# Preserving Peer Replicas By Rate-Limited Sampled Voting

Petros Maniatis
2150 Shattuck Av., Suite 1300
Berkeley, CA 94704
maniatis@intel-research.net
ACM #: 4617148
Advisor: Mary Baker

TJ Giuli
353 Serra St. Rm. 416
Stanford, CA 94305
giuli@cs.stanford.edu
ACM #: CR73908
Advisor: Mary Baker

Yanto Muliadi
111 N. Rengstorff Ave. Apt. 126
Mountain View, CA 94043
zyli@stanfordalumni.org
Advisor: Mary Baker

## 1. Introduction

[1]Academic publishing is migrating to the Web [20], forcing the libraries that pay for journals to transition from purchasing copies of the material to renting access to the publisher's copy [15]. Unfortunately, rental provides no guarantee of long-term access. Librarians consider it one of their responsibilities to provide future readers access to important materials. With millennia of experience on physical documents, they have techniques for doing so: acquire lots of copies of the document, distribute them around the world, and lend or copy them when necessary to provide access.

In the LOCKSS[2] program (Lots Of Copies Keep Stuff Safe), we model the physical document system and apply it to Web-published academic journals, providing tools for libraries to take custody of the material to which they subscribe, and to cooperate with other libraries to preserve it and provide access. The LOCKSS approach deploys a large number of independent, low-cost, persistent web caches that cooperate to detect and repair damage by voting in "opinion polls" on their cached documents. The initial version of the system [23] has been under test since 1999 at about 50 libraries world-wide, and is expected to be in production use at many more libraries in 2004. Unfortunately, the protocol now in use does not scale adequately, and analysis of the first design for a revised protocol [19] showed it to be insufficiently resistant to attack.

In this work [18], we present a design for and simulations of a new peer-to-peer opinion poll protocol that addresses these scaling and attack resistance issues. We plan to migrate it to the deployed system shortly. The new protocol is based on our experience with the deployed LOCKSS system and the special characteristics of such a long-term large-scale application.

## 2. Design Principles and Motivation

Distributed digital preservation, with its time horizon of many decades and lack of central control, presents both unusual requirements, such as the need to avoid long-term secrets like encryption keys, and unusual opportunities, such as the option to make some system operations inherently very time-consuming as a defense without sacrificing usability.

More specifically, digital preservation systems for content such as academic materials must, first, be very cheap to build and maintain, which precludes high-performance hardware such as RAID [21], or complicated administration. Second, they need not operate quickly. Their purpose is to prevent rather than expedite change to data. Third, they must function properly for decades, without central control and despite possible interference from attackers or catastrophic failures of storage media such as fire or theft. These features, combined with our experience building and maintaining other large-scale distributed systems, lead to the very conservative design principles we use:

*Cheap storage is unreliable.* We assume that in our timescale no cheap and easy to maintain storage is reliable [7]. Note that write-once media are at least as unreliable as disks, eliminating alternate designs dependent on storing documents or their hashes on CD-R (in our current deployment the CD-R containing the peer software is the cause of the vast majority of errors).

*No long-term secrets.* Or, to quote Diffie [10], "The secret to strong security: less reliance on secrets." Long-term secrets, such as private keys, are too vulnerable for our application. These secrets require storage that is effectively impossible to replicate, audit, repair or regenerate. Over time they are likely to leak; recovering from such leakage is extraordinarily difficult [9], [26].

*Use inertia.* The goal of a digital preservation system is to prevent change. Some change is inevitable, and the system must repair it, but there is never a need for rapid change. A system that fails abruptly, without warning its operators in time for them to take corrective action and prevent total failure [25], is not suitable for long-term preservation. Rate limiting has proved useful in other areas [28]; we can exploit similar techniques because we have no need for speed.

*Avoid third-party reputation.* Third-party reputation information is subject to a variety of problems, especially in the absence of a strong peer identity. It is vulnerable to slander and subversion of previously reliable peers. If evidence of past good behavior is accumulated, an attacker can "cash in" a history of good behavior in low-value interactions by defecting in a single high-value interaction [29].

*Reduce predictability.* Attackers predict the behavior of their

[1]This extended abstract describes collaborative work also conducted by Mary Baker (HP Labs), David Rosenthal (Stanford Libraries), and Mema Roussopoulos (Harvard University)

[2]LOCKSS is a trademark of Stanford University.

victim to choose tactics. Making peer behavior depend on random combinations of external inputs and internal state reduces the accuracy of these predictions.

*Intrusion detection is intrinsic.* Conventional intrusion detection systems are extrinsic to the application being protected. They have to operate with less than full information about it, and may themselves become a target. Systems with bimodal behavior [2] can provide intrinsic intrusion detection by surrounding good states with a "moat" of forbidden states that are almost never reachable in the absence of an attack, and that generate an alarm.

*Assume a strong adversary.* The LOCKSS system preserves e-journals that have intrinsic value and contain information that powerful interests might want changed or suppressed. Today, a credible adversary is an Internet worm whose payload attacks the system using tens of thousands of hosts. We must plan for future, more powerful attacks.

The LOCKSS design is very conservative, appropriately so for a preservation system. Our goal is to apply these principles to the design to achieve a high probability that even a powerful adversary fails to cause irrecoverable damage without detection.

## 3. LOCKSS Description

In this section we outline, describe and justify our new LOCKSS opinion poll protocol. We consider a population of peers preserving a copy of a single *archival unit* (AU), obtained from a publisher who is no longer available. We ignore the divide-and-conquer search for damage in a real, multi-file journal. Each peer uses one of a number of independent implementations of the LOCKSS protocol to limit common-mode failures. Each peer's AU is subject to the same low rate of undetected random damage caused by such events as bit flips on the magnetic storage medium or faulty, non-parity memory chips.

While a *peer* is any node that participates with benign or malicious intent in the LOCKSS protocol, we make the following distinctions between different types of peers in the rest of this paper:

- A *malign* peer is part of a conspiracy of peers attempting to subvert the system.
- A *loyal* peer is a non-malign peer, i.e., one that follows the LOCKSS protocol at all times.
- A *damaged* peer is a loyal peer with a damaged AU.
- A *healthy* peer is a loyal peer with the correct AU.

The overall goal of our design is that there be a high probability that loyal peers are in the healthy state despite failures and attacks, and a low probability that even a powerful adversary can damage a significant proportion of the loyal peers without detection.

A LOCKSS peer calls opinion polls on the contents of an AU it holds at a rate much greater than any anticipated rate of random damage. It invites into its poll a small subset of the peers it has recently encountered, hoping they will offer votes on their version of the AU. Unless an invited peer is busy, it computes a fresh digest of its own version of the AU, which it returns in a vote. If the caller of the poll receives votes that overwhelmingly agree with its own version of the AU (a *landslide win*), it is satisfied and waits until it has to call a poll again. If it receives votes that overwhelmingly disagree with its own version of the AU (a *landslide loss*), it repairs its AU by fetching the copy of a voter who disagreed, and re-evaluates the votes, hoping now to obtain a landslide win for its repaired AU. If the result of the poll justifies neither a landslide win nor a landslide loss (an *inconclusive* poll), then the caller raises an alarm to attract human attention to the situation.

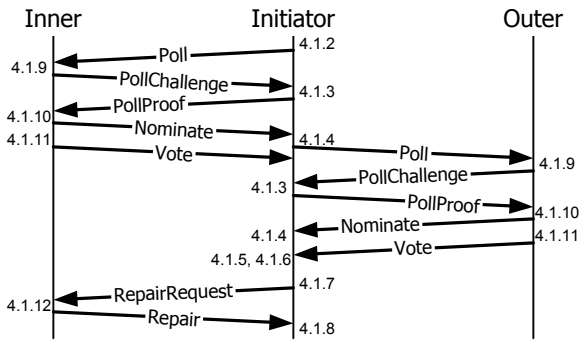


Fig. 1. The protocol messages exchanged during a poll between the poll initiator and poll participants. Time flows from top to bottom. Next to each phase in the execution of the protocol, we give the section number that provides the pertinent description.

The protocol supports two roles for participating peers. First, the *poll initiator* calls polls on its own AUs and is the sole beneficiary of the poll result. Second, the *poll participant* or *voter* is a peer who is invited into the poll by the poll initiator and who votes if it has the necessary resources. A voter need not find out the result of a poll in which it votes. Poll participants for a given poll are divided into two groups: the *inner circle* and the *outer circle*. Inner circle participants are chosen by the poll initiator from those peers it has already discovered. The initiator decides the outcome of the poll solely on inner circle votes. Outer circle participants are chosen by the poll initiator from peers nominated by inner circle voters. The initiator uses outer circle votes to perform discovery, i.e., to locate peers that it can invite into future inner circles for its polls. Newly discovered and already known peers are maintained in a list known as the *reference list*.

LOCKSS peers communicate in two types of exchanges. First, a poll initiator uses unicast datagrams to communicate with the peers it invites to arrange participation and voting in the poll. Second, a poll initiator may contact its voters to request a repair for its AU using a bulk transfer protocol. In both cases, communication is encrypted via symmetric session keys, derived using Diffie-Hellman key exchanges [11] between the poll initiator and each of its participants. After the poll, session keys are discarded. Figure 1 shows a typical message exchange between a poll initiator and its inner and outer circles.

The LOCKSS opinion poll protocol requires both poll initiators and voters to expend provable computational effort [14] in amounts related to underlying system operations (hashing of an AU), as a means of limiting Sybil attacks [13].

## 4. Attack Defenses

In this work, we primarily show how LOCKSS defends against the *stealth modification adversary*. The stealth adversary has to balance two goals: changing the consensus on the

target AU and remaining undetected. To achieve these goals, he must repeatedly find a healthy poll initiator, convince it that it has a damaged AU replica without causing it to raise any alarms, and then, if asked, conveniently offer it a repair with the bad version of the AU. This strategy relies primarily on building a "foothold" in loyal peers' reference lists by placing there as many malign peers as possible.

The stealth adversary acts in two phases. First he *lurks*, seeking to build a foothold in loyal peers' reference lists by exclusively nominating malign peers for a loyal peer's outer circle. In all other respects, however, he behaves as a loyal peer, voting and repairing with the correct version of the AU. Then he *attacks*, causing his malign peers to vote and repair using either the correct or the bad version of the AU, according to whether an opportunity presents itself. Specifically, malign peers vote with the correct copy unless the poll is *vulnerable*, i.e., its overwhelming majority of inner circle voters is malign. In vulnerable polls, malign peers vote with the bad copy, because by doing so they can convince the loyal poller that its AU needs a repair, without risking an inconclusive poll alarm.

Conservatively, we chose to model a stealth adversary with no resource constrains and an infinite number of identities (IP addresses).

To defend the LOCKSS system from attack, we make it costly and time-consuming for an adversary to sway an opinion poll in his favor or to waste loyal peers' resources. This means the protocol must

- prevent the adversary from gaining a foothold in a poll initiator's reference list and thus skewing the sampled voting process in his favor,
- make it expensive for the adversary to waste another peer's resources, and
- make it likely that the adversary's attack will be detected before it progresses far enough to cause irrecoverable damage.

We use provable recent effort, rate limiting, reference list churning, and obfuscation of protocol state to make it expensive and slow for an adversary to gain a significant foothold in a peer's reference list or waste other peers' resources. We raise alarms when we detect signs of attack.

### *4.1 Effort Sizing*

One application of our principle of inertia (Section 2) is that large changes to the system require large efforts. In a protocol where some valid messages cost nothing to produce, but cause the expenditure of great effort — e.g., a cheap request causing its recipient to hash a large amount of data — this principle is unfulfilled. To satisfy our inertia requirement in LOCKSS, we adjust the amount of effort involved in message exchanges for voting, discovery, and poll initiation, by embedding extra, otherwise unnecessary effort. Ultimately, the effort expended by a poller to invite a single voter into a poll is approximately equal to the effort expended by the voter in participating. Conversely, the effort expended by the poller is matched by the voter in constructing a vote.

### *4.2 Rate Limiting*

Another application of our principle of inertia (Section 2) is that the system should not change rapidly no matter how much effort is applied to it. We use rate-limiting techniques to implement this.

Loyal peers call polls autonomously and infrequently, but often enough to prevent random undetected damage from affecting readers significantly. This sets the effort required of the voters, and means that an adversary can damage a loyal peer only when that peer calls a poll. *The rate at which an attack can make progress is limited by the smaller of the adversary's efforts and the efforts of his victims.* The adversary cannot affect this limit on the rate at which he can damage loyal peers.

### *4.3 Timeliness Of Effort*

Our principle of avoiding third-party reputation leads us to use several techniques to ensure that only proofs of recent effort can affect the system. They prevent an adversary from exploiting evidence of good behavior accumulated over time.

Peers must supply a vote, and thus a proof of effort, to be admitted to the reference list. If several polls take place without the admitted peer taking part, perhaps because it died, the initiator removes it. If the admitted peer is invited and does take part in a poll, it must supply a vote and thus further proof of effort. After the poll the initiator "forgets" the peer if its vote affected the outcome of the poll.

By these means we ensure that any peer identity, whether loyal or malign, must continually be sustained by at least a minimum rate of expenditure of effort if it is not to disappear from the system. Although the lack of long-term secrets makes it cheap for an adversary to create an identity, sustaining that identity for long enough to affect the system is expensive. Unless the adversary's resources are truly unlimited, there are better uses for them than maintaining identities that do not contribute to his goals.

### *4.4 Reference List Churning*

The key to protecting a LOCKSS peer from AU subversion is to protect the sampling process from external bias. Bias is accomplished by populating a loyal peer's reference list with malign peers. LOCKSS combats bias using *churning*. At the end of each poll, peers from a short list of "friend" peers—peers assumed, sometimes falsely, to be trustworthy via out-of-band means—are added to the reference list along with new peers discovered in the outer circle. By churning these peers into the reference list, the initiator hampers attempts to fill the reference list with individual malign conspirators or with multiple identities of the same malign conspirator.

### *4.5 Obfuscation of Protocol State*

Our design principles (Section 2) include assuming a powerful adversary, capable of observing traffic at many points in the network. We obfuscate protocol state in two ways to deny him information about a poll other than that obtained from the malign participants.

First, we encrypt all but the first protocol message exchanged by a poll initiator and each potential voter, using a symmetric

key arranged afresh between the poller and each of its voters individually for each poll. Second, we make all loyal peers invited into a poll, even those who decline to vote, go through the motions of the protocol, behind the cover of encryption. This prevents an adversary from using traffic analysis to infer state such as the number of loyal peers who actually vote in a specific poll. Note that in our modeled adversary and simulations [18] we conservatively assume that the adversary *can* infer such information.

### *4.6 Alarms*

In accordance with our design principle that intrusion detection be inherent in the system, the protocol raises an alarm when a peer determines that a poll is inconclusive, suspects local spoofing, or has been unable to complete a poll for a long time. Raising an alarm is thus expensive; a significant rate of false alarms would render the system useless.

The expectation is that alarms result in enough loss to the adversary, for example by causing operators to remove damage, malign peers and compromised nodes, that a rational adversary will be highly motivated to avoid them, unless raising alarms is his primary goal.

## 5. Results

Our simulations [18] show that:

- When operating in the absence of attack, the system exhibits a low rate of false alarms, even in cases where peer storage fails unobtrusively at abnormally high rates.
- Using a combination of rate limiting and effort balancing, LOCKSS successfully resists attacks that can last decades. When under attack by a very powerful adversary, LOCKSS is able to resist irrecoverable corruption of the attacked document, even when up to one third of the loyal peer population is instantly compromised at the start of the lurking phase.
- If more than one third of the loyal peers are subverted, the probability of irrecoverable damage increases only gradually with the size of the subversion. Thus, as LOCKSS begins to fail, it degrades gracefully.

## 6. Related Work

In common with the Byzantine-fault-tolerance (BFT) literature (e.g., [3], [5], [17], [22]), our voting protocol derives an apparent prevailing opinion among a set of peers, some of whom are malicious. There are many differences; our population size is too large for BFT's global communication, we degrade gradually rather than mask the effects of failure or attack, and because we cannot assume an absolute upper bound on the malicious peers' resources we have to consider the possibility of being overwhelmed. We use sampling to avoid global knowledge or communication, rate-limiters to prevent our adversary's unlimited resources from overwhelming the system quickly, and integrated intrusion detection to preempt unrecoverable failure.

Our work has similarities with the anti-entropy protocol forming part of Bimodal Multicast [2], a reliable multicast protocol in which peers send digests of their message histories to randomly chosen other peers. Peers receiving these messages can detect omissions and request repairs from the peer that sent the digest. The system's name comes from its bimodal distribution of delivery probability, which is similar to our distribution of poll results absent an attack. As in our case, it exploits the properties of random graphs. As the authors acknowledge, the lack of voting among the peers leaves the anti-entropy protocol vulnerable to malign peers.

Our work also shares a goal with some of the first peer-to-peer systems including Freenet [6], FreeHaven [12], and the Eternity Service [1], namely to make it hard for a powerful adversary to damage or destroy a document in the system. The other key goal of these systems is to provide anonymity for both publishers and readers of content, which we do not share. It would make our system both illegal and unworkable, since we often preserve content that must be paid for.

Several studies have proposed a persistent, peer-to-peer storage service including Intermemory [4], CFS [8], Oceanstore [16], PAST [24], and Tangler [27]. Some (e.g., Oceanstore) implement access control by encrypting the data and thus do not solve our preservation problem, merely reducing it from preserving and controlling access to the content, to preserving and controlling access to the encryption key. Others (e.g., PAST) implement access control based on long-term secrets and smartcards or a key management infrastructure. Neither is appropriate for our application. Some (e.g., Intermemory) use cryptographic sharing to proliferate $n$ partial replicas, from any $m < n$ of which the file can be reconstituted. Others (e.g., PAST) replicate the entire file, as we do, but do not allow control over where the replicas are located. The goal of the LOCKSS system is to allow librarians to take custody of the content to which they subscribe. This requires that each library keep its own copy of a document it has purchased, not share the load of preserving a small number of copies.

## References


[1] Ross J. Anderson. The Eternity Service. In *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT 1996)*, Prague, Czech Republic, 1996.

[2] Kenneth Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[3] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI 1999)*, pages 173–186, New Orleans, LA, USA, February 1999. USENIX Association.

[4] Yuan Chen, Jan Edler, Andrew Goldberg, Allan Gottlieb, Sumeet Sobti, and Peter Yianilos. A Prototype Implementation of Archival Intermemory. In *International Conference on Digital Libraries*, pages 28–37, Berkeley, CA, USA, 1999.

[5] Benny Chor and Cynthia Dwork. Randomization in Byzantine Agreement. *Advances in Computing Research*, 5:443–497, 1989.

[6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Hannes Federrath, editor, *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66, Berkeley, CA, USA, July 2000. Springer.

[7] ConservationOnline. Electronic Storage Media. `http://palimpsest.stanford.edu/bytopic/electronic-records/electronic-storage-media/`, 2003.

[8] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area Cooperative Storage with CFS. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pages 202–215, Chateau Lake Louise, Banff, AB, Canada, October 2001.

[9] Don Davis. Compliance Defects in Public-Key Cryptography. In *Proceedings of the 6th USENIX Security Symposium*, pages 171–178, San Jose, CA, USA, July 1996.

[10] Whitfield Diffie. Perspective: Decrypting The Secret to Strong Security. `http://news.com.com/2010-1071-980462.html`, January 2003.

[11] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[12] Roger Dingledine, Michael J. Freedman, and David Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In Hannes Federrath, editor, *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 67–95, Berkeley, CA, USA, July 2000. Springer.

[13] John Douceur. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, pages 251–260, Boston, MA, USA, March 2002.

[14] Cynthia Dwork and Moni Naor. Pricing via Processing. In *Advances on Cryptology (CRYPTO 1992)*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1993.

[15] Michael Keller, Victoria Reich, and Andrew Herkovic. What is a Library Anymore, Anyway? *First Monday*, 8(5), May 2003. `http://www.firstmonday.org/issues/issue8_5/keller/index.html`.

[16] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, pages 190–201, Cambridge, MA, USA, November 2000.

[17] Dahlia Malkhi and Michael Reiter. Byzantine Quorum Systems. *The Journal of Distributed Computing*, 11(4):203–213, October 1998.

[18] Petros Maniatis, Mema Roussopoulos, TJ Giuli, David S. H. Rosenthal, Mary Baker, and Yanto Muliadi. Preserving Peer Replicas By Rate-Limited Sampled Voting. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 44–59, Bolton Landing, NY, USA, October 2003.

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

[20] Dru Mogge. Seven Years of Tracking Electronic Publishing: The ARL Directory of Electronic Journals, Newsletters and Academic Discussion Lists. *Library Hi Tech*, 17(1):17–25, 1999.

[21] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 109–116, Chicago, IL, USA, June 1988.

[22] Michael Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, Fairfax, VA, USA, November 1994.

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

[24] Antony Rowstron and Peter Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pages 188–201, Chateau Lake Louise, Banff, AB, Canada, October 2001.

[25] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, San Francisco, CA, USA, August 2002.

[26] Wietse Venema. Murphy's Law and Computer Security. In *Proceedings of the 6th USENIX Security Symposium*, San Jose, CA, USA, July 1996.

[27] Marc Waldman and David Mazières. Tangler: A Censorship-Resistant Publishing System Based On Document Entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001)*, pages 126–135, Philadelphia, PA, USA, November 2001.

[28] Matthew Williamson. Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. In *Proceedings of the 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, USA, December 2002.

[29] Nick Wingfield. EBay's Figurine Scandal: Auction Site Merchant Disappears With The Goods. Wall Street Journal, Feb 22nd, 2002.