Lavinia: An audit-payment protocol for censorship-resistant storage*

Cecylia Bocovich
University of Waterloo

John A. Doucette
New College of Florida

Ian Goldberg
University of Waterloo

Abstract

As distributed storage systems grow in popularity, there is now a demand for a reliable incentive and payment system to guarantee and reward the pristine storage of documents. However, many existing proof-of-retrieval and micropayment protocols are not secure in a censorship resistance setting, in which powerful adversaries may infiltrate a system or coerce the original publisher to remove content. Additionally, most existing censorship resistance systems lack a rigorous game-theoretic analysis. We propose Lavinia, an audit and payment protocol for censorship-resistant storage. Lavinia incentivizes document availability by providing micropayments to participating servers in exchange for honestly storing and serving content. Our protocol enables the implementation of a digital printing press as described in Anderson's Eternity Service: allowing *the publisher*, as opposed to public interest or an appointed editorial board, to decide whether a document is worth storing, and for how long. In addition to proving the security of our protocol, we provide an in-depth game-theoretic analysis and show that self-interested participants of our system will faithfully implement the desired behaviour and continue to store documents until their expiration date.

1 Introduction

Throughout history, the spread of information has been assisted by technological advances, but has also faced barriers in the form of censorship. With each new advance in technology that facilitates the spread of knowledge, ideas, and social understanding, there is an increase in the efforts of censors to limit this spread.

A popular example in the history of censorship and its resistance is the advent of the printing press [13]. Not only did the ability to print documents easily and efficiently result in the distribution of previously guarded works, it also led to an increase in the literacy rate of Europe. Despite censorship attempts, printed documents proved to be resistant to state-level attempts to remove them. Borders were difficult to patrol thoroughly, and the production of many copies of each text made them almost impossible to eradicate completely. The only important impediment to using a printing press was the acquisition of enough capital to purchase the requisite raw materials and labour.

Today, worldwide use of the Internet has enabled an even faster and further spread of ideas than the printing press, and provided the means for near-instantaneous conversations between physically and politically distant groups. However, although the Internet has made the distribution and mirroring of content easier and more cost effective than physical printing, it is also much easier to censor electronic content on a large scale due to the centralized nature of storage and routing services. For example, the Great Firewall of China [31], capable of filtering and inspecting all traffic that enters and leaves the country, is a much more practical and scalable censorship strategy than finding and searching the contents of every physical document that crosses the border. In the United States, the Digital Millennium Copyright Act (DMCA) provides an extremely flexible and versatile tool for commercial interests to target content providers and censor digital content from the web [28].

In an attempt to decrease the centralization of today's Internet services and provide Internet users with the censorship-resistant properties of the printing press, Anderson proposed the Eternity Service [2]. The Eternity Service is a description of an ideal digital printing press and with it Anderson outlines a conceptual framework for building censorship-resistant publishing systems in the context of modern digital communications. However, despite myriad attempts to build systems that fulfill Anderson's goals, many of which *do* provide strong

*This is an extended version of our paper published in Financial Cryptography and Data Security 2017 [6].

censorship resistance [8, 9, 26, 30], we are still removed from the model of the printing press. Existing systems impose barriers above and beyond the publisher simply paying for raw materials and labour, such as requiring the publisher to stay online, take responsibility for distributing their document, or operate without the guarantee that their document will remain in the system for the desired amount of time.

Censorship-resistant storage relies on a large number of geo-politically diverse participants providing bandwidth and storage space. A significant barrier to the adoption of existing systems has been the lack of incentives to participate honestly in a distributed storage system. Existing incentive models are unfit for censorship resistance because they rely on a centralized audit and payment system or lack a rigorous game-theoretic analysis of possible attempts to subvert the system and thus maximize earnings. Until recently, incentive systems also lacked a candidate electronic payment system with the security and anonymity properties necessary to provide micropayments to participant servers in exchange for their storage space and bandwidth. However, the development of cryptocurrencies has provided a new way to administer electronic payments and enforce *payment contracts*, similar to the original printing press.

In this paper, we propose Lavinia: a distributed audit and payment protocol for censorship-resistant storage in which publishers pay for the storage and bandwidth costs associated with distributing content securely in the presence of powerful censoring adversaries, and receive in return strong guarantees that their content will remain available for the specified amount of time. We give an extensive game-theoretic analysis of our protocol and show that rational, self-interested parties will implement our protocol faithfully, behaving no differently from an honest, altruistic, participant.

In Section 2, we discuss related work on distributed audit and payment protocols. We then give the models and definitions for censorship-resistant storage and payment contracts in Section 3. In Section 4 we describe the Lavinia protocol, and we show that self-interest results in honest participation in the protocol in Section 5. We give a security analysis in Section 6, and conclude in Section 7.

2 Related Work

Anderson first proposed a digital version of the printing press 20 years ago [2]. The Eternity Service is an ideal (yet unrealized) censorship-resistant publishing system that comprises properties such as plausible deniability for participating servers, anonymity for publishing authors, and notably a payment system to mimic the model of the printing press where a publisher pays to have her work replicated and distributed to readers in a way that is difficult for authorities to track and prevent.

Many existing censorship-resistant publishing systems rely on in-kind payments and reputation-management protocols to incentivize *honest* participation and to limit the effects of a storage-based denial-of-service attack, in which an adversary prevents the publication of new documents by filling up all available space. Tangler [30] assigns *storage credits* to participating servers, allowing them to store a set amount of content proportional to their own donated capacity. This gives them the option to "rent out" or donate their storage credits at their own discretion. However, Tangler does not provide a protocol for credit rentals or donations, leaving servers to adopt insecure or biased methods of collecting remuneration for their services. Furthermore, there is no audit process to guarantee that servers continue to store and serve uncorrupted documents over time. While Tangler does use a comparison of messages to inform other participating servers of nearby malicious servers, such a reputation system is not secure against a large number of colluding servers.

Free Haven [11] employs a more complex reputation management system in which servers assign a *reputation* and *credibility* value to all other known servers. Each of these two values is also accompanied by a *confidence rating* that reflects the depth of knowledge about the server in question. Servers broadcast *referrals* that contain suggestions for these values in the event of honest, malicious, or suspicious behaviour. Although such a system can pinpoint malicious servers, it does not defend against more complicated game-theoretic attacks in which an adversary behaves honestly but suspiciously in order to bait other servers into giving false reports.

Vasserman et al.'s one-way indexing scheme [29] solves the complexity of distributed trust assignments by using a centralized editorial board to curate content and defends against denial-of-service attacks by deleting unimportant documents from the system. This centralized design is not ideal for censorship resistance as users cannot store content that the editorial board deems to be uninteresting or offensive, unless it is also popular.

Although the development of an electronic *payment* protocol to incentivize censorship-resistant publishing is novel, it builds on related work in the area of distributed data storage and retrieval. A key problem in distributed storage is that once a document is stored, the server responsible for it may decide to discard the data or leave the storage network. Payment at the time of storage is therefore ineffective, and incremental payments require careful management of server reputations. There is a large body of work that addresses the problem of distributed payment in peer-to-peer systems through the use of micropayments, audits, and escrow services. Most

early systems relied on a centralized payment system or suffered from problems in scalability or anonymity [10], which render them unsuitable for censorship resistance. More recent systems tend to rely on escrow payments to incentivize storage, but still require centralized audits [15] or in-kind payments where publishers pay by offering access to unused CPU cycles or bandwidth [22]. These features are undesirable for censorship resistance, where centralized third parties are vulnerable to attack and publishers (e.g., political dissidents) should be allowed to cease interaction with the system after publication.

The proposed storage system most similar to our own [20] is fully distributed and provides micropayments in return for the periodic verification of storage. However, the proof of retrieval technique used to audit document availability allows servers to distinguish between auditors and regular users. This knowledge allows them to maximize profits by refusing to serve content to anyone but an auditor. While this model is appropriate for storing documents that are meant to be accessible by a single user, it does not fit the needs of a censorship-resistant publishing system in which content is meant to be accessed by many users, and *serving content is equally as important as storing it*. For this reason, our proposed solution will make users indistinguishable from auditors, forcing servers to deliver content for *every* access. We utilize a novel micropayment system in our protocol, similar to existing work, but with additional features that ensure suitability for censorship-resistant publication, storage, and retrieval.

3 Models and Definitions

3.1 Censorship-Resistant Storage

The structure of censorship-resistant publishing systems differs from that of traditional storage schemes. Censorship-resistant storage is largely decentralized and dynamic, involving a diverse and constantly changing set of servers. As with traditional printed documents, wide dispersal and redundancy are essential for increasing the likelihood of a document's continued existence over time in a digital setting. The dispersal of sensitive documents across multiple jurisdictions has important advantages: state-sponsored attempts to remove information from the system will not be able to reach a significant subset of servers, and a single entity's attempts to compromise each machine will not scale to physically separate servers.

Our payment and audit protocol will work with a wide variety of storage schemes, including many existing censorship-resistant publishing systems. We base our security and game-theoretic analysis on a general model of storage. Here we briefly describe existing censorship-resistant storage systems and define our general model.

File retrieval: Documents should be encrypted and split into multiple retrievable pieces using a threshold scheme [23]. The act of secret sharing provides honest servers with plausible deniability about what they are hosting, and encryption adds an extra layer of protection, preventing servers that have acquired multiple shares from using existing techniques to reconstruct the document. In some jurisdictions, this may afford them legal protections. We refer to a single document piece as a *file* f.

Many existing systems are built as overlays on top of structured peer-to-peer (P2P) networks such as distributed hash tables (DHTs[1]) [18, 21, 25]. Each file f is associated with a keyword, and the space of all keywords is partitioned among partipant servers to allow for efficient document storage and retrieval. BitTorrent [1] and Freenet [9] are examples of unstructured P2P systems. Rather than deterministically partitioning the keyspace among participant servers, documents are initially stored at one location, and then cached by additional servers when they are retrieved from the system.

In our model, each file f is stored in the system under a lookup key denoted $lookup(f)$. Performing a lookup for this key will return the server that is currently responsible for hosting that file. We assume that a lookup will be routed through, on average, a set of $k > 1$ servers on its way to the correct host. Additionally, we assume that given $lookup(f)$, a user is unable to discover all lookup keys necessary to reconstruct the entire document as in the one-way indexing technique [29]. This will provide servers and auditors with additional plausible deniability.

Redundancy: In the presence of an active censor, a high degree of redundancy ensures that a document does not become lost if some servers leave the system or refuse to serve content. We assume that the underlying storage scheme mirrors each file f on a set of $n \geq 2$ servers $server_{f1}, server_{f2}, \ldots, server_{fn}$. We also assume that the server responsible for the main copy of the file f (i.e., the server that is targeted by performing a lookup on key $lookup(f)$) has a way of contacting the mirroring servers.

Churn: Censorship-resistant storage systems are, by their nature, dynamic. We assume that new servers may join the system and that existing servers will leave. When a server joins the system, she becomes responsible for a subset of the system files. The new server may contact the server(s) that were previously responsible for her files, and any server operator may leave the system at any time, and may contact the server(s) responsible for

[1] Although many DHTs are vulnerable to Eclipse [7] or Sybil attacks [12], we note that securing DHT join and lookup protocols is an active area of research [4, 7, 24] and is outside the scope of this paper.

her files after she leaves.

3.2 Payment System

The main goal of providing compensation to participant servers in a censorship-resistant storage system is to incentivize the storage and availability of a document for an arbitrary amount of time. Anderson originally described an annuity that could accomplish this goal by "following data around", providing incremental payments to any server currently responsible for hosting and serving it [2]. Past precedent indicates that even small operators will go to great lengths to recover valuable missing data [16].

Recent innovations in cryptocurrencies have provided a means to create a travelling annuity. Funds may be transferred from a sending "wallet" to a recipient wallet with knowledge of the sending wallet's private key. This key can be easily transported along with a file or document, thereby following it around the storage system. In this section, we demonstrate the suitability of the Bitcoin cryptocurrency [19] for our protocol. However, our protocol will work with any payment system with the following properties: (1) coercion-resistant through geopolitical distribution or anonymization, (2) redeemable with a distributable secret, (3) time-locked where funds can be placed in escrow until a fixed time has passed, and (4) associated with an append-only log.

Bitcoin is coercion resistant through both geo-politcal distribution and optional anonymous extensions. As long as at least 50% of miners accept Lavinia transactions, there is a high probability that they will not be dropped from the system. Furthermore, Zerocash [5] may be used for anonymization, eliminating the ability to link payments with specific documents and thereby thwarting censorship attempts.

Bitcoins are redeemable with one or more secrets. To redeem (i.e., spend) a coin, a user must be able to sign a transaction with the private key associated with the coin's wallet. It has a time-lock feature, which allows the sender to specify a date before which the coin cannot be redeemed. The payment blockchain doubles as an append-only log, and the Bitcoin scripting language[2] allows the spender to enforce that specific values are added to this log before a payment can be redeemed.

In our system, a publisher Alice constructs a series of *payment contracts* $\mathscr{P}(X,t,S,v)$. Each contract places a set of funds X in temporary wallets with private keys $s_i \in S$. The funds cannot be removed until after the time t has passed, and upon redeeming these funds, the holders of the keys in S must publish the value v to the Bitcoin blockchain. The set of funds, $X = \{s_1 : x_1, \ldots, s_n : x_n\}$, specifies the amount of Bitcoins x_i that belongs to the wallet with private key s_i. The funds may not be redeemed without the cooperation of all recipients.

Each recipient (i.e., holders of the private keys in S) fulfills the contract $\mathscr{P}$ by having all recipients collectively sign and append a transaction to the blockchain that spends the coins to their own personal accounts and posts the now-public value v. We denote this *transaction* as $\mathscr{T}(proof(S),v)$. When the transactions are complete, the funds will be divided amongst the recipients in the amounts specified by X in the payment contract.

We can construct the payment contract

$$\mathscr{P}(\{s_1 : x_1, s_2 : x_2\}, t, \{s_1, s_2\}, v)$$

using the Bitcoin scripting language. When a transaction to spend a coin is processed, its input script is concatenated to the output script of the transaction that created the coin. Alice commits to $\mathscr{P}$ by submitting a transaction spending coins worth a total of $x_1 + x_2$ to the blockchain. This transaction contains the following output script:

```
timelock: t
Output: [
{x1,
    <Pubkey s1> <PubKey s2> 2
    OP_CHECKMULTISIG
    OP_SHA256
    <hash_of_v>
    OP_EQUALVERIFY },
{x2,
    <Pubkey s1><PubKey s2> 2
    OP_CHECKMULTISIG
    OP_SHA256
    <hash_of_v>
    OP_EQUALVERIFY }]
```

It requires a proof of knowledge of the secrets s_1 and s_2 in the form of signatures on the subsequent spend transaction. It also requires the next spend transaction to include the value v in its input script. It verifies this value by ensuring that it hashes to hash_of_v. The time lock ensures that the next spend transaction will not be submitted to the blockchain until the time t has passed.

The transaction $\mathscr{T}$ that fulfills this contract must have the following input script:

```
<v> <sig s1> <sig s2>
```

Each owner of a secret in the set S must provide a transaction with this input script to redeem their funds. This involves signatures from both parties on each transaction.

[2] https://bitcoin.org/

4 Lavinia Protocol

4.1 Overview

Lavinia allows a publisher to publish content, submit payments, and then vanish from the system completely—the continued availability of content is not contingent on the actions of the original publisher. This protects against out-of-band coercion tactics such as *rubber-hose cryptanalysis* in the case that the publisher is captured or prosecuted. Additionally, third-party benefactors may fund existing documents to increase the likelihood that they will remain in the system or extend the document's lifetime. This ensures that even popular content with higher bandwidth costs will remain in the system.

Micropayments to participating servers occur during audit periods chosen by the publisher or benefactor during the initial payment step. The publisher chooses a different auditing server for each audit period and places with them the responsibility of checking a file for availability at some time during that period, in exchange for a small remuneration. An auditor lacks sufficient evidence to prove her auditor status and requests the file as a regular user would, forcing the server to respond to both audits and regular requests for content. We place restrictions on the auditor by preventing her from learning which files she will audit until the previous audit period has passed. At that time, she may access the file's lookup key by searching the payment system's append-only log. Finally, we place additional incentives to ensure that all audits and remuneration occur in a timely manner at each audit time.

An important challenge associated with making censorship resistance a possibly profitable endeavor is ensuring that a participant is unable to game the system and receive payments without providing services. We assume that participants in the Lavinia protocol are rational and self-serving entities who will employ any means necessary to receive payments while incurring as few costs as possible in the form of storage space and network bandwidth. The fault tolerance features of our protocol will also defend against a small number of irrational, malicious participants. In Section 5, we show that exploiting this self-interest strengthens the censorship-resistant properties of the system and increases the likelihood that a document remains available until its expiration date.

Although Lavinia cannot directly help impoverished users publish documents safely, it does provide a way for third parties to help them more efficiently by allowing them to create payment contracts on behalf of the publisher or to suppliment existing documents. For example, concerned free speech advocates could form a fund to store documents they felt were meritorious, and perhaps even participate as servers in the storage system and host content for free.

4.2 Protocol Details

We give the full Lavinia protocol in Figure 1. During publication (or at any time throughout the life of the document) a publisher or benefactor, Alice, prepares payments for each of their files f stored in the system.

Alice first determines a set of times $T = \{t_1, \ldots, t_n\}$ that separate the audit periods during which she wishes her document to be checked for availability. (For convenience, let t_0 denote the time of publication of the document.) She creates a payment contract for each time $t_i \in T$. For example, if she wishes her document to be audited approximately once a month for two years, she would then create 24 payment contracts for each file f she uploads to the system. These contracts form an agreement between Alice, the servers hosting her shares, and the auditors responsible for ensuring her document's availability.

For each of Alice's contracts, she randomly generates new wallets with private keys skA and skS for an auditor and server, respectively. She then decides on the payment amounts $X = \{skA : x_a, skS : x_s\}$ for the auditor and server.

The time lock enforces that funds will not be transfered until the audit period ending at time t_i has passed. Let $kw(s,K)$ be a key wrapping function that encrypts a key K with a secret s. We assume that this key wrapping function is secure and that the ciphertext does not leak information about the secret s or the key K. Alice encrypts the auditor's secret with a random value r to produce the masked secret $v = kw(r, skA)$; v becomes the value that must be posted to the append-only log to redeem the contract, as described in Section 3.2.

Alice now constructs a contract

$$\mathscr{P}(\{skA_i : x_a, skS_i : x_s\}, t_i, \{skS_i, skA_i\}, kw(r_i, skA_i)).$$

for each audit period by spending coins to the newly created wallets, as described in subsection 3.2. This places funds for the file f in escrow with the server responsible for hosting the share and the auditor responsible for assuring its existence in the audit period $[t_{i-1}, t_i]$. She distributes the server secrets skS_i for each time t_i to the server hosting the file by encrypting them with the key wrapping function kw and a random value r_{s_i} to produce the masked server secrets $\{kw(r_{s_i}, skS_i)\}_{i=1}^{n}$. If the file changes hands (as in a dynamic storage system), the secrets $\{kw(r_{s_i}, skS_i)\}_{i=1}^{n}$ travel with it.

Alice then selects auditors for each of her contracts, and sends to each of them the beginning and end of their audit period, t_{i-1} and t_i, the lookup key for f encrypted with the previous auditor's secret, $kw(skA_{i-1}, lookup(f))$, and the masked auditor secret, $kw(H(f \| skA_{i-1}), skA_i)$, encrypted with a hash of the file

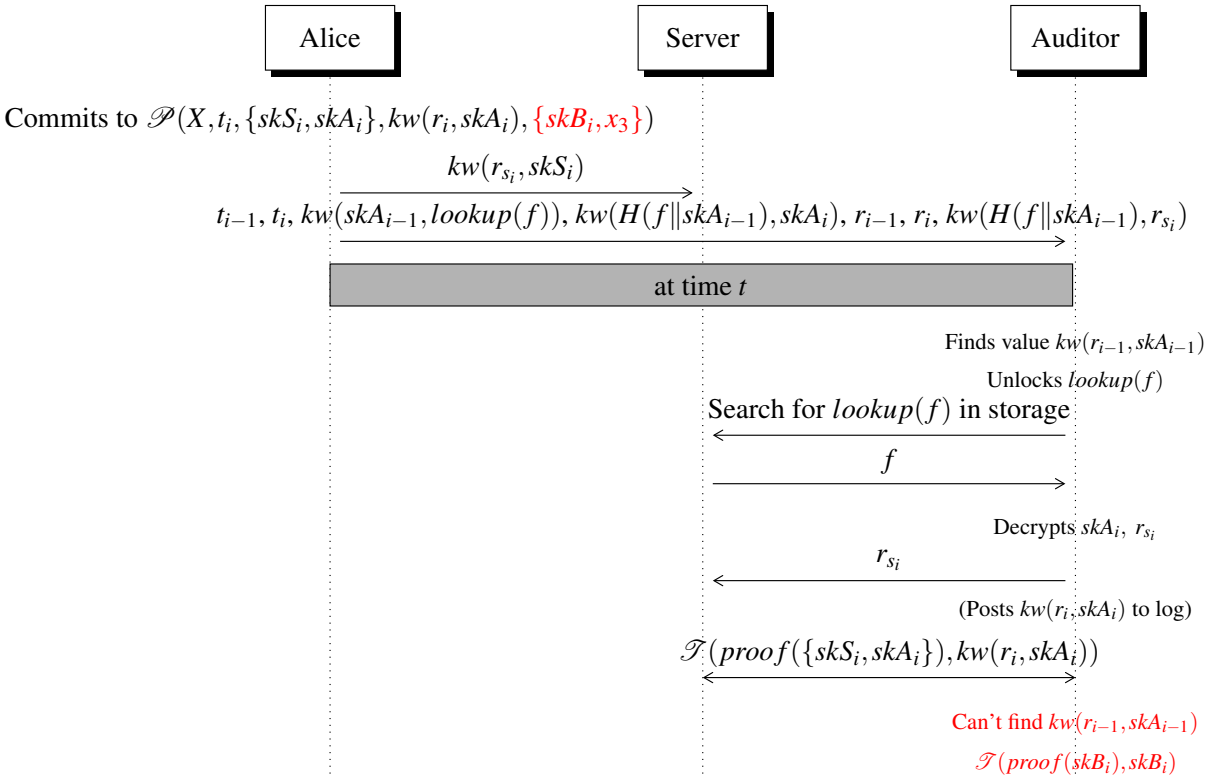

Figure 1: Protocol for setting up payments and auditing a file f during the period $[t_{i-1}, t_i]$, where $kw(s, K)$ is a function that encrypts a key K with a secret s. The burn procedure is shown in red.

f concatenated with the previous auditor's secret, where the cryptographic hash function H is both pre-image resistant and collision resistant. The first auditor for time period $[t_0, t_1]$ will also receive the value skA_0, randomly chosen by Alice. Finally, she sends the auditor the random value r_{i-1}, which is used to decrypt the previous auditor's secret, the random value r_i, which the auditor will use to encrypt her secret, and $kw(H(f\|skA_{i-1}), r_{s_i})$, which the auditor will later decrypt and send to the server to unlock skS_i. After this point, the publisher or other benefactor is free to cease all interaction with the system. The construction and distribution of the above payment information can be performed during or after the publication of the document. Note that it is in Alice's interest to construct these values honestly. An incorrect or insecure value that prevents a server or auditor from being paid or allows them to cheat the system will increase the probability that her files will be dropped.

To ensure that Alice's files will be audited during each time period, the server and auditor should not be paid before their audit period ends and they should not be able to audit the document before their audit period begins. To accomplish this, we use the payment system's time-lock feature and encrypt the lookup keys and secrets for auditor $[t_{i-1}, t_i]$ with the published value v of auditor $[t_{i-2}, t_{i-1}]$. In order for the auditor of period $[t_{i-1}, t_i]$ to unlock her secret skA_i, she must know skA_{i-1}.

This scheme has the advantage of enforcing the time lock with self-interest. The auditor $[t_{i-2}, t_{i-1}]$ cannot redeem her payment until time t_{i-1} has passed. When an auditor moves funds, she must also release her encrypted secret, $kw(r_{i-1}, skA_{i-1})$ to a publicly viewable append-only log. At this time, the auditor for the period $[t_{i-1}, t_i]$, who owns r_{i-1}, is able to compute her own secret and perform the audit of the file. If the previous auditor releases her secret ahead of time, she runs the risk of forfeiting her payment to the next auditor (since her secret will then be visible to that auditor). We note that if a server is temporarily unavailable at the time the auditor attempts to retrieve the file f, the auditor can continue to query for the document until her audit period has passed. We show the timeline for auditing a file f at audit time t_i in Figure 2.

Alice initializes this sequence by providing the first auditor with a randomly generated initialization key skA_0 and the following values:

$$t_1, lookup(f), kw(H(f\|skA_0), skA_1), r_1, kw(H(f\|skA_0), r_{s_1})$$

Note that this first auditor must still conduct a lookup of the file f to unlock her secret skA_1 and the server's random secret r_{s_1}.

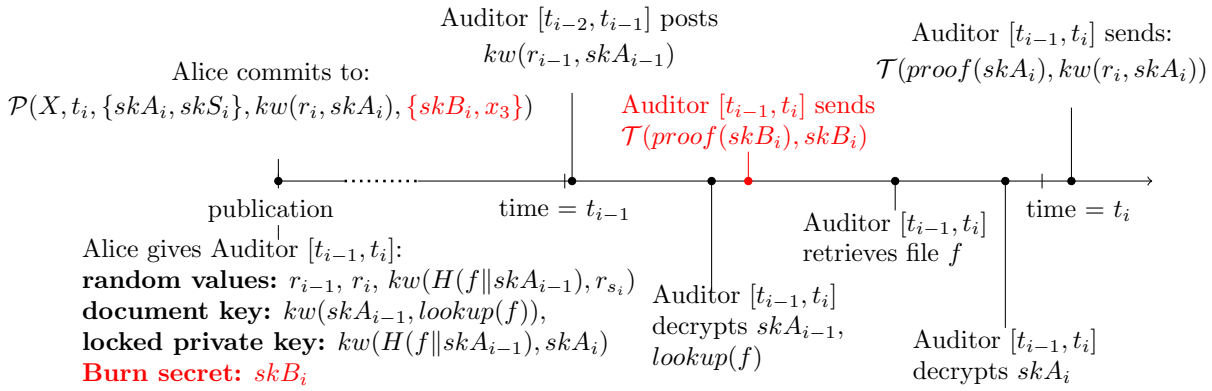

Figure 2: The timeline of an audit sequence for a file f from the perspective of its auditor for period $[t_{i-1}, t_i]$. If the auditor for period $[t_{i-2}, t_{i-2}]$ fails to complete their audit and post their private key, the next auditor will follow the burn procedure shown in red.

4.3 Burn contracts

A disadvantage of the method of sequential payments described above is the impact of an auditor leaving the system, even temporarily. If the previous auditor fails to release her information after time t_{i-1}, the auditor during $[t_{i-1}, t_i]$ will not be able to perform her audit or receive remuneration for her efforts. This in turn will prevent subsequent auditors from receiving the information needed to perform their audits, effectively terminating the revenue stream for the file. A malicious party could easily exploit this by posing as an auditor, and simply declining to perform her audit, or coercing an honest auditor into skipping a single payment on some targeted document. To avoid this, we extend the requirements of our payment system to allow an auditor to burn the previous auditor's payment after her time has passed. If an auditor at time t_i becomes aware that the previous audit failed, she will be able to burn the money in both her and her predecessor's accounts and forward the secret to the next auditor in the chain. In order to incentivize burning instead of complete inaction, we allow auditors to keep a small fraction of the profits they would have received if an audit were possible (though not so large that they would prefer burning payments to performing audits).

We define $\mathscr{P}(X, t, S, v, \{skB, x_3\})$ to be an extension of the payment contract in subsection 3.2 to allow Alice to specify a burn secret, skB_i and a payment amount x_3 for each time t_i. This will invalidate payments to the secrets in skS_{i-1} and skA_{i-1}, and pay the holder of this secret the amount x_3. The money is burned if and only if an auditor issues a transaction $\mathscr{T}(proof(skB_i), skB_i)$ where she posts skB_i to the log. Alice provides the auditor for period $[t_i, t_{i+1}]$ with the previous auditor's secret locked with their burn secret, $kw(skB_i, skA_i)$ at the initial time of payment. This will allow the auditor at time t_{i+1} to proceed as usual.

Each auditor will then receive the previous auditor's secret locked with the previous burn secret. In addition to preventing deliberate attacks on the chain of audits, this will incentivize auditors to complete their assigned audits before the next time period begins. We now give an implementation of the burn functionality in Bitcoin using the `OP_RETURN` call. The following Bitcoin script implements the payment contract:

$\mathscr{P}(\{skS_i : x_1, skA_i : x_2\}, t_i, \{skS_i, skA_i\}, kw(r, skA_i), \{skB_{i+1}, x_3\})$

```
Output: [
{x1,
    <hash_of_skB> OP_EQUAL
    OP_IF
        OP_RETURN //burns the money
    OP_ELSE
        <Pubkey skS> <Pubkey skA> 2
        OP_CHECKMULTISIG
        OP_SHA256
        <hash_of_kw(r, skA)>
        OP_EQUALVERIFY
    OP_ENDIF
}, {x2, //same as x1 script },
{x3,
    <hash_of_skB> <Pubkey skB>
    OP_CHECKSIG
}]
```

4.4 Choice of Auditors and Audit Times

Auditors can conceivably be any collection of entities willing to participate in the Lavinia protocol. We do not make any assumptions about whether or not they also participate as servers in the system. However, auditors

do need to be discoverable by Alice. For maximum security, Alice should choose a different auditor for each audit time and file. This requires a potentially large number of auditors. One way to increase the ease of distribution and discoverability is to make the set of servers and auditors one and the same. This would allow Alice to choose a random lookup key for each payment contract, and probabilistically ensure that no one auditor will be responsible for multiple audit times of a single file.

To reduce the ability of servers to guess future audit times, a publisher can choose times at random intervals, distributed according to a Poisson process. This defends against an attack in which servers only serve content during brief time windows around fixed intervals in an effort to distinguish between auditors and regular users.

We also note that any reader can claim to be an auditor for an audit time t, and servers are unable to verify her identity. Even if the server is certain about the next audit time, there will always be at least some period between the release of the previous secret and the retrieval of the document by the next auditor during which the servers will be forced to serve the document to all users.

5 Game Theory Analysis

In the Lavinia protocol, servers are incentivized to behave correctly by the potential profits they earn from delivering files. While individual operators might have nobler motives, we claim that the harnessing of the profit motive is actually an *advantage* of our system in many respects. Operators who see an opportunity for profit can go to great lengths to ensure the integrity of the system, and to ensure they are able to fulfill their obligations. However, the use of the profit motive also has a distinct disadvantage: profit-seeking operators will not necessarily conform to the desired protocols and behaviours of the system if they can find and implement a more profitable protocol, which may not include desired behaviours [14]. In this section, we show that rational, profit-maximizing server operators will follow the Lavinia protocol faithfully by continuously storing and serving documents to both regular users of the system and auditors.

Game theoretically, we model choices of servers within a censorship-resistant storage system as a *game* played by the set of server operators A, and denote player i by A_i. Each player operates one or more servers, all connected to the same network, and tries to maximize her own profits, but does not try to reduce the profits of other players (unless doing so increases her own profits).[3] We assume there are η servers in total, and denote the set of all servers by S, and server j operated by player i with $S_{i,j}$.

In our model, each player plays several families of games, in which they select a *strategy* in the form of a set of policy decisions (e.g., when to store a file or when to serve a requested file). Strategies are selected to maximize the *profit functions* of each player, potentially based on what the other players do. The set of strategies selected by all players is called a *strategy profile*. A strategy profile forms a *Nash equilibrium* when, even with complete knowledge of what the other players have done, no player could improve her profits by retroactively adopting a different strategy. An equilibrium is a *dominant strategy equilibrium* when no player could improve her profits, *regardless* of what the other players may or may not do. If the dominant strategy equilibrium is not unique (there exists, e.g., two equally good actions for a player to take), we assume that players prefer the strategy that is closest to the Lavinia protocol (a useful assumption in many game theoretic contexts [27]). This is essentially an assumption of sloth: no player should waste resources to change from the default client behaviour to something else, if there is no change in her overall profits.

5.1 The Static Game

To begin, we consider a simplified version of the storage system where the network topology is fixed. This environment is unrealistic, but could be a useful approximation of the network in the long run (i.e., after it has operated for a long time, and includes many players). Its study will also provide insights for the model considered in the next subsection, in which servers can both join and leave the network.

In this game operators must pick a strategy for operating their servers. An operator must adopt the following policies:

- A serving policy, Π_{share}, that specifies for each file f held by a server, whether or not to serve the file when it is requested. This policy is expressed as a set of probabilities $0 \leq \Pi_{share}(f) \leq 1$, each of which specifies the probability that the server responds to a request for file f.

- A storage policy Π_{store} that specifies whether or not to continue storing a file f expressed as a set of probabilities $0 \leq \Pi_{store}(f) \leq 1$.

- A routing policy Π_{route}, that specifies how a server s responds to lookup requests that are routed through it. This policy cannot depend on any particular f, but may instead depend on the server or lookup key. This policy can be split into three components: $\Pi_{route}(s, any)$, $\Pi_{route}(s, self)$ and $\Pi_{route}(s, others)$,

[3] We consider the impact of malicious servers in the next section.

respectively denoting whether the routing information from server s is sent at all, whether the routing information contains correct information about the keyspace managed by s, and whether the routing information for the keyspace of other servers is correct.

We further model global properties of the network with $\Gamma_{send}(f)$, and $\Gamma_{route}(s)$: the fraction of requests for a file f that are correctly routed (eventually) to the server storing f, and the fraction of routing traffic that passes through s as a fraction of all traffic expected to pass through s (i.e., $\frac{1}{\eta}$ of all traffic is expected to pass through s). Additionally, we denote by Γ_{hop} the average number of routing steps made by a given request. Finally, the function $\lambda_{BR}(f)$ denotes the ratio of lookup requests made by ordinary users to lookup requests made by auditors for a particular file f. The functions $g_{transmit}(f)$ and $c_{transmit}(f)$ denote the profit from sending f to an auditor, and the transmission cost of sending f to anyone, respectively. c_{route} similarly denotes the cost of sending routing information for one lookup key, and T is the total number of lookups into the system. We are now able to state a formal characterization of how rational actors will behave in the important set of static games of this kind (see Appendix A for the proof).

Theorem 1. *If every server in the network is a starting point for $\frac{1}{\eta}$ lookups, and no lookup will visit the same server more than once, then provided that for every server s in the storage system,*

$$\frac{\sum_f \frac{1}{\eta}(g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f))}{T} > c_{route}$$

and for every file f stored at s

$$\frac{1}{\eta}(g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f)) > c_{store}(f),$$

then there exists a dominant strategy Nash equilibrium where all servers adopt the strategy $\Pi_{store}(f) = 1$ and $\Pi_{send}(f) = 1$, for all f, and $\Pi_{route}(s, all) = R_s$, where R_s is the correct routing information for s.

The interpretation of this result is that, in a static system in which traffic levels for files are relatively constant in the longer term (i.e. $\lambda_{BR}(f)$ does not change much from the server's initial belief), rational servers will conform to the Lavinia protocol even if other servers behave irrationally, subject to some modest, realistic, constraints. Further, when more servers behave rationally, $\Gamma_{send}(f)$ increases, while $\Gamma_{hop}\Gamma_{route}(s)$ decreases, making the cost of irrational behaviour (relative to rational behaviour) increase. (See the proof in Appendix A for further details.) We conclude that this indicates the system should be quite stable in practice, once established.

We note that, although storage and bandwidth costs will vary by jurisdiction, the price of storage hardware at the moment amounts to approximately $0.03 per GB in the United States,[4] and the cost of bandwidth is approximately $10/month per Mbps,[5] which also equals $0.03 per GB. The profit from hosting a file, $g_{transmit}(f)$, should then be at least $(\eta + \lambda_{BR})\frac{\$0.03}{\text{GB}} \cdot |f|$.

5.2 Estimating λ_{BR}

Since the strategy adopted by the server is dependent on λ_{BR}, the ratio of unprofitable reader traffic to profitable auditor traffic, we now explain how servers might compute this quantity, and consequently compute their strategies.

If we assume that audit times are Poisson distributed, as mentioned above, then a server A_i still needs to estimate the frequency of non-audit traffic to compute λ_{BR}. In practice, the amount of non-audit traffic may change dramatically over time (e.g., making it an inhomogeneous Poisson process or a Cox process [17]). For example, one might expect a rapid increase in reader traffic if an important file is posted, and then later discovered and reported in the press. If A_i cannot model the change in the process's value over time, then it cannot reasonably decide whether to continue serving the file in response to sudden spikes in traffic (like a denial of service attack). It also cannot decide whether or not to continue storing the file if traffic grows too high (in the hope that traffic rates will decline again in the future), or to discard it (under the assumption that transmitting the file will never again be profitable).

In essence this is a traffic prediction problem, which is an active area of research. We suggest the use of a simple piece-wise linear approximation process [17], to estimate the current rate of requests. Since the auditor's request rates should not change over time, it can be estimated using a conventional maximum likelihood approach, where events take the form of a payment by an auditor. Thus, using the rate of payments for the file, λ_T, and the rate of total requests for the file, λ_f, a server can calculate $\lambda_{BR} = \frac{\lambda_f - \lambda_{\mathscr{T}}}{\lambda_{\mathscr{T}}}$.

5.3 Dynamic Behaviours

Having established that Lavinia is stable when the set of players is static, we now consider strategic behaviour in scenarios where servers can join and leave the network. In this section, we rely heavily on the presence of *cached* content in the storage system. When Alice publishes a document, she should store copies of each file, *along with its payment keys* at mirroring servers. We will

[4] http://www.mkomo.com/cost-per-gigabyte-update

[5] https://blog.cloudflare.com/the-relative-cost-of-bandwidth-around-the-world/

show in this section that servers have strong incentives for continuing to store this information in the long term.

After a server joins the network, it will be present in the routing information of all servers that point to its keyspace[6]. The only needed result is to show that newly joined servers will be able to acquire the content, and vitally, the payment information, associated with their assigned keyspace. We refer to this as the *mirroring subgame*. As stated, we assume that each file f is mirrored by at least two other servers s_{f1} and s_{f2}. The mirroring subgame is then a game played between a new server that wishes to join the network, which we call s_{new}, and three other servers, s_{orig}, s_{f1} and s_{f2}. We denote by $key(s)$ the identity determining the keyspace of a particular server. The other servers are defined as follows: s_{orig} is the server that holds some files that the new server s_{new} would like to take over. s_{f1} and s_{f2} are currently mirroring the file f. We derive the following equilibrium result (see Appendix A for the proof):

Theorem 2. *In the mirroring subgame, there is a Nash equilibrium where the joining server s_{new} offers a one-time payment to either s_{f1} or s_{f2}, selected randomly, and will receive f with certainty. Further, this amount is not more than half the long-run total value of the mirrored file, ensuring a long-run profit for s_{new}.*

We have now established that, under the assumption a joining server s_{new} will receive all income from acquired content, it can still plausibly acquire said content. Having established this, it is straightforward to show that servers are able to come and go from the network at will. To leave the network, a server simply copies the content to the server that will be responsible for the files after she leaves and mirrors the content at the new mirroring servers $server_{f1}, \ldots, server_{fn}$. These mirror servers will accept the extra load if the content is profitable to host in the first place, because it can be sold to future joining servers for a sum that will likely cover its costs, provided that network churn occurs frequently enough *relative to the storage cost of the content*. Since payment times are Poisson distributed, no particular block of time is worth more than any other in expectation, so servers cannot gain value by repeatedly joining and leaving. Note also that although many servers may thus end up with a given file and the associated payment keys, only the server reached by an auditor will receive the r_s value needed to unlock payment for that time period.

Under our assumptions regarding secure routing, Lavinia incentivizes an equilibrium where servers can join and leave at will, and where content will be stored redundantly. Coupled with the more robust equilibrium for a system with low churn, our results reinforce the idea that Lavinia satisfies the goals of our payment protocol.

[6]Note that we assume the presence of a *secure routing protocol*, in which there are protections against servers reporting incorrect routing information [7].

6 Security

We claim our audit protocol is secure if an attacker is unable to: (1) compute the value of *skA* for any audit time t, unless she is the auditor for time t, or time t has passed and the previous auditor has posted to the append-only log, (2) receive a payment for auditing a file f at time t, unless she has retrieved f from the system sometime after the previous audit time has passed, and (3) receive a payment for serving a file f at time t, unless she has served f after the previous audit time has passed. We defend against these attacks through the use of the key wrapping function kw and the cryptographically secure hash function H. For a full proof, see Appendix A.

While the security of the protocol itself guarantees that an auditor or server is unable to receive payment without faithfully implementing the protocol, there are a number of attacks that a malicious adversary willing to forego personal gains could employ to drop content from the system. We will now describe these attacks and their defenses.

Denial-of-Service (DoS) Attacks: As mentioned, a document that is frequently accessed will have a higher associated bandwidth cost for the hosting server. An adversary could flood the storage system with lookups in order to make content costly, incentivizing servers to stop serving certain files. There are defenses that servers could deploy individually, such as rate limiting by IP or requiring the performance of a small computational task, to limit the number of lookups by a single user. However, these techniques are useless against distributed attacks. In any case, we argue that a short-lived DoS attack will not result in content being dropped from system, but rather that servers will refuse to serve content only until the number of requests drop back to normal levels. A long-term DoS attack may be discoverable or too costly even for a state-level censor. Even the DDoS attack on Github by the Chinese government [3], which lasted five days, is still short term in the context of a document with a life span of multiple years.

Auditor-Server Collusion: An integral part of our protocol is that an audit must look no different from a regular request, forcing a server to deliver content at every request in return for possible payment. We show that an auditor lacks a sufficient amount of proof to reveal her status to the server without forfeiting her own payment. The only way for an auditor to prove her status before faithfully collecting the file f during the audit is to provide the server with $kw(H(f \| skA_{prev}), r_s)$ and lookup(f), allowing the server to retrieve *skS*. However, the server cannot validate that this signing key is correct

without the other signing key *skA*, and the only way for the server to validate it without serving the file to *any* auditor or reader that claims to possess *skA* is for the auditor to give the server $kw(H(f \| skA_{prev}), skA)$, forfeiting her payment.

Join-and-Leave Attacks: In an effort to inherit content from existing servers and drop it from the system entirely, an adversary can employ a join-and-leave attack. By repeatedly joining the network, an adversary will inherit a subset of documents from existing servers in the system. If the adversary leaves the system without replicating or moving these documents, the content will be lost. We argue that the existence of mirrored content and the profit motive will result in multiple redundant copies of each document, and that these copies may be found with minimal investigation.

False Payment Attacks: An adversary can attempt to trick servers or auditors into dropping a document from the system by issuing false payment contracts, forcing the server-auditor pairs to undergo the audit and payment protocol before she realizes that there are no funds associated with the provided payment keys. We argue that a document will still remain in the system as long as the original payments provided by Alice cover the marginal cost of participating in the additional malicious audits, which would be very small. Furthermore, the adversary is required to put some amount of funds in escrow and is unable to receive her funds until the audit time t has passed, allowing an auditor and server pair to race the adversary to complete the protocol and receive the additional payment. An adversary may try to overwhelm Alice's original contract by flooding the system with thousands of worthless ones. Such an attack is quite costly, in both computing resources and capital, as it requires a large amount of transactions in which the adversary must submit real payment contracts.

7 Conclusion

We have proposed Lavinia, a novel audit and payment protocol that incentivizes the continued availability of published content by remunerating server participation in a privacy-preserving manner. Lavinia provides a publisher with the means to specify an arbitrary storage time for her documents. The continued availability of stored documents is ensured by an audit and payment protocol, in which servers and auditors are compensated for ensuring that the document stays in the system until its expiration date. We provide a game-theoretic analysis that shows servers in the storage system acting on behalf of self-interest to maximize profits will participate honestly in the Lavinia protocol. With these requirements met, the Lavinia protocol provides the final pieces for a comprehensive realization of a true digital printing press for the Internet age.

Acknowledgements. We thank the anonymous reviewers for helping us to improve this work. We thank NSERC for grant STPGP-463324.

References

[1] BitTorrent. `http://www.bittorrent.com/`.

[2] R. Anderson. The Eternity Service. In *Pragocrypt 1996*, pages 242–252, 1996.

[3] S. Anthony. GitHub Battles "Largest DDoS" in Site's History, Targeted at Anti-Censorship Tools. *Ars Technica*, `http://arstechnica.com/security/2015/03/github-battles-largest-ddos-in-sites-history-targeted-at-anti-censorship-tools/`, Mar. 30 2015. [Online; accessed June 2016].

[4] B. Awerbuch and C. Scheideler. Towards a Scalable and Robust DHT. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 318–327, New York, NY, USA, 2006. ACM.

[5] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474, May 2014.

[6] C. Bocovich, J. A. Doucette, and I. Goldberg. Lavinia: An audit-payment protocol for censorship-resistant storage. In *Financial Cryptography and Data Security*, 2017.

[7] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-peer Overlay Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, Dec. 2002.

[8] I. Clarke, O. Sandberg, M. Toseland, and V. Verendel. Private Communication Through a Network of Trusted Connections: The Dark Freenet. [Online]. https://freenetproject.org /papers/freenet-0.7.5-paper.pdf, 2010.

[9] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[10] X. Dai, K. Chaudhary, and J. Grundy. Comparing and Contrasting Micro-payment Models for Content Sharing in P2P networks. In *Signal-Image Technologies and Internet-Based System, 2007. Third International IEEE Conference on*, pages 347–354. IEEE, 2007.

[11] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In *In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, 2000.

[12] J. R. Douceur. The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag.

[13] E. L. Eisenstein. *The printing press as an agent of change*, volume 1. Cambridge University Press, 1980.

[14] I. Eyal and E. Sirer. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In N. Christin and R. Safavi-Naini, editors, *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer Berlin Heidelberg, 2014.

[15] M. Gramaglia, M. Urueña, and I. Martinez-Yelmo. Off-line incentive mechanism for long-term P2P backup storage. *Computer Communications*, 35(12):1516–1526, 2012.

[16] A. Hern. Missing: Hard Drive Containing Bitcoins Worth £4m in Newport Landfill Site. *The Guardian*, 2013.

[17] W. A. Massey, G. A. Parker, and W. Whitt. Estimating the Parameters of a Nonhomogeneous Poisson Process with Linear Rate. *Telecommunication Systems*, 5(2):361–388, 1996.

[18] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[19] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf, 2008. [Online; accessed June 2016].

[20] N. Oualha and Y. Roudier. Securing P2P storage with a self-organizing payment scheme. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 155–169. Springer, 2011.

[21] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag.

[22] S. Seuken, D. Charles, M. Chickering, and S. Puri. Market Design & Analysis for a P2P Backup System. In *Proceedings of the 11th ACM conference on Electronic commerce*, pages 97–108. ACM, 2010.

[23] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[24] A. Singh, T.-W. Ngan, P. Druschel, and D. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, April 2006.

[25] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[26] A. Stubblefield and D. S. Wallach. Dagster: Censorship-Resistant Publishing Without Replication. Technical Report TR01-380, Houston, TX, USA, 2001.

[27] D. R. Thompson, O. Lev, K. Leyton-Brown, and J. Rosenschein. Empirical analysis of plurality election equilibria. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 391–398, 2013.

[28] J. M. Urban and L. Quilter. Efficient Process or Chilling Effects—Takedown Notices Under Section 512 of the Digital Millennium Copyright Act. *Santa Clara Computer & High Tech. LJ*, 22:621, 2005.

[29] E. Y. Vasserman, V. Heorhiadi, N. Hopper, and Y. Kim. One-Way Indexing for Plausible Deniability in Censorship Resistant Storage. In *2nd USENIX Workshop on Free and Open Communications on the Internet*. USENIX, 2012.

[30] M. Waldman and D. Mazieres. Tangler: A Censorship-Resistant Publishing System Based on Document Entanglements. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 126–135. ACM, 2001.

[31] P. Winter and S. Lindskog. How the Great Firewall of China is Blocking Tor. In *Proceedings of the 2nd USENIX Workshop on Free and Open Communications on the Internet*, 2012.

A Game Theory and Security Proofs

A.1 Proof of Theorem 1

Proof. We can now describe the utility (profit) functions for server operators in this game. The utility of an operator is given by the sum of the utilities of the servers they operate:

$$U_{opp}(A_i) = \sum_{S_{i,j} \in S_i} U_{ser}(S_{i,j})$$

while the utility of a server is given by:

$$U_{ser}(S_{i,j}) = \sum_f \Big\{ \Pi_{store}(f)\Pi_{route}(s,self)\Pi_{send}(f)\Gamma_{send}(f) \times (g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f)) - \Pi_{store}(f)c_{store}(f) - \Pi_{route}(s,any)c_{route}\Gamma_{route}(s)\Gamma_{hop}T \Big\}$$

That is, the utility function of the server is equal to the total expected profit from the policy adopted to serve and store each file f (or not), less the fixed costs imposed by the routing policy (i.e. the costs of sending routing information to those who ask for it).

Several facts are immediately apparent from this utility function:

1. Servers cannot benefit from adopting a policy $\Pi_{route}(s, others)$ that sends incorrect routing information, because their utility function is independent of the routing information sent. Similarly, adopting a policy for $\Pi_{route}(s, self)$ that incorrectly reports the keyspace for s is pointless, because $\Pi_{route}(s, self)$ affects *all* files, but $\Pi_{store}(f)$ provides a finer grained control over the same set, allowing servers to selectively serve only profitable files instead. Therefore, policies in which routing information is reported correctly (or not at all) dominate policies where routing information is reported incorrectly. Reporting truncated routing information is also not profitable, because its relatively small size in most distributed storage systems (e.g., about 4 KB in Chord for networks with millions of nodes) means transmission costs for full and partial information are essentially identical.

2. Servers should always respond to routing requests, provided that

$$(\sum_f \Pi_{store}(f) \times \Pi_{send}(f)\Gamma_{send}(f)(g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f)))/(\Gamma_{route}(s)\Gamma_{hop}T) > c_{route}.$$

That is, as long as the routing costs are lower than the ratio of expected profit for all files stored by the server, to the total number of routing requests the server handles, the server should serve files. In practice, most distributed storage systems (for instance, distributed hash tables) perform a lookup in a logorithmic number of steps, so number of queries for routing information that pass through a given node should scale as $\frac{\log \eta}{\eta}$. Since routing costs should be extremely small, we expect that for all practical values of these parameters, server operators should implement the routing protocol correctly.

3. Servers should send any file for which

$$g_{transmit}(f) > \lambda_{BR}(f)c_{transmit}(f)$$

as this means the expected value of transmitting the file is positive. Estimating $\lambda_{BR}(f)$, the average number of requests before an audit, is of course key to the policy adopted by the server for any particular file, and the server must assess whether sudden spikes in demand for a file signal a long-term shift in the rate, or just a short-term fluctuation. However, if it becomes unprofitable to serve the file, any interested user (who has already retrieved the file and who remembers the lookup keyword) can supplement the file's payments by creating a new (separate) audit contract on the same file, effectively increasing $g_{transmit}$. The contract could be for a shorter term than the original, allowing for temporary increases in times of high demand.

4. Servers should store any file for which

$$\Gamma_{send}(f)(g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f)) > \Pi_{store}(f)c_{store}(f)$$

that is, any file for which the expected profits of storing (and presumably serving) it for a given time period exceed the costs of storage. As in observation 3 above, estimating $\lambda_{BR}(f)$ and the long-term effects of short-term fluctuations will be important for the actual policy adopted by a given server.

The theorem follows fairly directly from observations 1–4 above. In the worst case, $\Gamma_{send}(f) = \frac{1}{\eta}$, meaning that requests for the file only reach the node when it is randomly selected as an entry point into the storage system,

which should happen $\frac{1}{\eta}$ times. Further, $\Gamma_{route}(s) \times \Gamma_{hop}$ must be no greater than 1, since every node is visited at most once per lookup. If

$$\frac{\sum_f \Gamma_{send}(f)(g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f))}{\Gamma_{route}(s)\Gamma_{hop}T} > c_{route}$$

for a server, then by observations 1 and 2 above, that server's profits are maximized by always providing *correct* routing information. It follows that $\Pi_{route}(s, all) = R_s$.[7]

Since sending correct routing information for all nodes is a dominant policy, and we have assumed that all stored files are profitable,[8] it follows directly from observation 3 together with the assumption that storage costs are positive (i.e., operators do not earn money by storing files alone), and the constraint

$$\frac{1}{\eta}(g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f)) > c_{store}(f)$$

$$\Rightarrow g_{transmit}(f) > \lambda_{BR}(f)c_{transmit}(f)$$

that profits are maximized by setting $\Pi_{send} = 1$ for every file.

The final part of the dominant strategy profile then follows directly from observation 4, together with the constraint that

$$\frac{1}{\eta}(g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f)) >$$

$$c_{store}(f)$$

.

□

A.2 Proof of Theorem 2

Proof. In the mirroring subgame, s_{new} wants to acquire all content (and associated payment information) with keys for the files it is acquiring from s_{orig}. It may issue payments to one or more of the other three servers to acquire this information. At the same time, s_{orig} wants to prevent s_{new} from taking over control of this keyspace. (In this subgame, we assume that s_{new} will be paid for files in this keyspace after acquiring them, rather than s_{orig}, who is paid for them now. We make the worst-case assumption that, despite the secure routing protocol, an auditor can still discover the copy of a file at s_{orig} if s_{new} does not have one.) The actions available to each player are respectively that: s_{new} may offer payments to each other player via the payment function p; s_{orig} may also offer payments to each other player, via the payment function p'; and each player with a copy of the file (and associated payment information) controls a policy Π_m, which takes on value 1 if they give the file to s_{new}, and value 0 otherwise. The game is played sequentially. First, players s_{new} and s_{orig} set the payment variables to any configuration they prefer. Then s_{f1}, s_{f2}, and s_{orig} each select a value of Π_m to adopt, in light of these variables. The game is played in a sequence of *rounds* unless and until s_{new} acquires f, or some other circumstance causes f to no longer be valuable. We assume that all nodes view money one time step in the future to be worth a fraction γ of the value today (because of the uncertain nature of future earnings). We define the utility functions for the players, for a single timestep, with respect to a single file f with expected long-term earnings of $\beta(f) = (g_{transmit}(f) - \lambda_{BR}(f)c_{transmit}(f) - c_{store}$ as:

$$U(s_{new}) = (\Pi_m(s_{f1}) \vee \Pi_m(s_{f2}) \vee \Pi_{m)}(s_{orig}))\beta(f)$$

$$-\Pi_m(s_{orig})p(s_{orig}) - \Pi_m(s_{f1})p(s_{f1}) -$$

$$\Pi_m(s_{f2})p(s_{f2}) + p'(s_{new})$$

$$U(s_{f1}) = \Pi_m(s_{f1})p(s_{f1}) +$$

$$\neg(\Pi_m(s_{f1}) \vee \Pi_m(s_{f2}))p'(s_{f1})$$

$$U(s_{f2}) = \Pi_m(s_{f2})p(s_{f2}) +$$

$$\neg(\Pi_m(s_{f2}) \vee \Pi_m(s_{f1}))p'(s_{f2})$$

$$U(s_{orig}) = \neg(\Pi_m(s_{f1}) + \Pi_m(s_{f2}) + \Pi_{m)}(s_{orig}))\beta(f) -$$

$$p'(s_{new}) - \Pi_m(s_{f1})p'(s_{f1}) - \Pi_m(s_{f2})p'(s_{f2}) +$$

$$\Pi_m(s_{orig})p(s_{orig})$$

In the long run, if s_{orig}, s_{f1} and s_{f2} all adopt $\Pi_m = 0$, their utilities are:

$$U(s_{f1}) = \sum_{i=0}^{\infty} \bar{p'}(s_{f1})\gamma^i = \frac{1}{1-\gamma}\bar{p'}(s_{f1})$$

$$U(s_{f2}) = \frac{1}{1-\gamma}\bar{p'}(s_{f2})$$

and

[7] Note that the proposed bound is not unrealistic. The marginal cost of transporting a 4 KB Chord finger table at the present time is approximately 10^{-7} USD. Therefore, if the total traffic on the network is approximately 10,000,000 lookups per audit period, a total expected payment of 1 USD/period for *all* stored files combined would suffice. This also represents the worst case value (i.e. when no other nodes in the network respond to any routing requests). Consequently, we posit that all realistic scenarios conform to this requirement.

[8] Again, in a static network, this assumption is realistic. Documents that were not expected to be profitable should not have been stored in the first place.

$$U(s_{orig}) = \frac{1}{1-\gamma}(\beta(f) - \bar{p'}(s_{f1}) - \bar{p'}(s_{f2}))$$

where $\bar{p'}$ is the *average* payment given to s_{f1} or s_{f2} in the long run. For $U(s_{orig})$ to be positive in the long run, it follows that $\beta(f) > \bar{p'}(s_{f1}) + \bar{p'}(s_{f2})$. However, if $\bar{p'}(s_{f1}) \neq \bar{p'}(s_{f2})$, then s_{new} may pay a lesser amount to the less-well paid one to induce mirroring. Maximum resistance to mirroring is achieved when $\bar{p'}(s_{f1}) = \bar{p'}(s_{f2})$, so for $U(s_{orig})$ to be positive, and each other server paid as much as possible, it must be the case that $\bar{p'}(s_{f1}) = \bar{p'}(s_{f2}) = \frac{\beta(f)}{2}$. If the average payments exceed this amount in the long run, then s_{orig} would have been better off allowing mirroring to occur.

Given the utility functions above, it is obvious that a one-time payment of $\frac{1}{1-\gamma}\frac{\beta(f)}{2} + \varepsilon$ exceeds the long-term expected earnings of either s_{f1} or s_{f2}, and will thus be accepted by a profit maximizing agent. Further, s_{new} should be happy to make such a payment, provided their discounting factor γ is similar to those of the mirroring servers, as they will still recover half the long-run value of the file. Since s_{orig} makes no payments if defection occurs, there is no incentive to avoid offering a payment of $\frac{\beta(f)}{2}$ to both s_{f2} and s_{f1}. □

It is interesting to note that, if the joining node s_{new} is aware of additional caches (e.g., s_{orig} nodes that continued storing content after previous joins), then an even lower price can be offered. We speculate that, in the long run, an equilibrium is reached when the storage costs of f for the caching nodes are approximately equal to the expected profit from joins that use the strategy outlined above.

Importantly, the above proof depends on the exact values of γ that are used. We suppose that rational agents have similar values of γ, because it represents the uncertainty about the future (which should affect all similarly in expectation). However, in practice, human operators may have wildly different γ values. We note that if s_{f1} and s_{f2} have different values of γ, but do not know each others' exact γ value, they should still accept offers based on a globally average γ, under the belief that their opponent (who will eventually be offered the same value) is just as likely to have an above-average γ as a below-average one. Additionally, the proof depends on all players having some reasonable estimate of $\beta(f)$. Ultimately this could be obtained through incrementally increasing the offers made by the p function in s_{new}'s equilibrium strategy. s_{orig} has no incentive to make offers that produce negative income (it is not malicious, merely profit maximizing), and so should give up when the bid values exceed $\frac{1}{1-\gamma}\frac{\beta(f)}{2}$.

Note that the equilibrium we describe is not necessarily unique. However, if s_{new} plays according to the equilibrium, the other players will maximize their profits by doing so as well. Our result therefore provides an *upper* bound on the price s_{new} must pay to acquire content during a join. Another obvious, and seemingly preferable, policy is one where s_{orig} gives s_{new} the content freely. This has the same expected value for s_{orig} as the equilibrium in our proof, and is better for s_{new}, but is *not* an equilibrium of the outlined game. If s_{orig} does not make randomized offers to the caching players, than s_{orig} could improve its profits by making no payments to those players, and also not providing f to s_{new}. Consequently, play of this kind is unstable. Indeed, there is no equilibrium where s_{new} can pay s_{orig} less for the file than it could pay either caching player. We emphasize again however, that altruistic operators might be happy to mirror content free of charge (especially since doing so has the same expected profits), so our provided results are for the worst case.

A.3 Audit Protocol Security Proof

We claim our audit protocol is secure if an attacker is unable to:

1. compute the value of skA for any audit time t, unless they are the auditor at that time, or time t has passed and they are the auditor in the subsequent time period,

2. receive a payment for auditing a file f at time t, unless they have retrieved the file f from the system sometime after the previous audit time, t_{prev}, has passed, and

3. receive a payment for serving a file f at time t, unless they have served the file f some time after the previous audit time, t_{prev}, has passed.

Theorem 3. *The Lavinia audit protocol is secure.*

Proof. We first prove that an attacker is unable to perform attack (1) above. Let us assume for the sake of contradiction that an attacker can learn information about the auditor's secret skA. There are only four messages that depend on skA: $kw(r, skA)$, $kw(H(f\|skA), skA)$, $kw(H(f\|skA), r_s)$, and $kw(skA', skA)$ as well as the protocol for commiting to a contract $\mathscr{P}$ and proving the possession of a secret in the transaction $\mathscr{T}$. Learning skA from any of these messages, without knowledge of r or skA' violates our assumption on the security of the key-wrapping function kw, the collision and pre-image resistance of the hash function H, and the requirements we placed on our payment system in subsection 3.2.

We now show that an attacker cannot perform attack (2). If the attacker is not tasked with auditing the file f at time t, they do not possess $kw(r, skA)$ and therefore cannot perform a transaction $\mathscr{T}(proof(skA), kw(r, skA))$ and cannot receive a payment for auditing the file. Let us then assume that the attacker is an auditor at this time and possesses the values t, $kw(lookup(f), skA_{prev})$, $kw(H(f \| skA_{prev}), skA)$, r_{prev}, r, and r_s. The auditor can not fulfill the contract $\mathscr{P}$ without the secret skA. To unlock this secret, the auditor must possess the previous audit key skA_{prev} and the file f. Since skA_{prev} will not become available until the audit time t_{prev} has passed, the attacker is unable to collect a payment before time t_{prev}. Additionally, the auditor has no way of knowing which file f they are in charge of until they unlock the lookup key. This can only be done with skA_{prev}. Therefore, the attacker can not collect payment unless they have retrieved the file f from storage after t_{prev} has passed. As we mentioned in subsection 4.4, as long as Alice chose her auditors reasonably, there is a very small probability that an auditor will have cached the file f from a previous audit time.

Finally, we show that an attacker cannot perform attack (3) above. A server can only receive the payment for serving file f at time t if it has the corresponding payment key skS. This is locked with r_s, which may be sent by the auditor at time t. This value is locked with the same key as skA. This reduces the attack to performing attack (2). □