Permacoin: Repurposing Bitcoin Work for Data Preservation

Andrew Miller[1], Ari Juels[2], Elaine Shi[1], Bryan Parno[3] and Jonathan Katz[1]

[1]University of Maryland
[2]Cornell Tech (Jacobs)
[3]Microsoft Research

Abstract

Bitcoin is widely regarded as the first broadly successful e-cash system. An oft-cited concern, though, is that mining Bitcoins wastes computational resources. Indeed, Bitcoin's underlying mining mechanism, which we call a scratch-off puzzle (SOP), involves continuously attempting to solve computational puzzles that have no intrinsic utility.

We propose a modification to Bitcoin that repurposes its mining resources to achieve a more broadly useful goal: *distributed storage of archival data*. We call our new scheme *Permacoin*. Unlike Bitcoin and its proposed alternatives, Permacoin requires clients to invest not just computational resources, but also storage. Our scheme involves an alternative scratch-off puzzle for Bitcoin based on Proofs-of-Retrievability (PORs). Successfully minting money with this SOP requires *local, random access to a copy of a file*. Given the competition among mining clients in Bitcoin, this modified SOP gives rise to highly decentralized file storage, thus reducing the overall waste of Bitcoin.

Using a model of rational economic agents we show that our modified SOP preserves the essential properties of the original Bitcoin puzzle. We also provide parameterizations and calculations based on realistic hardware constraints to demonstrate the practicality of Permacoin as a whole.

1 Introduction

> "We are justified in supposing that the contents of the Royal Library, if not wholly destroyed, were at least seriously diminished in the fire of 48 B.C." – Peter M. Fraser, on the destruction of the Ancient Library of Alexandria [16]

Bitcoin [27] is an exceptionally successful e-cash system based on the equivalence "time = money." Clients (nodes) in Bitcoin's peer-to-peer network invest computational *time* in order to mint *money* in the form of a currency called Bitcoins or BTC. The operation by which clients generate coins is called *mining*. We refer to the basic unit of mining work in Bitcoin as a scratch-off puzzle (SOP). In Bitcoin today, nodes mine coins by solving SOPs that involve finding preimages under a hash function. Correctly solving an SOP constitutes a *proof of work* [14], i.e., computational investment.

At the time of writing, mining a Bitcoin block (batch of coins) requires about 2^{55} hash computations. (For perspective, this is also the expected effort required to crack a DES key.) The Bitcoin network mines a block roughly every ten minutes, and thus consumes massive computing resources and natural resources such as electricity, prompting widespread concern about waste. The Bitcoin FAQ[1] says this about the issue:

Question: Is [Bitcoin] not a waste of energy?

Answer: *Spending energy on creating and securing a free monetary system is hardly a waste.... [Banks] also spend energy, arguably more than Bitcoin would.*

Question: Why don't we use calculations that are also useful for some other purpose?

Answer: *To provide security for the Bitcoin network, the calculations involved need to have some very specific features. These features are incompatible with leveraging the computation for other purposes.*

Indeed, researchers have struggled to identify useful computational tasks outside Bitcoin, e.g., protein folding problems [3], that *also* have the predictable solution times and efficient public verifiability required for Bitcoin.

1.1 Goal and approach

We show that *Bitcoin resources can be repurposed for other, more broadly useful tasks*, thereby refuting the widespread belief reflected in the Bitcoin FAQ. We propose a new scheme called *Permacoin*. The key idea in our scheme is to make Bitcoin mining depend upon *storage* resources, rather than *computation*. Permacoin then utilizes storage resources in the Bitcoin network.

Concretely, Permacoin involves a modified SOP in which nodes in the Bitcoin network perform mining by constructing a *Proof of Retrievability (POR)* [17]. A POR proves that a node is investing memory or storage resources to store a target file or file fragment. By building a POR-based SOP into Bitcoin, our scheme creates a system of *highly distributed, peer-to-peer file storage* suitable for storing a large, publicly valuable digital archive F. Specifically, our aim is to distribute F to protect it against data losses associated with a single entity, e.g., the outages or wholesale data losses already incurred by cloud providers [21].

In contrast to existing peer-to-peer schemes [9, 23], our scheme doesn't require an identity or reputation system to ensure storage of F, nor does it require that F be a popular download. We achieve file recoverability based strictly on

[1]Referenced 6 Apr. 2013 at https://en.bitcoin.it/wiki/FAQ.

clients' incentives to make money (mine Bitcoins).

1.2 Challenges

In constructing our SOP in Permacoin based on Proofs of Retrievability, we encounter three distinct challenges.

A standard POR involves a single prover holding a single file F. In our setting, however, multiple clients collectively store a large dataset F (too large for a single client) in a distributed manner. Of these an *adversarially selected fraction* may act maliciously. The first challenge in creating our SOP is to construct an adversarial model for this new setting, and then present a *distributed* POR protocol that is secure in this model. Assuming that clients have independent storage devices, we prove that with our SOP, for clients to achieve a high rate of mining, they must store F such that it is recoverable.

Additionally, we must ensure that clients indeed maintain independent storage devices. If, for instance, clients pooled their storage in the cloud to reduce their resource investment, the benefits of dataset-recovery robustness through distribution would be lost. Thus a second challenge in our SOP construction is to ensure that clients must make use of *local* storage to solve it.

To enforce locality of storage, we introduce into our POR a pair of novel features. First, block accesses depend on a client's *private key*, which is used to secure her Bitcoins and which she is therefore not likely to share but will instead only store locally. Second, these accesses are made sequentially and pseudorandomly. Thus fetching blocks remotely from a provider would incur infeasibly high communication costs (e.g., extremely high latency). We show using benchmarks how our SOP scheme thus takes advantage of practical network-resource limitations to prevent dangerous storage pooling.

Finally, to ensure incentives for client participation, it is important for our new storage-based SOP to preserve the economic structure of Bitcoin itself. We present an economic model showing how to parameterize our SOP to achieve this property.

1.3 Contributions

In summary, our contributions are as follows.

- *Bitcoin resource recycling:* Globally, our proposal Permacoin shows how to modify Bitcoin to *repurpose* the computing and natural resources that clients invest in mining for a general, useful goal and thus *reduce waste*. We consider this our central contribution.
- *POR distribution:* We show how to construct a distributed POR in a peer-to-peer setting. By incentivizing local storage of private keys and penalizing storage outsourcing, our scheme encourages local storage by participants and thus physically robust file dispersion.
- *Modeling:* We introduce a new, general model for Bitcoin tasks (Scratch-Off Puzzles) and accompanying models of adversarial and economically rational participant behavior and costs. We thereby expand the design space of Bitcoin and offer tools to analyze new Bitcoin variants, including resource-recouping schemes such as Permacoin.

Permacoin recovers a substantial portion of the resources invested in coin-mining infrastructure. Repurposing a larger fraction would be desirable, of course. We hope that Permacoin encourages new techniques for better repurposing, and the harvesting of resources other than distributed storage.

2 Preliminaries and Background

We now introduce some general terminology for Bitcoin and our proposed scheme, Permacoin.

Associated with every epoch in Bitcoin is a unique puzzle ID (denoted puz) known to all participants. In a given epoch, all miners attempt to solve an SOP specified by puz. Solving the SOP involves making random guesses at solutions. An SOP is defined in terms of the following two functions, which any client may compute:

- $\mathsf{Guess}(\mathsf{puz}) \rightarrow \mathsf{ticket}$: A randomized algorithm that generates a candidate solution to the SOP, which we call a *ticket*.
- $\mathsf{IsWinningTicket}(\mathsf{puz}, Z, \mathsf{ticket}) \rightarrow \{0, 1\}$: A function that outputs 1 if ticket represents a solution to the SOP and outputs 0 otherwise. Input Z specifies the level of hardness for the SOP determined by the Bitcoin network.

If $\mathsf{IsWinningTicket}(\mathsf{puz}, Z, \mathsf{ticket}) = 1$, then we call ticket a *winning ticket* for the epoch.

All miners race to generate tickets via Guess until a winning one is found. This ticket is published by the winning miner, who receives coins as a reward. Publication of a winning ticket marks the end of the current epoch and the beginning of a new one.[2] Therefore, the length of each epoch is a random variable. (We model epoch lengths later in the paper.) In the Bitcoin system, an epoch currently has an average length of ten minutes.

Bitcoin specifics. In Bitcoin, solving an SOP validates a recent history of transactions in the system, and is called *mining a block*. A solution is called the *header* for the mined block. The SOP involves repeatedly hashing candidate headers, specifically a prefix puz and a guess ticket (usually called a nonce in Bitcoin specifications), until an image results that is less than a target value Z. (The smaller Z is, the harder the SOP.) The two corresponding functions are:

[2] Ideally, an epoch ends when a winning solution is found. Due to propagation delays in the actual Bitcoin network, however, collisions may occur in which winning solutions compete for publication. This happens rarely, and the resulting temporary fork is usually resolved in the next epoch when a unique winning solution is identified by consensus in the network. The losing block is referred to as a *stale block*. An estimated 2% of produced blocks are stale in Bitcoin [12].

$\mathsf{Guess}(\mathsf{puz}) \xrightarrow{\$} \mathsf{ticket};$

$\mathsf{IsWinningTicket}(\mathsf{puz}, Z, \mathsf{ticket}) = \{\mathrm{H}(\mathsf{puz}||\mathsf{ticket}) \overset{?}{\leq} Z\},$

where H is a hash function (`SHA-256` in Bitcoin).

The header prefix $\mathsf{puz} = v \parallel B_l \parallel MR(x) \parallel T$ includes software version number v, previously mined block header B_l, Merkle-tree root $MR(x)$ over new transactions x, and T, the current time expressed in seconds (since 1970-01-01T00:00 UTC).

A successful miner obtains payment by including in x a payment of freshly generated coins to herself.[3] In general, a miner j identifies herself via a pseudonymous public key pk_j that she publishes to enable verification of her digitally signed transactions.

We defer further details, e.g., on parameter choices, until later in the paper.

Bitcoin design challenge. The Bitcoin SOP was designed to achieve several properties essential to the Bitcoin system:

1. *Predictable effort:* The Bitcoin system adjusts the hardness of mining (via Z) once every 2016 blocks to ensure an epoch length of approximately ten minutes. This ongoing calibration requires a simple and precise characterization of the hardness of mining: To find a ticket satisfying $H(\mathsf{puz}||\mathsf{ticket}) \leq Z$ in Bitcoin simply requires R/Z computations on average, where R is the size of the range of H (2^{256} for `SHA-256`).

2. *Fast verification:* While solving SOPs is resource-intensive in Bitcoin, verification of solutions must be inexpensive, to permit rapid verification of transactions and coin validity by any user. Verifying the correctness of a winning ticket, i.e., that $H(\mathsf{puz}||\mathsf{ticket}) \leq Z$, requires only one hash computation.

3. *Precomputation resistance:* Without knowledge of puz, it is infeasible for a client to execute Guess or otherwise perform useful precomputation of a winning ticket in Bitcoin. As puz relies on transaction history, it cannot feasibly be guessed prior to the beginning of an epoch. In other words, puz may be modeled as a fresh, random value in every epoch.

4. *Linearity of expected reward:* The expected reward per unit of work is approximately constant, even for very small investments of computational effort. This prevents large participants from monopolizing the system and driving out ordinary participants.

[3] In addition to freshly generated coins, miners also receive payment in the form of "fees" attached to each transaction by users. The rate of new coin generation is scheduled to gradually diminish and then (in 200 years) cease. At this point the process of "mining" will nonetheless continue, sustained by transaction fees alone. For simplicity, in the remainder of the paper we use "the reward" to refer to either kind of payment.

There are few known puzzles that meet all of these criteria. Our aim here is to construct a puzzle that *also* satisfies:

5. *Repurposing:* Resources committed to mining can be repurposed for useful tasks that are independent of the Bitcoin system.

Proofs of Retrievability. We address criterion 5 in our proposed scheme Permacoin by recycling storage resources, thus requiring Bitcoin / Permacoin puzzle solvers to store useful data. A successful SOP solution, i.e., winning ticket, in our scheme takes the form of a (partial) Proof of Retrievability (POR).

We refer the reader to [17] for details on PORs, whose use we briefly review here. A basic POR takes the form of a challenge-response protocol in which a Prover P demonstrates its possession of a file F, and the fact that it can be correctly retrieved, to a Verifier V. To audit P's possession of F, V may issue a random challenge c at any time; it receives a response r, which it can verify without possessing F.

There are many variant POR schemes in the literature. (See Section 9.) Let $F := (F_1, F_2, \ldots, F_n)$ denote a dataset consisting of n sequential segments. We make use of a simplified scheme with public verifiability.

- $\mathsf{Setup}(F) \rightarrow (\widehat{F}, \mathsf{digest})$. P encodes F using an erasure code. Then P computes a Merkle tree whose leaves are segments of the encoded F (with their indices) and whose root is digest. Let $\widehat{F}$ denote the encoded form of F and its accompanying Merkle tree.

- $\mathsf{Prove}(\mathsf{puz}, R, \widehat{F}) \rightarrow \{F_{r_i}, \pi_{r_i}\}_{r_i \in R}$. Let $R := \{r_1, \ldots, r_k\} \in [n]^k$ denote a set of random challenge indices selected by V. P outputs a proof that for each challenge index $r_i \in R$, $\widehat{F}$ contains F_{r_i} and the accompanying path π_{r_i} in the Merkle tree.

- $\mathsf{Verify}(\mathsf{digest}, R, \{F_{r_i}, \pi_{r_i}\}_{r_i \in R}) \rightarrow \{0, 1\}$. V validates the Merkle path π_{r_i} for each segment F_{r_i} against digest.

PORs provide a strong guarantee, namely that with overwhelming probability, if P provides correct responses, F can be retrieved *completely* from P. That is, thanks to erasure-coding, every bit of F can be recovered.

Our adaptation of PORs for Permacoin, however, differs from previously proposed PORs in two main ways. First, in our case, V is the entire Bitcoin / Permacoin network. Unlike previous schemes, however, we let the challenge c be generated *non-interactively* by a client executing Guess.

Second, in our setting, every client can act as a prover (if it successfully mines blocks). So the number of possible provers is large (thousands of clients). Additionally, the target dataset F is quite large, so each client j holds only a portion of F. Thus we distribute F across multiple provers. The POR generated by a prover / client is *partial*, in the sense

that it covers only the client's stored portion of F. While distributed PORs have been previously explored in, e.g., [4], previous schemes have involved small numbers of provers acting synchronously. In our setting, not only is the number of provers large, but proofs are generated asynchronously, as a byproduct of mining.

Another distinctive feature of our setting is that most PORs aren't explicitly verified. In order to *try to solve* an SOP, a client must generate PORs on its blocks of F. But a client that never mines a block successfully may never release any POR. In this sense, PORs *implicitly* incentivize correct storage: a client stores F_i in order to be able to generate a correct POR, whether or not the POR is ever in fact verified.

In Section 4, we specify how POR functions are integrated into Guess and IsWinningTicket to construct Permacoin's SOP.

3 Security Assumptions

In the Bitcoin system, clients are pseudonymous. Each client j has a key pair $(\mathsf{sk}_j, \mathsf{pk}_j)$ used to sign (validate) her transactions. Additionally, there are no pre-established identities, and clients may create new identities as desired at any time. We retain these properties in our proposed scheme.

Permacoin makes use of an extremely large, *high-entropy*, [4] archival data file F (notionally, F may be 1 petabyte in size). Consequently, F is too large for storage by individual peers, which must instead store fragments of F. A benefit of distributing F across peers, however, is that it becomes more durable, i.e., able to survive infrastructure damage, as specifically desired for high-value files. Even if a fraction of peers go offline or behave maliciously, F remains recoverable.

Globally, therefore, our security goal is to preserve F even in the face of failures, benign or malicious, of a fraction of clients in the Bitcoin network.

We make the following three important assumptions in Permacoin that are distinct from those in Bitcoin.

File distribution. We assume (for the sake of presentation) that F emanates from a single authoritative *dealer* that digitally signs file blocks. We assume that newly created clients can download fragments of F on demand. The dealer might remain continuously online to serve these blocks. In this case, storage of F in Bitcoin provides a hedge against a dealer failure. (For example, the Library of Congress might serve out its collection F. But in case of a failure, as during the recent U.S. government shutdown [18], F would be recoverable from the Bitcoin network.)

In brief, we assume that fragments of an authoritatively generated file F may be downloaded on demand by clients. We make no further assumptions of centralization. In practice, we imagine that the functionality of the trusted dealer will be provided by the Bitcoin network itself; we discuss possible mechanisms for this in Section 8.

Limited adversary. We assume an adversary that controls a small (minority) fraction of clients. Given a recent result [15] showing that an adversary that controls 1/4 of clients can subvert the Bitcoin network, limited adversarial control is a fundamental requirement of Bitcoin, and not just of our scheme. We assume that the remaining clients act independently and, in particular, assume that they behave in an economically rational manner. That is, they seek to maximize their gain in Bitcoin mining and limit their resource investment.

Local private-key storage. We assume that a substantial fraction of clients do not share their private signing keys with external entities. It is the signing key sk_j of a client j that entitles him to the reward of Bitcoins associated with successful block mining. Consequently, sharing this key means sharing a client's coins—and exposing them to theft with no ability to trace them and thus no recourse or indemnification should a provider be breached or embezzle coins. Additionally, many Bitcoin miners today perform mining on special-purpose local devices, such as ASIC miners [22]. Thus, we assume that a substantial fraction of clients in the Bitcoin network store their private keys locally.

We construct our SOP such that efficiently solving it requires *continuous use of a miner's private key*. Given our assumption of local key storage, therefore, we are able to show that clients in Permacoin perform mining locally and thus that fragments of F are distributed across distinct clients and enjoy the full physical distribution and robustness of a true peer-to-peer network.

4 Scheme

Our idea, at a high level, is to build a scratch-off-puzzle out of a Proof-of-Retrievability, such that the only way effective way to solve the puzzle is to store portions of the public dataset. In the following sections, we describe how we design the puzzle to ensure that **(a)** users reliably store a subset of the data, **(b)** participants assign themselves mostly non-overlapping subsets of data to ensure good diversity, and **(c)** the entire dataset is recoverable with high probability from the contents of participants' local storage devices.

4.1 A Simple POR Lottery

To reduce the energy wasted by Bitcoin's current proof-of-computation lottery, we propose replacing it in Permacoin with a POR lottery. In a POR lottery, every scratch-off attempt can be associated with the effort of computing a POR. There are at least two issues that must be addressed:

- *Choosing a random subset of segments based on each participant's public key.* Since each participant may not have sufficient storage to store the entire dataset, we have each participant choose a random subset of segments of the data to store, based on the hash of their public key.

[4] We must assume that F is high-entropy (i.e., that it has been optimally compressed) and that no party can efficiently recompute "on-the-fly" a significant fraction of the file (as would be the case, for example, if the file contained the output of a pseudorandom function for which some party knows the seed). We thank Sarah Meiklejohn for pointing out this implicit assumption in the original version of this paper.

- *Non-interactive challenge generation.* In traditional PORs, a verifier sends a random challenge to a prover, and the prover answers the challenge. In our system, the verifier is the entire Bitcoin network, and the challenge must be generated non-interactively.

  Thus, we have the participants generate challenges based on the publicly known epoch-dependent puzzle ID puz. A valid challenge is computed as $H(\text{puz}||s)$ for some string s of the prover's choice.

Our strawman protocol is described in Figure 1.

4.2 Local-POR Lottery

One drawback of the strawman POR lottery (Figure 1) is that it does not incentivize *distributed* storage, which undermines our goal of long-term, resilient data-storage.

In particular, participants can potentially benefit from economies of scale if they outsource the puzzle solving process to a cloud server, including the storage and computation necessary. In fact, several companies have begun to offer hosted Bitcoin mining services [6].

If most users outsource the puzzle solving process to the cloud, then Permacoin's distributed computational and storagee network would effectively become centralized with a few companies. To increase our resilience against correlated disasters, we wish to increase the geographical diversity of the storage. Therefore, we wish to disincentivize users from outsourcing their storage to cloud providers.

We now propose a new *local*-POR lottery mechanism (see Figure 2) that discourages users from outsourcing puzzle solving to the cloud.

Idea 1: Tie the payment private key to the puzzle solution. Our first idea is to tie to the puzzle solution to the private key to which the lottery reward is paid out. This key must be kept private in order to claim the reward for oneself. By tying the payment private key to the puzzle solution, a user must reveal her private key to the cloud if she wishes to reduce her own costs by outsourcing the puzzle to the cloud.

As mentioned earlier (Section 3), we assume that at least a fraction of the users will choose not to entrust the cloud with their payment private keys.

Idea 2: Sequential and random storage access. We also need to discourage a user from outsourcing storage to the cloud, but performing computation locally on her own machine. To achieve this goal, we craft our puzzle such that access to storage is sequentialized during the scratch-off attempt. Furthermore, the storage access pattern is random (based on outcomes of calling a random oracle) and cannot be precomputed ahead of time. Thus, if the data is stored in the cloud and the computation is performed locally, many round-trips must be incurred during the scratch-off attempt, which will reduce the user's chance of finding a winning ticket.

Boosting recoverability with erasure codes. As in standard proof-of-retrievability schemes, we boost the probability of successful recovery through erasure coding. In the setup phase, we erasure code a dataset containing f segments into rf segments, where $r > 1$, such that any f segments suffice to recover the dataset.

4.3 Floating-Preimage Signature Scheme

For the signing operation in Figure 2, one simple option would be to use the commonly adopted RSA signature scheme. However, RSA signatures (and most other commonly-used signatures) require asymmetric operations, which impose a high computational overhead. This is undesirable for our purposes; for every dollar a participant invests in mining equipment, we would prefer as much of it as possible to be spent on storage devices (which simultaneously provide additional utility through our scheme) rather than signature-computing processors (which do not). From a back-of-the-envelope calculation, if an RSA signature were used, and if we choose parameters to ensure half of each invested dollar is spent on storage, then each puzzle iteration would have to add a half-megabyte of data to the corresponding proof (see Section 7), which would place an impractical burden on the Bitcoin network.

Instead, we propose a highly efficient multi-use hash-based signature scheme which we call a *floating preimage signature* (FPS). This scheme is an instance of the generalized Bos-Chaum signature scheme [30], to which we refer for a better description of standard hash-based signatures. In brief, the secret key consists of a set of randomly generated strings; the public key is the root digest of a Merkle tree with these strings at the leaves. A message is signed by pseudorandomly selecting a subset of leaves to reveal.

Our puzzle requires a multi-use signature scheme allowing $k+1$ signed messages, where the first k are performed during the k iterations of the scratch-off, and the additional $(k+1)$-th is used to spend the Bitcoin reward after successful mining. However, we cannot directly employ any standard multi-use signature scheme (e.g., those based on Merkle signatures [30]), since we require a special non-outsourceable property.

[Non-outsourceability:] *During a scratch-off, successfully computing the k signatures necessary for a ticket requires possession of a large fraction of the private keys (preimages), such that with high probability, anyone able to produce k signatures will be able to produce the $(k+1)$-th signature (used to pay the reward).*

Our basic, *stateful* FPS signature scheme is illustrated in Figure 3 and defined in Figure 4. Next we describe the modifications needed to achieve the non-outsourceable property.

Adjustments to the basic FPS scheme. Based on the basic scheme in Figure 4, we make the following modifications to ensure non-outsourceability, i.e., to ensure that the server can make the $(k+1)$-th signature with high probability if it can successfully sign the first k messages during the scratch-off. Particularly, for the $(k+1)$-th message, we allow the server (i.e., signer) to reveal any q' out of $4q'$ randomly chosen unused leaves to produce a valid signature.

- **Setup.** The dealer computes and publishes the digest of the entire dataset, consisting of n segments

  A participant with public key pk chooses a subset S_{pk} of segments to store:

$$\forall i \in [\ell] : \text{ let } \mathbf{u}[i] := H_0(\mathsf{pk}||i) \mod n, \qquad S_{\mathsf{pk}} := \{\mathbf{u}[i]\}_{i\in[\ell]}$$

  where ℓ is the number of segments stored by each participant. The participant stores $\{(\mathbf{F}[j], \pi_j)|j \in S_{\mathsf{pk}}\}$, where π_j is the Merkle proof for the corresponding segment $\mathbf{F}[j]$.

- **Scratch-off.** Every scratch-off attempt is seeded by a random string s chosen by the user. Let puz denote a publicly known, epoch-dependent, and non-precomputable puzzle ID. A participant computes k random challenges from its stored subset S_{pk}:

$$\forall i = 1, 2, \ldots, k : \quad r_i := \mathbf{u}[H(\mathsf{puz}||\mathsf{pk}||i||s) \mod \ell] \tag{1}$$

  The ticket is defined as:

$$\mathsf{ticket} := (\mathsf{pk}, s, \{\mathbf{F}[r_i], \pi_i\}_{i=1,2,\ldots,k})$$

  where π_i is the Merkle proof for the r_i-th segment $\mathbf{F}[r_i]$.

- **Verify.** The Verifier is assumed to hold the digest of F. Given a $\mathsf{ticket} := (\mathsf{pk}, s, \{\mathbf{F}[r_i], \pi_{r_i}\}_{i=1,2,\ldots,k})$ verification first computes the challenged indices using Equation (1), based on pk and s, and computing elements of $\mathbf{u}$ as necessary. Then verify that all challenged segments carry a valid Merkle proof.

Figure 1: A simple POR lottery.

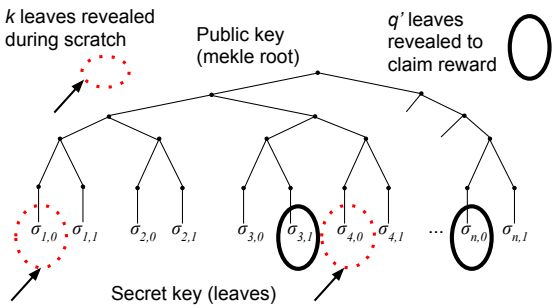


Figure 3: Illustration of the floating preimage signature signature scheme.

Lemma 1. *Let q denote the number of leaves revealed for the first k signatures. For the $(k+1)$-th signature, suppose that q' out of $4q'$ leaves must be revealed. Let the number of leaves $L = 2kq + 8q'$.*

Suppose that the client reveals $L/2$ or more leaves to the server. Then, 1) the server can sign the $(k+1)$-th message except with negligible probability; and 2) anyone else without knowledge of the leaves cannot forge a valid $(k+1)$-th signature for a different message (except with negligible probability).

Proof. (sketch.) For the final, $(k+1)$-th message, $4q'$ unused leaves are drawn randomly as a challenge using the hash function, and the server is allowed to select any q' of these to reveal. We know that among the $4q'$ leaves, the server in expectation has $2q'$ of them, since it knows half of the leaves. It is not hard to show using a Chernoff-like bound that the server can successfully produce the final signature (containing q' out of $4q'$ signatures). Based on this argument, for the server to succeed in signing during scratch-off with any non-negligible probability, it must know enough of the leaves to be able to sign the $(k+1)$-th message with overwhelming probability.

It is not hard to show that the probability the $(k+1)$-th signature happens to be a valid signature for any other given message is negligible in the security parameter if $q' = O(\lambda)$. In particular, to compute the probability for the $(k+1)$-th signature to be a valid signature for a different message, consider the probability that a randomly chosen $4q'$ leaves out of $8q'$ contains q' specific leaves contained in the $(k+1)$-th signature. By a Chernoff bound, this probability is $\propto \exp(-cq')$ for some appropriate constant $c > 0$. □

Parameterizations and security. In order for this to be a secure, unforgeable signature scheme for all $k+1$ messages, we can set $L = 2kq + 8q'$, $q = O(\lambda)$ and $q' = O(\lambda)$. The proof of this is standard and deferred to our online full version. [26]

However, we observe that the first k signatures (performed during scratch off) actually need not be unforgeable signatures. In fact, due to the above Lemma 1, we just have to set our parameters such that any rational user will store at least $L/2$ leaves on the server.

Therefore, in practice, we can set $q = 1$ for all the internal signatures during scratch off. However, for the $(k+1)$-th signature, we set $q' = O(\lambda)$, and the signer must choose $4q'$ leaves and reveal any q' of them. In this case, if the client withholds $L/2$ leaves from the server, the server must in expectation contact the client $k/2$ times during the scratch-off attempt – in Section 5, we show that the cost of transmitting even small packets of data greatly exceeds the cost of simply

- **Setup.** Let $r > 1$ denote a constant. Suppose that the original dataset F_0 contains f segments.

  First, apply a maximum-distance-separable code and encode the dataset F_0 into F containing $n = rf$ segments, such that any f segments of F suffice to reconstruct F_0. Then, proceed with the **Setup** algorithm of Figure 1.

- **Scratch-off.** For a scratch-off attempt seeded by an arbitrary string s chosen by the user, compute the following:

$$\begin{aligned}
&\sigma_0 := 0 \\
&r_1 := \mathbf{u}[H(\mathsf{puz}||\mathsf{pk}||s) \mod \ell] \\
&\text{For } i = 1, 2, \ldots, k: \\
&\quad h_i = H(\mathsf{puz}||\mathsf{pk}||\sigma_{i-1}||\mathbf{F}[r_i]) \quad (*) \\
&\quad \sigma_i := \mathsf{sign}_{\mathsf{sk}}(h_i) \\
&\quad r_{i+1} := H(\mathsf{puz}||\mathsf{pk}||\sigma_i) \mod \ell
\end{aligned}$$

  The ticket is defined as:

$$\mathsf{ticket} := (\mathsf{pk}, s, \{\mathbf{F}[r_i], \sigma_i, \pi_{r_i}\}_{\forall i=1,2,\ldots,k})$$

  where π_{r_i} is the Merkle proof of $\mathbf{F}[r_i]$.

- **Verify.** Given $\mathsf{ticket} := (\mathsf{pk}, s, \{\mathbf{F}[r_i], \sigma_i, \pi_{r_i}\}_{\forall i=1,2,\ldots,k})$, verification is essentially a replay of the scratch-off, where the signing is replaced with signature verification. This way, a verifier can check whether all iterations of the scratch-off were performed correctly.

Figure 2: Local-POR lottery.

computing scratch-off iterations locally. Therefore, a rational user would not outsource its computation yet withhold $L/2$ or more leaves.

5 To Outsource or Not to Outsource

As mentioned earlier, since we tie possession of newly minted coins to a user's private key in Permacoin, we assume that a substantial fraction of users will *not* entrust their private keys to a service provider and risk theft of their coins.

A user j who only stores her private key sk_j locally can choose between two ways of storing her assigned blocks of F: a *local storage* device or *outsourced storage* leased from a remote cloud storage service. (A combination of the two is also possible.) We now analyze the storage choice of rational participants, those seeking to maximize their return on mining by achieving the lowest expected cost per SOP. We argue that rational users will choose *local storage* to drive down their resource costs.

In both the local storage and outsourced storage scenarios, the user locally provisions a basic computational resource (incurring the hardware costs of a motherboard, CPU, and RAM and power costs, but not of a substantial storage medium). The cost differences for the two storage scenarios—again, favoring local storage—stem from the following:

Cost of Storage and I/O: In the local scenario, a client's costs are its investment in storage equipment for mining, specifically, for the purchase of RAM or SSD. (These costs may be characterized in terms of equipment depreciation.)

In the outsourced scenario, a client's costs include the: 1) Cost of storage and disk I/O charged by the service provider; 2) Cost of network bandwidth, including that of the network link provided by an ISP, and the cost per GB of network transfer charged by a service provider. In our setting, storage of the file F can be amortized across multiple users, so we assume the storage cost and disk I/O cost are close to zero. What remains is the cost of network I/O.

We show that based on typical market prices today, the costs of storage and I/O are significantly cheaper for the local storage option.

Latency: By design, our SOP sequentially accesses blocks in F in a random (pseudorandom) order. The resulting, unpredictable fetches penalize outsourced storage, as they introduce substantial latency: a single round-trip for every fetched block, which is vastly larger than disk I/O latency. This latency overhead reduces a miner's chance of finding a valid puzzle solution and winning the reward when the number k of outsourced fetches is large.

If each block fetch incurs roundtrip latency τ, then for large k, the total incurred latency $k\tau$ may be quite large. For example, with $k = 6000$, one iteration parameter we analyze below, and a 45ms roundtrip latency, typical for regional internet accesses, $k\tau$ would be 4.5 minutes—almost half the length of an epoch. Boosting to $k > 13,333$ would render $k\tau$ *larger* than an average epoch, making outsourcing infeasible.

Of course, if $k\tau$ is small enough, a client can parallelize fetches across SOP guesses. It is helpful to quantify formally the value of time, and penalties associated with latency, as we do now.

5.1 Stochastic model

We now present a stochastic model that offers a quantitative comparison of the economics of local vs. outsourced storage.

- $\mathsf{KeyGen}(y, \ell)$: Let L denote the number of Merkle leaves. Pick $\{\sigma_1, \ldots, \sigma_L\}$ at random from $\{0,1\}^{O(\lambda)}$. Construct a Merkle tree on top of the L leaves $\{\sigma_1, \ldots, \sigma_L\}$, and let digest denote the root's digest.

  The secret key is $\mathsf{sk} := \{\sigma_1, \ldots, \sigma_L\}$, and the public key is $\mathsf{pk} := \mathsf{digest}$. The signer and verifier's initial states are $\Omega_s = \Omega_v := \emptyset$ (denoting the set of leaves revealed so far).

- $\mathsf{Sign}(\mathsf{sk}, \Omega_s, m)$: Compute $H(m)$ where H is a hash function modeled as a random oracle. Use the value $H(m)$ to select a set $I \subset \mathsf{sk} - \Omega_s$ of q unrevealed leaves. The signature is

$$\sigma := \{(\sigma_i, \pi_i)\}_{i \in I} \text{ in sorted order of } i$$

  where π_i is the Merkle proof for the i-th leaf. Update signer's state $\Omega_s := \Omega_s \cup I$.

- $\mathsf{Verify}(\mathsf{pk}, \Omega_v, m, \sigma)$: Use $H(m)$ to select an unused set I of leaves, of size q. Parse $\sigma := \{(\sigma_i, \pi_i)\}_{i \in I}$ (in sorted order of i). Verify that each π_i is a correct Merkle proof for the i-th leaf node σ_i where $i \in I$. Output 1 if all verifications pass. Otherwise output 0.

  Finally, update verifier's state $\Omega_v := \Omega_v \cup \sigma$.

Figure 4: An FPS signature scheme.

Table 1: Notation used for system parameters

f	# segments necessary for recovery
m	total # segments stored by good users during recovery
n	total # encoded segments
ℓ	# segments assigned to each identity
k	# iterations per puzzle
B	# size of each block (bytes)

The notation used for the parameters of our scheme are summarized in Table 1.

We consider a stochastic process in which a single-threaded mining process is trying to find a ticket. This mining thread will keep computing the iterations sequentially as described in Figure 2. At any time, if another user in the network finds a winning ticket first, the current epoch ends, and the mining thread aborts the current scratch-off attempt and starts a new attempt for the new epoch.

We consider the following cost metric: *expected cost invested until a user succeeds in finding one ticket*. Every time a user finds a ticket (before anyone else finds a winning ticket), the user has a certain probability of having found a winning ticket, and hence being rewarded.

Game with a giant. We can think of this stochastic process as a user playing a game against a *giant*. The giant models the rest of the network, which produces winning tickets at a certain rate. The stochastic process in which the giant produces winning tickets is a *memoryless* process. At any time, the remaining time T it takes for the giant to find a winning ticket follows an exponential distribution. The expectation of T is also the *expected epoch length*. In Bitcoin, as noted, the difficulty of its SOP is periodically adjusted with respect to the computational power of the network to keep the expected epoch length at about 10 minutes.

If the giant generates a puzzle solution, it is immediately communicated to the user, who aborts her current attempt. Thus the stochastic process can be modeled as a Markov chain as follows:

- Every iteration takes t time, and costs c.
- If k iterations are finished (before the giant wins), a user finds a ticket (which may or may not be a winning ticket). In this case the user gets a positive reward in expectation.
- Let s_i denote the state in which the user has finished computing the i-th iteration of the puzzle.
- If $i < k - 1$: with probability p, the giant does not win in the current iteration, and the state goes from s_i to s_{i+1}. With probability $1 - p$, the giant wins, and the state goes back to s_0, i.e., the current epoch ends, and a new scratch-off attempt is started. Suppose that the expected epoch length is T; then it is not hard to see that $p = 1 - t/T$ given that the stochastic process of the giant winning is memoryless.
- In state s_{k-1}, with probability 1, the state goes back to s_0. Furthermore, in state s_{k-1}, with probability p, the user will finish computing all k iterations — in which case another random coin is drawn to decide if the ticket wins.

We analyze the stationary distribution of the above Markov chain. Let π_{k-1} denote the probability that it is in state s_{k-1}. It is not hard to derive that $\pi_{k-1} = (1-p)p^{k-1}/(1-p^k)$. Therefore, in expectation, $1/\pi_{k-1}$ time is spent between two visits to the state s_{k-1}. Every time the state s_{k-1} is visited, there is a p probability that the user will finish all k iterations of the puzzle. Therefore, in expectation, $1/(\pi_{k-1}p)$ time (in terms of number of iterations) is spent before the user finds a

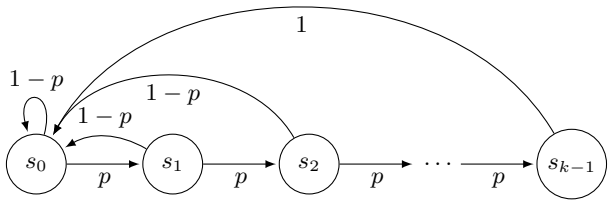


Figure 5: Markov chain model for a sequential mining process that resets if the epoch ends during an iteration.

ticket before the giant does. If a user finds a ticket before the giant does, we call this a "success". Hence, we have that

$$\mathbf{E}[\text{expected cost per success}] = \frac{c(1-p^k)}{(1-p)p^k}$$

5.2 Local Storage vs. Outsourced Storage

Based on the above analysis, we now plug in typical practical values for the parameters and investigate the economics of local vs. outsourced storage.

Local storage. The cost of a scratch-off attempt depends on two things, the power consumed and the cost of the equipment. We consider two hardware configurations,

1. with SSD drives as the storage medium; and
2. using RAM as the storage medium.

Both are typical configurations that an amateur user can easily set up. Note that while it is possible to optimize the local hardware configuration further to have better amortized cost, it is outside the scope of this paper to do so, since our goal is to show that, even for an amateur user, local mining is economically superior to outsourced storage mining.

First we estimate the cost of local mining using an SSD and standard CPU. Today, the cost of a desktop containing a high-end processor (Intel Core i7, 3.4GHz and 8 virtual cores) is approximately $500. The cost of a 100GB SSD is about $100. Amortized over three years, the effective cost is $6.34e$-6 $/second. We measured the power consumption while mining to be about 40 watts; assuming an electricity cost of 15 cents/kWh, the energy cost of mining is $1.67e$-6 $/second in power. Note the mining cost is dominated by equipment, not power. The latency for a single disk read of up to 4096 bytes is measured at approximately 30 microseconds.

We assume for now that the size of a file segment is 64 bytes, and every puzzle iteration requires hashing a single leaf with two 120-bit secrets ($y = 1$). Computing a hash over a message of less than 128 bytes takes no more than ~ 15 microseconds on an ordinary CPU, suggesting that for a single-threaded mining program, the SSD and CPU would be in approximately equal utilization. Thus assuming an average of 30 microseconds per iteration, the cost of mining with a local SSD is roughly $3.2e$-10 $/iter.

Next we consider the cost of local mining using RAM rather than an SSD. A 2GB stick of DDR3 SDRAM can be purchased for about $20, and has a data transfer rate of $12,800$ megabytes per second. Assuming a segment size of 64 bytes, the average throughput of this memory is approximately 200 million puzzle iterations per second. This is faster than a single-threaded CPU performing signing operations can keep up with. On the other hand, many desktop computers have a graphics processor (GPU) that can be used to accelerate Bitcoin mining. Taking one example, the ATI Radeon 6750 costs $100, consumes 150 watts, and can perform 150 million Bitcoin hashes per second. Thus, under this scheme the GPU would be utilized approximately as much as the RAM.

Outsourced storage. The cost of outsourced storage mining may vary according to the pricing of the specific service provider. Our goal is to show that under most conceivable scenarios for outsourced mining, local mining will be superior. To demonstrate this, we consider a wide spectrum of cost ranges for the outsourced storage setting, and show that even when we unfairly favor the outsourced option by assuming aggressive lower bounds for its cost, the local option is still more more economical.

We consider multiple cost configurations for the outsourced storage option:

1. *EC2.* First, we rely on the pricing representative of today's high-end cloud providers. In particular, our estimates are based of Amazon EC2's pricing. EC2 charges 10 cents per gigabyte of transfer, and a base rate of 10 cents for the smallest virtual machine instance.

2. *Bandwidth + CPU.* Amazon EC2's setup is not optimized for constant-use high-bandwidth applications. Other rental services (such as `http://1gb.com/en/`) offer "unmetered" bandwidth at a fixed monthly cost. To model this, we consider a cost lower bound by assuming that the cloud provider charges nothing, and that the user only needs to pay for its local CPU and the bandwidth cost charged by the ISP.

   Internet transit costs are measured in $ per mbps, per month. Costs have diminished every year; the median monthly cost of bulk bandwidth during 2013 has been estimated at $\$1.71/mbps$, corresponding to 0.53 cents per gigabyte under constant use.[5] Each puzzle iteration requires transferring a file segment.

   Since the SSD accounts for about 16% of the equipment cost in the local SSD configuration, and the CPU is approximately matched with the SSD in terms of utilization, for this model we assume that the latency is equiv-

[5] According to an October 2013 press release by market research firm TeleGeography: `http://www.telegeography.com/press/press-releases/2013/10/08/ip-transit-port-upgrades-yield-steeper-price-declines-for-buyers/index.html`

alent, but reduce the local equipment and power cost by 16%.

3. *CPU only or bandwidth only.* We consider an even more aggressive lower bound for outsourcing costs. In particular, we consider a scenario in which the user only needs to pay for the local CPU; or she only needs to pay the ISP for the bandwidth.

   While this is not realistic today, this lower bound models a hypothetical future world where cloud costs are significantly lowered, or the scenario where a powerful adversary can reimburse users' mining costs assuming they join its coalition.

Findings. Table 2 compares the costs of local mining to those of outsourced storage.

Notice that in our protocol in Figure 2 one tunable parameter is the number of bytes that must be transferred between the server and the client per iteration if storage were to be outsourced to a server. In general, when more bytes are transferred per iteration, the bandwidth cost per iteration also increases. In Table 2 we assume a conservative parameter setting where only 64-byte segments are transferred.

Although latency is listed in the second-leftmost column, the effect of latency is not accounted for in the rightmost Total cost column, since this depends on the number of iterations of the puzzle. Figure 6 illustrates that cost effectiveness diminishes when the number of iterations is increased sufficiently. The figure suggests that under almost all scenarios, local mining is strictly more economical than outsourcing storage, regardless of the number of iterations k for the scratch-off attempt. We stress that this is true even when 1) the local mining user did not spend too much effort at optimizing its hardware configuration; and 2) we give the outsourced storage option an unfair advantage by using an aggressive lower bound for its costs. Recall that local mining saves in cost for two reasons: 1) local storage and I/O costs less than remote (in the latter case the client has to pay for both the storage, disk I/O, and network bandwidth); and 2) lower storage I/O latency gives the user an advantage in the stochastic lottery against the "giant".

The only exception is the "CPU only" entry in Table 2 — in this case, the user is not paying anything for bandwidth, and the only cost is for the CPU hashing operation. In this case, the cost per iteration is lower for the outsourced option than for the local CPU/SSD option (though even here GPU/RAM with local storage remains more efficient). However, longer roundtrip latency to the remote storage will penalize the user during the mining. Therefore, even in this case, we could discourage outsourced storage by setting k very large (thousands of iterations), so that the effect of longer storage I/O latency dominates. For the rest of our analysis, we do include the price of bandwidth in our model and so small values of k are sufficient.

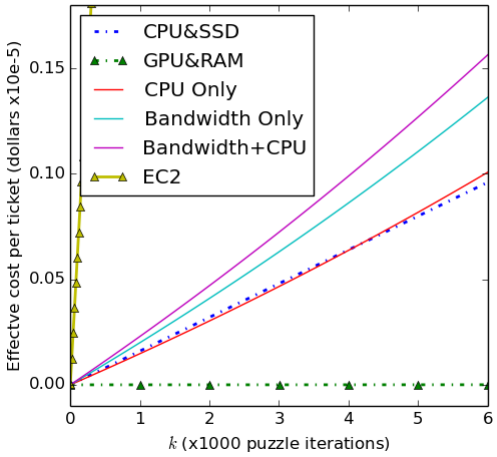


Figure 6: Cost effectiveness versus number of iterations k, for different hardware configurations. Note that for $k > 4e3$ iterations, the CPU/SSD configuration with local storage is more cost effective than the CPU-only (zero-cost bandwidth) with remote storage.

5.3 Partial Storage Analysis

While the above analysis shows that a puzzle-solving configuration using local storage is typically more cost-effective than a configuration using remote cloud storage, we also wish to consider a hybrid-strategy, in which the user chooses to store just a fraction $\gamma < 1$ of her assigned segments on the cloud. When a puzzle iteration calls for one of the remotely stored blocks, the client decides either to fetch it from the server or abort the attempt. For space, we include this analysis only in the full version, and here just describe our result. For all configurations that globally favor local storage in our analysis above, an economically rational client will favor full local storage over any partial local storage and for any hybrid strategy.

6 File-Recovery Analysis

We now give further details on our adversarial model in Permacoin—as well as our assumptions about the behavior of rational players, as motivated by our economic analyses above. Then we present a bound on the probability of recovering the archived dataset F.

Adversarial model. Our adversarial model includes two main assumptions:

- *Bounded control.* An adversary may control a coalition of clients in the network (e.g., through malware, bribery, etc.). We assume, however, that this coalition is of bounded size and, in particular, the adversary cannot control the entire network, i.e., that at least a fraction of clients are uncompromised. We call these clients *rational*.

- *Recoverable file segments.* We assume that among the

Table 2: Costs per iteration for different mining configurations in Permacoin (mining with local storage vs. three forms of cloud storage). Latency is the time to compute one iteration of the puzzle. Effective Latency accounts for the pipelining of computation and storage requests. Equipment is the fixed cost of the system. Total cost per iteration is shown assuming the transfer of a 64-byte segment.

Model	Latency	Eff. Lat.	Equipment	Power	Bandwidth	Total
CPU & SSD	45μs	30μs	\$600	$40W$	n/a	\$2.10e-10/iter
GPU & RAM	600ns	300ns	\$700	$190W$	n/a	\$5.04e-14/iter
EC2	30ms	0	$\$0.10/s$	n/a	\$.10/GB	\$8.39e-7/iter
CPU + BW	30ms	15μs	\$500	$33.6W$	\$5.3e-3/GB	\$4.04e-10/iter
CPU Only	30ms	15μs	\$500	$33.6W$	n/a	\$8.76e-11/iter
BW Only	30ms	n/a	n/a	n/a	\$5.33e-3/GB	\$3.16-10/iter

rational users, a fraction will contribute all their segments in *local storage* during file recovery. We call these clients *good*, and denote their number by g.

In favor of this second assumption, we note that any entity interested in recovering F may be willing to pay users to do so under extraordinary circumstances. (In this sense, Permacoin is a globally distributed analog of the Amazon Glacier service, in which it is cheap to archive data, but relatively expensive to perform recovery.)

Rational-client model. The design of Permacoin incentivizes a rational player to adopt the following behaviors, which we assume in our security analysis:

- *Honestly generated keys.* As discussed in Section 3 economically rational players choose their public/secret keys honestly.
- *Local storage.* As shown in our economic analysis in Section 5, rational clients will store all of their assigned segments locally. Further, we show below that the additional cost of omitting any fraction of the assigned segments increases sharply with the fraction omitted.

Recoverability bound. Rational users will store their assigned segments entirely. Some, however, may not choose to contribute their segments during the process of recovering F. (They may, for instance, be offline or have dropped out of the system.) We now show how to bound the probability of recovery failure under a relatively weak assumption: that the total number of segments contributed by rational players to recovery is at least a constant fraction of all assigned segments.

The proof is omitted due to space limitations, but may be found in the full version of our paper [26]. It proceeds in two steps. First, we first prove recoverability of F with high probability assuming that segments thrown away by users are selected *randomly*. Then, we show that even if these segments are selected arbitrarily (e.g., if the users favor certain segments over others—or even act maliciously), recoverability still holds, except with negligible probability.

Let $m = \ell g$ be the total number of file segments stored by good users, where g is the number of good users, and ℓ is the number of segments stored by each user. The m file segments are chosen randomly from the set of n encoded segments. We model this as throwing m balls into n bins, and we wish to cover $f = (1 - \frac{1}{\alpha})n$ number of bins, where $\alpha > 1$ is some constant related to the coding rate of the MDS code. Let X be the random variable denoting the number of empty bins after m throws.

Lemma 2 (Recoverability under randomly chosen stored set.)**.** *Let $\frac{f}{n} = 1 - \frac{1}{\alpha}$ for some constant $\alpha > 1$. Let $\frac{m}{n} = r > \ln \alpha > 1 - \frac{1}{\alpha}$ for some constant r. Then,*

$$\Pr[\textit{recovery failure}] = \Pr[X > n - f] < \exp(-n\epsilon)$$

where $\epsilon = \epsilon(\alpha, r) > 0$ is some constant dependent on α and r.

The proof (included in the full version of this paper) follows from a standard balls and bins analysis. The lemma can be strengthened to allow the adversary to selectively choose to omit up to a constant fraction of the total segments stored by good users.

7 Parameterization and Microbenchmarks

In this section we provide evidence of the feasibility of Permacoin and suggest how in practice to determine the remaining parameters. We are particularly interested in determining the size of the largest dataset we could reasonably store by repurposing the Bitcoin network. To calculate this value, we assess the storage capacity of the network and determine what verification costs can be tolerated by the network.

In every Bitcoin epoch, a block containing a sequence of transactions and an SOP solution is broadcast to every node in the network. Each node must verify both the transactions and the proof-of-work. Our scheme only affects the SOP solution (this is in contrast to other proposed Bitcoin modifications, such as Zerocoin [25], which leave the SOP unchanged but change the procedure of transaction validation). In addition to validating each new block as it is broadcast, new clients that join the network for the first time, or clients that rejoin the network after some dormant period (e.g., mobile clients), must download and validate all of the blocks generated during some offline period. The maximum tolerable validation cost should certainly be less than 10 minutes (the average epoch

time), since otherwise even the online participants would not be able to keep up. We show reasonable parameter settings for which an ordinary computer can still validate a week's worth of segments in approximately 6 seconds.

Cost of Validation. The cost of validating one iteration of our SOP consists of **(a)** computing the hash of one file segment (equal to $m = F/b$) and two 120-bit secrets, **(b)** verifying a Merkle tree proof for the segment by recomputing $\log_2(rF)$ hashes, and **(c)** verifying a Merkle tree branch for the secrets by computing $\log_2(k + \lambda)$ hashes. Note that the validation cost does not depend on δ, the storage capacity of the network relative to the file.

In Figure 7, we show the validation cost for $k = 20$ iterations of a puzzle using a 20-terabyte file, as a function of the segment size. Figure 7(a) shows the cost in terms of verification time, and Figure 7(b) in terms of proof size.

For a reasonable choice of $r \approx 4$, the cost of validating a proof is approximately 6 milliseconds (based on measured time to compute SHA1 hashes). One new (valid) proof is generated in each epoch; assuming the average epoch time remains 10 minutes, it would take a participant *6 seconds to validate a week's worth of blocks*. For comparison, this is approximately a thousand times more expensive than Bitcoin's current puzzle proof, which requires computing only a single hash (two invocations of SHA256). The proof size in this case would be about 20KB. The average Bitcoin block is approximately 200KB, so our scheme would only increase the average block size by about 10%.

Parameter Choice and ASIC Mining. An underlying assumption about Bitcoin's incentive structure is that a powerful miner cannot earn disproportionately more revenue in expectation than an ordinary individual (i.e., "one-CPU-one-vote" [27]). Because the original hash-based proof-of-work operation is simple, it can be computed efficiently with a small circuit, and the only way to effectively mine faster is to replicate the basic circuit.

During the first few years after Bitcoin's introduction, mining participants primarily used ordinary CPUs, but shortly thereafter transitioned to repurposed graphics cards (GPUs) which provided much more cost-effective hashing power. Now, five years since Bitcoin's introduction, the most cost effective mining is performed using efficient ASICs designed and manufactured solely for the purpose of Bitcoin mining. These ASICs consist of multiple copies of the same circuit on a small chip, which means that the most cost-effective technology remains accessible in small quantities to ordinary users, at approximately a proportional price. The smallest ASIC available on the market today is the ASICMINER "Block Erupter Sapphire", which costs $30, and achieves a cost-effectiveness of 11 megahashes/sec/$. Another ASIC company, Butterfly Labs, also sells a reel of unpackaged chips, for $75 per chip (in minimum lots of 100) at a cost-effectiveness of 53 megahashes/sec/dollar[6]. The feasibility of our system relies on choosing parameters to preserve this economic structure as much as possible.

We model the economics of participation among large and small users by considering cost-effectiveness of configurations attainable given a fixed equipment budget. Our storage-based puzzle alters the economic structure by introducing a base cost: the amount of storage assigned to each identity, m, determines the minimum cost of a suitable storage device. A smaller value of m means users with small budgets can participate more effectively. On the other hand, larger users have little incentive to use additional identities; they could instead achieve proportionally increased mining power by purchasing more devices and filling them with multiple copies of the same dataset segments, thus contributing less to the recovery probability. We would like to set m as large as possible, while preserving the low "barrier-to-entry" that enables ordinary users to participate with the most cost-effective configuration.

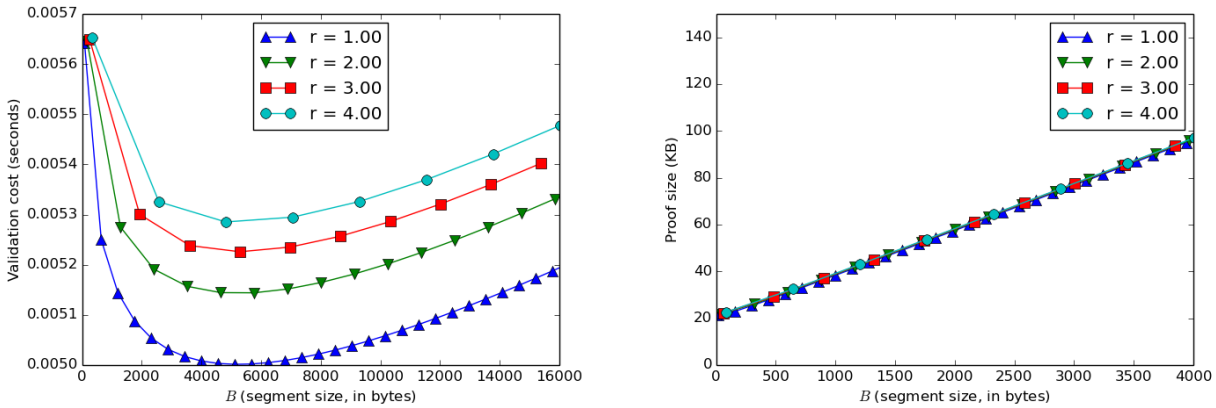
We consider two mining configurations, based on two widely available choices for storage: a) a solid-state storage device SSD, and b) random access memory (DDR3 SDRAM). In either case, it is also necessary to perform hashing operations. Since existing Bitcoin mining ASICs perform essentially the same operation, we can use them as a benchmark: the $30 Block Erupter ASIC could compute approximately 333 million puzzle iterations per second.

The throughput of a high-performance SSD is on the order of 25,000 per second for random reads of 4KB blocks [35], whereas a DDR3-1600 memory module can support 200 million fetches per second (of 64B segments). Thus it would take thousands of SSDs to saturate the computational capacity of a small ASIC, but a single small ASIC would roughly match the fetch rate of one set of RAM.

Next we consider several possible settings for the per-identity storage requirement ℓ. We assume that the most cost effective strategy for $\ell > 2GB$ is to purchase sets of RAM and ASICs in approximately equal amounts (illustrated in Figure 8). This is because we are not modeling the cost of memory buses (i.e., the motherboard), and so two sets of 2GB RAM cost about as much as one set of 4GB of RAM, yet results in twice the total throughput. Thus the effect of increased ℓ is to raise the minimum cost of participation. We believe it would be reasonably consistent with Bitcoin's existing economic structure to choose $\ell = 4GB$, in which case the *minimum equipment investment to participate* with an efficient configuration is only $60.

Storage Capacity of the Permacoin Network. If everyone in the Bitcoin network had always used our scheme rather than the original protocol, investing an equal amount of money to acquire and operate mining equipment, how much storage capacity could we expect to recycle? Miners in Bitcoin are essentially anonymous, so it is difficult to precisely estimate the number of distinct participants and the distribution of the computational power contributed. Nonetheless, we have several available approaches. First, at the time of

[6]See `https://products.butterflylabs.com/homepage-subproducts/65nm-asic-bitcoin-mining-chip.html`



(a) (b)

Figure 7: Estimated cost of validation (in seconds) vs. segment size, for a puzzle with $k = 20$ iterations, using a 20 terabyte dataset F for varying coding rates r.

writing, there are estimated to be about 30,000 known nodes on the network. We may be willing to suppose, as an upper bound, that each node participates an equal amount. However, not all connected nodes necessarily mine at all, and in fact not every miner needs to operate a network-visible node (instead, miners may connect directly to mining pool operators). If each user contributes 4GB of storage, we would have a total capacity of 120 terabytes.

Another approach is to observe the overall hashing capacity ("hashpower") of the network and estimate the amount of money spent on mining equipment. The current hashpower of the network is 4e15 hashes per second. The most cost-effective currently available mining equipment is the line of Avalon ASICs, whose costs is approximately 50e6 hashes per second per dollar. As a lower bound, if everyone used these ASICs, then at least $80 million dollars has been spent on equipment.[7] If this infrastructure investment were entirely redirected toward SSD drives, assuming $70 for 100GB, the result would be a total network storage capacity of 100 petabytes. The cost-density of RAM is lower ($20 for 2 gigabytes); given an approximately equal investment in hashing devices to saturate it, the result would be an overall network storage capacity of 4 petabytes.

Feasibility example. Assuming we've identified the storage capacity of the network – suppose we take it to be 1 petabyte — what is the largest size file we could hope to safely store? Could we store 208 TB of data (one popularly cited estimate of the storage requirement for the print collection of the Library of Congress [24])? We find that yes, using our scheme this is possible. Using parameters $r = 4$, and assigning segments in contiguous segments of 1 MB, we achieve a validation cost of less than 3 milliseconds and an error probability less then $e^{-10^{10}}$.

[7] This is an underestimate, since older FPGA and GPU mining equipment contributes less proportionally (relative to its cost) to the overall hashpower. Also, although some hashpower also comes from the "idle cycles" of ordinary CPUs, the extreme disparity in cost effectiveness between ASICs and ordinary CPUs leads us to conclude that the vast majority of mining income is awarded to participants with dedicated hardware.

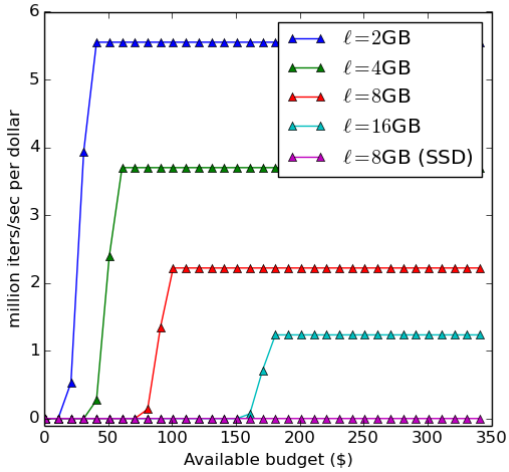


Figure 8: Cost effectiveness for various per-identity storage requirements (ℓ), as a function of per-user equipment budget.

8 Enhancements and Discussions

Stealable Puzzles. In Permacoin, local storage of private keys is essential to ensure strong physical distribution of data. We assumed in our adversarial model that users do not share their private keys in order to outsource mining to an external entity, as this entity could then steal a user's mined coins. However, there are numerous defenses the user may employ against this theft, which make it less effective as a deterrent. For the service provider to steal the reward, it would have to reveal the user's public key, which would alert the user and lead to a race condition as both try to get their version of

the block accepted. Additionally this would allow the user to provide evidence of the server's theft, tarnishing the server's reputation, and potentially enabling legal contract enforcement. A variant of our scheme uses generic zero-knowledge SNARKs (e.g., [29]) to nullify these defenses, allowing any server to steal the reward and evade detection. In the full version of our paper, [26] we provide the details of this construction and show that it adds no overhead for ordinary participants, yet is practical as a deterrent against outsourcing.

Supporting Updates. So far we have considered the archiving of a static dataset. While it is plausible that archival data may not often need to change, we would prefer to incorporate additional data incrementally, and to garbage collect data that is no longer desired. We describe some standard techniques that can be added to our system to efficiently support updates.

We first consider the case where some file segments update, but the file size remain unchanged. Later, we will discuss how to support additions and deletions.

Incrementally updatable erasure codes. Using a standard maximum-distance-separable erasure code, updates to a single segment may result in a constant fraction of the encoded segments needing to be updated. This would not only result in high computational cost during updates, but would also require each user responsible for storing the updated segment to download the new segment.

To address this issue, we could instead employ an incrementally updatable erasure code [7, 34]. One suitable scheme [34] relies on the following main idea: updates are not applied to the main encoded file, but instead are aggregated in a hierarchical log that is merged incrementally. Using a special FFT-like encoding scheme, each update results in updates to $O(\log n)$ segments in an amortized sense. Standard deamortization techniques for such hierarchical data structures can be used to achieve $O(\log n)$ worst-case cost instead of amortized cost, at the cost of a constant-factor larger storage requirement. Using this scheme, for each update, each user would have to update $O(\frac{\ell \log n}{n})$ segments on average, where ℓ is the number of segments allocated to a user. In other words, a user needs to download a new segment every $O(\frac{n}{\ell \log n})$ update operations. On average, the total storage utilized in these schemes at any time is *over-provisioned* by a factor of two compared to the size of the dataset.

Decentralized Update Approval. One option would be to allow the trusted dealer to approve each update, and to sign each new Merkle root digest resulting from adding (or modifying) a segment to the data set. A more satisfying way would be to rely on the majority voting mechanism already inherent to Bitcoin. Specifically, Bitcoin provides a mechanism by which a majority of participants vote on transactions that extend an append-only log; users attach fees to their transactions in order to pay the participants to include them. It would be natural to extend this mechanism to approve updates, so that users would pay fees to add new data to the global data set; additionally, segments could be removed if periodic fees are not added.

Upon on approval of an update, the trusted dealer (or its distributed instantiation by Bitcoin majority voting), must compute the new digest of the dataset. Using an incrementally updatable erasure code [7, 34], updating each segment results in updates to $O(\log n)$ encoded segments, resulting in $O(\log^2 n)$ cost for the Merkle tree digest update.

Note that the updates added by a user are public; if privacy is desired, the user should use a standard encryption algorithm and only publish ciphertexts. To avoid linking multiple ciphertexts to the same origin, the user should connect to the network via an anonymizing overlay such as Tor [13], using a separate circuit for each update.

Encouraging Honest Identity Generation. As mentioned in Section 6, our security holds when rational users choose their identities honestly, leading to a random (rather than adversarially chosen) storage assignments. We therefore wish to design a system that provides the right incentives. Our idea is to associate a proof-of-work with the identity generation process, thus making selective identity generation more costly. We discuss this approach in more detail in the full version of this paper.

Encouraging Better Use of Additional Storage. Although we have set the per-identity storage requirement ℓ in Permacoin small enough so that an ordinary user can afford to participate, we expect that many users will purchase additional storage devices to participate competitively. We would like to encourage them to store many different segments, rather than multiple copies of the same assigned segments, thus increasing the diversity of storage. It would seem difficult to achieve such an incentive while preserving the linearity-of-expected-reward requirement. Our approach here is to suggest that the behavior of Bitcoin participants may in fact be better predicted by *prospect theory* [36] rather than *expected utility*; participation in our scheme more closely resembles participation in state lottery gambles (hence our use of the term *scratch-off lottery puzzle*). Evidence from the Powerball lottery suggests that affluence correlates positively with the variance and the size of the jackpot [28]. Using this as a heuristic, we suggest that participants who can afford to purchase many storage devices may prefer a puzzle lottery with a higher difficulty (i.e., higher variance) and a commensurately larger reward, even if the expectation is the same.

A Weaker Trusted Dealer Assumption. In practice, the trusted dealer in Permacoin would be replaced by the function of the network itself. While our scheme incentivizes the storage of files, there is no built-in incentive to distribute files to other users. However, even a fraction of altruistic users may form an effective distribution network by sharing segments voluntarily, in which case the storage incentive still helps by offsetting the storage cost. Note that this is similar to how Bitcoin already operates, where the network distributes copies of historical data to new users despite the mining incentives having nothing to do with this function.

Overall Impact. How much would actually be gained if Per-

macoinwere fully successful (for example, if Permacoin were adopted as a modification to Bitcoin – or, alternately, if Permacoin as an independent system surpasses Bitcoin, just as Facebook surpassed MySpace in the 2000's)?

First, note that in the previous section, we estimated the storage capacity of the network if Permacoin had hypothetically been used instead of Bitcoin from the beginning. The distinction is that the current hardware investment in Bitcoin has already been made – at best, we can hope Permacoin leads to better utilization of *future* equipment investments in Bitcoin-like systems. We remark that the observed hashpower of the Bitcoin network has grown steadily since its inception, and has at least doubled every two months between March 2013 and March 2014.[8] If growth continues, then repurposing the additional investment will lead to a benefit.

Although Permacoin reclaims additional benefit from mining hardware that is simultaneously employed in maintaining the currency, the performance of the storage service we obtain is poor compared to typical cloud storage. Large datacenters enjoy economies of scale and save costs on bulk power and cooling; even Amazon Glacier, which, as mentioned in Section 6, most closely matches our operating points (cheap to store, expensive to recover), is much cheaper per gigabyte and offers similar features like geographic replication of data. However, Permacoin offers a *greater* degree of geographic diversity, and, more importantly, dispersion across more administrative domains. Indeed, full decentralization is the key design goal in Bitcoin, and our approach confers the same property to data preservation.

9 Related Work

Permacoin spans three main areas of literature.

Proofs of Storage. There is an expansive literature on PORs. The original construction [17] has been followed by variants, including some with improved efficiency [5, 33], public verifiability [17, 33], distributed variants, e.g., [4], and, most recently, the ability to support file systems, rather than the static files of the original construction [34]. A related vein of literature on Proofs of Data Possession (PDPs) [2] considers proofs that most, but not necessarily all of a file is stored. PORs, unlike PDPs, also support a feature called *extraction*, that allows file recovery through the challenge-response interface. We don't make use of extraction in our proposal here.

Bitcoin. A purely digital implementation of money has been a sought-after goal since the development of blind signatures by Chaum in 1982 [8]. Previous attempts (e.g., [11]) awarded monetary value to computational work, but required an external timestamp service to process transactions in sequence. The novel approach underlying Bitcoin [27] is to relate the problem of committing transactions to the problem of Byzantine consensus in a public p2p network, for which the use of computational puzzles can overcome a lack of pre-established identities [1]. The result is a system which is computationally very expensive, yet encourages a high degree of participation and tolerates substantial misbehavior without relying on strong identities. The economic assumptions underlying Bitcoin are not yet fully understood. When computational resources are allocated equally among individuals, cooperation according to Bitcoin's reference implementation has been shown to be a Nash equilibrium, but not a unique one [22]; when an individual or coalition wields a third or more of the overall power, then it is more profitable to deviate from the protocol [15]. Numerous variations of Bitcoin have been proposed, such as Litecoin, which use different puzzle schemes to alter the incentive structure, for example to encourage participation with commodity (rather than customized) hardware. Ours is among the first proposals for puzzles with beneficial side-effects. In Primecoin [20], mining yields Cunningham chains of prime numbers, though these have no known uses and the puzzle scheme is not proven to satisfy any security definition. The "proof-of-stake" [19] technique proposes to eliminate computational work altogether; this could be an even greater benefit than our approach, although the security tradeoffs are as of yet unclear.

P2P File Systems and Incentives. Since 2000 there has been substantial research on peer-to-peer systems for persistent storage of data [23, 32] based on distributed-hash-table routing techniques (e.g., Pastry [31]). The most popular systems reward users for contributing their resources to the network. Most incentive schemes are based on reciprocation, and are designed to prevent *free-riding* (i.e., users that consume resources without contributing their fair share). In particular, BitTorrent [9] users that contribute upstream bandwidth are rewarded with bandwidth from other users and thus can download files faster. Peer-to-peer document backup systems [10] have been proposed in which users swap contracts by which they agree to store copies of others' files.

Our peer-to-peer storage system in Permacoin is distinguished from these in two main ways. First, the entire target archive is replicated throughout the entire *global* network, rather than among a small cross section of peers. This yields the highest probability of file recovery, and thus is especially suitable for data of great public significance. Second, the high cost of global redundancy is offset by the fact that our system recycles computational resources that are already consumed on the Bitcoin network.

10 Conclusion

We have presented Permacoin, a modification to Bitcoin that recycles the enormous investment of computational resources in order to serve a useful auxiliary purpose. Our approach has been to replace the underlying computational SOP in Bitcoin with one based on Proofs-of-Retrievability. We have shown that in a model involving rational participants and a resource-limited adversary, participants in Permacoin must locally store diverse portions of a large dataset F, ensuring full data recovery with high probability. Our analysis has shown that the system would be feasible, preserving the es-

[8] Read from `http://blockchain.info/charts/hash-rate` on March 11.

sential incentive structures underlying Bitcoin and imposing only a minor overhead. Given the size of the existing Bitcoin network, we estimate that our scheme would recycle enough resources to store at least a "Library of Congress" worth of data (i.e., two hundred terabytes) in a globally distributed network.

Acknowledgements. We thank the anonymous reviewers for their invaluable suggestions, and members of the Bitcoin community (especially Gregory Maxwell) for early discussions leading to this work. This research was funded in part by NSF under grant numbers CNS-1223623 and CNS-1314857, and by a Google Faculty Research Award.

References

[1] J. Aspnes, C. Jackson, and A. Krishnamurthy. Exposing computationally-challenged byzantine impostors. *Department of Computer Science, Yale University, New Haven, CT, Tech. Rep*, 2005.

[2] G. Ateniese et al. Provable data possession at untrusted stores. In *ACM CCS*, pages 598–609, 2007.

[3] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande. Folding@ home: Lessons from eight years of volunteer distributed computing. In *Parallel & Distributed Processing (IPDPS)*, pages 1–8, 2009.

[4] K. D. Bowers, A. Juels, and A. Oprea. HAIL: a high-availability and integrity layer for cloud storage. In *ACM CCS*, pages 187–198, 2009.

[5] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. In *CCSW*, pages 43–54, 2009.

[6] D. Bradbury. Alydian targets big ticket miners with terahash hosting. `http://www.coindesk.com/alydian-targets-big-ticket-miners-with-terahash-hosting/`, August 2013.

[7] N. Chandran, B. Kanukurthi, and R. Ostrovsky. Locally updatable and locally decodable codes. *(to appear) TCC*, 2014.

[8] D. Chaum. Blind signatures for untraceable payments. In *Crypto*, volume 82, pages 199–203, 1982.

[9] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[10] B. F. Cooper and H. Garcia-Molina. Peer-to-peer data trading to preserve information. *ACM TOIS*, 20(2):133–170, 2002.

[11] W. Dai. b-money, 1998.

[12] C. Decker and R. Wattenhofer. Information propagation in the Bitcoin network. In *IEEE P2P*, 2013.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[14] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1993.

[15] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *In FC'14)*, March 2014.

[16] P. M. Fraser. *Ptolemaic Alexandria*, volume 3. Clarendon Press Oxford, 1972.

[17] A. Juels and B. S. Kaliski Jr. PORs: Proofs of retrievability for large files. In *ACM CCS*, pages 584–597, 2007.

[18] C. Kellogg. Government shutdown closes Library of Congress – online too. *Los Angeles Times*, 1 Oct. 2013.

[19] S. King. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. `http://www.peercoin.net/bin/peercoin-paper.pdf`, 2012.

[20] S. King. Primecoin: Cryptocurrency with prime number proof-of-work. `http://www.primecoin.org/static/primecoin-paper.pdf`, 2013.

[21] R. K. Ko, S. S. Lee, and V. Rajan. Understanding cloud failures. *IEEE Spectrum*, 49(12), 28 Nov. 2012.

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

[23] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, et al. Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201, 2000.

[24] P. Lyman, H. Varian, J. Dunn, A. Strygin, and K. Swearingen. How much information? Technical report, UC Berkeley, 2000.

[25] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[26] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for data preservation. online full version: `http://cs.umd.edu/~amiller/permacoin_full.pdf`, March 2014.

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

[28] E. Oster. Are all lotteries regressive? evidence from the powerball. *National Tax Journal, June*, 2004.

[29] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[30] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Information Security and Privacy*, pages 144–153. Springer, 2002.

[31] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[32] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201. ACM, 2001.

[33] H. Shacham and B. Waters. Compact proofs of retrievability. In *Asiacrypt*, pages 90–107, 2008.

[34] E. Shi, E. Stefanov, and C. Papamanthou. Practical dy-

namic proofs of retrievability. In *ACM CCS*, pages 325–336, 2013.

[35] A. L. Shimpo. The Seagate 600 & 600 Pro SSD review. URL: www.anandtech.com/show/6935/seagate-600-ssd-review/5, 7 May 2013.

[36] A. Tversky and D. Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *J. Risk Uncertainty*, 5(4):297–323, 1992.