4.3.2, different mintettes will have earned fees for different transactions, and only the central bank can allocate fees appropriately.)

As RSCoin is intended as a framework rather than a stand-alone cryptocurrency, there are multiple ways to consider embedding these operations into the blockchain. The simplest way is to define a new cryptocurrency in which these semantics are included in the specification of transactions and blocks; i.e., in which there are special transaction types for each of the operations and blocks are expected to have the structures defined above.

On the other end of the spectrum, we consider how to embed these semantics into an existing cryptocurrency like Bitcoin, to allow an instantiation of RSCoin as an overlay — albeit one that would require a fork of the network — rather than a separate architecture. Operations (1) and (4) are relatively simple, as coin generation transactions already exist, and include the collection of fees. To thus generate n new units of currency, the bank could include a coin generation transaction $\mathsf{tx}(\emptyset \xrightarrow{n} \mathsf{addr}_{\mathsf{bank}})$, where $\mathsf{addr}_{\mathsf{bank}}$ is an address owned by the bank. Similarly, to achieve operation (4), the bank could include a coin generation transaction $\mathsf{tx}(\emptyset \xrightarrow{f} \mathsf{addr}_{\mathsf{m}})$, where f represents the fees owed to m. The only potential limitation of this approach is that coin generation transactions are not signed in Bitcoin, so — if necessary, as in practice only the central bank can produce higher-level blocks anyway — coin generation transactions could be modified to require a signature from the central bank. To achieve operation (2), the bank could store some units of currency in a master address $\mathsf{addr}_{\mathsf{bank}}$ and include in txset_i a transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{bank}} \xrightarrow{n_{pk}} pk_{\mathsf{bank}}^{(i+1)})$. If other transactions are expected from $\mathsf{addr}_{\mathsf{bank}}$ to $pk_{\mathsf{bank}}^{(i+1)}$, then n_{pk} could take on some special semantic value, but otherwise it can be arbitrary. Similarly, to achieve operation (3), the bank could include in txset_i a transaction $\mathsf{tx}(pk_{\mathsf{bank}}^{(i+1)} \xrightarrow{n_{\mathsf{m}}} pk_{\mathsf{m}})$ for each mintette m authorized for period_{i+1}, where again if necessary n_{m} could be a value of some special semantic significance. As we expect that $pk_{\mathsf{bank}}^{(i+1)}$ will not be used again, we expect that $n_{pk} = |\mathsf{DPK}_{i+1}| \cdot n_{\mathsf{m}}$, so that the units of currency can be moved into $pk_{\mathsf{bank}}^{(i+1)}$ and then moved out again immediately, resulting in a balance of zero.

If transactions are embedded in this manner into blocks, then some of the values that were stated as explicit components of blocks can now be made implicit or removed entirely. For example, the signature $\sigma_{\mathsf{bank}}^{(\mathsf{m})}$ in a lower-level block b produced in period_{i+1} can be retrieved from B_i as the transaction described for operation (3) above, and thus does not need to be included.

3.6 Incentivizing mintettes

One might naturally imagine that this structure, as currently described, places the significant burden on the central bank of having to merge the distinct blocks from each mintette into a consistent history. By providing appropriate incentives, however, we can create an environment in which mintettes communicate with each before providing their blocks to the central bank, so that ultimately the presented ledger is consistent before the bank even sees it. If mintettes deviate from the expected behavior then, as we describe in Section 4.3, they can be held accountable and punished accordingly (e.g., not chosen for future periods or not given any fees they have earned).

Section 3.3 describes one direct incentive for mintettes to collect transactions, which is fees. While these are currently quite low in most existing cryptocurrencies, they are expected to rise as block rewards decrease — in cryptocurrencies that follow this model, such as Bitcoin — and can be increased by consensus at any point in time; e.g., miners could collectively agree to stop accepting transactions that don't include a sufficient fee. As we describe in Section 4.3.2, mintettes are rewarded only for *active* participation, so that an authorized mintette still needs to engage with the system in order to earn fees. Section 3.5 describes another direct incentive, which is the authorization of mintettes by the central bank. For semantic purposes, the value n_{m} used to authorize each mintette for the next period could be arbitrarily small. As an incentive, however, this value could be larger to directly compensate the mintettes for their services.

Finally, we expect that the central bank could be a national or international entity with existing relationships with, e.g., commercial banks. There thus already exist strong business incentives and regulatory frameworks for such entities to act as honest mintettes.

3.7 Setting system parameters

As described, the system is parameterized by a number of variables, such as the length of epochs, the length of a period, and the number of mintettes. Most of these parameters are determined by the specifics of our consensus mechanism, so we defer a detailed discussion to Section 4, but nevertheless provide a hint of these choices here.

The length of an epoch for an individual mintette is entirely dependent on its interaction with other mintettes (as described in detail in Section 4.3). Mintettes that interact more frequently with other mintettes will therefore have shorter epochs than ones that interact less frequently. There is no limit on how short an epoch can be, and the only upper limit is that an epoch cannot last longer than a period.

It might seem desirable for periods to be as short as possible, as ultimately a transaction is sealed into the official ledger only at the end of a period. To ease the burden on the bank, however, it is also desirable to have longer periods, so that central banks have to intervene as infrequently as possible (and, as we describe in Section 5.1, so that central banks can potentially perform certain optimizations to reduce transaction bloat). In Section 4.2 we describe methods by which mintettes can "promise" (in an accountable way) to users that their transactions will be included, so that in practice near-instantaneous settlement can be achieved even with longer periods, so long as one trusts the mintette. (This is akin to SPV clients in Bitcoin, which replace the need to download the entire blockchain with trust in the difficulty of the mining process.) Nevertheless, we do not expect periods to last longer than a day.

For the purposes of having a fair and competitive settlement process, it is desirable to have as many mintettes as possible; as we see in Section 4.4.1, this is also true from a performance perspective, as the performance of the RSCoin system (measured in the rate of transactions processed) scales linearly with the number of mintettes. Adding more mintettes, however, also has the effect that they earn less in transaction fees, so these opposing concerns must be taken into account when settling on a concrete number (to give a very rough idea, one number that has been suggested [2] is 200).

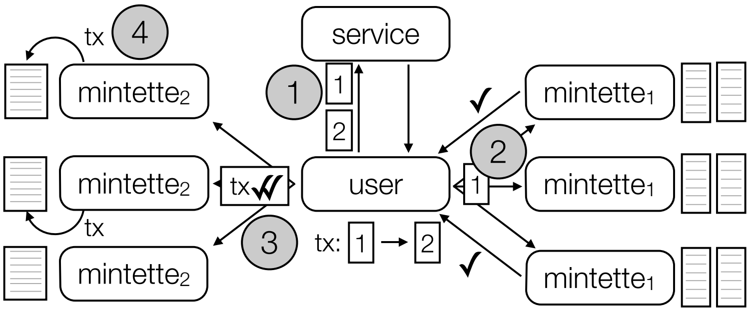


Figure 2: The proposed protocol for validating transactions; each mintette m_i is an owner of address i. In (1), a user learns the owners of each of the addresses in its transaction. In (2), the user collects approval from a majority of the owners of the input addresses. In (3), the user sends the transaction and these approvals to the owners of the transaction identifier. In (4), some subset of these mintettes add the transaction to their blocks.

4 Achieving Consensus

In Section 3.6, we outlined various reasons that mintettes are motivated to be not only individually honest in collecting transactions, but also to collectively agree on a consistent history to ensure the minimal necessary intervention by the central bank at the end of a period. In this section, we describe *how* the mintettes can come to this consensus.

As described in the introduction, one of the major benefits of centralization is that, although the generation of the transaction ledger is still distributed, consensus can be reached in a way that avoids the wasteful proofs-of-work required by existing cryptocurrencies. In traditional cryptocurrencies, the set of miners are neither known nor trusted, meaning one has no choice but to broadcast a transaction to the entire network and rely on proof-of-work to defend against Sybil attacks. Since our mintettes are in fact authorized by the central bank, and thus both known and — because of their accountability — trusted to some extent, we can avoid the heavyweight consensus requirement of more fully decentralized cryptocurrencies and instead use an adapted version of Two-Phase Commit (2PC), as presented in Figure 2.

4.1 Threat model and security

We always assume that the central bank is honest and that the underlying cryptography is secure; i.e., no parties may violate the standard properties offered by the hash function and digital signature. Honest mintettes follow the protocols as specified, whereas dishonest mintettes may behave arbitrarily; i.e., they may deviate from the prescribed protocols, and selectively or broadly ignore requests from users. Finally, honest users create only valid transactions (i.e., ones in which they own the input addresses and have not yet spent their contents), whereas dishonest users may try to double-spend or otherwise subvert the integrity of RSCoin.

We consider two threat models. Our first threat model assumes that each transaction is processed by a set of mintettes with an honest majority; this is different from assuming that a majority of all mintettes are honest, as we will see in our description of transaction processing in Section 4.2. Our second threat model assumes that no mintette is honest, and that mintettes may further collude to violate the integrity of RSCoin. This is a very hostile setting, but we show that some security properties still hold for honest users. Additionally, we show that mintettes that misbehave in certain ways can be detected and ultimately held accountable, which may serve as an incentive to follow the protocols correctly.

In the face of these different adversarial settings, we try to satisfy at least some of the following key integrity properties:

- **No double-spending**: Each output address of a valid transaction will only ever be associated with the input of at most one other valid transaction.
- **Non-repudiable sealing**: The confirmation that a user receives from a mintette — which promises that a transaction will be included in the ledger — can be used to implicate that mintette if the transaction does not appear in the next block.
- **Timed personal audits**: A user can, given access to the lower-level blocks produced within a period, ensure that the implied behavior of a mintette matches the behavior observed at the time of any previous interactions with that mintette.
- **Universal audits**: Anyone with access to the lower-level blocks produced within a period can audit

all transactions processed by all mintettes. In particular, mintettes cannot retroactively modify, omit, or insert transactions in the ledger.

- **Exposed inactivity**: Anyone with access to the lower-level blocks produced within a period can observe any mintette's substantial absence from participation in the 2PC protocol. (In particular, then, a mintette cannot retroactively act to claim transaction fees for services not provided in a timely manner.)

To see how to satisfy these security properties, we first present our basic consensus protocol in Section 4.2, and then present in Section 4.3 ways to augment this protocol to achieve auditability. We then prove in Section 4.3.1 that at least some subset of these security properties can be captured in both our threat models, and that exposure may disincentive mintettes from violating those that we cannot capture directly.

4.2 A basic consensus protocol

To begin, the mintettes divide up the space of possible transaction identifiers, so that each mintette m is responsible for some subset, or "shard." For reliability and security, each shard is covered by (potentially) multiple mintettes, and everyone is aware of the owner(s) of each. We use $\mathsf{owners}(\mathsf{addrid})$ to denote the set of mintettes responsible for addrid. Recall that $\mathsf{addrid} = (\mathsf{tx}, i, v)$, where tx specifies the transaction in which addr received value v, so that the same set of mintettes will own all of the output addresses in a transaction. For simplicity, we therefore use the notation $\mathsf{owners}(S_{\mathsf{out}})$ below (where S_{out} is the list of output addresses for a transaction).

In each period, each mintette m is responsible for maintaining two lists: a list of unspent transaction outputs, denoted utxo, and two lists of transactions seen thus far in the period, denoted pset and txset respectively (the former is used to detect double-spending, and the latter is used to seal transactions into the ledger). The utxo list is of the form $\mathsf{addrid} \mapsto (\mathsf{addr}, v)$, where $(\mathsf{addrid} \mapsto (\mathsf{addr}, v)) \in \mathsf{utxo}$ indicates that addrid had not acted as an input address at the start of the period but has since sent value v to addr and $(\mathsf{addrid} \mapsto (\bot, \bot)) \in \mathsf{utxo}$ indicates that addrid has not yet spent its contents. The pset list is of the form $\mathsf{addrid} \mapsto \mathsf{tx}$, where $(\mathsf{addrid} \mapsto \mathsf{tx}) \in \mathsf{pset}$ indicates that addrid has acted as an input address in transaction tx. We assume that each mintette starts the period with an accurate utxo list (i.e., all transactions within the mintette's shard in which the outputs have not yet been spent) and with an empty pset.

At some point in the period, a user creates a transaction. The user[2] can now run Algorithm 4.1.

Algorithm 4.1: Validating a transaction, run by a user

```
Input: a transaction $\mathsf{tx}(S_{\mathsf{in}} \xrightarrow{n} S_{\mathsf{out}})$ and period identifier $j$
bundle ← ∅
//first phase:  collect votes
forall the $\mathsf{addrid} \in S_{\mathsf{in}}$ do
    $M \leftarrow \mathsf{owners}(\mathsf{addrid})$
    forall the $\mathsf{m} \in M$ do
        $(pk_{\mathsf{m}}, \sigma) \leftarrow \mathsf{CheckNotDoubleSpent}(\mathsf{tx}, \mathsf{addrid}, \mathsf{m})$
        if $(pk_{\mathsf{m}}, \sigma) = \bot$ then
            return $\bot$
        else
            $\mathsf{bundle} \leftarrow \mathsf{bundle} \cup \{((\mathsf{m}, \mathsf{addrid}) \mapsto (pk_{\mathsf{m}}, \sigma))\}$
//second phase:  commit
$M \leftarrow \mathsf{owners}(S_{\mathsf{out}})$
forall the $\mathsf{m} \in M$ do
    $(pk_{\mathsf{m}}, \sigma) \leftarrow \mathsf{CommitTx}(\mathsf{tx}, j, \mathsf{bundle}, \mathsf{m})$
```

In the first phase, the user asks the relevant mintettes to "vote" on the transaction; i.e., to decide if its input addresses have not already been used, and thus certify that no double-spending is taking place. To do this, the user can compute the owners for each input address, and send the transaction information to these mintettes, who each run Algorithm 4.2. We omit for simplicity the formal description of an algorithm $\mathsf{CheckTx}$ that, on input a transaction, checks that the basic structure of the transaction is valid; i.e., that the collective input value is at least equal to the collective output value, that the input address identifiers point to valid previous transactions, and that the signatures authorizing previous transaction outputs to be spent are valid.

---

[2] We refer to the user here and in the sequel, but in practice this can all be done by the underlying client, without any need for input from the (human) user.

Algorithm 4.2: CheckNotDoubleSpent, run by a mintette

```
Input: a transaction $\mathsf{tx}_c$, an address identifier $\mathsf{addrid} = (\mathsf{tx}, i)$ and a mintette identifier $\mathsf{m}$
if $\mathsf{CheckTx}(\mathsf{tx}_c) = 0$ or $\mathsf{m} \notin \mathsf{owners}(\mathsf{addrid})$ then
    return $\perp$
else
    if $(\mathsf{addrid} \in \mathsf{utxo}_\mathsf{m})$ or $((\mathsf{addrid} \mapsto \mathsf{tx}_c) \in \mathsf{pset}_\mathsf{m})$ then
        $\mathsf{utxo}_\mathsf{m} \leftarrow \mathsf{utxo}_\mathsf{m} \setminus \{\mathsf{addrid}\}$
        $\mathsf{pset}_\mathsf{m} \leftarrow \mathsf{pset}_\mathsf{m} \cup \{(\mathsf{addrid} \mapsto \mathsf{tx}_c)\}$
        return $(pk_\mathsf{m}, \mathsf{Sig.Sign}(sk_\mathsf{m}, (\mathsf{tx}_c, \mathsf{addrid})))$
    else
        return $\perp$
```

Briefly, in Algorithm 4.2 the mintette first checks if the current transaction is valid and if the input address is within its remit, and returns $\perp$ otherwise. The mintette then proceeds if the address identifier has either not been spent before (and thus is in utxo), or if the address identifier has already been associated with the given transaction (and thus the pair is in pset). In those cases, it removes the address identifier from utxo and associates it with the transaction in the period transaction set pset; these actions are idempotent and can be safely performed more than once. The mintette then returns a signed acknowledgement to the user. If instead another transaction appears in pset associated with the address identifier, then the address is acting as an input in two different transactions — i.e., it is double-spending — and the mintette returns $\perp$.

At the end of the first phase, an honest user thus will have received some signatures (representing 'yes' votes) from the owners of the input addresses of the new transaction. Users should check the signatures returned by these mintettes and immediately return a failure if any is invalid. Once the user has received signatures from the majority of owners for each input, she can now send the transaction, coupled with a "bundle of evidence" (consisting of the signatures of the input mintettes) to represent its validity, to the owners of the output addresses (who, recall, are the same for all output addresses). These mintettes then run Algorithm 4.3.

Algorithm 4.3: CommitTx, run by a mintette

```
Input: a transaction $\mathsf{tx}(S_\mathsf{in} \xrightarrow{n} S_\mathsf{out})$, a period identifier $j$, a bundle of evidence
       $\mathsf{bundle} = \{((\mathsf{m}_i, \mathsf{addrid}_i) \mapsto (pk_i, \sigma_i))\}_i$, and a mintette identifier $\mathsf{m}$
if $\mathsf{CheckTx}(\mathsf{tx}) = 0$ or $\mathsf{m} \notin \mathsf{owners}(S_\mathsf{out})$ then
    return $\perp$
else
    $d \leftarrow 1$
    forall the $\mathsf{addrid} \in S_\mathsf{in}$ do
        forall the $\mathsf{m}' \in \mathsf{owners}(\mathsf{addrid})$ do
            if $(\mathsf{m}', \mathsf{addrid}) \in \mathsf{bundle}$ then
                $(pk, \sigma) \leftarrow \mathsf{bundle}[(\mathsf{m}', \mathsf{addrid})]$
                $d' \leftarrow d \wedge H(pk) \in \mathsf{DPK}_j$
                        $\wedge\ \mathsf{Sig.Verify}(pk, (\mathsf{tx}, \mathsf{addrid}), \sigma)$
            else
                $d \leftarrow 0$
    if $d = 0$ then
        return $\perp$
    else
        $\mathsf{utxo}_\mathsf{m} \leftarrow \mathsf{utxo}_\mathsf{m} \cup S_\mathsf{out}$
        $\mathsf{txset}_\mathsf{m} \leftarrow \mathsf{txset}_\mathsf{m} \cup \{\mathsf{tx}\}$
        return $(pk_\mathsf{m}, \mathsf{Sig.Sign}(sk_\mathsf{m}, \mathsf{tx}))$
```

In Algorithm 4.3, a mintette first checks the transaction and whether it falls within its remit. The mintette then checks the bundle of evidence by verifying that all — or, in practice, a majority — of mintettes associated with each input are all included, that the input mintettes were authorized to act as mintettes in the current period, and that their signatures verify. If these checks pass and the transaction has not been seen before, then the mintette adds all the output addresses for the transaction to its utxo list and adds the transaction to txset. The mintette then sends to the user evidence that the transaction will be included in the higher-level block (and a user can use this evidence to implicate the mintette if this is not the case).

At the end of the period, all mintettes send txset to the central bank, along with additional information in order to achieve integrity, which we discuss in the next section.

Security In our first threat model, where all transactions are processed by a set of mintettes with honest majority, it is clear that (1) no double-spending transactions will be accepted into txset by honest mintettes, and (2) the confirmation given to a user in Line 17 of Algorithm 4.3 can be wielded by the user as evidence that the mintette promised to seal the transaction. Thus, in our first threat model — in which all transactions are processed by a set of mintettes with honest majority — the first and second integrity properties in Section 4.1 are already satisfied by our basic consensus protocol.

4.3 Achieving auditability

While our basic consensus mechanism already achieves some of our desired integrity properties (at least in our weaker threat model), it is still not clear that it provides any stronger notions of integrity, or that it provides any integrity in a more hostile environment. To address this limitation, we present in this section a way to augment both the lower-level blocks discussed in Section 3.4 and the basic consensus mechanism. At a high level, a mintette now maintains a high-integrity log that highlights both its own key actions, as well as the actions of those mintettes with whom it has indirectly interacted (i.e., from whom it has received signatures, ferried through the user, in the process of committing a transaction).

In more detail, each mintette maintains a log of absolutely ordered actions along with their notional sequence number. Actions may have one of three types: Query, Commit and CloseEpoch. The Query action signals an update to pset as a result of an input address being assigned to a new transaction (Line 6 of Algorithm 4.2), so for this action the log includes the new transaction. The Commit action signals an update to utxo and txset as a result of receiving a new valid transaction (lines 15 and 16 of Algorithm 4.3, respectively), so for this action the log includes the transaction and its corresponding bundle of evidence.

To facilitate the CloseEpoch action, each mintette stores not only the log itself but also a rolling hash chain; i.e., a *head* that acts as a witness to the current state of the log, so $h_{seq} = H(a_{seq} \| h_{seq-1})$, where a_{seq} is the log entry of the action and h_{seq-1} is the previous head of the chain.

To share this witness, mintettes include a signed head in every message they emit; i.e., in line 7 of Algorithm 4.2 and line 17 of Algorithm 4.3, the mintette m computes $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(sk_{\mathsf{m}}, (\mathsf{tx}_c, \mathsf{addrid}, h, seq)$ (where h is the head of its chain) rather than $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(sk_{\mathsf{m}}, (\mathsf{tx}_c, \mathsf{addrid}))$, and outputs $(pk_{\mathsf{m}}, \sigma, h, seq)$. Now that mintettes are potentially aware of each others' logs, the CloseEpoch action — which, appropriately, marks the end of an epoch — includes in the log the heads of the other chains of which the mintette is aware, along with their sequence number. This results in the head of each mintette's chain depending on the latest known head of both its own and other chains; we refer to this phenomenon as *cross-hashing* (which, in effect, implements a cryptographic variant of vector clocks [24]).

4.3.1 Security

Lemma 4.1. *In both of our threat models, the augmented consensus protocol outlined above provides timed personal audits, universal audits, and exposed inactivity (as defined in Section 4.1).*

Proof. (Informal.) To prove that our protocol provides timed personal audits, observe that if the log reported by any mintette (or equivalently its hash at any log position) forks at any point from the record of a user or other mintette, then the signed head of the hash chain serves as evidence that the log is different. To remain undetected, the mintette must therefore provide users with the signed head of a hash chain that is a prefix of the actual hash chain it will report. Both the Query and Commit messages leading to a signed hash, however, modify the action log. Providing an outdated hash thus would not contain the latest action, so again there is evidence that such an action should have been recorded (in the form of the signed response to the message that should prompt the action), which also incriminates the mintette. Thus a mintette that does not wish to be detected and incriminated may only refrain from responding to requests requiring actions that would change its log.

To prove that our protocol provides universal audits and exposed inactivity, we first note that, despite the lack of synchronization between mintettes within periods, we can detect when an action is committed to a mintette log a 'significant time' after another action. This is due to the fact that the second message of the 2PC protocol that users send to mintettes carries the hash heads from all input mintettes involved. This forms a low-degree random graph with good expansion properties, and we expect that in a short amount of time mintettes will have hash chains dependent on the hash chains of all other mintettes. Thus, if two actions are separated by a sufficiently long period of time, it is extremely likely that a head dependent on the first action has propagated to a super-majority of other mintettes. Checking this property allows us to detect which came first with very high probability. Using this observation, everyone may audit claims that a mintette contributed to an action (e.g., processing the first query of the 2PC protocol for a valid transaction) in a timely fashion, by using the process above to detect whether the claimed action from the mintette is or is not very likely to have come after the same action was committed by all other mintettes concerned. □

4.3.2 The role of the bank

Using this augmented protocol, at the end of the period the central bank receives from each mintette not only the set of transactions that it wants to include, but also a log detailing its actions throughout the period. Using the set of transactions, the bank can merge them and publish the set in the resulting higher-level block. If double-spending does occur, then the bank can — using the sets of transactions and, if necessary, the logs — identify the mintette(s) responsible, punish them accordingly, and remove the conflicting transactions from the set before publishing the block.

Using the provided logs, we also present a mechanism to allow active and live mintettes to be remunerated through transaction fees. Recall from Section 3.6 that these fees serve as incentive for honest work done in support of the double-spending prevention protocol, and that mechanisms to collect fees are already present in existing cryptocurrencies such as Bitcoin (albeit not widely used).

The central bank keeps a tally of the mintettes that were involved in certifying the validity of input addresses; i.e., those that replied to the first phase of the 2PC protocol. The central bank then creates transactions that distribute the fees to specific mintettes, according to their contribution, and includes them in the higher-level block. The choice to reward input mintettes is deliberate: in addition to providing a direct incentive for mintettes to respond in the first phase of the protocol, it also provides an indirect incentive for mintettes to respond in the second phase, as only a transaction output that is marked as unspent can later be used as an input (for which the mintette can then earn fees). Thus, rewarding input mintettes provides incentive to handle a transaction throughout its lifetime.

The action logs play a crucial role in fee allocation. In particular, the "exposed inactivity" security property prevents an inactive mintette from becoming active at a later time and claiming that it contributed to previous transactions, as an examination of the action logs can falsify such claims. Additionally, if fee allocation is determined on the basis of a known function of the action logs, anyone with access to the action logs can audit the actions of the central bank.

Finally, we mention that although the logs are sent only to the central bank, the expectation is that the central bank will publish these logs to allow anyone to audit the system. As we assume the central bank is honest, this does not present a problem, but in a stronger threat model in which less trust were placed in the central bank, one might instead attempt to adopt a broadcast system for distributing logs (with the caveat that this approach introduces significantly higher latency).

4.4 Performance

4.4.1 Theoretical analysis

Looking back at the algorithms in Section 4.2, we can get at least a theoretical estimate of the communication and computational complexity of the system. Denote by T the set of transactions that are generated per second; by Q the number of mintettes that own each address; and by M the number of total mintettes.

For a transaction with m inputs and n outputs, a user sends and receives at most mQ messages in the first phase of the 2PC protocol (line 5 of Algorithm 4.1) and sends and receives at most Q messages in the second phase (line 12). For the user, each transaction thus requires at most $2(m+1)Q$ messages.

In terms of the communication complexity per mintette, we assume that each mintette receives a proportional share of the total transactions, which is ensured as the volume of transactions grow, by the bank allocating shards of equal sizes to all mintettes. Then the work per mintette is

$$\frac{\sum_{\mathsf{tx}\in T} 2(m_{\mathsf{tx}}+1)Q}{M}.$$

In particular, this scales *infinitely*: as more mintettes are added to the system, the work per mintette decreases (in a linear fashion) and eventually goes to zero.

4.4.2 Experimental analysis

To verify these performance estimates and to measure the latency a typical user would experience to confirm a transaction, we implemented the basic consensus mechanism presented in Section 4.2 and measured

Benchmark	μ (s^{-1})	σ
Hash	1,090,119.5	35,761.03
Sign	6718.97	106.77
Verify	4726.67	53.41
Check tx	3671.63	54.7
Query msg	1078.06	62.4
Commit msg	884.29	19.57

Table 2: Micro-benchmarks at the mintettes

its performance on a modest cluster hosted on Amazon's Elastic Compute (EC2) infrastructure. Our implementation consists of 2458 lines of Python code: 1109 lines define the core transaction structure, cryptographic processing, and 2PC protocols as a Twisted service and client; 780 lines are devoted to unit and timing tests; and 569 lines use the Fabric framework to do configuration, deployment management (DevOps), live testing, and visualizations. All cryptographic operations use the OpenSSL wrapper library `petlib`, and we instantiate the hash function and digital signature using SHA-256 and ECDSA (over the NIST-P224 curve, as optimized by Käsper [12]) respectively. The implementation and all configuration and orchestration files necessary for replicating our results are available under a BSD license.

Our experimental setup consisted of 10 mintettes, each running on an Amazon EC2 `t2.micro` instance in the EU (Ireland) data center (for reference, each cost $0.014 per hour as of May 2015). We assigned three mintettes to each shard of the transaction space, so a quorum of at least two was required for the 2PC. A different set of 10 servers on the same data center was used for stress testing and to estimate the peak throughput in terms of transactions per second. Each of those test machines issued 1000 transactions consisting of two inputs and two outputs. For wide area networking latency experiments we used a residential broadband cable service and an Ubuntu 14.02.2 LTS Linux VM running on a 64-bit Windows 7 laptop with a 2.4 GHz i7-4700MQ processor and 16GB RAM.

Table 2 reports the mean rate and the standard deviation of key operations we rely on for RSCoin.[3] *Hash*, *Sign* and *Verify* benchmark the number of basic cryptographic operations each mintette can perform per second (including the overhead of our library and Python runtime).

For the other benchmarks, we consider a single transaction with one input and two outputs (as of September 2014, 53% of Bitcoin transactions had this structure, so this is a reasonable proxy for real usage). The *check tx* benchmark then measures the rate at which a mintette can parse and perform the cryptographic checks associated with this transaction. This involves a single signature check, and thus its difference from the *Sign* benchmark largely represents the overhead of parsing and of binary conversion in Python. Guided by this benchmark, we chose to represent ECDSA public keys using uncompressed coordinates due to orders-of-magnitude slowdowns when parsing keys in compressed form.

The *query msg* and *commit msg* benchmarks measure the rate at which each mintette can process the first and second message of the 2PC respectively for this transaction. These include full de-serialization, checks from persistent storage of the `utxo`, cryptographic checks, updates to the `utxo`, signing, and serialization of responses. These benchmarks guided our design towards not synchronizing to persistent storage the `utxo` before each response, and relying instead on the quorum of mintettes to ensure correctness (a design philosophy similar to RAMCloud [23]). Persisting to storage before responding to each request slowed these rates by orders of magnitude.

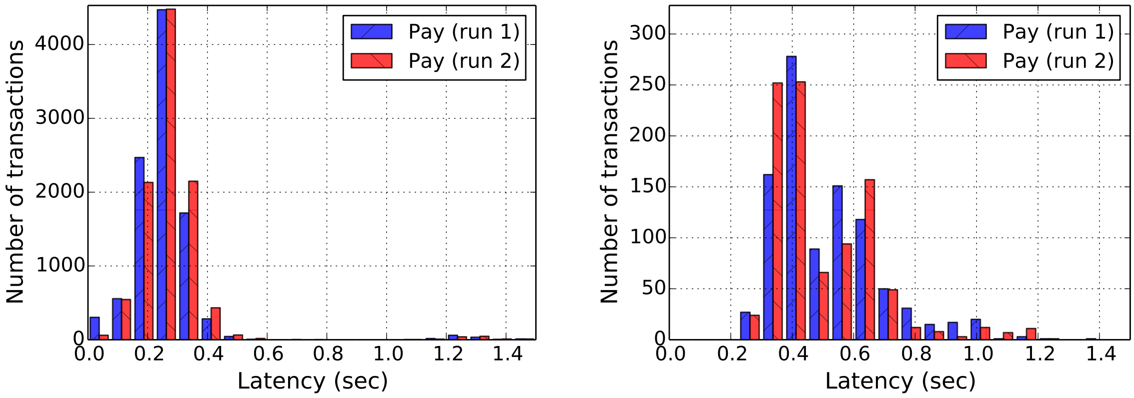
Figure 3 illustrates the latency a client would experience when interacting with the mintettes. Figure 3(a) illustrates the experiments with client machines within the data center, and point to an intrinsic delay due to networking overheads and cryptographic checks of less than 0.5 seconds. This includes both phases of the 2PC.

Over a wide area network the latency increases (Figure 3(b)), but under the conditions tested, the latency is still usually well under a second for the full 2PC and all checks. We note that no shortcuts were implemented: for each transaction, all three mintettes for each input were contacted and expected to respond in the first phase, and all three mintettes responsible for the new transaction were contacted and have to respond for the second phase. In reality, only a majority need to respond before concluding each phase, and this may reduce latency further.

Figure 4 plots the throughput of the system as we increase the number of mintettes from 1 to 10, under the load of 10 synthetic clients, each pushing 1000 transactions. As expected, when fewer than three mintettes are available the throughput is roughly flat (fewer than 400 transactions per second), as both phases of the

---

[3] All measurements were performed on a single thread on a single core, using a reactor pattern where networking was necessary.



(a) Local area network (EC2)

(b) Wide area network (Broadband)

Figure 3: Latency, in seconds, to perform the 2PC to validate a payment for a transaction with freshly issued coins as inputs (run 1), and transactions with two arbitrary previous transactions as inputs (run 2).

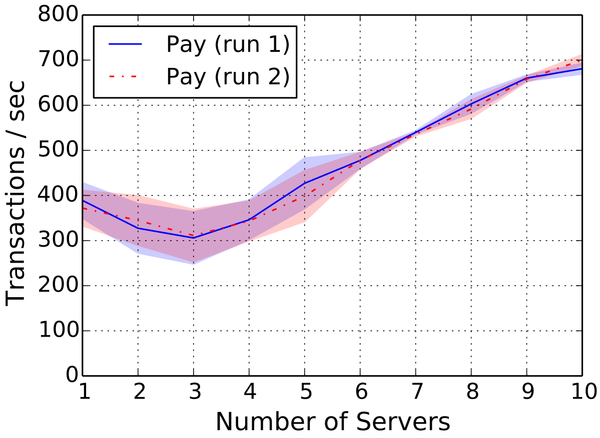


Figure 4: Throughput (mean and standard error), in transactions per second, as a function of the number of mintettes, for transactions with two freshly issued coins as inputs (run 1) and transactions with two arbitrary previous transactions as inputs (run 2).

2PC need to contact all mintettes. Once more than the minimum of three mintettes are available the load is distributed across them: the first phase need to access at most six mintettes (three for each of the two transaction inputs), and the second phase at most three mintettes. This load per transaction is independent of the number of mintettes and as a result the throughput scales linearly, as predicted in Section 4.4.1. After the initial three mintettes, each new mintette adds approximately 66 additional transactions per second to the capacity of the system.

The gap between the micro-benchmarks relating to the message processing for the two phases (1,078.06 s^{-1} and 884.29 s^{-1} respectively) and the rate of transactions observed under end-to-end conditions (approximately 400 s^{-1}) indicates that at this point bandwidth, networking, or the interconnection with the process are scaling bottlenecks for single mintettes. In particular no pipelining was implemented as part of the client (although the mintettes support it) and thus every request initiates a fresh TCP connection, with the slowdowns and resource consumption on the hosts this entails.

5 Optimizations and Extensions

In Sections 3 and 4, we presented a (relatively) minimal version of RSCoin, which allows us to achieve the basic integrity and scalability properties that are crucial for any currency designed to be used on a global level. Here, we briefly sketch some extensions that could be adopted to strengthen either of these properties, and leave a more detailed analysis of these or other solutions as interesting future research.

5.1 Pruning intermediate transactions

At the end of a period, the central bank publishes a higher-level block containing the collection of transactions that have taken place in that time interval; it is only at this point that transactions are officially recorded in the ledger. Because mintettes provide evidence on a shorter time scale that a user's transaction is valid and will be included in the ledger, however, users might feel more comfortable moving currency multiple times within a period than in traditional cryptocurrencies (in which one must wait for one or several blocks to avoid possible double-spending).

It therefore might be the case that at the end of a period, the central bank sees not just individual transactions, but potentially multiple "hops" or even whole "chains" of transactions. To limit *transaction bloat*, the bank could thus prune these intermediate transactions at the end of the period, so that ultimately only the start and end points of the transaction appear in the ledger, in a new transaction signed by the central bank.

On its surface, this idea may seem to require a significant amount of trust in the central bank, as it could now actively modify the transaction history. The action logs, however, would reveal the changes that the bank had made and allow users to audit its behavior, but nevertheless the alterations that could be made would need be significantly restricted.

5.2 Further incentives for honest behavior

In addition to the existing incentives for honest behavior outlined in Sections 3.6 and 4.3.2, mintettes could adopt a sort of proof-of-stake mechanism, in which they escrow some units of currency with the central bank and are allowed to collate only a set of transactions whose collective value does not exceed the escrowed value. If any issue then arises with the transactions produced by the mintette (e.g., it has accepted double-spending transactions), the central bank can seize the escrowed value and remove the double-spending transactions, so the mintette ultimately pays for this misbehavior out of its own pocket (and maybe even pays additional fines).

This mechanism as described is not fully robust (as in particular the mintette might accept many expenditures of the same unit of currency, not just two), but it does have an interesting effect on the length of periods. In particular, the length of earlier periods will necessarily be quite small, as mintettes will not have much capital to post. As mintettes accumulate stores of currency, however, periods can grow longer. This is a fairly natural process, as it also allows for a trial period in the beginning to ensure that authorized mintettes don't misbehave, and then for a more stable system as a set of trustworthy mintettes emerges.

5.3 Multiple banks and foreign exchange

In a global setting, one might imagine that each central bank could develop their own version of RSCoin; this would lead, however, to a landscape much the same as today's Bitcoin and the many altcoins it has inspired, in which multiple implementations of a largely overlapping structure lead to an *infrastructure fragmentation*: bugs are replicated across codebases and compatibility across different altcoins is artificially low.

An attractive approach is for different central banks to instead use the same platform, to prevent this fragmentation and to allow users to seamlessly store value in many different currencies. While this allows the currencies generated by different central banks to achieve some notion of interoperability, we still expect that different blockchains will be kept separate; i.e., a particular central bank does not — and should not — have to keep track of all transactions that are denominated in the currency of another central bank. (Mintettes, however, may choose to validate transactions for any number of central banks, depending on their business interests.)

While every central bank does not necessarily need to be aware of transactions denominated in the currency of another central bank, this awareness may at times be desirable. For example, if a user would like to exchange some units of one currency into another belonging to a central bank that is relatively known to and trusted by the first (e.g., exchange GBP for USD), then this should be a relatively easy process. The traditional approach is to simply go to a third-party service that holds units of both currencies, and then perform one transaction to send units of the first currency to the service, which will show up in the ledger of the first currency, and another transaction to receive units of the second currency, which will show up in the ledger of the second currency.

Although this is the approach by far most commonly adopted in practice (both in fiat currency and cryptocurrency markets), it has a number of limitations, first and foremost of which is that it is completely opaque: even an outside observer who is able to observe both ledgers sees two transactions that are not linked in any obvious way. One might naturally wonder, then, if a more *transparent* mechansim is possible, in which the currency exchange shows up as such in the ledger. One simple approach for this works as follows. If Alice holds m units of currency c_1 and wishes to exchange these for Bob's n units of currency c_2, then they can create a transaction with two inputs and two outputs: one input is Alice's address $\mathsf{addr}_A^{(c_1)}$, to which m units of currency c_1 have been sent and the other is Bob's address $\mathsf{addr}_B^{(c_2)}$, to which n units of currency c_2 have been sent, and the output addresses are $\mathsf{addr}_A^{(c_2)}$ (to receive value n) and $\mathsf{addr}_B^{(c_1)}$ (to receive value m). After creating and signing this transaction, they can send it to the respective mintettes for c_1 and c_2. If the mintettes simply ignore the components of the transaction that are not denominated in a currency they recognize, then the transaction will ultimately be incorporated into the ledgers for both c_1 and c_2. Anyone with access to both of these blockchains can thus verify that this exchange took place.

6 Related Work

Much of the research on cryptocurrencies either has analyzed the extent to which existing properties (e.g., anonymity and fairness) are satisfied or has proposed new methods to improve certain features. We focus on those works that are most related to the issues that we aim to address, namely stability and scalability.

Much of the work on these two topics has been attack based, demonstrating that even Bitcoin's heavyweight mechanisms do not provide perfect solutions. As demonstrated by Eyal and Sirer [7] and Garay et al. [8], an attacker can temporarily withhold blocks and ultimately undermine fairness. Babaioff et al. [1] argued that honest participation in the Bitcoin network was not sufficiently incentivized, and moreover Johnson et al. [10] and Laszka et al. [14] demonstrated that in fact some participants might be incentivized to engage in denial-of-service attacks against each other. Karame et al. [11] and Rosenfeld [25] consider how an adversary might take advantage of both mining power and the network topology to execute a double-spending attack. On the positive side, Kroll et al. [13] analyzed a simplified model of the Bitcoin network and concluded that Bitcoin is (at least weakly) stable.

In terms of other constructions, the work perhaps most related to our own is Laurie's approach of designated authorities [15]. This solution, however, does not describe a consensus mechanism or consider a centralized entity responsible for the generation of a monetary supply. The RSCoin framework is also related to the approaches adopted by Ripple and Stellar, although the underlying consensus mechanisms [17, 26] and goals of the systems are quite different.

Finally, our approach borrows ideas from a number of industrial solutions. In particular, our two-layered approach to the blockchain is in part inspired by the Bitcoin startup Factom, and our consensus mechanism is in part inspired by Certificate Transparency (CT) [16]. In particular, RSCoin, like CT, uses designated authorities and relies on transparency and auditability to ensure integrity of a ledger, rather than full trust in a central party.

7 Conclusions

In this paper, we have presented the first cryptocurrency framework, RSCoin, that provides the control over monetary policy that entities such as central banks expect to retain. By constructing a blockchain-based approach that makes relatively minimal alterations to the design of successful cryptocurrencies such as Bitcoin, we have demonstrated that this centralization can be achieved while still maintaining the transparency guarantees that have made (fully) decentralized cryptocurrencies so attractive. We have also demonstrated, via the construction of a new consensus mechanism and measurement of its performance, that centralization allows RSCoin to use a significantly simpler consensus mechanism than that of traditional cryptocurrencies, which in turn leads to a more scalable system that completely avoids the wasteful hashing required in proof-of-work-based systems.

Acknowledgements

We thank Robleh Ali, Simon Scorer, Alex Mitchell, and John Barrdear from the Bank of England and Ben Laurie from Google for interesting discussions. George Danezis is supported by EPSRC Grant EP/M013286/1 on "Strengthening anonymity in messaging systems."

References

[1] M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar. On Bitcoin and red balloons. *SIGecom Exchanges*, 10(3):56–73, 2011.

[2] Bank of England. Private communication, 2015.

[3] Bank of England. One bank research agenda, 2015. `www.bankofengland.co.uk/research/Documents/onebank/discussion.pdf`.

[4] B. Bernanke, Nov. 2013. `qz.com/148399/ben-bernanke-bitcoin-may-hold-long-term-promise/`.

[5] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Research perspectives and challenges for Bitcoin and cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2015.

[6] European Central Bank. Virtual currency schemes - a further analysis, Feb. 2015. `www.ecb.europa.eu/pub/pdf/other/virtualcurrencyschemesen.pdf`.

[7] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Proceedings of Financial Cryptography 2014*, 2014.

[8] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Proceedings of Eurocrypt 2015*, 2015.

[9] HM Treasury. Digital currencies: response to the call for information, Mar. 2015. `www.gov.uk/government/uploads/system/uploads/attachment_data/file/414040/digital_currencies_response_to_call_for_information_final_changes.pdf`.

[10] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore. Game-theoretic analysis of DDoS attacks against Bitcoin mining pools. In *Workshop on Bitcoin Research*, 2014.

[11] G. Karame, E. Androulaki, and S. Capkun. Double-Spending Fast Payments in Bitcoin. In *Proceedings of ACM CCS 2012*, 2012.

[12] E. Käsper. Fast elliptic curve cryptography in openssl. In G. Danezis, S. Dietrich, and K. Sako, editors, *Financial Cryptography and Data Security - FC 2011 Workshops, RLCPS and WECSR 2011, Rodney Bay, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers*, volume 7126 of *LNCS*, pages 27–39. Springer, 2011.

[13] J. A. Kroll, I. C. Davey, and E. W. Felten. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Proceedings of WEIS 2013*, 2013.

[14] A. Laszka, B. Johnson, and J. Grossklags. When Bitcoin mining pools run dry: A game-theoretic analysis of the long-term impact of attacks between mining pools. In *Workshop on Bitcoin Research*, 2015.

[15] B. Laurie. An efficient distributed currency, 2011. `www.links.org/files/distributed-currency.pdf`.

[16] B. Laurie. Certificate transparency. *Commun. ACM*, 57(10):40–46, 2014.

[17] D. Mazières. The Stellar consensus protocol: a federated model for Internet-level consensus, 2015. `www.stellar.org/papers/stellar-consensus-protocol.pdf`.

[18] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing Bitcoin work for data preservation. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2014.

[19] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. `bitcoin.org/bitcoin.pdf`.

[20] Nasdaq. Nasdaq launches enterprise-wide blockchain technology initiative, May 2015. `www.nasdaq.com/press-release/nasdaq-launches-enterprisewide-blockchain-technology-initiative-20150511-00485`.

[21] K. J. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. In *Proceedings of the Irish Signals and Systems Conference*, pages 280–285, 2014.

[22] D. O'Leary, V. D'Agostino, S. R. Re, J. Burney, and A. Hoffman. Method and system for processing Internet payments using the electronic funds transfer network, Nov. 2013.

[23] J. K. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. M. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. The case for ramcloud. *Commun. ACM*, 54(7):121–130, 2011.

[24] M. Raynal and M. Singhal. Logical time: Capturing causality in distributed systems. *IEEE Computer*, 29(2):49–56, 1996.

[25] M. Rosenfeld. Analysis of hashrate-based double-spending, Dec. 2012. `bitcoil.co.il/Doublespend.pdf`.

[26] D. Schwartz, N. Youngs, and A. Britto. The Ripple protocol consensus algorithm, 2014. `ripple.com/files/ripple_consensus_whitepaper.pdf`.