Filecoin: A Cryptocurrency Operated File Storage Network

1e96a1b27a6cb85df68d728cf3695b0c46dbd44d
filecoin.io

July 15, 2014

Abstract

Filecoin is a distributed electronic currency similar to Bitcoin. Unlike Bitcoin's computation-only proof-of-work, Filecoin's proof-of-work function includes a proof-of-retrievability component, which requires nodes to prove they store a particular file. The Filecoin network forms an entirely distributed file storage system, whose nodes are incentivized to store as much of the entire network's data as they can. The currency is awarded for storing files, and is transferred in transactions, as in Bitcoin. Files are added to the network by spending currency. This produces strong monetary incentives for individuals to join and work for the network. In the course of ordinary operation of the Filecoin network, nodes contribute useful work in the form of storage and distribution of valuable data.

1 Introduction

Many computer systems store and access data via commercial service providers. At present, a handful of large providers serve most of these markets. New market entrants are rare, as direct competition at full scale with incumbent providers is virtually impossible. This makes it hard to optimize certain inefficiencies. For example, the data transfer bottleneck today is mostly last-mile ISPs; both the internet backbone and local area networks are orders of magnitude faster. Distributed services, in which agents have individual incentives to store data and optimize local distribution, could provide vastly better solutions.

In addition, one major goal of distributed storage systems is to ensure the preservation of important files. In this respect, current storage systems are brittle. First, they tend to be centrally managed by one service provider, linking the fate of the files stored to the fate of that organization. Second, in widely-used file retrieval schemes such as HTTP, a file is identified by its location rather than by its content. This makes file availability depend upon the uptime of specific hosts, rather than the existence of the file anywhere in the network. Third, widely-used schemes place the perpetual burden of serving a file on the original creator—either by hosting the file herself, or hiring a provider to do so—which is often unsustainable, particularly for large files such as scientific data sets. What is needed is a globally distributed network whose individual agents can serve any file requests and are strongly incentivized to do so.

Blockchain-based cryptocurrencies, a recent development, can organize and incentivize large networks of machines to perform computations as proofs-of-work. In Bitcoin [7], these computations tend to be useless. Others have attempted to organize more useful algorithmic work; for example, Primecoin [4] rewards finding chains of prime numbers. However, these systems could also perform other kinds of useful work in clear immediate demand, such as storing and distributing files. In this paper, we propose a solution that incentivizes file storage using a Bitcoin-like cryptocurrency network, whose work function incorporates proofs-of-retrievability.

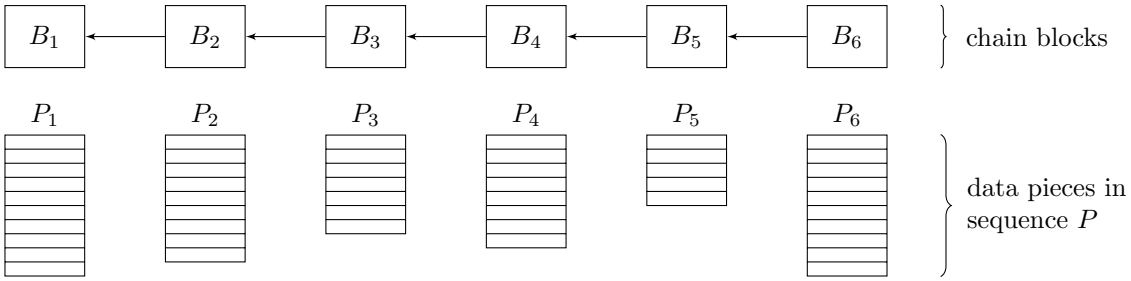
2 Design

Like Bitcoin, Filecoin implements a transaction ledger via a blockchain, in which each block must be accompanied by a proof-of-work based on a cryptographic hash function. The proof-of-work parameter is dynamically adjusted so that one block occurs roughly every ten minutes, as in Bitcoin. Further, as in most blockchain-based constructions, clients should only consider transactions to be committed after a sequence

of several valid blocks, where the number of blocks is determined by the assumed restrictions on the power of the adversary (e.g., that no adversary can control a majority of the network's computation power). However, Filecoin makes a number of central changes to the standard Bitcoin-style design.

2.1 The Piece Set

The first key component of Filecoin is the addition of a growing sequence of data *pieces*,[1] which form the files stored by the network. A *piece* is an opaque segment of data of fixed size S, a tunable network parameter.[2] Each piece introduced to the Filecoin blockchain appears in some block B_t, via a special Put transaction (discussed in Section 2.2). The *piece set* P is ordered chronologically. As in Bitcoin, all nodes must keep the entire blockchain in local storage, but pieces are distributed among all Filecoin nodes. This scheme allows the Filecoin blockchain to provide storage of data orders of magnitude larger than the the blockchain itself.



$$P = P_1||P_2||...||P_t$$

2.2 Put and Get Transactions

The second key addition in Filecoin is the specification of Put and Get transactions. Put transactions add files to the network's storage: each Put transaction includes a list of piece records introducing new pieces. A *piece record* is a value of the form:

$$\mathsf{record}_i = (\mathcal{H}(p_i), \mathcal{H}(\sigma_i), \mathsf{pk}_i, \mathsf{reward}_i)$$

where $\mathcal{H}(p_i)$ is the cryptographic hash of a piece, σ_i is a sequence of authenticators for p_i (Section 2.5), pk_i is a verification key (Section 2.5), and reward_i are reward parameters (Section 2.7). When a Put transaction is finalized—mined as part of a block—all pieces identified by the piece records are deemed stored by the network.

A Get transaction is used to surface a specific piece previously stored on demand. When a Get transaction is finalized, the pieces listed are transmitted to the transaction issuer, as discussed in Section 2.4. These two transactions extend the Bitcoin protocol with a standard way to store and retrieve files, which are made up of concatenated pieces.

2.3 Incentivizing Piece Storage via Challenges

Get transactions are not enough to incentivize miners to store all pieces, as miners cannot predict the distribution of future Get transactions and their expected reward. Filecoin's most important departure from Bitcoin-style cryptocurrencies is a modification of the work function. In order to succeed in mining a block, on top of the usual hash-based proof-of-work, miners also must prove that they currently store a particular set of "challenge" pieces.

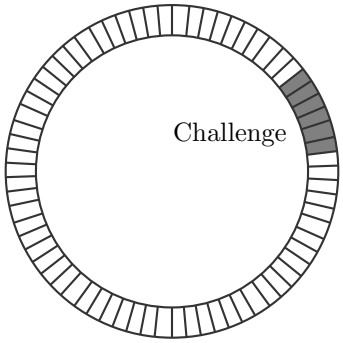
[1] Files imply data segments of arbitrary size, while Filecoin uses fixed-size data blocks. To avoid confusion, *block* refers to a block forming the Filecoin blockchain, and *piece* refers to a block of data stored by the blockchain.

[2] Each piece introduces overhead in the blockchain, which every node must store. Choice of S also depends on the parameters of the proof-of-retrievability scheme (Section 2.5).

For example, consider a blockchain storing records of pieces ending with block B_{t-1}. In order to add block B_t to the blockchain, a miner must prove that they are storing a particular sequence of pieces, as determined by the partially-mined block B'_t (which includes a Bitcoin-style proof-of-work), described in Section 2.6. Specifically, the starting piece index i_t is determined as follows:

$$i_t = \mathcal{H}(B'_t) \mod |P|$$

This index points to a sequence of pieces p_i in P, for i ranging from i_t through $(i_t + k - 1)$, where k is a tunable difficulty parameter. This index choice samples pieces uniformly. The size of the challenge k gives the flexibility of making challenges harder or easier to match node capacity. For example, $k = 6$ would require proving pieces i through $i+5$. We issue challenges of contiguous pieces because sets of non-contiguous pieces would yield a super-linear expected return on storage, incentivizing formation of large mining pools. As in Bitcoin, such pools would threaten to allow an adversary to control a majority of the network.



Piece Set P

If a miner wishes to extend the blockchain by mining block B_t, and claim any rewards for doing so, it must provide a proof demonstrating that it is currently storing this sequence of challenge pieces p_i:

$$\mathsf{PieceProof}(p_i, t) = (\mathcal{H}(B'_t||p_i), \pi_i)$$

where B'_t is the partial block constructed thus far and determines the challenge, and the values $(\mathcal{H}(B'_t||p_i), \pi_i)$ constitute the miner's proof (Section 2.5) that it is storing the challenge pieces. As the total storage of the network—the pieces in P—grows well beyond the storage capacity of individual nodes, miners are incentivized to acquire whichever pieces are covered by the fewest other nodes, since these have a significant chance of yielding a profit upon a future block-minting challenge.

Systems like Filecoin must take into account the possibility that all copies of a particular piece have disappeared. This would render some challenges unsolvable, and is possible due to large network failure and data loss. In the construction discussed so far, this is not a problem because each block challenge depends on its partial block B'_t including the proof-of-work. Two nodes that find different valid proofs-of-work will be issued different piece challenges.

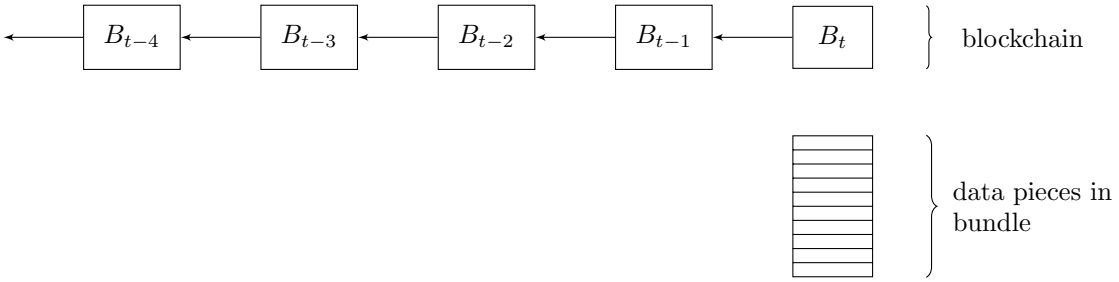
2.4 Dispersing Pieces in the Bundle

As discussed so far, Filecoin nodes have an incentive to hoard pieces, to retain the rarest for themselves and thereby increase their expected block-minting reward. Filecoin provides a piece dispersion mechanism to prevent monopoly or hoarding in general. When mining a new block, nodes must also present a set of pieces. This set is called the *bundle* and includes all the pieces p_i (and corresponding σ_i) referenced in the new blockchain head. It has the challenge pieces, and pieces in Get or Put transactions. This bundle approach solves multiple problems at once:

- It discourages hoarding and prevents nodes from exploiting long-lasting piece monopolies.
- It ensures that pieces are sent to issuers of Get transactions.
- It ensures that new pieces in Put transactions are seen by many nodes, so that they are quickly adopted and benefit from redundant storage.

- It provides starting sets of pieces to new miners joining the network.
- It mitigates the proof-of-retrievability forgery discussed in Section 2.5.

The size of the bundle, on the order of hundreds of transactions, is only a small fraction of that of the blockchain (which, in the Bitcoin network, currently exceeds 15 GB).



2.5 Verification via Compact Proofs-of-Retrievability

The Filecoin blockchain must be verifiable by any node in the network, including new nodes who have no prior knowledge. Filecoin proofs $\mathsf{PieceProof}(p_i, t)$ have two components: $\mathcal{H}(B'_t||p_i)$ and π_i. The first component, the hash value $\mathcal{H}(B'_t||p_i)$, allows any node, given the original piece p_i, to verify that the miner of block B_t was in possession of p_i at the time of mining the block B_t. The second component, π_i, is a compact proof-of-retrievability, as defined below, that can be verified by any node, even without the original piece p_i.

Since verification based on the hash value $\mathcal{H}(B'_t||p_i)$ requires access to the original piece p_i, it would be prohibitively expensive to validate the entire blockchain this way. Instead, Filecoin nodes conduct this expensive verification only on the latest blocks as they are minted (whose corresponding pieces are circulated in the form of the head "bundle" accompanying each block, as described in Section 2.4), and rely on the compact proof-of-retrievability π_i to validate blocks earlier on the blockchain.

We now describe the use of proofs-of-retrievability in more detail. Most proof-of-retrievability schemes involve two parties, a *client* who preprocesses data and a *server* who stores the processed data. At any point, the client can issue a *challenge* to the server, who must then calculate the corresponding *proof*. Usually, the challenges are generated and verified using a secret key known only to the client. However, recent schemes, such as the constructions of Shacham and Waters [8] and of Ateniese et al. [1], have the following desirable properties.

1. They are publicly verifiable: challenges can be issued and verified by anyone, not just the original client.
2. They are compact in bandwidth: the challenges and proofs are small enough to include in the blockchain.

Here, we adapt these schemes to be used in the blockchain by storing the public keys in the piece records, issuing the challenges by hashing prospective partially-minted blocks, and storing the proofs-of-retrievability on the resulting block headers.

$$
\begin{aligned}
(\mathsf{sk}_i, \mathsf{pk}_i, \sigma_i) &\leftarrow \mathsf{PoR.Setup}(p_i) \\
\mathsf{challenge} &\leftarrow \mathsf{PoR.Challenge}(\mathsf{pk}_i, \mathcal{H}(B'_t||i)) \\
\pi_i &\leftarrow \mathsf{PoR.Prove}(\mathsf{challenge}, p_i, \sigma_i) \\
&\quad\ \mathsf{PoR.Verify}(\mathsf{pk}_i, \pi_i) \in \{0, 1\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{record}_i &= (\mathcal{H}(p_i), \mathcal{H}(\sigma_i), \mathsf{pk}_i, \mathsf{reward}_i) \\
\mathsf{PieceProof}(p_i, t) &= (\mathcal{H}(B'_t||p_i), \pi_i)
\end{aligned}
$$

The PoR operations can be instantiated by the publicly verifiable scheme of Shacham and Waters [8]. For each piece p_i in a prospective new block, the miner generates a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. We store $\mathcal{H}(\sigma_i)$ and pk_i

as part of the piece record in the piece Put transaction. The authenticators σ_i must be verified during Put transaction verification, as discussed in Section 2.6. Any nodes who wish to prove possession of the original piece p_i must then store both p_i and its authenticators σ_i. The piece challenge is determined by the block, including challenge randomness $\mathcal{H}(B'_t||i)$. The proof is stored on the final block. This yields a compact, publicly verifiable way to check the entire blockchain. However, this adaptation of the publicly-verifiable Shacham-Waters scheme [8] has a disadvantage in our context: if some attacker has already mined some past block successfully, then such an attacker could retain that block's secret keys sk_i, and use them to forge valid proofs-of-retrievability π_i without possessing the original piece data p_i or σ_i. We guard against this forgery by also requiring the value $\mathcal{H}(B'_t||p_i)$ to appear in the newly-minted block, and requiring the miner to distribute the original piece data (p_i, σ_i) (as discussed in Section 2.4), which nodes can then check against the hash output $\mathcal{H}(B'_t||p_i)$. With this mitigation, under some circumstances Filecoin can provide even stronger security guarantees than Bitcoin: notably, even an adversary with 51% of the hashing power of the network, who generates and stores sk_i and thus can forge the corresponding proofs-of-retrievability, would still also need to expend resources to store or acquire the data pieces at the time of forgery.

2.6 Block Construction and Verification Procedures

A new Filecoin block B_t is mined by preparing the new transactions it will include, constructing the block, and preparing its corresponding bundle.

- **Transactions** are assembled into the standard Merkle tree construction, with root txRoot.
  - Put transactions include the sequence of new piece records $\mathsf{record}_i = (\mathcal{H}(p_i), \mathcal{H}(\sigma_i), \mathsf{pk}_i, \mathsf{reward}_i)$.
  - Get transactions include the value $\mathcal{H}(p_i)$, and the value $\mathsf{PieceProof}(p_i, t)$ as described in Section 2.3.
  - Other types of transactions can be included as in Bitcoin.
- **Block** $B_t = (\mathsf{parent}_t, \mathsf{txRoot}_t, \mathsf{nonce}_t, \mathsf{PoW}_t, \mathsf{PoR}_t)$, where:
  - parent_t is $\mathcal{H}(B_{t-1})$.
  - txRoot_t is the transaction Merkle tree root.
  - nonce_t is a nonce chosen, as per the Bitcoin proof-of-work scheme, so that:

    $$\mathsf{PoW}_t = \mathcal{H}(\mathsf{parent}_t||\mathsf{txRoot}_t||\mathsf{nonce}_t) < 2^\ell$$

    for dynamically-adjusted hashing difficulty parameter ℓ.
  - PoW_t is the proof-of-work output value, which determines the challenge, as specified in Section 2.3.
  - PoR_t is the sequence of values $\mathsf{PieceProof}(p_i, t)$ for each piece p_i in the piece challenge.
- **Bundle** data includes all pieces in the current block's challenge set, as well as those referenced by all Put and Get transactions in the block.

Once the block B_t is successfully constructed, the miner broadcasts the block to the rest of the network. Any other member can verify block B_t according to the following procedures.

1. Verify parent_t is the previous blockchain head.
2. Verify all transactions:
   - Perform Bitcoin-style transaction verification, including checking balance changes are valid.
   - For each Get transaction:
     - check p_i and σ_i are in the bundle,
     - check $\mathcal{H}(p_i)$ matches,
     - check $\mathsf{PieceProof}(p_i, t)$ as described in Section 2.5.
   - For each Put transaction, verify each new piece record:
     - check p_i and σ_i are in the bundle,
     - check $\mathcal{H}(p_i)$ and $\mathcal{H}(\sigma_i)$ match,
     - check reward_i is smaller than transaction issuer's balance,
     - check pk_i is a pair of elements of the appropriate group as described in Shacham and Waters [8],

- check σ_i has been constructed correctly, according to the procedure below.

3. Verify $\mathsf{PoW}_t = \mathcal{H}(\mathsf{parent}_t || \mathsf{txRoot}_t || \mathsf{nonce}_t)$ and $\mathsf{PoW}_t < 2^\ell$.
4. Verify all $\mathsf{PieceProof}(p_i, t)$ in PoR_t as described in Section 2.5.

It is important to verify σ_i have been constructed correctly, as an adversary could provide invalid σ_i so that other miners are sometimes unable to construct valid proofs. Checking σ_i involves checking that the authenticators have been constructed correctly:

$$e(\sigma_{i,j}, g) = e(\mathcal{H}(j) \cdot u^{p_{i,j}}, v)$$

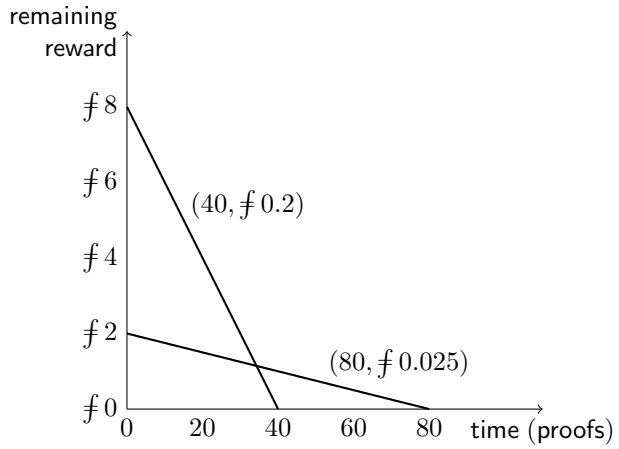
where j indexes into each subdivision of the piece p_i and its authenticators σ_i as described in the Shacham-Waters construction [8]. Checking all authenticators in σ_i for every Put transaction would introduce significant computational overhead. It is possible to perform a more efficient check by verifying only a random subset of authenticators. We note that in this case, all verifiers must select this subset deterministically by evaluating a hash function on PoW_t, or they might disagree on whether to accept or reject a block.

2.7 Reward Parameters and Secondary Markets

Filecoin allows Put transactions to specify piece rewards. This construction enables users to spend Filecoin currency to strengthen the incentives of storing particular pieces. Reward parameters reward_i for each piece must be specified in the value record_i in the corresponding Put transaction. Transaction issuers must pay for all rewards up-front, effectively placing the currency into a fund for the pieces in question. The reward function design space is ample, so we describe only a straightforward construction and leave others for future development. Our function uses parameters:

$$\mathsf{reward}_i = (\mathsf{TTL}_i, \mathsf{RPP}_i)$$

where TTL_i is a time-to-live for p_i, and RPP_i is a reward-per-proof. Our function awards an additional RPP_i Filecoin to miners that successfully prove possession of p_i in a storage challenge, up to TTL_i times. This is a total of $(\mathsf{RPP}_i \cdot \mathsf{TTL}_i)$ Filecoin, which the Put issuing user must pay up-front. These two simple parameters give users significant freedom in controlling the incentives of storing particular pieces. Though it is measured in "challenges proved", the TTL corresponds to the lifetime of a piece in the network, or the duration of the incentive. The RPP corresponds to the strength of the incentive. The figure below illustrates tradeoffs between two $(\mathsf{TTL}_i, \mathsf{RPP}_i)$ configurations.



In the most straightforward setting of the reward parameters, challenge rewards for a given piece p_i are dispensed at a constant rate (as in the charts above), and the sum of all challenge rewards issued adds up to the amount of currency required to insert the piece p_i in its initial Put transaction. Other parameter settings could set these rewards to decrease by some other, nonlinear function (perhaps exponentially), or could specify that the total reward extracted exceeds that inserted (thereby creating an inflationary currency). Note

that subtly different reward schemes can yield drastically different piece storage distributions, or potentially break the incentive structure altogether. Altering the reward scheme must include careful analysis of the resulting market equilibria. For instance, if the currency is too inflationary, then attackers may benefit from adding large amounts of "dummy data" that they can easily reproduce without incurring the cost of storage (e.g., the outputs of a pseudorandom function for which they know the secret key), and thereby gain a net advantage in the challenge reward system over the long term.

Nodes can also profit from the Filecoin network in ways other than by directly redeeming block-minting rewards. Miners can set additional transaction fees, as determined by the total demand of Filecoin clients who are using the network to execute monetary transactions. In addition, since piece data is likely to be stored and demanded elsewhere, and since even those miners with significant hashing power are unlikely to store all of the resulting challenge pieces themselves, it is likely that secondary markets in piece data will emerge. In these secondary markets, even nodes who are unable to provide much hashing power can still profit from their storage capacity. Additionally, nodes can provide services such as data processing, generating additional income from the data they already store. Whether or not these transactions occur on the Filecoin blockchain, they contribute to the profit of Filecoin nodes, and thus to the continued storage and distribution of useful data pieces.

2.8 Consensus Mechanisms

In Filecoin, block mining represents useful work: previously stored files have been proven to remain in the network, new files have been added, and new transactions have been issued. In Bitcoin, block mining also provides the useful service of issuing transactions, despite the intrinsically useless proofs-of-work. These are performed in order to ensure that the network achieves consensus on the ledger.

In the version of Filecoin described in this paper, the useful storage service is layered on top of Bitcoin consensus: after a proof-of-work is solved, the miner must then also provide a corresponding proof-of-retrievability. The storage service is useful, but the network still wastes vast computational resources in performing the proof-of-work component of the consensus mechanism. One alternative approach would be to use proofs-of-retrievability directly in the consensus protocol. In this case, in order to produce a block, a miner would first be required to prove retrievability of challenge pieces as determined by the hash of the previous block, followed by a much easier proof-of-work. Using this mechanism would change the Bitcoin-style assumptions required: specifically, the requirement that no adversary controls 51% of the hashing power would be replaced by a tunable tradeoff between storage and computation, with independent difficulty parameters. However, this approach also has a considerable vulnerability. Since secondary markets are likely, a computationally powerful adversary might take advantage of them to bypass the proof-of-retrievability requirement altogether, and only compete computationally with a small fraction of the network.

Instead, we observe that the Filecoin storage service can also be based on any robust distributed ledger, such as proof-of-stake-based systems [2, 5], or any Byzantine consensus mechanism [3, 6]. Since Filecoin's goal is to make data available widely and cheaply, and to repurpose wasted computational resources to useful tasks, in a future version of Filecoin we propose replacing proof-of-work in the consensus mechanism entirely.

3 Conclusion

We present a new cryptocurrency and file storage network called Filecoin. Filecoin enables outsourcing of data storage to a fluid distributed network of service providers. Individual providers are incentivized to allocate their storage resources to cover all requested data pieces, since any such piece may be the subject of a profitable future block-mining challenge. Parties may opt to join or leave the network at will, without compromising the robustness of the system. Tunable parameters trade off between churn and resiliency, replication factors, and consensus strategies.

References

[1] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *Advances in Cryptology–ASIACRYPT 2009*, pages 319–333. Springer, 2009.

[2] V. Buterin. Ethereum <https://ethereum.org/>, Apr. 2014.

[3] M. Castro, B. Liskov, et al. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[4] S. King. Primecoin <http://primecoin.io/bin/primecoin-paper.pdf>, Apr. 2014.

[5] S. King and S. Nadal. Peercoin <http://peercoin.net/assets/paper/peercoin-paper.pdf>, Apr. 2014.

[6] L. Lamport. Byzantizing paxos by refinement. In D. Peleg, editor, *Distributed Computing*, volume 6950 of *Lecture Notes in Computer Science*, pages 211–224. Springer Berlin Heidelberg, 2011.

[7] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[8] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.