Establishing a Web of Trust in Sampled Voting Systems

Rachel Greenstadt, greenie@eecs.harvard.edu Geoffrey Goodell, goodell@eecs.harvard.edu Ian Becker, ibecker@fas.harvard.edu Mema Roussopoulos, mema@eecs.harvard.edu

TR-09-04



Computer Science Group Harvard University Cambridge, Massachusetts

Establishing a Web of Trust in Sampled Voting Systems

Abstract

We propose a fully distributed PKI for authenticated peer discovery in P2P systems, and we specify a variety of localized recovery procedures that can be used to complement intrusion detection in such systems. Our work is motivated by LOCKSS, a P2P system used for the preservation of online published material. LOCKSS relies on network-layer addresses for authentication, making it vulnerable to powerful adversaries capable of controlling ISPs and subverting Internet routers. Moreover, LOCKSS relies upon the assumption that the adversary can be globally eradicated from the system in response to an intrustion detection alarm, and this may not be feasible. We extend the LOCKSS protocol to incorporate the distributed PKI, and we demonstrate the efficacy of a variety of local alarm procedures.

1. INTRODUCTION

Document publishing has entered the Digital Age, and many publishers are making their publications available electronically on their web sites. Instead of allowing libraries to purchase the material, publishers are now renting access to their web sites to the libraries. Librarians are concerned that future access to important material depends on publishers who might misbehave, go out of business, or simply not desire or be able to afford to preserve access.

The LOCKSS system is a peer-to-peer digital preservation system that aims to approximate the traditional purchase model in which libraries pay for and own content distributed by the publishers. LOCKSS strives to achieve consensus within a community of peers. Each peer maintains its own low-cost, persistent cache of documents and periodically conducts "opinion polls" on each of its documents using a sample of other peers in the system. These polls allow the peer to detect and repair damage to its documents.

The LOCKSS design is documented in a series of publications [22], [25], [31], [32]. The design assumes a powerful adversary that has unlimited computational power, unlimited identities, and the ability to subvert and control some proportion of the peers participating in the system. We believe that these assumptions are realistic, as governments and other powerful organizations often try to suppress or destroy information distasteful to them. Incentive to control the dissemination of information is strong within our society. For example, the Church of Scientology fought hard to suppress its secret documents [21], and an article by George Bush, Sr. discussing his reasoning for not removing Saddam Hussein from power in Iraq in 1991 disappeared from *Time*'s online archives [5].

The success of the LOCKSS design depends upon two important assumptions. The first assumption is that routers are incorruptible; LOCKSS authenticates peers with their IP addresses. While the LOCKSS design has demonstrated the ability to withstand adversaries with unlimited computational power, identities, and subversion of peers, LOCKSS remains vulnerable to adversaries who can control ISPs or subvert Internet routers.

The second assumption in the design of LOCKSS is that intrusion detection alarms lead to a complete eradication of the adversary from the system; when an alarm occurs, humans get involved. The operators of all of the LOCKSS peers work together to sort out which peers are malign and which copy of the document in question is authentic. This intrinsic intrustion detection is an important feature of LOCKSS, but published work to date examining the LOCKSS protocol does not consider what happens *after* an alarm is raised. Humans are not infallible, and this vision of complete global defeat of the adversary is not realistic.

This paper provides the following contributions:

• To provide strong identities, we augment LOCKSS with a fully distributed Public Key Infrastructure (PKI) that is robust against adversaries who can *control network infrastructure* in addition to having an abundance of computational power, launching Sybil attacks [13], and using exploits to subvert a substantial minority of the participating peers. ¹.

¹We assume an adversary has an abundance of computational power, but not enough to arbitrarily invert cryptographic functions

We note that the authors of LOCKSS categorically exclude the possibility of using private keys, arguing that this would insinuate the use of long-term secrets and thus introduce substantial vulnerability. We demonstrate that it is possible to perform regular key rotations in such a manner as to avoid much of the risk associated with the storage and maintenance of long-term secrets [10]. Furthermore, we argue that the benefits associated with strong authentication outweigh the costs associated with the use of public keys.

- Our approach provides a general mechanism for authenticated discovery in peer-to-peer systems. Previous approaches to fully distributed PKI have fallen back on centralized mechanisms, typically because clients in the system seek to verify the authenticity of particular peers [9], [36], [29]. In contrast, clients in a P2P system are only interested in obtaining a resource from the system as a whole rather than from a particular peer. Consequently, it is sufficient to find some set of trusted peers that are capable of providing the resource. We exploit this characteristic of P2P networks in order to build a distributed PKI that can succeed where other attempts have failed.
- We explore a variety of localized alarm procedures for performing recovery following intrusion detection. These include procedures that leverage the web of trust that the peers build during peer discovery. We find that localized alarm procedures in which peers heal compromised nodes near the root of their web of trust are effective in protecting against adversaries attempting to corrupt documents.

2. BACKGROUND: LOCKSS OVERVIEW

In this section, we give an overview of how the LOCKSS protocol works, and we discuss the limitations of using IP addresses as authenticators. Details and justification of the design decisions behind the protocol can be found in prior publications [22], [25], [31], [32].

In LOCKSS, each participating peer runs a persistent web cache. The peer crawls the websites of publishers who have agreed to have their content preserved and downloads copies of published material (e.g. academic journals) to which the library in question has subscribed. The cached information is then used to satisfy requests from the library's users when the publisher's website is unavailable. Library budgets set aside for preservation are generally quite small [3], so the LOCKSS design calls for using low-cost hardware and software that requires minimal operation and administration. Any storage media is unreliable given a sufficiently long time horizon [15], [28], so the system must be robust against natural bit degradation as well as attacks from powerful adversaries who may try to modify or censor content that they consider offensive.

Each peer maintains two lists: a *friends list* and a *reference list*. The reference list is a list of peers that the peer in question has recently discovered in the process of participating in the LOCKSS system. The friends list is a list of peers (*friends*) that the peer knows externally and with whom it has an out-of-band relationship before entering the system. When a peer joins the system, his reference lists starts out containing the peers on his friends list.

Content within the LOCKSS system is organized into *Archival Units* (AUs), e.g., a year's run of an academic journal. Periodically, at a rate faster than the rate of natural bit degradation, each peer conducts an *opinion poll* on an AU. The peer takes a random sample of peers from its reference list and invites the chosen peers to vote on the AU by sending hashes of their individual copies of the AU to the peer initiating the poll. Hereafter, we refer to the peer who initiates the poll as the *poller* and the invited peers as the *voters*.

The poller compares the votes it receives with its local copy. If an overwhelming majority of the hashes received agrees with the poller's hash, then the poller concludes that its copy is good, and it resets a refresh timer to determine the next time to check this AU. If an overwhelming majority of hashes disagree, then the peer fetches a *repair* by obtaining a copy of the AU from one of the disagreeing peers and re-evaluating the votes it received. If there is neither landslide agreement nor landslide disagreement, then the poll is deemed *inconclusive* and the poller raises an alarm. Because natural storage degradation is a relatively infrequent occurrance, it is unlikely that many peers will simultaneously experiencing degradation. If an inconclusive poll results, it is an indication that an attack might be in progress. LOCKSS uses alarms as

a way of performing intrusion detection, so that when an attack is suspected, humans are called upon to examine, heal, and restart the system.

At the conclusion of a poll, the poller updates its reference list as follows. First, it removes those peers that voted in the poll so that the next poll is based on a different sample of peers. Second, the poller replenishes its reference list by adding *nominated peers* and peers from the friends list. Nominated peers, or *nominees*, are peers that are introduced by the voters when the voters are first invited to participate in the poll. Nominees are used solely for discovery purposes so that the poller can replenish its reference list. Nominees vote on the AU, but their votes are not considered in determining the outcome of the poll. Instead, their votes are used to implement admission control into the reference list. Nominees whose votes agree with the poll outcome are added to the reference list. The contents of the reference list determines the outcome of future polls. Adding more friends to the reference list than nominees makes the poller vulnerable to targeted attacks aimed at its friends. Adding more nominees than friends to the reference list increases the potential for Sybil attacks.

The LOCKSS protocol is designed so that for each request a peer A makes of another peer B, it costs A at least as much effort to make the request as it does for B to satisfy the request. For example, simply sending an invitation to a peer to vote in a poll is much cheaper than voting, which requires hashing an AU. LOCKSS, therefore, inflates the cost of calling a poll by requiring the poller to provide a proof of some recent provable computational effort to the voter as a form of payment for its vote. This prevents an adversary interested in denial-of-service from calling many spurious polls to waste resources of participating peers and prevent them from calling their own polls. For details and justification of this and other design decisions, we refer the reader to the LOCKSS literature [23], [22].

The LOCKSS designers anticipate and have developed defenses against adversaries with a variety of goals including denial-of-service adversaries that attempt to keep peers overwhelmed with spurious poll invitations so that they are unable to conduct their own polls, adversaries that try to discredit the system by raising alarms unnecessarily, and adversaries attempting to obtain content to which they do not have authorized access. The most important adversary highlighted is the *stealth modification* adversary, whose goal is to use the LOCKSS protocol to modify content in the system without raising alarms. We focus on the stealth modification adversary in this paper. The strategy of this adversary consists of two phases. In the first phase, the adversary lurks, nominating identities that are under his control to attempt to gain a foothold in the reference lists of the peers holding AUs it wishes to modify. In the second phase, the adversary attacks, attempting to damage AUs by winning polls and convincing pollers to repair using the adversary's version of the AU. It is not enough for the adversary to convince a poller once to repair using the adversary's version of the AU. To be successful, the adversary must continue to maintain a strong foothold in the reference list of the poller and to win all subsequent polls initiated by the poller.

2.1 Problems with basing Identity on IP Addresses

Using network addresses as authenticators creates the fundamental problem that the security properties of LOCKSS depend upon the security properties and trust semantics of the underlying network-layer infrastructure. This is not only an aesthetic problem that can complicate the adaptation of a system to overlays and non-IPv4 networks, but it has considerable security implications as well.

We outline four difficulties in particular. First and foremost, IP addressing is designed to facilitate the flow of packets through the network; arguably, it was never intended to provide security. Whenever we consider passing unauthenticated messages from one network to another through intermediaries, we must concern ourselves with the risks of malicious behavior on the part of the intermediaries as well as the networks at the ends of the connection. So, we have a man-in-the-middle attack.

LOCKSS assumes that man-in-the-middle attacks are launched against LOCKSS peers only by hosts that live in the same local subnet as the peers. That is, a host A that resides on the same subnet as a LOCKSS peer B, can spoof messages with B as the source destination and any IP address as the target destination, as well as messages with any IP address as the source destination and B as the target destination. It turns out, we don't really need to be on the network of the address

we are spoofing, even if all networks in the system are using ingress filtering at its edge routers. An adversary that controls a router can spoof any packet that passes through it. Despite asymmetric routing, often the forward and reverse paths will traverse some or even all of the same providers, and thus we observe how controlling a single ISP or a few machines owned by a single ISP may be sufficient to provide effective spoofing. We claim that this is particularly likely, especially given that some Internet Service Providers and may have ulterior motives for censoring documents, and other providers are inexorably tied to governments that may have an interest in squelching certain forms of expression [18].

Additionally, both network address translation (NAT) and dynamic addressing are widespread in practice, and they both have important effects on network topology and the validity of underlying presumptions about which addresses belong to which real-world entities. In essence, we do not wish to promote a system that encourages a library to enter a contract with an ISP to obtain a static IP address; this may even be prohibitively expensive in certain parts of the world.

3. Related Work

3.1 Other Preservation Systems

The LOCKSS literature [22], [23] contains extensive discussion of other preservation systems, such as RAID [27] and peer-to-peer storage systems, and how they compare to the LOCKSS protocol.

Some examples of distributed storage systems based upon distributed hast table (DHT) technologies are OceanStore [19], PAST [33], and CFS [8]. Some of these systems achieve preservation by distributing r partial replicas of content; the content can then be reconstituted from any q < r replicas. In LOCKSS, for administrative and legal reasons, each library peer must store and have control over its own whole replica.

Other storage systems provide features that accommodate specific architectural goals. For example, Free Haven [12], Freenet [6], the Eternity Service [2] and Tangler [35] share with LOCKSS the aim to resist censorship. However, some of these systems (Free Haven, Freenet, and the Eternity Service) also aim to preserve the anonymity of publishers and subscribers in the system. In LOCKSS, anonymity is not an option because publishers insist that libraries pay for content before making it available to their local patrons.

Introducing the use of private keys offers two approaches to preservation. One approach builds into the system a means by which we can ensure that users receive the same document that they had at some earlier time. This could be done, for instance, with signed hashes of documents that are replicated throughout a storage system that are periodically checked for agreement against the original document. A system such as Samsara [7] could be used to implement this approach. LOCKSS uses a second approach, which creates a means by which individual users of the system check with other users to ensure that their copies match. The former aims to achieve preservation of individual copies in isolation, whereas the latter aims to promote consensus among members of a community. One can argue that one method is fundamentally superior to the other with respect to achieving long-term preservation of a set of documents within a community, but such arguments are beyond the scope of this paper. We believe that both schemes have their merits, but we focus primarily on the latter approach, considering ways to enhance the security of sampled voting systems such as LOCKSS.

3.2 Public Key Infrastructure

Public Key Infrastructures are essential to distributed authentication systems and the trust management infrastructures necessary to support such systems. Our proposed modification to the LOCKSS system employs a web of trust [4] based upon X.509 certificates [34], [17].

One major issue in any PKI system is *certificate revocation*—how to announce to the world that a certificate is invalid before its nominal time of expiration. The standard way to accomplish this is to have a *Certification Authority* (CA) periodically issue a Certificate Revocation List (CRL). The entities in the system check for updates to this list as they choose to do so. A number of types of CRLs have been proposed:

- Complete or Base The CRL lists all certificates that have been revoked, but that have not yet expired. This has the drawback of potentially growing extremely large if there is a large number of peers or rapid key turnover.
- Delta This CRL contains a list of all the certificates revoked since the last Base CRL. This can be updated more frequently than the Base CRL, and peers can download the Delta CRL instead of the whole Base CRL every time. This uses the "Delta CRL Extension" to X.509.
- Redirect or Referral This CRL is distributed across a set of servers that each maintains a piece of the full CRL, allowing partitioning of the CRL so that load can be distributed across many servers. This capitalizes on the "CRL Issuing Distribution Point" extension in X.509.
- Windowed Key Revocation This attempts to limit the size of the CRL by tying the lifetime of an entry in the CRL to the time that a certificate would be cached in the system, instead of tying it to the certificate's expiration. [24]

A thorough discussion of the different types of CRLs can be found in [14].

In traditional distributed systems, entities need to know where the latest version of the CRL can be found. This is typically accomplished by having the central CA maintain the list, and either host the list or provide the location of the list to the entities in the system. In a peer-to-peer system, placing this responsibility on one or a small set of peers is undesirable. The peers chosen to host these services become trust and performance bottlenecks as well as prime targets for attackers.

One alternative to CRLs is Rivest's online Suicide Bureaus [30]. In this system, one or more agents declares itself to be a "suicide bureau." If there is more than one, they form an association and certify the members of the association. These agents are assumed to share a high-speed, reliable network. When an entity in the system creates a key pair, it registers the key with one of the suicide agents. In order to revoke the key, the entity signs a "suicide note" and transmits it to one of the suicide bureaus. Whenever any of the suicide bureaus gets such a note, the bureau broadcasts the note to the other bureaus. Whenever the entity needs to prove to another entity that its key has not been revoked, the entity can obtain a current "certificate of health" for the key from the suicide bureau, certifying that the bureau has not received any suicide notes for that key. While this is an interesting alternative to CRLs, it requires that the suicide bureaus are continuously online, have sufficient awareness of each other to form the association, and share a high-speed network. These conditions are likely not to hold in a distributed, peer-to-peer environment. Thus, the suicide bureaus will not be able to stay synchronized with the suicide notes, and they will not be able to produce credible certificates of health.

Another alternative to CRLs is online certificate checking, such as the Online Certificate Status Protocol (OCSP) [26]. Under this system, when an entity wants to determine the validity of a certificate, it makes a status request to the OCSP server for that certificate. Generally, this is the CA that issued the certificate. The main advantage of this approach is that the entity in question can determine how often to check the status of a certificate, rather than having to get updates on the schedule of a CA maintaining a CRL. Thus, each entity can set its own acceptable risk threshold with regard to staleness of certificate validity information.

4. REVISED PROTOCOL

In this section, we augment LOCKSS with a public key infrastructure to provide support for a more effective security model. We leverage the nomination mechanism used by LOCKSS for peer discovery to build our PKI. We employ certificates with capabilities specified by X.509. We assert that given some peer P and its list of friends, then for each friend F, peer P signs an X.509 certificate indicating that F is his friend. Therefore, in X.509 parlance, each peer P acts as a Certification Authority (CA), certifying the public keys of each peer in his friends list. Thus, each certificate in the LOCKSS system certifies an out-of-band "friend" relationship between peers.

4.1 Terms

Formally, we say that a peer P considers an X.509 certificate C to be a *valid certificate* if C satisifies the following criteria:



Fig. 1. A trust web. The reference list for A is a subset of the set of peers reachable from A in the graph. B shows the result of revoking the certificate issued by peer A to B. The grayed out part of the graph represents peers no longer trusted by A.

- C has been activated and is not expired. (The current time is within the bounds specified by X.509 parameters specified by C.)
- Peer P has not received any notification that C has been revoked.
- Peer P has not received any notification that the public key associated with C is not valid.

We say that each peer in the network manages a *trust web*, which is a set of peers and certificates. Each certificate can be construed as a directed edge from the certificate issuer to its target. Specifically, if a and b are peers, and c is a peer with trust web W_c , then we say that W_c contains a directed edge from a to b if c is in possession of a valid certificate signed by a that contains the public key belonging to b. Figure 1a shows a web of trust that a peer A has built while participating in the LOCKSS system. Peers B, C, and D are the peers in A's friends list. That is, A has exchanged public keys with these peers and has a certificate pertaining to each.

Further, consider the transitive relationship provided by certificates. We define a *certification path* as an ordered list of certificates $\{R_1, ..., R_n\}$ such that for all values of i such that $1 \le i \le n-1$, there exists a directed edge from R_i to R_{i+1} . We say that peer R recognizes peer S as *authentic* if there exists a certification path from R to S. In Figure 1a, A trusts node G because there exists a certification path from A to G.

4.2 Changes to the polling mechanism

The LOCKSS protocol specifies that at the beginning of a poll, peers in the reference list of the poller may nominate other peers for the poller's consideration. Our model requires that a peer R nominating another peer S must present an X.509 certificate containing R as the issuer and S as the target of a chain of certificates leading from R to S. The certificate(s) may include X.509 policies such as path length restrictions or restrictions on the scope of entities that may be signed by the target(s). A peer S is thus added to the reference list of a poller only if a certification path exists between the poller and S. A peer S can remain in the reference list of the poller only so long as at least one path exists from the poller to S. These certification paths provide an "audit trail" with which the poller can verify the chain of trust relationships leading to any given peer in its reference list.

In addition, a peer may impose requirements on the peers it invites into polls based upon the structure of its trust web. For example, a peer R may require that there must exist some number of distinct paths to a peer S in its reference list before inviting S to participate in a poll. Additionally, a peer calling a poll may use the structure of the trust web to enforce balance in the set of peers invited. We do not specify nor simulate any such policies in this paper. We make two small but powerful changes to the LOCKSS protocol to support strong identities. First, we require conversations between the poller and the voters to begin with an authenticated Diffie-Hellman exchange [11]. This guarantees the identity of the two parties, and it also provides perfect forward secrecy for LOCKSS communication. Second, a voter R who nominates another peer S will provide the poller with the network address of S and a certification chain between R and S. If for whatever reason the poller deems the chain received to be invalid (e.g. as the result of policy), then the poller disregards the nomination and does not add S to its reference list or its trust web.

4.3 Key rotation

As with any public key infrastructure, keys in the LOCKSS PKI may be lost, compromised, or rotated to prevent dependence on long-term secrets; we cannot assume that keys will persist forever. In fact, one of the principal design goals of LOCKSS was to avoid dependence on long-term secrets [10]. For this reason, we expect that peers will periodically rotate their keys. Our design stipulates that the certificates used for LOCKSS must expire within a reasonable amount of time as dictated by the specified frequency of key rotation.

X.509 allows certification authorities to use **notBefore** and **notAfter** values to define a time interval during which a given certificate has validity. A peer can rotate a key by issuing a certificate for the new key signed by its existing key. The certificate must include a **notBefore** date indicating that the validity period of the new certificate begins before the validity period of the old certificate ends. Individual peers are responsible for attaining new versions of keys prior to their expiration.

4.4 Revocation of keys and certificates

Our mechanism for *revocation* provides a means by which peers can issue messages indicating changes in their trust relationships. Our system supports two kinds of revocation:

- Revocation of Certificates. Trust relationships between pairs of peers are mutable; in particular, a peer may occasionally have an interest in cancelling a certificate that it had issued earlier. For example, this may occur because a peer decides that another peer is untrustworthy or because a peer wants to retract an erroneously generated certificate (see Figure 1b).
- 2) Revocation of Keys. Occasionally, a peer may find that it has an interest in immediately cancelling its key pair. The primary reason for this would be that the peer's private key has been leaked or otherwise compromised, though other circumstances could force a peer to surrender its key as well. For simplicity, each peer maintains a self-signed certificate to itself and the distinction between the two categories vanishes. In our system, revocation of keys is implemented as revocation of self-signed certificates.

We wish to avoid having special peers that handle key revocation information, in keeping with our goal of complete decentralization of the PKI. We use an online checking procedure to accomplish this.

When a peer P is choosing peers to invite to a poll, the peer finds in its trust web a chain $(P, P_1, ..., P_n)$ leading to that peer. P then contacts each peer P_i along the chain and asks if the certificate linking that P_i to P_{i+1} is still valid. If it is, then P continues following the chain. If not, then P asks P_i if there is a newer certificate. If there is a newer certificate, then the P updates its trust web and continues along the chain. If not, then the peer removes the link from the trust web and picks a different peer to invite. This design can be implemented by having an Online Certificate Status Protocol (OCSP) server at each peer. This design has the advantage of allowing the peer to determine for itself when to check the validity of the certificates in its trust web.

Performing online checking to build a list of voters could take a long time if the trust chains that must be verified are long, but this is dwarfed by the cost of the proofs of effort pollers must produce to call polls. If intermediate peers in the trust chain are down or refuse to answer the OCSP query, it may be impossible to verify a chain to a particular peer in order

to invite that peer to the poll. In this case, the peer can make decisions about whether to use stale information or exclude the peer in question from the poll.

An online checking procedure ensures that a peer can determine its own policy for acceptable recency of revocation information by using a pull-based model for the revocation information. It is possible, for instance, for peer A to decide that it absolutely must have up-to-the-minute validity information, and so it must check every traversed link in the trust web for every poll A calls. Peer B, however, may decide that, as long as it has verified a link within the past week, it need not do so again when it calls a poll. Alternately, peer B could employ an optimization strategy of periodically checking some or all of the certificates in its trust web in order to ensure that fewer links would need to be validated during a poll.

We considered push-based models of propagating revocation information, but in those models both A and B are at the mercy of the timings of the other peers in the system.

5. Alarms

In LOCKSS, the *inconclusive poll alarm* is raised by a poller whose version of an AU fails to win a poll with a supermajority. This leads us to the question of what happens when an alarm is called. The original LOCKSS design was focused on intrustion detection; making sure alarms were called in time before the adversary has caused irrevocable damage. What happened afterwards was largely out of the scope of their research. The main assumption of the designers is that the recovery procedure after an alarm is raised is global: all the human operators of the peers are assembled, together they determine which copy of the AU is the correct one, and they identify and fix any peers that may have been compromised by the adversary.

We believe that it is unrealistic to assume that the operators are able to fix the system globally. First, in a large LOCKSS system, it is highly unlikely that a significant subset of the operators, much less all of them, can be assembled to deal with an alarm. The only peers a peer can reasonably be expected to contact are the ones in its "friends list," since by definition, those are the peers with whose operators the peer's operator has a real-life relationship. Thus, we would like to make alarms local to the peer calling the alarm, directly involving only the peer's friends, rather than global to the system. Second, depending upon the tactics of the adversary and the skill of the operators of the peers, it may be difficult or impossible to determine whether a peer has been compromised. Moreover, depending on the skill of the operators, it may not be possible to ensure that the compromised peers get fixed.

We define several different kinds of localized alarm procedures: *MUTE*, *RESET*, *HEAL-RESET*, and *TRUSTWEB-HEAL*. These alarms are illustrated in Figure 3.

The *MUTE* alarm (Figure 3a) is our baseline mechanism. When a peer calls an alarm, the peer ignores it and continues. This allows us to see how much damage the adversary can cause over time if the peers do not take any countermeasures.

The *RESET* alarm is a local procedure where the peer calling the alarm has no ability to affect the other peers in the system. The peer can only change its own local state, including its trust web and its reference list. In this alarm (Figure 3b), when a peer calls an alarm, the peer discards the contents of its reference list, and resets its reference list to the contents of its friends list, just as when the peer first joined the system. It then resumes the LOCKSS protocol.

HEAL-RESET and *TRUSTWEB-HEAL* are alarms in which the peer that called the alarm tries to affect (*heal*) other peers in the system. In particular, the peer calling the alarm contacts its friends' operators and asks them to examine their system and rectify any subversion. Additionally, the peer can ask its friends' operators to make the same request of their friends' operators. To understand which peers could realistically expect to be healed, we need to classify the peers in the system. This classification is summarized in Figure 2.

To begin, we define two general types of peers, *loyal* and *malign*. Loyal peers follow the LOCKSS protocol and honestly attempt to preserve their AUs. Loyal peers can be either *healthy*, meaning they have a correct copy of the AU, or *damaged*, meaning that they have a corrupted (adversary's) copy. Malign peers attempt to subvert the LOCKSS system, and do not necessarily follow the protocol. Malign peers fall into several categories. *Subverted* peers were once loyal peers, but they have fallen under the control of the attacker through some means, such as a virus, trojan horse, or other vulnerability in

Loyal	Healthy—has a good copy of AU
	Damaged—has a bad copy of AU
Malign	Subverted—compromised peer, can be healed
	Evil—operated by the adversary

Fig. 2. Classification of Peers



Fig. 3. Examples of what happens to peer *R*'s trust web after various alarms. Malign nodes are marked in gray, loyal nodes in white. (a) The original trust web, or MUTE alarm. (b) RESET alarm, (c) HEAL-RESET/TRUSTWEB-HEAL pessimistic, depth 1, (d) TRUSTWEB-HEAL optimistic, depth 2, (e) TRUSTWEB-HEAL pessimistic, depth 2, (f) TRUSTWEB-HEAL optimistic, depth 2

the peer's software. The true owners of these peers may not realize that the peer has become subverted. If alerted to this fact, however, the owners may be able to cleanse the peers, thus "healing" them - reverting them to the loyal state (albeit probably damaged). *Evil* peers are owned by the attacker, and as such, they are not susceptible to being "healed." We assume that evil peers have duped some loyal peers into including them in their friends lists.

We assume a peer can "heal" its friends (transform them from subverted to damaged) with some probability p. This probability represents the willingness and ability of the operators to heal their subverted peers.

One of the benefits of maintaining the web of trust is having state information that allows a peer to do recursive healing to a certain depth. One could imagine doing similar recursive healing by asking friends of friends and so on, rather than the peers in the web of trust, but this alarm would become large and global quickly. A peer can leverage the web of trust to perform local, targeted healing of its reference list. In addition, a peer may have nominated malign peers that were nominated to him by his subverted or evil friends. With the web of trust, the newly healed peer can revoke these certificates and other peers in the system will learn not to trust them. Without this information, these peers could continue to be used.

We consider two alarms that make use of healing: *HEAL-RESET* and *TRUSTWEB-HEAL*. *HEAL-RESET* alarms (Figure 3c) involve the poller healing its friends and then resetting its reference list to its friends list. *TRUSTWEB-HEAL* alarms have two forms:

• *Pessimistic*—In this situation, a peer *P* suspects that some members of its reference list are subverted, but it does not know which. First, *P* asks its friends to heal themselves. Then, *P* asks nodes at depth 2 in its trust web to heal

themselves, and so forth until some depth d. The peer assumes that all of the peers in question accede to the request, and successfully heal themselves if they are in fact subverted. If a peer P' does not provide an explicit indication that the peer has been subverted, then P assumes that it is loyal (i.e. not subverted). The peer then examines its trust web, and cuts all links that are more than d hops away. Thus the peer is essentially taking the pessimistic position that all peers more than d hops away are suspect, and it ceases to trust them. Pessimistic alarms are shown in Figure 3c and e.

• *Optimistic*—As with the Pessimistic alarm, given depth *d*, the peer asks peers up to *d* hops away in the trust web to heal themselves. Certificates formerly issued by subverted peers are now considered invalid, since the peer will revoke the certificates it signed, so chains containing subverted peers are severed at the link following the subverted peer. However, chains in which no subverted peers were found are left intact, reflecting a more optimistic attitude. This strategy allows the peer to retain the peer discovery and trust relationships it has built. Optimistic alarms are shown in Figure 3d and 3f.

6. EVALUATION

The original LOCKSS authors evaluated the security of its protocol using simulations. In order to provide reasonable comparisons and extend their work, we decided to use their simulator, Narses [16], to evaluate our work.

6.1 The LOCKSS Simulator

This section describes the LOCKSS simulations and the Narses environment. The environment is described in more detail in the LOCKSS literature [22]. Narses is a publicly-available Java-based discrete-event simulator that can scalably simulate large numbers of peers over long periods of time. Both Narses and the baseline LOCKSS extension to Narses are publicly available [1]. Each simulation run consists of 1000 peers, each storing an AU that takes 120 seconds to hash. Each peer calls a poll on its AU every three months, inviting twenty voters per poll; each voter nominates ten peers. With twenty voters and provable computational effort requirements, polls each take about six simulated hours. Time necessary to perform cryptographic operations and checks against the trust webs is negligible in comparison.

We assign peers to each other's friends lists based upon a cluster topology in which each peer has at least thirty friends. The adversary is simulated as a large, multi-homed peer with sufficiently many network interfaces and addresses to carry out its attacks. The number of peers controlled by the adversary is a parameter of the simulation. We consider such peers to have been subverted at the beginning of the simulation.

To simulate the web of trust provided by the distributed PKI, we provide each peer with a trust graph whose edges represent certificates. When new peers are nominated, chains of certificates are added to the trust graph. Every time a peer P_1 is added to the reference list of peer P_0 , peer P_0 checks its graph to ensure that a path exists from P_0 to P_1 . For the purposes of our simulations, we presume perfect online checking of certificates; peers are always online when asked to verify the validity of a certificate.

Our simulations evaluate the behavior of alarm mechanisms and the web of trust used to support those mechanisms. We have several objectives. First, we show that it is possible to derive substantial resistance against adversaries even if we restrict alarms to a local scope. Second, we show that localized healing techniques are robust even if individual healing attempts are unsuccessful. Finally, we explore the benefits of using trust graphs to verify the authenticity of peers.

Given the rate at which polls are called, we do not believe that the performance overhead of maintaining the web of trust or doing cryptographic operations to be significant, since these performance costs are vastly overshadowed by the proofs of effort required at each poll. In addition, we assume that the strong identities provided by our system successfully defeat man-in-the-middle attacks.

6.2 Simulating Alarms

We define *irrevocable damage* as a state of the system in which the adversary damages more than half of the copies of the AU in the system by winning polls called by loyal peers, thus convincing them to repair their copies with the adversary's



Fig. 4. MUTE alarm (does nothing). Numbers above the tickmarks in (a) indicate the proportion of damaged copies at the time of the first alarm.



Fig. 5. RESET alarm (resets the poll initiator's reference list to its friends list)

copy. In previous work on LOCKSS [22], the authors simulate the behavior of the system for ten years of the attack phase, stopping at the occurrance of the first alarm. Since the simulations had shown that the first alarm was called prior to the adversary having achieved irrevocable damage, their results were quite favorable. This was reasonable given the presumption of a global alarm that would completely restore the system to a healthy state. The LOCKSS authors were primarily concerned with demonstrating that the system would not reach a state of irrecoverable damage by the time that the first alarm is called; their results point to the conclusion that it is possible for the system to recover following an alarm. Their simulations presumed an initial subversion ratio of at most forty percent.²

Realistically speaking, we observe that since an alarm does not mark the end of the lifetime of a particular AU, we have an interest in examining the behavior of the system even after alarms have been called. Therefore, we designed a series of experiments to simulate the behavior of alarms described in Section 5.

First, we extend the experiments proposed by the original LOCKSS authors to accomodate a twenty-year simulation window. We presume a *stealth adversary* whose behavior consists of two phases; each phase lasts for a duration of ten simulated years. In the first phase, the adversary lurks, nominating "Sybil" peers (new identities controlled by the adversary) and thus attempting to get a foothold in the reference lists of the loyal peers. In the second phase, the adversary attacks, attempting to damage documents by winning polls and issuing repairs to peers consisting of bad copies. Specifically, the adversary attacks when it knows that it is guaranteed to have secured the supermajority necessary to win the poll; that is to say that the adversary only attacks when an overwhelming majority of invited peers are controlled by the adversary.

All simulations run for the full twenty years. When alarms are called, we have individual peers modify their internal state according to the provisions of the particular alarm mechanism that we are simulating, and we measure the proportion of copies of the AU that are damaged at the end of the simulation.

As a baseline, we simulate behavior of the *MUTE* alarm. This alarm does essentially nothing, which is to say that no peers modify their internal state when such alarms are called. Figure 4a shows the proportion of replicas that the adversary manages to damage by the end of each twenty-year simulation; the error bars indicate the variation in the damage achieved over multiple runs. The diagonal line represents the proportion of subverted peers, which serves to indicate the degree of

²This is more than the 33% failures tolerated by Byzantine systems.



Fig. 6. HEAL-RESET alarm (same as RESET, plus heals friends)



Fig. 7. TRUSTWEB pessimistic alarm, depth 1, heal probability 0.5

damage at the beginning of the run. The horizontal line indicates fifty percent damage; bars above this line indicate cases in which the adversary managed to achieve irrecoverable damage. Figure 4b shows the mean foothold ratio that the adversary manages to achieve in the reference lists of loyal peers. The three lines indicate the mean foothold at the end of the lurking phase (ten years), the mean foothold achieved at the end of the simulation (twenty years), and the maximum mean foothold achieved during the course of the simulation.

Our simulations show what can be expected: if peers do not respond to alarms, then the adversary generally manages to sustain its foothold, and the proportion of damaged copies continues to increase. Note in particular that if the adversary controls more than 27 percent of the peers, then we can expect irrevocable damage within a decade following the start of the attack phase.

The numbers printed above the error bars in Figure 4a indicate the percentage of copies the adversary has managed to damage at the time that the first alarm is called, confirming the findings of the earlier work that conclude that if the adversary controls less than forty percent of the peers, then he cannot cause irrevocable damage before the first alarm. This gives us hope of devising a localized alarm mechanism capable of resisting the stealth adversary. The remainder of our simulations examine the various localized alarm mechanisms proposed in Section 5.

Our first local alarm, *RESET*, is fairly passive: it simply resets the poll initiator's reference list to be equal to its friends list. We show the results in Figure 5. Overall, we found a marked improvement in resistance against the adversary; however, the adversary still achieves irrecoverable damage in simulations involving an initial subversion ratio of 0.28 or more. This is to be expected, since the friends lists still contain subverted peers throughout the simulation, giving the adversary the opportunity to rebuild its foothold in the reference lists after the reset. Figure 5b also shows how systematic resetting of the friends lists can cause the adversary to lose its foothold; note that the mean adversary foothold at the end of the simulation is smaller than its foothold at the conclusion of the lurking phase.

A more active approach to dealing with alarms involves localized *healing* of subverted peers, in which the poll initiator contacts specific other peers in the system and manages to transform them from subverted to loyal. The most basic healing alarm we examine, *HEAL-RESET*, behaves like the aforementioned *RESET* alarm, except that the poll initiator also manages to heal its friends. Our results, shown in Figure 6, illustrate a dramatic improvement over alarms without healing and demonstrate that localized healing is sufficient to limit the effect of the adversary. Figure 6a illustrates that use of the



Fig. 8. TRUSTWEB optimistic alarm, depth 1, heal probability 0.5



Fig. 9. TRUSTWEB optimistic alarm, depth 1, heal probability 1.0

HEALRESET alarm manages to avert irrecoverable damage in nearly all cases in which the initial subversion ratio is less than 0.4. Also, Figure 6b demonstrates that the adversary loses a substantial proportion of its foothold following the onset of its attack phase.

Trust webs provide a richer infrastructure within which peers have non-repudiable knowledge of who is responsible for which nominations and which peers are friends with each other. As such, it is possible for peers to behave more conservatively during alarms; for example, a peer may be able to isolate suspicious peers on its reference list and reset its reference list to what it would have been had it not received nominations from the suspicious peers. Additionally, because a peer has deeper knowledge of the topology of friend relationships, it can potentially hold a friend responsible for its friends; thus, a peer may be able to heal not only its friends but also specific friends of its friends during an alarm. Our *TRUSTWEB-HEAL* alarms employ simulation parameters with these extended capabilities in mind:

- Depth 1 vs. Depth 2. To some extent, the use of trust graphs allows us to hold our friends responsible for the behavior of their friends. Depth 2 simulations allow an examination of the effect of asking friends to heal their friends, in addition to healing themselves.
- **Optimistic vs. Pessimistic.** This parameter determines whether to preserve links to friends of peers not found to be subverted beyond the given depth for healing, as described in Section 5.
- Heal Reliability (probability 1 vs. probability 0.5) For various reasons, it may be unreasonable to assume that peers will reliably heal themselves when asked. For example, system administrators of other peers may not always be willing or able to perform healing operations when asked. Performing simulations without a guarantee of reliability allows us to determine the sensitivity of our claims to practical concerns related to human interaction and error.

A pessimistic *TRUSTWEB-HEAL* alarm with depth 1 (a peer heals its friends list and cuts everyone else from its trust graph) is identical to *HEAL-RESET*, aside from the revocation of certificates signed by healed peers. Our results (not shown) reflect this similarity.

Figure 11 shows our results for *TRUSTWEB-HEAL* alarms at depth 2. We find some improvement over the corresponding alarms with depth 1, again supporting our assertion that a local alarm can be effective and also indicating that it is possible to derive benefit from healing peers within a wider scope of locality than friends lists.

Figures 7 and 10 demonstrate the effects of unreliable healing; i.e. peers receiving requests to heal remain subverted with



Fig. 10. TRUSTWEB pessimistic alarm, depth 2, heal probability 0.5



Fig. 11. TRUSTWEB pessimistic alarm, depth 2, heal probability 1.0

probability 0.5. We observe that even for cases in which half of the requested heal operations fail, the results of simulation are remarkably similar to their counterparts in which heal requests are reliable.

Finally, we consider optimistic *TRUSTWEB-HEAL* alarms, in which the nominations of peers not found to be subverted are preserved in the trust graph. We find that these results, as seen in Figure 9, are very similar to this pessimistic case. These results, combined with the results for the *RESET* alarm, suggest that it is the number of peers healed, rather than direct modifications to the poll initiator's reference list, that results in the most significant improvement.

7. FUTURE WORK

The security of LOCKSS depends on peers choosing peers on their friends list that are operated by librarians acting in good faith. If an *evil* peer socially engineers its way onto friends lists, it may still cause considerable damage. We believe we can augment the web of trust maintained by each peer to use a trust metric, such as that used by Advogato [20] to limit this damage. The Advogato trust metric is designed to provide assurance that no single peer or small group of peers accrues disproportionate influence on the trust decisions of other peers.

New software vulnerabilities and exploits tend to arise at a rate greater than once every twenty years. It is conceivable that an adversary will try to reinfect machines after losing some to healing. We would like to simulate the performance of our alarm mechanisms if the system is subject to periodic infections. This may require us to be more agressive in our healing (further recursive levels of depth).

Also, we would like to simulate imperfect online checking of revocation information; right now, peers are assumed to be always up and responding to OSCP queries.

While this paper addresses improving recovery procedures following intrusion detection, we believe that strengthening the intrusion detection itself may provide benefits. For example, inconclusive poll alarms can be more effective at detecting adversarial activity if we augment the LOCKSS protocol with local hashing. In a version of LOCKSS augmented in this fashion, each peer would keep a hash of its copy of the AU. While these hashes may become corrupted, they would serve as an extra check on the stealth adversary. Then, when a peer calls a poll and its hash is determined to be valid, it knows that it should not receive a repair. If the vote indicates that it should repair, then it would conclude that adversarial activity is likely with high probability, and it would call an inconclusive poll alarm. Peers might be tempted, using this augmentation,

to only call polls when its hash does not match its document, but this would eliminate the intrusion detection benefits of alarms. In addition, the peer would not have been participating in others' polls and performing peer discovery, so it would have stale information about the system when it needs to call a poll. Thus, even if the system were to be augmented in this manner, peers would still need to conduct polls at regular intervals.

Finally, many aspects of the LOCKSS system have potential uses outside the realm of library preservation of digital journals. As such, it may be of interest to consider potential application of the LOCKSS system to networks of different size and scope, as well as to networks designed to deliver other kinds of content. We would also explore how our authentication mechanism might be useful in other peer-to-peer systems.

8. CONCLUSION

The LOCKSS system aims to allow libraries to preserve and archive electronic documents, achieving consensus about the correct version of a document despite unreliable storage media and despite adversaries attempting to modify content they consider offensive. By removing the LOCKSS protocol's reliance on network layer addresses for authentication, we strengthen its security model, enabling LOCKSS to resist stealth modification attacks from adversaries that can control ISPs or Internet routers. Our decentralized PKI provides a general mechanism that can be used to perform authenticated peer discovery in peer-to-peer systems.

While previous work on LOCKSS has focused on intrusion detection, we have provided specific and effective mechansim for recovery following intrusion. Specifically, we have strengthened LOCKSS by presenting a localized mechanism for dealing with *inconclusive poll* alarms. This mechanism allows us to limit the damage caused by a stealth adversary that has compromised up to roughly 40% of the peers in the system. Our changes present a realistic means of containing the *stealth adversary* and allowing nodes to achieve the benefits of LOCKSS without fear.

REFERENCES

- [1] Project: LOCKSS. http://sourceforge.net/projects/lockss/.
- [2] R. 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.
- [3] Association of Research Libraries. ARL Statistics 2000-01. http://www.arl.org/stats/arlstat/01pub/intro.html, 2001.
- [4] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. Technical Report 96-17, 28, 1996.
- [5] G. H. W. Bush and B. Scowcroft. Reasons not to invade iraq. http://www.thememoryhole.org/mil/bushsr-iraq.htm.
- [6] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In H. 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] L. P. Cox and B. D. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pages 120–132, Bolton Landing, NY, USA, Oct. 2003.
- [8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area Cooperative Storage with CFS. In SOSP, 2001.
- [9] T. Dierks and C. Allen. RFC 2246: The TLS protocol version 1, Jan. 1999. Status: PROPOSED STANDARD.
- [10] W. Diffie. Perspective: Decrypting the Secret to Strong Security. http://news.com.com/2010-1071-980462.html, Jan. 2003.
- [11] W. Diffie and M. E. Hellman. New Directions in Cryptography. IEEE Transactions on Information Theory, IT-22(6):644–654, Nov. 1976.
- [12] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In H. 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] J. Douceur. The sybil attack. In Proceedings of the IPTPS02 Workshop, 2002.
- [14] S. Fairbrother. Certificate revocation in public key infrastructures. http://www.giac.org/practical/GSEC/Scott_ Fairbrother_GSEC.pdf, 2003.
- [15] C. Fullmer. Storage and Multimedia: The Facts and More. http://www.cse.ucsc.edu/classes/cmpe003/Fall02/L11_ch6. pps, 2002.
- [16] T. Giuli and M. Baker. Narses: A Scalable, Flow-Based Network Simulator. Technical Report arXiv:cs.PF/0211024, Computer Science Department, Stanford University, Stanford, CA, USA, Nov. 2002.
- [17] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile, Jan. 1999.

- [18] Human Rights Watch. Silencing the net: The threat to freedom of expression. Human Rights Watch Report, 8(2), May 1996.
- [19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. 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, Nov. 2000.
- [20] R. Levien. Advogato's trust metric. http://www.advogato.org/trust-metric.html/.
- [21] J. Lippard and J. Jacobsen. Scientology v. the internet: Free speech & copyright infringement on the information super-highway. 3(3):35-41, 1995. http://www.skeptic.com/03.3.jl-jj-scientology.html.
- [22] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, M. Baker, and Y. 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, Oct. 2003.
- [23] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, M. Baker, and Y. Muliadi. Preserving Peer Replicas By Rate-Limited Sampled Voting in LOCKSS. Technical Report arXiv:cs.CR/0303026, Stanford University, Mar. 2003.
- [24] P. McDaniel and S. Jamin. Windowed key revocation in public key infrastructures. Technical Report CSE-TR-376-98, 13, 1998.
- [25] N. Michalakis, D.-M. Chiu, and D. S. H. Rosenthal. Long Term Data Resilience Using Opinion Polls. In 22nd IEEE Intl. Performance Computing and Communications Conference, Phoenix, AZ, USA, Apr. 2003.
- [26] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 2560: X.509 Internet Public Key InfrastructOnline Certificate Status Protocol - OCSP, June 1999.
- [27] D. A. Patterson, G. Gibson, and R. 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.
- [28] C. Quirke. Hard Drive Data Corruption. http://users.iafrica.com/c/cq/cquirke/baddata.htm, Nov 2002.
- [29] B. Ramsdell. S/mime version 3 certificate handling, June 1999.
- [30] R. L. Rivest. Can we eliminate certificate revocations lists? In Financial Cryptography, pages 178–183, 1998.
- [31] D. S. H. Rosenthal and V. 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.
- [32] D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, and M. Baker. Economic Measures to Resist Attacks on a Peer-to-Peer Network. In Proceedings of Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, June 2003.
- [33] A. Rowstron and P. 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, Oct. 2001.
- [34] International Telecommunication Union. Recommendation X.509. Data Networks and Open System Communications The Directory: Authentication Framework, 1997. Version 8/1997.
- [35] M. Waldman and D. 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, Nov. 2001.
- [36] P. R. Zimmermann. PGP source code and internals. MIT Press, 1995.