LOCKSS: Lots Of Copies Keep Stuff Safe

David S. H. Rosenthal
LOCKSS Program, Stanford University Libraries, CA

Abstract

Over the last 12 years the LOCKSS Program at Stanford has developed and deployed an open source, peer-to-peer system now comprising about 200 LOCKSS boxes in libraries around the world preserving a wide range of web-published content. Initially supported by NSF, and subsequently by the Mellon Foundation, Sun Microsystems and NDIIPP, the program has since 2004 been sustainable, funded by the libraries using it. The program won an ACM award for breakthrough research in fault and attack resistance in peer-to-peer systems.

Since it was designed initially for e-journals, the system's design is unusual; it is driven primarily by copyright law. The design principles were:

- Minimize changes to existing legal relationships such as subscription agreements.
- Reinstate the purchase model of paper. Each library gets its own copy to keep and use for its own readers as long as it wants without fees.
- Preserve the original, just what the publisher published, so that future readers will see all the intellectual content, including the full historical context.
- Make access to the preserved content transparent to the reader.

1 Introduction

Over the last twelve years the LOCKSS[1] Program at the Stanford Libraries has developed and deployed a free, open source, peer-to-peer system now comprising about 200 LOCKSS boxes in libraries around the world preserving a wide range of web-published content. The impetus for the program came from Stanford Libraries' pioneering work in e-journals; in May 1995 their HighWire Press [12] unit unveiled the Web edition of the *Journal of Biological Chemistry*. The sudden advent and immediate popularity of the Web versions of expensive and important academic journals forced an unexpected and unwelcome transition upon librarians: from *purchasing* a copy of the journal to *renting* access to the publisher's copy. While rental satisfied the librarians' responsibility to current readers, it did not satisfy their responsibility to future readers. It did not provide "perpetual access".

2 Design

One way to satisfy the goal of providing libraries with perpetual access to the e-journal content to which they subscribed was commonly suggested. This involved the concept of a *third-party archive*, which would acquire e-journal content from the publisher and re-publish it. In addition to subscribing to the journal itself, libraries would subscribe to the archive. If they found themselves having to cancel a subscription, they could apply to the archive to access the archive's copy of content the journal had published during the period their subscription was valid.

This concept was presented as cancellation insurance. But another way of looking at it was as a second instance of exactly the problem it was trying to solve. Access to the canceled content would be provided only as long as the library continued to subscribe to the archive. This fundamental flaw in the third-party archive model was compounded by the very significant legal and business difficulties to which it gave rise.

The nature of technology markets, with increasing returns to scale [3] or network effects, is to be dominated by a single supplier. Thus it was likely that there would be one such archive containing the content from *every* publisher. Publishers were to be asked to give, or license on non-commercial terms, their intellectual property assets to this archive, which was both an *actual* competitor, in that the funds libraries used to pay their archive subscription came from the same budget as their subscriptions to the publishers, and a *potential* competitor, in that it held and was able to re-publish the content of every publisher. The negotiations with the publishers leading to an agreement to provide the archive with content would thus be prolonged, difficult and expensive[2].

[1]LOCKSS is a trademark of Stanford University.

[2]This problem was not to be resolved until the Mellon Foundation applied its very considerable leverage in 2002.

The LOCKSS Program was started with a “Small Grant for Exploratory Research” (SGER) from the NSF. Prolonged and expensive negotiation with lawyers from the publishers was not simply unattractive, it was unaffordable. The design of the system was thus driven by the need to satisfy the demands of both copyright law (in particular the Digital Millennium Copyright Act (DMCA) [1]) and the librarians. The original design principles were:

- Minimize changes to existing legal relationships such as subscription agreements.
- Reinstate the purchase model of paper, so that each library gets its own copy to keep and use for its own readers as long as it wants without fees.
- Preserve the original, just what the publisher published, so that future readers will see all the intellectual content, including the full historical context.
- Make access to the preserved content transparent to the reader, so that librarians would not need to educate their readers to take advantage of the system.

These principles led to the key aspects of the original design [33]:

- Publishers would grant permission for subscribers to collect and preserve the content to which they subscribed by placing a permission statement on a web page visible only to subscribers.
- Libraries would collect the content, including the permission statement, by crawling the publisher’s web site using their subscription access. This avoided the need to develop a separate mechanism to determine whether a library had permission and a subscription for the content. If they did not they wouldn’t be able to get the content. And it avoided the need to modify the publisher’s existing systems, something they were reluctant to do.
- Libraries’ preserved content would act as a proxy for the publisher’s web site. If the publisher was unable to supply it, or refused to do so because the subscription was no longer valid, the preserved content would be supplied. Otherwise, the publisher’s content would be supplied to the library’s reader.
- Only the library’s readers would have access to the content, and the publisher would see all accesses to the content as if they had been made directly. This ensured that the system would not add to the risk of content theft, nor prevent publishers profiting from the hits on their web pages.

This design led to an unusual opportunity. Clearly, any system for long-term digital preservation must be highly fault-tolerant, and thus replicated. Because each library had to maintain its own copy of the content as evidence that it was doing so with the permission of the publisher, the system as a whole had many replicas. Instead of the question normally at the heart of the design of fault-tolerant systems, “how few replicas do we need?”, the question for the LOCKSS system was “what can we do with the abundance of replicas to increase fault-tolerance?”.

The obvious next question, too infrequently asked of digital preservation systems [41], is “against what threats is the content being preserved?”. These threats would cause the faults the system must tolerate. Many digital preservation systems have implicitly adopted the threat model of Jeff Rothenberg’s influential 1995 article [35]; this has focused attention to an unwarranted extent on the threat of format obsolescence [31]. The LOCKSS system’s threat model [34] included format obsolescence [32] and “bit rot” [30], but was also based in the experience of:

- Paper research libraries, which regularly encounter attempts by fanatics [43] and governments [25] to suppress information, and disasters such as fire [2] and flood [21].
- Large computing facilities, whose experience shows that data loss is frequently caused by operator error [28] and insider abuse [17].

Libraries in the paper world are used to peer-to-peer relationships, such as inter-library loan and copying. A peer-to-peer architecture for the LOCKSS system was thus natural, and essential to defense against threats such as operator error and insider abuse. These threats spread via the trust relationships inherent in centralized, hierarchically controlled systems.

3 Implementation

The initial NSF SGER supported the development of prototype nodes, called LOCKSS boxes, that formed a peer-to-peer network. Each held its library’s copies of the subscribed content, which it obtained by crawling the publisher’s web site. The nodes communicated with each other, locating other copies of the same content and regularly comparing them, using a peer-to-peer *anti-entropy protocol* based on voting on the message digest (digital hash) of the content. If, using this protocol, a node discovered that its copy disagreed with the consensus of the other nodes, it could request that its copy be repaired from a node that did agree with the consensus. A node receiving a repair request would supply the repair only if it remembered agreeing with the requester as to the content in the past. Thus the protocol could be used only to repair a previously good copy obtained from the publisher, not to violate the publisher’s copyright by obtaining an initial copy from another library. The prototype was implemented as a long-lived daemon process in Java.

The prototype was deployed in 2000 to 6 libraries using test content from HighWire Press journals *Science* and the *British Medical Journal*, by kind permission of their publishers. The test program was later roughly doubled in size with funding from the Mellon Foundation and Sun Microsystems.

Experience with the prototype was encouraging enough that the Mellon Foundation and the NSF funded the development of a production system. This was a complete re-write of the system, incorporating the lessons of the prototype:

- The prototype clearly demonstrated that the LOCKSS approach of a peer-to-peer distributed digital preservation system with low-cost nodes at individual libraries was technically, legally and economically feasible.
- The prototype showed that crawling the publisher’s web sites was a viable method for collecting the content,

and that it allowed a very faithful representation of the reader's experience to be preserved. However, the variation among publishers and the extent of the detailed control of the crawling process required to keep relations with the publisher harmonious, required that publisher-specific knowledge be encapsulated in a set of Java classes interfacing to the rest of the daemon via a "plug-in" interface.

- Adequate reliability required that the nodes in the LOCKSS network be extremely secure. Administering systems to this level of security was seen as difficult for typical library staff [11]. The production system was packaged as a *network appliance* based on the security-conscious OpenBSD operating system, but running from a read-only CD rather than from software installed on a read-write hard disk [29]. "Live CDs" like this have become popular.
- The prototype's anti-entropy protocol used IP multicast, which turned out to be impractical. It was replaced by a TCP-based gossip protocol called LCAP (Library Content Audit Protocol) version 1.

Further research showed that the design of LCAP version 1 was flawed; it was vulnerable to several potential attacks [22], and it was later discovered that the improved version 2 protocol under development to replace it was vulnerable to a related attack. A team under Prof. Mary Baker at Stanford's Computer Science Dept. developed an entirely new approach to tolerating faults in and attacks against peer-to-peer networks without requiring central administration or long-term secrets [19, 8], that exploited the abundance of replicas as the basis of its defenses. For this work they were awarded "Best Paper" at the 2003 SOSP workshop, and an ACM Student Research award. This research forms the basis for the current, version 3, anti-entropy protocol although the complete set of defenses has yet to be fully implemented.

4 Deployment

The production system was deployed in a beta test to 50 libraries in 2002, and went in to production in 2004. The number of LOCKSS boxes in production use is about 200. A LOCKSS box preserving all the content available for the public network that libraries use to preserve subscription and open access needs at least 2 terabytes of storage.

LOCKSS boxes' ease of use and simple you-scratch-my-back-I'll-scratch-yours organizational model proved attractive in fields other than e-journals. With support from the Library of Congress' National Digital Information Preservation Program (NDIIPP), a group of libraries in the South-East of the U.S. came together as the MetaArchive [38] to preserve each other's collections of local culture and history. This was the first Private LOCKSS Network (PLN), a network of nodes separate from the main, public network open only to members of a specific organization. PLNs have proliferated since, preserving many other genres of content, including state records, government documents, and datasets. The largest LOCKSS boxes in use in PLNs have about 16 terabytes of storage, they are the nodes in the CLOCKSS network. This is a community-governed dark archive building comprehensive collections of content from large and small e-journal publishers. There will eventually be about 15 nodes in the network scattered across the globe.

5 Interoperability

There are four aspects of interoperability among digital preservation systems of interest; *Content*, *Metadata*, *Audit* and *Module* interoperability.

5.1 Content Interoperability

As technologies evolve and institutions rise and fall, it will at times be necessary to transfer content from one digital preservation system to another.

LOCKSS boxes contain collections of web content, the result of web crawls. The standard for representing such collections is WARC files [15], an enhancement of the Internet Archive's ARC file format [5], which packs the content and headers of many URLs into a single large file.

LOCKSS box content can be exported as ARC files in one of two ways; the LOCKSS box can export the ARC files directly, or the Heritrix web crawler [23] can crawl the original web site using a special proxy implemented in the LOCKSS box. In both cases the resulting ARC file appears as if Heritrix had crawled the original web site without the involvement of the LOCKSS box[3]. Note that the special proxy used by Heritrix could be used by any other Web crawler; exactly the same content can be harvested from the LOCKSS box at exactly the same URLs where it was originally found. Thus, although it can use ARC files as an export format, it is in no way dependent on doing so.

LOCKSS boxes can also import content in ARC format directly. In this case the result is as if the LOCKSS box had crawled the original web site instead of Heritrix. Content has been transferred from the Internet Archive's Archive-It service [14] to a PLN run by the University of Rochester. ARC file import is a special case of LOCKSS boxes' ability to import packaged content. Variants of this capability are used by the CLOCKSS program to import e-journal source file content in the formats used by Elsevier [7] and Springer. Again, although LOCKSS boxes can use ARC as an import format, they are in no way dependent on doing so.

These mechanisms are being updated to use the standard WARC format. LOCKSS currently stores web content as individual files in a POSIX file system. This part of the system is also being upgraded to store the content in WARC files, in a way similar to that of the Wayback Machine [13].

5.2 Metadata Interoperability

There are two kinds of metadata of interest for preservation; *format* and other metadata relevant to ensuring the future readability of the content, and *bibliographic* metadata relevant to finding the content.

The standard for format metadata for Web content is the MIME type in the HTTP headers, and additional "magic number" information in the HTTP payload. LOCKSS boxes preserve both the HTTP headers and the payload for every URL, and thus have adequate format metadata for rendering.

The LOCKSS approach for format obsolescence is to preserve the original bits and, if necessary, transparently create a

[3]Except for some additional HTTP headers

temporary access copy of the content in a less-obsolete format when a reader requests access [32]. There is thus no need to collect and preserve the output from format identification and verification tools such as JHOVE [10]; it is in any case doubtful whether doing so actually contributes to future readability [31].

The widely-accepted Dublin Core standard for bibliographic metadata is useful but not in practice strict enough to enable interoperability without extensive human intervention [36]. This is not feasible at the scale involved in preserving e-journals.

LOCKSS boxes are capable of extracting the article-level bibliographic metadata contained in the Web content they harvest. Most e-journal publishers include the DOI and Dublin Core metadata in the Web articles they publish, some in HTML meta-tags, some in the text. The LOCKSS plugins contain publisher-specific code that knows where this information can be found, how to convert it to a internal standard representation, and how to relate it to a system-wide database of journal- and volume-level metadata called the "title database" (TDB).

The German Research Foundation (DFG) is funding LuKII, a collaboration between the German National Library (DNB), Humboldt University and the LOCKSS team to investigate the use of METS to facilitate the metadata aspects of interoperation between the DNB's KOPAL system and a LOCKSS PLN run by Humboldt University [36].

5.3 Audit Interoperability

The requirements for audits of digital preservation systems are discussed in Section 6.1. Here we simply observe that if two or more systems claim to be preserving the same content, it would be useful for them to be capable of performing a *mutual audit*, proving to each other that they each had a copy of the content in question, and that their copies were the same. A protocol for doing so would be valuable; none has so far been standardized.

This process of mutual audit is the heart of the LCAP anti-entropy protocol. One LOCKSS box calls a *poll* on some content C by inviting a number of other boxes holding the same content and supplying them with a random nonce $N1$. Invitees, or *voters*, each generate a second random nonce $N2$, compute the digest $H(N1, N2, C)$ and return $N2, H(N1, N2, C)$ to the poller. The poller tallies these votes by comparing the values they contain for the hash with the voter's values of $N2$ and C, with the values the poller computes using its value of C and the various values of $N2$ in the votes [4]. If the values agree, the voter has proved to the poller that their copies of the content C are identical.

The messages in the LCAP protocol are in XML and contain only opaque string identifiers for the content, hash values and time stamps. The actions needed to respond to them are simple and well-defined. These attributes make the LCAP protocol a suitable candidate for a mutual audit protocol standard.

Detection of loss or damage to a replica is one half of an anti-entropy protocol. In the digital preservation literature this half is usually described as a *fixity check*, and is normally assumed to be accomplished by means of a message digest stored in associated with the content it refers to. A mismatch between the stored digest and the stored content indicates damage. But is this damage to the content, or to the digest? Insider abuse or external attack, both realistic threats, could lead to both the content and the digest being modified so as still to match, so a match is not actually a guarantee of fixity. Further, either or both of the content and the digest could be lost. Simple digest comparisons are not adequate even for internal fixity checks; cryptographic techniques to ensure the integrity of the digest are needed.

[4]The precise details of this protocol and how they defend against the various possible attacks are described in [19].

The other half of an anti-entropy protocol is *repair* of any damage or loss that is detected. Without a repair protocol, damage will simply accumulate and eventually overwhelm any replication scheme [4]. The LCAP protocol implements repair (see Section 3), and can do so in a way that prevents violations of copyright.

5.4 Module Interoperability

The LOCKSS team hopes shortly to start working, with funding from the Library of Congress, towards making the core LCAP version 3 anti-entropy protocol available for reuse by other systems in the form of a Java library. This work would initially be targeted towards the problem of auditing content preserved in cloud storage systems. It will also incorporate lessons from the LuKII project [36].

6 Lessons

6.1 Audit

Funders pay for digital preservation in the present in the expectation of receiving value, access to preserved content, in the future. This business model is analogous to insurance, and like insurance in the absence of regulation based on auditing would be open to fraud, abuse and incompetence.

Discussions of how digital repositories should be audited have focused primarily on ISO9000-style checklists of written policies. Clearly, these are useful both internally, as models of good policies, and externally, to distinguish between repositories that are adhering to current best management practices and those that are not. It must be noted that audits of this kind are time-consuming, expensive, and cannot be automated. Their cost-effectiveness is yet to be established.

Nevertheless, it is also important that repositories containing digital objects be audited by third-party auditors to confirm that (a) they contain the digital objects they are supposed to, and (b) that the objects are undamaged. Even the best-documented policies do not guarantee these essential attributes, nor will audits of written policies establish that the attributes hold.

Clearly, the idea that a human could audit a system containing terabytes of data by accessing a few documents and reading them to see if they "looked OK" is laughable. A successful digital preservation system must be extraordinarily reliable; keeping a petabyte for a century with a 50% chance that every bit survives requires a bit half-life roughly 60 million times the age of the universe. Although the observed reliability of state-of-the-art storage systems fails to meet this goal by a factor of at least 10^9, it is still high enough to pose a significant problem for auditors [30]. They have two possible approaches; they can *sample* the archive's content or they can conduct a *full* audit of the entire content. To detect the low rates of damage actually observed, let alone the possibility of targeted attacks on particular objects, in a timely fashion requires a full audit.

Thus, just as the system internally must conduct fixity checks on every bit of each of its replicas at regular intervals [4], so must external auditors. Since the majority of the cost of these checks is in the I/O to access the objects [40], the cost of external audits can be greatly reduced if they can be combined with the internal fixity checks by accessing the object once then performing both checks.

Even if the access restrictions on deposited content allowed it, performing an audit by extracting most or all the repository's content and transferring it to a third-party auditor is not feasible at the scale of current and future repositories. Both the cost [18] and the time required would be prohibitive. Further, experience shows that at large scales such mass transfers of digital objects are themselves subject to errors [42]. These errors would lead to false negatives, the auditor would report that an object had been damaged when in fact the copy in the archive was undamaged.

On the other hand, trusting the repository to self-audit and report the results is not adequate. Recent financial scandals such as Enron [20] and Lehman Bros. [44] show all too clearly the problems caused by too credulous an auditor. In an environment where a single data loss incident could destroy a repository's credibility, and thus its business model, the temptation to cover up data loss is strong. Despite anecdotal reports of such losses, no repository appears willing to report one. Suggestions [30] for an anonymized incident reporting system similar to NASA's Aviation Safety Reporting System [24] have gone nowhere.

Suggestions are frequently made that third parties audit repositories using the preserved hashes of digital objects [39]. When temptations to subvert the audit exist, this conventional use of hashes is inadequate. To understand why, consider the following example. The auditor knows that the repository should contain digital object X and that the hash of $X = H(X)$ should be Y. The auditor asks the repository what $H(X)$ is. The repository replies Y. What has this established? Merely that the repository knows $H(X)$. It could know this without containing a good copy of X. For example, it could have ingested X, computed $H(X)$, stored $H(X)$ and discarded X. When the auditor asks for $H(X)$, it replies with the stored value of $H(X)$. Alternatively, the repository could search the Web for keywords from X [27], notice that someone else had posted X and $H(X)$, and not even bother with all that ingesting and computing.

Thus there is an apparent dilemma. It isn't feasible to extract the content, so the repository has to perform the audit. But if the repository performs the audit, the result isn't credible.

The requirements for a third-party auditor of a large digital repository are thus:

1. The auditor must not trust the repository being audited. Doing so renders the audit worthless.

2. The auditor must not depend on extracting content from the repository. Doing so renders the audit unaffordable.

3. The auditor must not depend on processing the content as it is being ingested into the repository. Since experience shows that content ingest is a large part of the cost of preservation, doing so would render the audit unaffordable. It would also lock that content in the repository into forever being audited only by the one auditor in place during ingestion.

4. The audit must not depend on metadata, such as digests, about the content being perfectly preserved in a registry. Doing so turns the registry itself into a repository, which must itself be audited. In other words, the auditor must be fault-tolerant.

5. Ideally, the audit checks should be share access to the content with the repository's internal fixity checks.

It appears that no existing audit technology satisfies all these requirements. Some audit technologies are assessed against them in Appendices A and B.

Section 2 showed that the network effects inherent in digital publishing made it likely that one digital preservation system would dominate the market, and thus that all copies of most preserved content would reside in a single organization's preservation system. If we add the requirement that third-party audits be possible even in this case, it seems probable that no *possible* audit technology can satisfy them. Thus, to the obvious all-eggs-in-one-basket risks of a digital preservation mono-culture must be added the likely impossibility of performing a trustworthy audit of how well the mono-culture is performing.

6.2 Transparency

The LOCKSS system was designed to address the problem that moving Web content somewhere else reduces its value by breaking the links to it, and by reducing the ranking search engines give it. Because the LOCKSS daemon was designed to act as a proxy from the original publisher's web site, not as a substitute web server, the preserved content remained accessible at its original URL. Both internal and external links continued to work without needing to be rewritten. Rewriting links is time-consuming and error-prone.

The LOCKSS system was transparent to readers, it just made the content much more durable. The problem was that, precisely because it was transparent, it did not visibly provide any value. In fact, if it was installed and configured as originally designed, only a Web expert could detect that it was doing anything at all. This made it hard to persuade libraries to pay for it. In order to get paid, the capability for LOCKSS to act as a substitute Web server, and rewrite the links, had to be added. This made access to the preserved content not transparent, and added complexity to the user experience.

The recent Memento proposal [45] addresses this issue in a more comprehensive way by extending the underlying HTTP mechanisms rather than, as with proxying or URL rewriting, trying to evade them.

6.3 Licensing

In many common cases content interoperability is not just a technical problem. If the content is copyright, and even open access Web content must be treated as copyright, transferring it might violate terms of the license agreement under which it was originally preserved, or the DMCA, or whatever more restrictive copyright law might be in force at the time of transfer.

The choice of license is thus important for preservation. Creative Commons licenses [6] permit all the activities necessary for preservation including content transfer and should be used if at all possible. Failing that, transfer to a successor archive without further negotiation should be allowed under the license; the "orphan works" problem [26] shows that by the time such negotiations are needed they may no longer be possible.

7 Conclusion

Experience with the LOCKSS system has shown that a distributed, peer-to-peer system based on collaboration between libraries is a both a technically and an organizationally viable way to preserve subscription e-journals and other copyright content. It has demonstrated content interoperability with other preservation systems, and work is under way to explore the other aspects of interoperability.

Fundamentally, digital preservation is a problem in computer, or rather information security. The design and evaluation of digital preservation systems thus requires an adversarial mindset, a "Black Hat" approach. The lesson for interoperability between repositories is that the repository you are interoperating with may not be your friend.

A Audit Control Environment

The Audit Control Environment (ACE) [39] claims to implement third-party audits of digital objects in repositories:

> "our methodology allows a third-party independent auditor to verify the integrity ... of an archived digital object" [39]

It does so by storing an *integrity token* associated with each digital object in the archive. The token contains the hash of the object together with cryptographic information sufficient to detect any corruption of or tampering with the token, using technology due to Haber *et al.* [9].

An audit consists of a request from the third-party auditor to the Audit Manager (ACE-AM) module to:

- evaluate the validity of the integrity token, and if it is valid
- hash the object and compare the result with the hash in the integrity token,
- report success or failure of the comparison to the auditor (the Integrity Management System ACE-IMS).

To avoid the need to extract each object to be audited from the repository and transmit it to the auditor, the Audit Manager runs inside the repository itself:

> "The ACE Audit Manager (ACE-AM) is local to an archiving node whose main function is to pass information between the archiving node and the ACE-IMS. ... It then retrieves the digital object's integrity token, computes the hash of the object, and sends this information to the ACE-IMS." [39]

As a result, the system as described fails to implement a third-party audit. The auditor asks the repository to audit itself; it trusts code it believes (apparently without evidence) is running there to do so and report the results correctly. The auditor only receives a message from the repository saying "this object is good, it hashes to H"[5], it has no independent evidence that the object is in fact good.

Although as described [39, 40] ACE's Audit Manager runs in an untrusted environment local to the repository and thus does not implement a credible third-party audit, the ACE team also describe [16] a true third-party audit in which the Audit Manager runs inside the auditor's trusted environment and each object to be audited is transferred there. This is feasible for small collections, but for large collections this requires sampling, rendering the audit unlikely to detect loss or damage at the rates actually observed 6.1.

Assessing ACE against the requirements of Section 6.1 shows that it does not satisfy any of them:

1. ACE trusts the repository being audited, since the Audit Manager runs in the repository's environment.
2. If it is not to trust the repository being audited, it must extract the entire content from the repository on each audit.
3. ACE must see the content as it is being ingested into the repository in order to compute the hash in the integrity token without trusting the repository to do so.
4. ACE depends on the repository preserving metadata, the integrity token, undamaged indefinitely. ACE is certainly capable of *detecting* damage to the token but provides no mechanism for *recovering* from such damage. It is thus not fault-tolerant.
5. ACE is not able to combine its hashing with the repository's internal fixity checks without trusting the repository.

B Shah *et al.*

Shah *et al* [37] describe a system that implements a third-party audit of digital objects in a repository without transferring them to the auditor on each audit. They propose that the repositories store both the data in encrypted form and the key that encrypted it[6]. Each audit must therefore confirm that both the encrypted data and the key are present and undamaged, but the auditor must not be able to determine the key.

Simplifying Shah *et al.* greatly, digital object X is sent to the repository in encrypted form together with its key K. The repository publishes the hash of the encrypted data $H(E(K, X))$, and a one-way cryptographic transform of the key $T(K)$. The owner of the data contracts with an auditor to perform regular audits of object in the repository, by sending it $H(E(K, X))$ and $T(K)$. The auditor must reliably remember these values. The auditor initializes the audit process by

[5]Step 3 in the description of the operations of the Audit Trigger, a component of the Audit Manager running in the repository, starts:

> "If the integrity token is verified to be intact in Step 2, the Audit Trigger computes a hash of the given object and compares it to the one in the integrity token." [39]

It is thus easy to see that the repository could, by extracting the hash from the integrity token, construct a correct response to the Integrity Management System without hashing the object, or even having a copy of the object to hash.

[6]There are concerns about the use of encryption for long-term data storage. It does not appear essential to this scheme that the data be stored in encrypted form, only that it be encrypted with its key during ingest, extraction and processing

obtaining the encrypted object $E(K, X)$ from the repository. Note that the auditor does not know K and cannot invert $T()$ to obtain it, so does not know X.

The auditor computes $H(E(K, X))$ to confirm that the repository currently has an undamaged copy of $E(K, X)$. It then generates and securely and reliably remembers a set of N *challenges*. Each challenge consists of a random number R and the corresponding hash $H(R, E(K, X))$. It then discards $E(K, X)$.

On each of the subsequent N audits, the auditor removes a challenge at random from the set, sends R to the repository, and waits for it to respond with $H(R, E(K, X))$. If this response matches the challenge, the auditor knows that the repository has an undamaged copy of $E(K, X)$. It must then confirm, using its remembered copy of $T(K)$, that the repository also has an undamaged copy of K. The details of how this is accomplished without revealing K to the auditor are too complex to describe here.

After N such audits, the set of challenges will be empty. The auditor must be re-initialize the process by obtaining the encrypted object $E(K, X)$, generating and remembering a set of N new challenges, and then again discarding $E(K, X)$.

This is a rigorous third-party audit that requires the auditor to access the digital object only once in every N audits, does not trust the repository being audited and does not reveal the object to the auditor. It does depend on reliable long-term storage of secrets (the remaining challenges) by the auditor. Like ACE, it can be applied in cases where the digital object is stored by only one repository, although such cases should be deprecated.

Assessing this system against the requirements of Section 6.1 shows that it fully satisfies one and partially satisfies some others:

1. It does not trust the repository being audited.
2. It requires, on average, data equivalent to only $1/N$ of the repository's content ($E(K, X)$) to be extracted per audit.
3. It must see data equivalent to the content ($E(K, X)$) as it is being ingested into the repository.
4. It depends on the auditor preserving metadata, the challenges, undamaged for a period. It is thus not fully fault-tolerant.
5. It can combine its hashing with the repository's internal fixity checks.

C Acknowledgments

This work was supported by the LOCKSS Alliance, the CLOCKSS Archive, and the Library of Congress. Grateful thanks are also due to the publishers who give permission for LOCKSS and CLOCKSS boxes to preserve their content and its metadata, to the LOCKSS engineering team, and to the authors and maintainers of the open source infrastructure upon which we depend; OpenBSD, Linux, and the Java environment. Special thanks are due to Michael Lesk for our initial SGER. Opinions and conclusions are the author's alone.

References

[1] 105th Congress, United States of America. Public Law 105-304: Digital Millennium Copyright Act (DMCA), October 1998.

[2] American Library Assoc. German Library Fire Destroys Thousands of Rare Books. `http://www.ala.org/ala/alonline/currentnews/newsarchive/alnews2004/september2004abc/weimar.cfm`, September 2004.

[3] W. Brian Arthur. *Increasing Returns and Path Dependence in the Economy*. University of Michigan Press, 1994.

[4] Mary Baker, Mehul Shah, David S. H. Rosenthal, Mema Roussopoulos, Petros Maniatis, TJ Giuli, and Prashanth Bungale. A Fresh Look at the Reliability of Long-term Digital Storage. In *Proceedings of EuroSys2006*, Leuven, Belgium, April 2006.

[5] Mike Burner and Brewster Kahle. ARC File Format. `http://www.archive.org/web/researcher/ArcFileFormat.php`, September 1996.

[6] Creative Commons. Web site. `http://creativecommons.org/`.

[7] Elsevier. ScienceDirect: Product Specifications. `http://info.sciencedirect.com/implementation/implementing/sdos/`, March 2006.

[8] TJ Giuli, Petros Maniatis, Mary Baker, David S. H. Rosenthal, and Mema Roussopoulos. Resisting Attrition Attacks on a Peer-to-Peer System. In *Usenix Annual Technical Conf.*, Anaheim, CA, USA, April 2005.

[9] Stuart Haber and W. Scott Stornetta. How to Time-stamp a Digital Document. *Journal of Cryptology: the Journal of the International Association for Cryptologic Research*, 3(2):99–111, 1991.

[10] Harvard University Library. Format-Specific Digital Object Validation. `http://hul.harvard.edu/jhove/`, 1999.

[11] Cormac Herley. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *NSPW '09: Proceedings of the 2009 workshop on New security paradigms workshop*, pages 133–144, New York, NY, USA, 2009. ACM.

[12] HighWire Press. About HighWire Press. `http://highwire.stanford.edu/about/`, 2009.

[13] Internet Archive. The Internet Archive Wayback Machine. `http://www.archive.org/web/web.php`.

[14] Internet Archive. Welcome to Archive-It. `http://www.archive-it.org/`.

[15] ISO. ISO 28500:2009 Information and documentation - WARC file format. `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=44717`, May 2009. Draft at `http://bibnum.bnf.fr/WARC/warc_ISO_DIS_28500.pdf`.

[16] Joseph JaJa. Personal communication, March 2010.

[17] Michelle Keeney, Eileen Kowalski, Dawn Cappelli, Andrew Moore, Timothy Shimeall, and Stephanie Rogers. Insider Threat Study: Computer System Sabotage in Critical Infrastructure Sectors. `http://www.secretservice.gov/ntac/its_report_050516.pdf`, May 2005.

[18] Allan Leinwand. The Hidden Cost of the Cloud: Bandwidth Charges. `http://salon.com/tech/giga_om/tech_insider/2009/07/17/the_hidden_cost_of_the_cloud_bandwidth_charges/`, July 2009.

[19] Petros Maniatis, Mema Roussopoulos, TJ Giuli, David S. H. Rosenthal, and Mary Baker. The LOCKSS Peer-to-Peer Digital Preservation System. *ACM Transactions on Computer Systems*, 23(1):2–50, 2005.

[20] Bethany McLean and Peter Elkin. *Smartest Guys in the Room: The Amazing Rise and Scandalous Fall of Enron*. Portfolio Hardcover, 2003.

[21] Sami Menafee. Drivesavers cuts price for lost data recovery. `http://www.findarticles.com/cf_dls/m0NEW/n39/20324409/p1/article.jhtml`, February 1998.

[22] Nikolaos Michalakis, Dah-Ming Chiu, and David S. H. Rosenthal. Long Term Data Resilience Using Opinion Polls. In *22nd IEEE International Performance Computing and Communications Conference*, Phoenix, AZ, USA, April 2003.

[23] Gordon Mohr, Michael Stack, Igor Ranitovic, Dan Avery, and Michele Kimpton. An Introduction to Heritrix: An Open Source Archival Quality Web Crawler. In *IWAW04: Proceedings of the International Web Archiving Workshop*, September 2004.

[24] NASA. Aviation Safety Reporting System. `http://asrs.arc.nasa.gov/`, July 2008.

[25] National Security Archive. CIA STALLING STATE DEPARTMENT HISTORIES. `http://www.gwu.edu/~nsarchiv/NSAEBB/NSAEBB52/`, July 2001.

[26] Marybeth Peters. The Importance of Orphan Works Legislation. `http://www.copyright.gov/orphan/`, September 2005.

[27] T. Phelps and R. Wilensky. Robust Hyperlinks Cost Just Five Words Each. Technical Report UCB–CSD-00-109, UC Berkeley, January 2000.

[28] J. Reason. *Human Error*. Cambridge University Press, 1990.

[29] David S. H. Rosenthal. A Digital Preservation Network Appliance Based on OpenBSD. In *Proceedings of BSDcon 2003*, San Mateo, CA, USA, September 2003.

[30] David S. H. Rosenthal. Bit Preservation; A Solved Problem? In *iPRES2008*, September 2008.

[31] David S. H. Rosenthal. Format Obsolescence: Assessing the Threat annd the Defenses. *Library High Tech*, 28(2), June 2010. To appear.

[32] David S. H. Rosenthal, Thomas Lipkis, Thomas S. Robertson, and Seth Morabito. Transparent format migration of preserved web content. *D-Lib Magazine*, 11(1), January 2005.

[33] David S. H. Rosenthal and Vicky Reich. Permanent Web Publishing. In *Proceedings of the USENIX Annual Technical Conference, Freenix Track*, pages 129–140, San Diego, CA, USA, June 2000.

[34] David S. H. Rosenthal, Thomas S. Robertson, Tom Lipkis, Vicky Reich, and Seth Morabito. Requirements for Digital Preservation Systems: A Bottom-Up Approach. *D-Lib Magazine*, 11(11), November 2005.

[35] Jeff Rothenberg. Ensuring the Longevity of Digital Documents. *Scientific American*, 272(1), 1995.

[36] Michael Seadle. Archiving in the Networked World: Interoperability. *Library High Tech*, 28(2), June 2010. To appear.

[37] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to Keep Online Storage Services Honest. In *HOTOS XI, 11th Workshop on Hot Topics in Operating Systems*, May 2007.

[38] K. Skinner and M. Schultz, editors. *A Guide to Distributed Digital Preservation*. Educopia Institute, 2010.

[39] Sangchul Song and Joseph JaJa. New techniques for ensuring the long term integrity of digital archives. In *dg.o '07: Proceedings of the 8th annual international conference on Digital government research*, pages 57–65. Digital Government Society of North America, 2007.

[40] Sangchul Song and Joseph JaJa. Techniques to audit and certify the long-term integrity of digital archives. *Int. J. Digit. Libr.*, 10(2-3), August 2009. DOI: 10.1007/s00799-009-0056-2.

[41] Robert F. Sproull and Jon Eisenberg. Building an Electronic Records Archive at the National Archives and Records Administration: Recommendations for a Long-Term Strategy. `http://www.nap.edu/catalog/11332.html`, June 2005.

[42] Jonathan Stone and Craig Partridge. When the CRC and TCP checksum disagree. In *SIGCOMM*, pages 309–319, 2000.

[43] Jennifer Tom. When Mutilators Stalk the Stacks. `http://gort.ucsd.edu/preseduc/bmlmutil.htm`.

[44] Anton R Valukas, editor. *Lehman Brothers Holdings Inc. Chapter 11 Proceedings Examiners Report*. Jenner and Block, March 2010.

[45] Herbert van de Sompel, Michael L. Nelson, Robert Sanderson, Lyudmila L. Balakireva, Scott Ainsworth, and Harihar Shankar. Memento: Time Travel for the Web. `http://arxiv.org/abs/0911.1112`, November 2009.