Preventing History Forgery with Secure Provenance

Ragib Hasan
University of Illinois at Urbana-Champaign
Radu Sion
Stony Brook University

Marianne Winslett
University of Illinois at Urbana-Champaign
Email:(hasan,winslett)@cs.uiuc.edu, sion@cs.stonybrook.edu

May 24, 2009

Abstract

As increasing amounts of valuable information are produced and persist digitally, the ability to determine the origin of data becomes important. In science, medicine, commerce, and government, data provenance tracking is essential for rights protection, regulatory compliance, management of intelligence and medical data, and authentication of information as it flows through workplace tasks. While significant research has been conducted in this area, the associated security and privacy issues have not been explored, leaving provenance information vulnerable to illicit alteration as it passes through untrusted environments.

In this paper, we show how to provide strong integrity and confidentiality assurances for data provenance information at the kernel, file system, or application layer. We describe Sprov, our provenance-aware system prototype that implements provenance tracking of data writes at the application layer, which makes Sprov extremely easy to deploy. We present empirical results that show that, for real-life workloads, the runtime overhead of Sprov for recording provenance with confidentiality and integrity guarantees ranges from 1% – 13%, when all file modifications are recorded, and from 12% – 16%, when all file read and modifications are tracked.

1 Introduction

On May 3, 2006, the Picasso painting *Dora Maar au Chat* (Dora Maar with Cat) was auctioned at Sotheby's for US $95 million, becoming one of the most expensive paintings in the world. At the same time, someone listed several paintings for sale on eBay, supposedly complete with Picasso's signature. Though eBay quickly removed those listings, many purported Picasso paintings are still on the market.

How do art buyers authenticate paintings? Many factors play a role, but one key element is provenance records that list the ownership history of an item and the actions performed on it. Long used in archives, art, and archaeology, recently provenance has become essential for digital documents in financial, commercial, medical, scientific, and legal contexts. Such information often originated in a remote location, has been processed by multiple parties, and has resided in potentially untrustworthy storage. In such situations, it is increasingly important to know where the information comes from and how it has been processed and handled. In other words, to be able to trust the information in the document, we need to know its provenance.

For example, scientists need to keep track of data creation, ownership, and processing workflow to ensure a certain level of trust in their experimental results. The National Geographic Society's Genographic Project and the DNA Shoah project (for Holocaust survivors searching for remains of their dead relatives) both carefully track the processing of DNA samples. Individuals who submit DNA samples for testing through these programs want strong assurances that no unauthorized entities will be able to see the provenance of the samples and, for example, provide that information to insurance companies or anti-Semitic organizations.

Regulatory and legal considerations mandate provenance assurances for businesses as well. The US Sarbanes-Oxley Act [U.S02] sets prison terms for chief financial officers who sign their names on incorrect corporate financial statements. As a result, CFOs have become very interested in provenance information that authenticates the path that a financial report took during its development, the data that contributed to the report, and the people who worked on the report. The Gramm-Leach-Bliley Act [Con99] and Securities and Exchange Commission rule 17-a [The03b] also require documentation and audit trails for financial records, as do many non-financial compliance regulations. For example, the Health Care Portability and Accountability Act (HIPAA) mandates logging of access and change histories for medical records [Cen96].

Provenance is also important for patents, proving authorship, and other intellectual property litigations, to support rights assertions in court [Gob02]. Often claimants must produce evidence of prior work and proof of their research chronology. A secure provenance chain is ideally suited for this. To be admissible in US courts, digital and physical evidence must have a firmly established chain of custody. In high-stakes cases, lawyers for the defense often allege that evidence has been tampered with (e.g., O. J. Simpson's glove), whether or not it has been. Conversely, innocent people have been framed and convicted using tampered evidence. Even in large US cities, police departments have little technological assistance in securing chains of custody for evidence.

The provenance of data and programs is central to digital forensics and post-incident investigation of intrusions [Gob02, SPG05]. If operating systems are augmented with provenance mechanisms, we can expand provenance tracking to system files and installation kits. System binaries can include provenance information and record all the changes made to them. If the integrity of such a provenance chain can be ensured, forensic investigators can detect and trace changes to a binary introduced by an attacker. Similarly, secured provenance information can serve as a certificate of authenticity during software distribution and updates.

Provenance tracking of physical artifacts is relying increasingly on digital shipping, manufacturing, and laboratory records, often with high-stakes financial incentives to omit or alter entries. For example, pharmaceuticals' provenance is carefully tracked as they move from the manufacturing laboratory through a long succession of middlemen to the consumer. Clinical trials of new medical devices and treatments involve detailed recordkeeping, as does US Food and Drug Administration testing of proposed new food additives. The dates and places of residence of each US cow are recorded and the cow itself is tagged; if it is found to have mad cow disease, these records are used to track down its old farm-mates and slaughter them, with heavy financial consequences for their owners.

To help manage the above processes, digital provenance mechanisms support the collection and persistence of information about the creation, access, and transfer of data. Provenance mechanisms enable authorized parties to later evaluate rights claims, data flow integrity, authenticity, and ultimately, establish an acceptable level of trust in the data.

While significant research has been conducted on how to collect, store, and query provenance information, the associated integrity and privacy issues have not been explored. Yet unless the integrity of provenance information is ensured, it cannot be effective as a tool for document trust assessment. Otherwise, as information crosses application and organization boundaries and passes through untrustworthy environments, its associated provenance information becomes vulnerable to illicit alteration and should not be relied upon.

For example, consider the repudiation incentives in the following *real-life* anonymized medical litigation scenario, illustrated in Figure 1. Dana visited Dr. Alice for consultation. Dr. Alice referred her to Dr. Bob for tests, and sent Dana's medical records to Dr. Bob, who failed to analyze the test results properly. Dr. Bob provided these reports along with other information to Dr. Charlie, who treated Dana accordingly. When Dana subsequently suffered from health problems related to the incorrect diagnosis, she sued Alice and Charlie for malpractice. To establish Bob's liability for the misdiagnosis, Alice and Charlie hired Audrey as an expert witness. Audrey used the provenance information in Dana's medical records to establish the exact sequence of events, which in this case implicated Bob. If Bob had been innocent, Alice and Charlie should not be able to collude and falsely implicate him. Similarly, if Bob altered his faulty diagnosis in Dana's medical records after the fact, Audrey should be able to detect that.

Making provenance records trustworthy is challenging. Ideally, we need to guarantee *completeness* – all relevant actions pertaining to a document are captured; *integrity* – adversaries cannot forge or alter provenance records; *availability* – auditors can verify the integrity of provenance information; *confidentiality* – only authorized parties should read provenance records; and *efficiency* – provenance mechanisms should have low overheads.

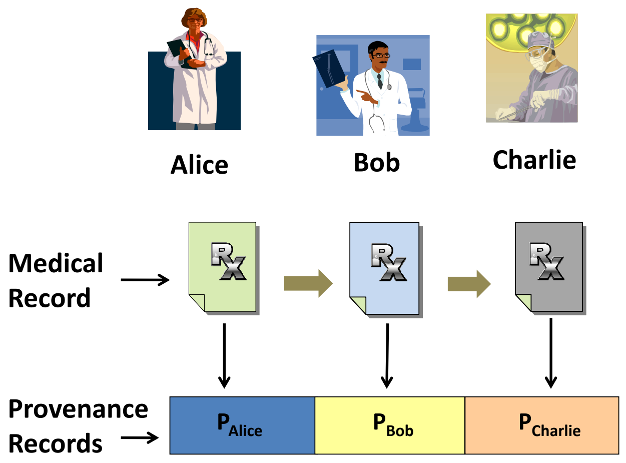

Figure 1: Example scenario: Dana goes to Dr. Alice, who refers her to Dr. Bob for a test, and the test results are then sent to Dr. Charlie. Because of a misdiagnosis by Dr. Bob, Dana suffers health problems, and sues Dr. Alice and Dr. Charlie, who want to use the provenance ($P_{Alice}|P_{Bob}|P_{Charlie}$) of Dana's medical records to prove their innocence. Dr. Bob has an incentive to hide his actions by retroactively altering his entries in Dana's medical records, and tampering the provenance record P_{Bob} of his diagnosis to match.

1.1 Contributions

In this paper, we propose and evaluate mechanisms for secure document provenance that address these needs. The contributions of this paper are as follows:

1. a cross-platform, low-overhead architecture for capturing provenance information at the application layer and guaranteeing its security;
2. a proof of correctness for our approach to secure provenance;
3. an implementation of our approach for file systems; and
4. an experimental evaluation that shows that our approach to provenance collection with security guarantees introduces modest overheads at run time: 1%–13% for most real-life workloads (when tracking write operations); and 8%–10% with Postmark and 12%–16% with real-life workloads (when tracking both read and write operations).

1.2 Organization

The rest of the paper is organized as follows. Section 2 defines the secure provenance problem and the terms associated with it. We present our approach to addressing these problems in Section 3, provide proofs of correctness in Section 4, and discuss potential enhancements in Section 5. We describe a proof-of-concept implementation and experimental results in Section 6. Section 7 discusses previous work on provenance, and we conclude in Section 8.

2 Threat Model

We use the term **document** to refer to the data item for which provenance information is collected, such as a file or database tuple. At the IT layer, the **provenance** of a document is the record of actions taken on that document over its lifetime. Each access to a document D may generate a **provenance record** P. The types of access that should generate a provenance record depend on the domain, as do the exact contents of the record, but in general P may include the identity of the accessing principal; a log of the access actions (e.g., read, write) and their associated data (e.g., the bytes of the document or its metadata that were read/written); a description of the environment at the time of the action, such as the time of day and the software environment; and confidentiality- and integrity-related components, such as cryptographic signatures, checksums, and keying material. A **provenance chain** for document D is a non-empty time-ordered sequence of provenance records $P_1 | \cdots | P_n$.

In a given security domain (organization), **users** are principals who read and write documents and their metadata. Each organization has one or more **auditors**, who are principals authorized to verify the integrity of provenance records associated with documents. A *superauditor* is a set of one or more auditors, such that for every provenance record in a chain, at least one auditor in the set is authorized to access all details of that record. Documents move from one location to another, as email attachments, FTP transfers, file copy operations, or by other means. Provenance chains move with the documents. When a user modifies a document, a new provenance record describing the modifications is appended to the provenance chain, and the user permits a non-empty subset of the auditors to read the new record. **Adversaries** are principals from inside or outside an organization who have access to a document and its provenance chain and want to alter them inappropriately, as discussed below.

In general, provenance collection mechanisms become harder to subvert when they are implemented at lower levels of a system. However, we can never track provenance perfectly, because a provenance tracking system implemented at a particular level of the system is oblivious to attacks that take place outside the view of that level. For example, suppose that we implement provenance tracking for tuples inside a database management system. An adversary can subvert provenance collection by opening the database file with a private copy of the database management system that has provenance collection turned off, or by using a file editor to modify the database file. Suppose instead that we implement provenance tracking in the OS kernel. If the kernel is not running on hardware that offers special security guarantees, an intruder can take over the machine, subvert the kernel, and circumvent the provenance system.

Given a guarantee that users could never circumvent the provenance mechanisms, we could reliably track all information flows by recording all information that each user reads, in addition to what they write. Unfortunately, provenance tracking for read operations is especially easy to subvert. Even with a trusted pervasive hardware infrastructure and provenance tracking at every level of the system, a malicious user who can read a document can always memorize and replicate portions of it later, minus the appropriate provenance information. Since we cannot fully monitor the information flow channels available to attackers, our power to track the origin of data by monitoring read operations is limited. For situations where it is important to track reads, we provide an optional low cost provenance collection mechanism in Section 6. In the rest of the paper, we confine our attention to tracking actions that modify documents.

Consider the version history that would result if a document were created and subsequently edited and transferred from user to user by a sequence of permitted actions, with provenance information correctly and indelibly recorded all along the way. We call this a *plausible history* for the resulting document and its provenance chain.

Definition 1. *Suppose that a sequence of users carries out permitted actions on a newly created document D, and correctly records a provenance chain record for every action. The resulting provenance chain is a* **plausible provenance chain** *for D.*

There are so many potential uses for provenance information that no single set of security guarantees is a perfect match for all potential applications. We target applications whose provenance integrity needs are met by the following guarantee: *auditors can determine whether the provenance chain associated with a document is plausible.* Such applications are common. For example, a retail pharmacy will not accept a shipment of drugs unless it can be shown that the drugs have passed through the hands of certain middlemen. If a criminal wants to sell drugs manufactured by an unlicensed company, he will want to forge a provenance chain that gives the drugs a more respectable history, so that he can move them into the supply chain. Our approach detects that the new chain is forged. The criminal will not want to take drugs manufactured and distributed through legitimate channels, strip off their distribution provenance records, and replace it by a record showing that he distributed the drugs himself, as this new plausible history for the drugs makes them worthless. Similarly, there is little danger that someone will remove the provenance chain associated with a box of Prada accessories, and try to pass them off as another brand. Instead, the incentive is to pass off non-Prada accessories as Prada, and we detect this attack.

The plausibility of a particular provenance chain depends on the background knowledge of the viewer and the information included in the chain. For example, suppose that Audrey audits a one-record chain that says that Bob wrote file D at time t. If the chain is properly formed by Bob, and no one has tampered with D or the chain, then presumably the chain and D should pass the audit. But given additional information about the state of the file system where D resided, and the application Bob used to access D, then Audrey would be able to recognize that some plausible chains could not really have occurred in practice. For example, perhaps Bob was not a user in the system at time t. Or perhaps Bob did not have the file system privileges needed to write D. Perhaps Bob was authorized to write D, but the system was down at time t. Perhaps "Bob" is actually a database management system, and the user who was running the database application would not have been permitted to make those changes in the database file D. In general, all such semantics- and environment-aware restrictions on plausible chains must be enforced by policies established by higher levels of provenance chain audit software. These policies can either be checked by the higher levels themselves in a separate pass over the chain, or by a routine passed down to lower levels for enforcement. For example, in some distributed systems it is reasonable to assume that the times recorded in successive provenance chain records will increase monotonically. Such temporal checks can be performed by a routine passed down from higher levels, when appropriate for a particular deployment environment. In this paper, our goal is to create a generic auditing facility, with no application-specific background knowledge. Thus we will assume that if Bob signed a provenance record saying that he performed a particular action, then it was possible for him to perform that action.

It may be helpful to consider some short concrete scenarios that may give rise to implausible chains:

- **I1:** An adversary acting alone removes one or more other principals' records from the beginning or middle of a provenance chain.
- **I2:** An adversary acting alone adds records in the beginning or the middle of the chain.
- **I3:** After the chain contains subsequent records by non-colluding parties, two colluding adversaries who have contributed records to a provenance chain add records of other non-colluding users between theirs.

For example, suppose that we have a provenance chain A B C D in which, for simplicity, each record is denoted by the identity of its corresponding principal. Suppose that colluding users A and C add records between their own, corresponding to fabricated actions by non-colluding parties B, D, or E, and then give the chain to an auditor. The auditor will detect the violation.

- **I4:** After the chain contains subsequent records by non-colluding parties, two colluding adversaries who have contributed records to a provenance chain remove the record of a non-colluding user between theirs.

For example, suppose that colluding users A and C remove the record made by non-colluding user B, then give the chain to an auditor. The auditor will detect the violation.

A user Ed may be able to access an old version of a document and its chain; for example, the file system may store all versions of a file. Or, if Ed can read the last few records in the chain, he may be able to undo the actions recorded in them, outside the view of the provenance system. Or he may be able to obtain an old version of the document from outside the system. No matter how he gets an old version of the document, clearly he can truncate the provenance chain to create a plausible history for the old version. Then Ed and his colleagues can perform actions on the old version as desired, while the provenance system adds the appropriate new records to the chain. The resulting chain gives a plausible history for the document. However, constraint I4 prevents Ed from attributing any of the new writes to those who are not collaborating with him. For example, suppose Ed undoes records C and D, where D is a stamp of approval from the Food and Drug Administration. His collaborator C then alters the old version of the document, generating a new and different provenance record C'. Suppose that they then reaffix the old FDA stamp of approval D without the FDA's help and cooperation, and give the chain A B C' D to an auditor. The auditor will detect the tampering, because the chain does not correspond to any plausible history.

- **I5:** A user who contributed a record to a chain subsequently claims that that record was forged.
- **I6:** An adversary claims that a valid provenance chain for one document belongs to a document with different contents.

- **I7:** An adversary alters a document without appending an appropriate provenance record to its chain.

We also provide two confidentiality guarantees:

- **C1:** Any auditor can verify the integrity of a chain without accessing any of its confidential components. Unauthorized access to confidential provenance record fields is prevented.
- **C2:** The set of parties originally authorized to read the contents of a particular provenance record for D can be further restricted by subsequent writers of D.

Our integrity and confidentiality assurances rest upon certain assumptions. First, the keys used to create and read chain records are never compromised or revoked. Second, auditors can use a trusted source to obtain the public keys of users who create chain records, and all such sources agree on what those public keys are. Third, there is always at least one auditor. Fourth, if we need to be able to check that a document matches its declared history, the document must have a superauditor throughout its lifetime. Fifth, auditors perform their audits correctly and report the results correctly. Relaxation of these assumptions for long-lived chains is interesting area for future research. For example, what is the best way to preserve integrity and confidentiality guarantees if keys can be revoked or stolen? What should one do when brute-force attacks on current cryptography schemes become practical due to hardware advances? How can we guarantee availability of the public key that a particular user had when they wrote a provenance record long ago?

Any auditor will be able to determine whether the current contents of D match the contents claimed in the last record of D's chain. Under certain conditions, auditors will also be able to verify that the sequence of actions described in D's chain would produce D. For this additional check to be possible, the set of auditors carrying out the audit must be authorized to see the details of each action in the chain, i.e., must be a superauditor. Further, the actions described in the chain must be either replayable or reversible, so that the auditors can determine whether the actions could have the claimed effect. In this paper, we adopt a very simple formalization of these ideas, wherein either all action descriptions are reversible for all documents, all are replayable for all documents, or else auditors cannot guarantee that all the previous versions of a document are as specified in its provenance history. Further, if actions are supposed to be reversible or replayable, then any history that involves a non-reversible or non-replayable action is not plausible. We will formalize these concepts after introducing a particular representation for provenance chains.

The choice between audits with or without replay/undo is *fundamental.* Audits are much faster and easier without replay/undo, but also much weaker: they only check the integrity of the provenance chain and verify that the final state of the document is as specified at the end of the chain. They ignore the actions in the chain, and thus do not address the question of whether the actions could actually have produced this final state. We will formalize this difference after introducing a particular representation for provenance chains.

3 A Secure Provenance Scheme

We propose a solution composed of several layered components: encryption for sensitive provenance chain record fields, a checksum-based approach for chain records, and an incremental chained signature mechanism for securing the integrity of the chain as a whole. For confidentiality (C1), we deploy a special keying scheme based on broadcast encryption key management [HS02, KSW06] to selectively regulate the access for different auditors. Finally, for confidentiality (C2), we use a cryptographic commitment based construction. In the following, we describe these components.

3.1 Chain Construction

Provenance records (**records** for short) are the basic units of a provenance chain. Each record P denotes a sequence of one or more actions performed by one principal on a document D:

$$P \quad = \quad \langle U, W, hash(D), C, public, I \rangle,$$

where

- U is an identifier for the principal;
- W is an opaque representation of the sequence of document modifications performed by U;

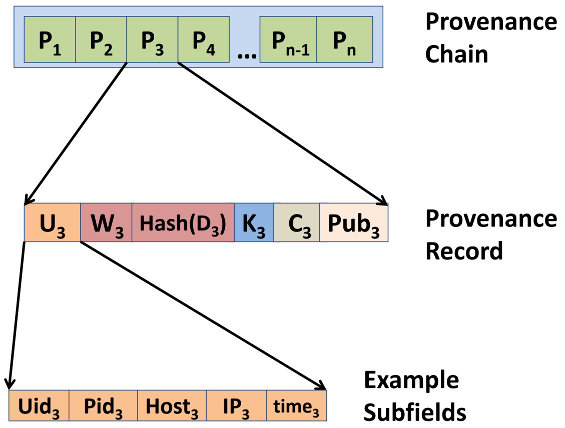

Figure 2: Structure of a provenance chain, showing a provenance record and the fields of its user identity component.

- $hash(D)$ is a cryptographic hash of the contents of D, including all aspects of D being tracked by the provenance system;
- C contains a record integrity checksum;
- *public* is an optional public key certificate for user U;
- I contains keying material for interpreting the preceding fields.

As a practical matter, at the start of an editing session, the provenance system should verify that the current contents of D match its hash value stored in the most recent provenance record.

We discuss each of these fields in the following subsections.

3.2 Confidentiality

Let w be a representation of the sequence of document modifications just now performed by U. The choice of representation for w is dictated by the application domain; for example, w could be a file diff, a log of changes, or a higher-level semantic representation of the alterations. The representation should be reversible or replayable, if auditors must perform strong plausibility checks. Given w, W is an encrypted version of w:

$$W = E(w)$$

U should identify the principal in a manner appropriate for the application domain. For example, in a file system we can define U as:

$$U = \langle userID, pid, port, ipaddr, host, time \rangle.$$

Auditors must obtain U's public key from a trusted source. The auditor can consult an online authority, or extract U's key from the *public* field of the record, if *public* has been signed by an authority that the auditor trusts. In this paper, U and *public* are in plaintext. In practice, however, if U and *public* are sensitive, they can be stored in encrypted form as long as *all* auditors can decrypt them. For example, they can be encrypted with a secret key known to all auditors.

In the remainder of this section, we discuss options for E that satisfy the C1 confidentiality requirement.

Straw Man Choices of E. If all auditors should be able to read all provenance records, we can encrypt all w using a single secret key k shared with the auditors via a central keystore:

$$E(w) = e_k(w).$$

If only a subset of the auditors should be able to read w, then U can encrypt one copy of w for each auditor that U trusts:

$$E(w) = \{e_{K_A}(w) : K_A \text{ is the public key of } A, \text{ an auditor } U \text{ trusts}\}$$

This is inefficient, as U has to include multiple copies of w, which may be quite large. One solution approach is to let each auditor in the provenance system correspond to an auditor *role* in the larger environment, and use machinery external to the provenance system to determine who can get access to the auditor role and to track the activities of auditors. Still, the number of different auditor roles can become quite large in a real-world institution. For example, a medical record might be audited by lab technicians, billing people, the patient, her guardians, physicians, and insurers, each with different rights to see details of what was done to the record. Hence, the use of auditor roles external to the provenance system needs to be coupled with internal measures to minimize the size of records.

To save space, U can encrypt w with a session key k unique to this record, and also include the (shorter) session key k encrypted with the public keys of the trusted auditors. That is,

$$E(w) = e_k(w)$$

and

$$I \quad = \quad \{e_{K_A}(k) : K_A \text{ is the public key of } A, \text{ an auditor } U \text{ trusts}\}$$

In the rest of this paper, we assume that each record employs a session key. Still, I may have to include many encrypted auditor keys.

Broadcast Encryption for E. To reduce the number of keys that must be included in P, we employ broadcast encryption [HS02, KSW06]. We illustrate with a specific instance, but any broadcast encryption approach can be used here.

Given a set of up to n auditors during the lifetime of D, we build a broadcast encryption tree of height $\lceil \log N \rceil$. Each leaf corresponds to one auditor, and each node contains a public/private key pair in a PKI infrastructure. We give all the public keys in the tree to every user and auditor. In addition, we give each auditor the private keys in all the nodes on the path from its own leaf to the root.

To allow all auditors to access a record, we encrypt the session key k with the public key in the root of the tree and store it in I. Any auditor can then decrypt W using the private key in the root node (known only to auditors). Similarly, if U trusts a single auditor A, we encrypt k with the public key k_A at the leaf for A, so that $I = e_{k_A}(k)$. If U trusts an arbitrary subset S of auditors, we set

$$I = \{e_{k_B}(k) : B \; is \; the \; public \; key \; of \; a \; node \; in \; R\},$$

where R is a minimal-size set of tree nodes such that the set of descendants of R includes all the leaves for auditors in S, and no other leaves. This approach can significantly reduce the size of I, as on average I should include only $\log(n - |S|)$ copies of k.

In some application domains, we may need finer-grained control over which auditors and users can see which details of a record, rather than using a single session key to encrypt all sensitive fields in a record. For example, we might be willing to let the billing auditors decrypt U but not W. In this case, one option is to use additional session keys for the other sensitive fields of the record, so that we can control exactly which auditors can see which fields.

Threshold Encryption. To address separation-of-duty concerns, we can partition the set of auditors into groups, so that decryption of W requires joint input from at least one auditor from each group. Alternatively, we can require that at least k different authorized auditors act jointly to decrypt W. For these two approaches, we employ secret sharing and threshold cryptography for I [Sha79]. Under the first approach, each group has a different share of the session key i, and we use the broadcast encryption keys to encrypt those shares. Each auditor can decrypt the share for her group. Under the second approach, there are as many different shares as auditors, and a minimum threshold number of auditors must collaborate to decrypt W.

3.3 Integrity

Principle C1 says that every auditor can check the integrity of every provenance chain, even if she cannot decrypt some of its W fields. In some application domains, it is appropriate to allow every user to act as an auditor in this weak sense. In this situation, we can define the integrity checksum field C of a record P as:

$$C = S_U(hash(U, W, hash(D), public, I) \mid C'),$$

where S_U means that user U signs the hash with his or her private key, and C' is the checksum of the previous record in the chain, if any. We refer to this approach as **signature-based checksums**, as it creates a signature chain that ensures that implausible chains will be detected. For an auditor Audrey to check the integrity of the chain, she must obtain the public keys for the users who wrote the chain, as discussed earlier. Audrey can then verify the integrity of the provenance chain by parsing it from beginning to end and using the C values to verify the integrity of each record. She can also verify that the current contents of D match the value $hash(D_n)$ stored in the last record P_n of the chain, by hashing exactly the aspects of D's metadata and internal contents that are being tracked by the provenance system. In this paper, two documents D and D' are the same iff they are the same from the viewpoint of the provenance system, i.e., $hash(D) = hash(D')$. For example, if we only track the set of bytes stored in a document, then all empty documents are the same document. The choice of which aspects of a document to track is application-dependent.

Audrey cannot check that D_n was computed properly from D_{n-1} unless she can obtain the session key k_n and decrypt W_n, and a reversible representation was used for actions. To verify that all actions in the chain were performed correctly, Audrey must retrieve her keys from the broadcast encryption tree and use them to decrypt the I field of each record. The I field holds the session key for each record, which she can use to obtain the w field. From D_n and reversible w_n, Audrey can compute D_{n-1} and verify that it matches its hash in P_{n-1}. Audrey can repeat this process with w_{i-1}, continuing until the entire evolution of D has been verified. Audrey can check replayable actions in a similar manner. If Audrey is not authorized to access all of the session keys for D, then she can only verify that the most recent j records match the contents of D, where session key k_{n-j} is the most recent session key that she cannot access. To verify the entire chain, she must find enough colleagues she trusts to form a superauditor.

This suggests the following formalization of replayability and reversibility, where D_0 is a nonexistent document with no hash:

Definition 2. *The action described in provenance chains can be* **replayed** *if for every document D with associated plausible provenance chain $P_1 \cdots P_n$ and all $0 < i < n$, the value $hash(D_i)$ recorded in P_i is the same as that produced by a superauditor simulating the actions recorded in $P_1 \cdots P_i$, and then hashing the resulting content.*

Definition 3. *The actions described in provenance chains can be* **reversed** *if for every document D with associated plausible provenance chain $P_1 \cdots P_n$ and all $0 < i < n$, the value $hash(D_{i-1})$ recorded in P_{i-1} is the same as that produced by a superauditor simulating the undoing of the actions recorded in $P_i \cdots P_n$, and then hashing the resulting content.*

These two definitions have some important ramifications. First, they assume that the superauditor can simulate the action recorded in a provenance record, as that action would have been performed by the user who wrote that record. A different user performing the same action might obtain a different result. For example, a variety of final states may result from Bob, Carl, or the bank trying to withdraw \$100 from Bob's checking account at different times.

Second, the exact choice of content to track in provenance records has a big impact on whether actions can be replayed or reversed. As a very simple example, suppose that metadata regarding the last write time of documents is to be tracked. All tracked metadata is part of D's content, and must be included when computing $hash(D)$. If D is a file, any auditor can find this metadata in the file system and include it in the hash of the *current* version of D. To check this metadata for an *old* version of D, auditors need to obtain the *old* last write time either from the provenance chain or an external source, e.g., from a versioning file system. Similarly, if metadata regarding a file's name, path, permissions, etc., is to be tracked, then actions can only be reversed or replayed if the file system or the provenance record itself makes that information available to auditors. Thus to support a wide variety of applications, implementations of our approach need to be flexible regarding the exact content included in $hash(D)$. Further, to support replay and reversal for a wide variety of applications, implementations also need to allow additional fields to be added to the record data structure and included in the checksum computation.

Third, the choice of content to track and the desired precision of any undo/redo checks dictate which operations must be monitored, and what aspects of those operations must be described in w. For example, if file permissions are tracked, then all operations that could change those permissions (e.g., `chmod`) must be within the view of the provenance collection system. If an application needs very precise undo/redo checks, then even more operations must be within view. For example, suppose that auditors must be able to check whether U would have been able to write file D at the time a particular provenance record was generated. To enforce this check, the provenance system must track all changes in group membership.

Recall that audits are weaker if they do not include replay/undo. We formalize this distinction in the following definitions.

Definition 4. *A provenance chain is syntactically well-formed if for each record $\langle U, W, hash(D), public, I, C\rangle$ of the chain, all of the following hold:*

- *U is a user with public key p.*
- *$hash(D)$ is the correct length for the output of function $hash()$.*
- *C is well-formed, i.e., the value of C is what would be produced by using the private key corresponding to p to sign $hash(U, W, hash(D), public, I) \mid C'$, where C' is the checksum stored in the previous record (if any) of the chain.*

Digital signatures have the property that any user who can see the chain and knows p can check that C is well-formed.

Definition 5. *A provenance chain is* **weakly plausible** *for document D if it is well-formed and $hash(D)$ is the document hash value stored in the last record of the chain.*

In a later section, we prove that audits without replay/undo only guarantee that chains are weakly plausible. From a theoretical point of view, weak plausibility checks may seem too weak to be useful. In practice, though, weak plausibility checks suffice in many application domains, if we add just a tiny bit of application-specific knowledge to the checking procedure. For example, consider the electronic equivalent of branded merchandise (e.g., Prada handbags or Toyota auto parts) that should only be manufactured and distributed by approved organizations. In the first record for such a chain, U's public key certificate should be signed by an approved factory; in the second, it should be signed by an approved distributor. The auditor can use those domain-dependent checks to limit the signing authorities it trusts. The collision-resistance properties of cryptographic hashes make it essentially impossible for a retailer to substitute one record (e.g., an approved manufacturer) for another (a pirate) in a chain, without changing the checksum that was used to compute the following record's checksum.

Even without replay/undo, an auditor may be able to tell that the chain is inconsistent with the current contents of D. For example, suppose that a chain record says that all appendices to a report were deleted, and no subsequent record added any appendices. Then application-domain-dependent reasoning can allow an auditor to conclude that something is wrong if the document still contains appendices.

3.4 Fine-Grained Control Over Confidentiality

As mentioned earlier, the all-or-nothing approach to allowing auditors to view sensitive fields will be too coarse-grained for some applications. Sometimes it may be hard to foresee which fields may become sensitive over time, especially for a long-lived document that may cross boundaries between organizations. For example, the confidentiality needs for the testing of a particular National Geographic DNA sample may be met perfectly by a particular set of auditors and session keys, as long as the sample stays at its original processing location (the University of Arizona). However, a very small percentage of samples produces ambiguous or seemingly unlikely results (rare genotypes), and these are sent for additional rounds of testing at other labs. When a sample's chain is sent out to a lab in England, the details of previous testing should be eliminated to prevent bias in interpreting the results of the new rounds of tests.

To provide flexibility in such situations without a proliferation of broadcast encryption keys, we can use cryptographic commitments [Blu81] for subfield and field data that may eventually be deemed sensitive. With such a scheme, we can selectively omit plaintext data entirely when sending D's chain to a new organization, regardless of whether such a need was foreseen when setting up the session key(s) for D. The plaintext information can be restored to the chain if, for example, D later finds its way back to its original organization.

To achieve this level of control without a proliferation of encryption keys, we replace each potentially sensitive plaintext subfield s inside W by its commitment before computing the checksum for P:

$$comm(s) = hash(s, r_s),$$

where r_s is a sufficiently large random number.

During construction of the signature-based checksum, the provenance system uses these hashes instead of the actual data items. For example, the names of the tests performed in W can be replaced by commitments. When the University of Arizona sends the chain to an internal party trusted to view the plaintext version of W, both the commitments and the plaintext name of each unusual test s and r_s will be included as usual in the provenance record. When the University of Arizona sends the chain to a lab in England, Arizona can remove the plaintext for s and r_s and send only their commitments. Since the chain checksums were computed using the commitments rather than the plaintext data, the English lab can still verify the integrity of the chain. Access to sensitive values is prevented until the chain returns to the University of Arizona, which can reinstate the plaintext in the chain. If the English lab chooses to send out the sample for additional testing, it may choose to omit all the plaintext from all the Arizona records of the chain. This level of flexibility would be awkward to build into the provenance system using only session keys, but is easily accomplished with commitments. To verify that a document matches its reported history, a superauditor must include auditors with access to the sensitive values (e.g., one auditor each from the English and Arizona labs).

3.5 Spiral Provenance Chains

Provenance information tends to grow very fast, and can become several magnitudes larger than the document itself and require compaction [CJR08]. In this section, we show how chains can be compacted by removing irrelevant records, without recomputing signatures or compromising chain integrity checks. The removed records can be reinstated later on if desired.

The *integrity spiral*, a redundant, multiply-linked chaining mechanism, is conceptually similar to skip lists [Pug90]. The idea is to compute the checksum(s) of each provenance record by combining the hash of the current record with multiple previous checksums. More precisely, instead of storing a single checksum in each provenance record, we can store R separately signed checksums, where R is called the *dimension* of the spiral. The checksum of the first provenance record in a chain is computed as usual. For subsequent records, the checksum C_i stored in the i-th provenance record is computed as follows:

$$C_i = C_{i_1}, \cdots, C_{i_R},$$

where for $1 \leq j \leq R$,

$$C_{i_j} = S_{U_i}(hash(U_i, W_i, hash(D_i), public_i, I_i), i - f_j(i), C_{f_j(i)_j}).$$

The function $f_j(i)$ maps i to a positive integer less than i. The expression $i - f_j(i)$ tells which previous provenance record the checksum $C_{f_j(i)}$ was taken from.

To ensure that the chain is completely linked together, we require that $f_1(i) = i - 1$, for $i > 1$. The appropriate choice of the remaining f_j functions depends on the application domain. For example, one option is to use checksums from previous records at distance $1, 2, 4, \cdots 2^{R-1}$. Another option is to partition the chain by domains, and use the checksum from the last record of each partition. Whatever the choice, auditors need to know how to find the appropriate checksums to include when computing C_{i_j}; if necessary, $i - f_j(i)$ can be included explicitly in the provenance record to tell them where to find the checksum.

Advantages. An auditor can verify the integrity of the entire chain, or use the spiral to skip integrity checks for selected portions of the chain. For example, if an auditor does not want to check the integrity of records originating from her own domain, she could skip over most or all of them and verify the integrity of all other records. Or, if the chain is constructed so that all records of a particular type of event are linked by a given dimension of the spiral, the auditor can check the integrity of just the records of events that interest her. Of course, if there is an integrity violation in a record that the auditor does not check, then she will not realize

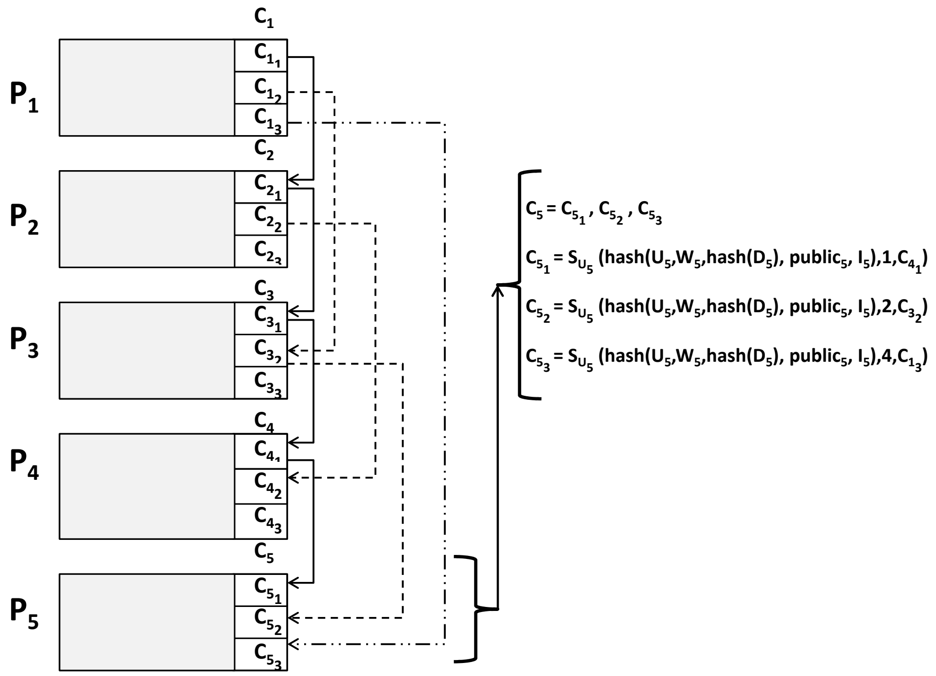
C_1
C_{1_1}
C_{1_2}
C_{1_3}
P_1
C_2
C_{2_1}
C_{2_2}
C_{2_3}
P_2
C_3
C_{3_1}
C_{3_2}
C_{3_3}
P_3
C_4
C_{4_1}
C_{4_2}
C_{4_3}
P_4
C_5
C_{5_1}
C_{5_2}
C_{5_3}
P_5
$C_5 = C_{5_1}, C_{5_2}, C_{5_3}$
$C_{5_1} = S_{U_5}$ (hash(U_5,W_5,hash(D_5), public$_5$, I_5),1,C_{4_1})
$C_{5_2} = S_{U_5}$ (hash(U_5,W_5,hash(D_5), public$_5$, I_5),2,C_{3_2})
$C_{5_3} = S_{U_5}$ (hash(U_5,W_5,hash(D_5), public$_5$, I_5),4,C_{1_3})

Figure 3: Example of a spiral provenance chain with $R = 3$, i.e., 3 checksums per provenance entry. For example P_5's checksum C_5 has 3 separate checksums in it. Each checksum is computed by concatenating the hash of the current content of P_5 with a checksum from a previous entry. In the example, C_{5_1}, C_{5_2}, and C_{5_3} are computed using C_{4_1}, C_{3_2}, and C_{1_3} respectively.

that the document's history is not plausible.

Integrity-preserving Summarization. A multiple-checksum spiral allows us to remove records from a provenance chain while still allowing integrity verification of the remaining records. If record j occurs after record i in the chain, and one or more of the checksums stored in record j is computed using a checksum C from record i, then we say that records i and j are *directly linked* by C. Checksum C from record i serves as a witness that record i preceded record j in the original chain. Even if we remove all the records between i and j from the chain, auditors will still be able to check the integrity of i and j and use C to verify that record i preceded record j in the original chain. This is consistent with I1, as the field $i - f_j(i)$ of the checksum tells auditors whether intermediate records have been skipped over. More generally, suppose that certain records in a chain are considered important according to domain-specific criteria. Consider the subchain containing just the important records. Let us add additional records to that subchain, until each record in the subchain is directly linked to its successor. The choice of which additional records to include can be guided by the records' brevity, importance, or other domain-dependent criteria. Then if we compact the original chain by removing all records not in the subchain, auditors can still verify the integrity and ordering of all records in the subchain and observe how many records have been omitted.

4 Proofs of Correctness

In this section, we prove that the non-spiral formulation of provenance chains satisfies the integrity, confidentiality, and privacy properties outlined in Section 2. We refer to an audit carried out by a superauditor as a *superaudit*. We defer consideration of the use of cryptographic commitments until the end of the section.

Theorem 1. *Suppose that a superaudit of document D with provenance chain P fails. Then P is not a plausible provenance chain for D.*

Theorem 2. *Suppose that an audit of document D with provenance chain P fails. Then P is not a weakly plausible provenance chain for D.*

Proof. We prove the theorem for a superaudit. The theorem for ordinary audits will follow from the proof of the theorem for superaudits.

The audit may have failed because the auditor could not parse one or more provenance records. If non-optional fields are missing or garbled, then the history is not plausible. The *public* field is optional; if it is missing or garbled, that cannot have caused the audit to fail. Instead, the auditor will have looked for an online source of information about the public key of the user. If W is not encrypted, then the auditor will ignore the contents of I; so in that case, the contents of I cannot have caused the audit to fail.

Recall that the cryptographic hash function $hash()$ used to hash documents is known to all auditors. The audit may have failed because an auditor detected that the contents of D do not match the value $hash(D)$ that should be stored in the final record of the chain $P = P_1, \ldots, P_n$ (attacks I6 and I7). In this case P is not a plausible history for D, because the contents of D (i.e., whatever aspects of D's internal contents and metadata that are being tracked) are not what P says they should be.

The audit may have failed because the chain is reversible or replayable, and a superauditor determined that the sequence of actions stored in P does not produce the contents of D (attacks I6 and I7). In this case P is not a plausible history for D, either.

The audit may have failed because a superauditor was supposed to replay or reverse the actions recorded in D's provenance records, but no auditor was able to decrypt the W field of some record. Recall that when such an audit may be needed, we assume that a superauditor will exist throughout the document's lifetime. Thus if D still exists, it should have a superauditor. We trust all auditors to behave correctly; thus if no auditor can decrypt I or W in some record, then the I or W field must be garbled or tampered. In that case, the history recorded in P is not plausible.

The audit may have failed because a superauditor was supposed to replay or reverse the actions recorded in D's provenance records, but the action recorded in the field of some record was not replayable or reversible. If such an audit may be needed, then all user actions are supposed to be replayable or reversible, and non-replayable or non-reversible actions are not permitted. Thus any history that involves such actions is not plausible.

Every provenance record includes the identity of the user who signed that record, in plaintext. Recall that we assume that auditors always have a trusted source of information about public keys of users mentioned in provenance records, and that all such sources agree with one another. Thus every auditor can determine which user is listed in each record. If the auditor's trusted sources do not know of any public key associated with that user, then no such user actually exists, and therefore P is not plausible. Otherwise, the user actually exists, and the auditor will be able to find the public key for that user, either from the record itself or from an external source. If a provenance record's *public* field contains anything other than a public key certificate signed by an authority that the auditor trusts, then the auditor will ignore the *public* field, so this cannot have caused a failure. Since we assume that all authorities agree on what the public key is for a user, the audit cannot have failed due to disagreements between these authorities.

The only other possible cause for audit failure is a problem with a checksum in some record of the provenance chain. Let P_i be the first record in the chain where such a problem occurs. Let U, C, I, and W be the user identity, checksum, keying material, and action description stored in P_i, respectively; every user can inspect the chain and see these values. Let *public* be the public key of U obtained by the auditor. Let C' be the checksum stored in the previous record in the chain, which has already passed the auditor's inspection. Recall that to create C, U is supposed to sign $(hash(U, W, hash(D_i), public, I) \mid C')$, where $hash(D_i)$ is a hash of the then-current version of D. As $hash(D_i)$ is given elsewhere in P_i, the auditor can form the same tuple $(hash(U, W, hash(D_i), public, I) \mid C')$, and use *public* and C to see if that tuple was actually signed by U.

If the signature fails this check, then the audit will fail. One possible cause of the failure is that the private key corresponding to *public* was not used to sign C. Because we assume that private keys are never compromised, this failure means that U did not actually sign C. Thus P does not give a plausible history for D, because either U did not carry out W as stated, or else some user (possibly even U) has tampered with the contents of P_i.

Otherwise, the private key corresponding to *public* was used to sign the checksum (which, due to our assumption that private keys are never compromised, means that U really did sign C (I5)), but the tuple U signed was not $(hash(U, W, hash(D_i), public, I) \mid C')$. If this check fails, then P_i cannot plausibly have been produced by U performing W on D_{i-1}, and the theorem follows.

It is instructive to analyze the possible causes of this latter failure. One possibility is that U did not form P_i properly, in which case we say that U tampered with P_i. A second possibility is that U did form P_i properly, but someone tampered with P_i afterward, making the chain implausible. A third possibility is that P_i was formed properly and was not tampered with; in that case, the checksum C' that the auditor found in the previous record must be different from the checksum used to compute C. Since the audit succeeded up through the previous record P_{i-1}, and U formed P_i properly, C' must not have been the checksum in the last record of the chain at the time U created P_i. Thus someone subsequently tampered with the part of the chain just before P_i. The tampering must involve inserting a different record in front of P_i (attacks I2 and I3), removing the record right in front of P_i (attacks I1 and I4), or both (e.g., the user who created the previous record has altered its contents and re-signed it; or P_i has been plucked from its chain and inserted into a different one). These are the only possibilities, because any other kind of tampering would have been detected earlier in the chain. $\square$

Theorem 3. *Suppose that provenance chain P is not a plausible history for document D. Then any superaudit of D and P will fail.*

Theorem 4. *Suppose that provenance chain P is not a weakly plausible history for document D. Then any audit of D and P will fail.*

Proof. Again, we prove the theorem for superaudits. The theorem for ordinary audits will follow from the proof of the theorem for superaudits.

Let the chain be $P = P_1 \cdots P_n$. Since P is not a plausible history for D, the sequence of users named in P cannot possibly have carried out the actions listed in P and thereby produced D. Let $P_i = \langle U, W, hash(D_i), public, I, C\rangle$ be the first record in the chain where user U cannot possibly have applied permitted action W to document D_{i-1} and correctly generated P_i to record the action. If no such record exists, then it must be the case that P is empty. If P is empty, then any auditor will detect that D's chain is empty, and the audit will fail. In the remainder of the proof, we assume that such a record P_i exists.

There are several possible reasons why a user U cannot possibly have applied permitted action W to document D_{i-1} and correctly recorded the activity in P_i:

1. No such user U can have existed.
2. Action W would not have been permitted.
3. If user U had applied permitted action W to document D_{i-1}, then the correct description of the activity would be different from P_i, for one of the following reasons.
   - The *public* field would be different.
   - The I field would be different.
   - The $hash(D_i)$ field would be different.
   - The checksum field C would be different.

In the remainder of the proof, we argue that if any one of these conditions holds, then the superaudit will detect it.

No such user can have existed. Recall that our definition of plausibility does not rely on any background knowledge regarding the actual set of users on any system. Thus there are only two reasons why no such user can have existed. The first possibility is that U is syntactically malformed. In this case, any auditor will detect the problem, and the audit will fail. The second possibility is that U is syntactically well-formed, but does not satisfy our assumption that every user has a public key that an auditor can obtain from a trusted source, and all such sources agree on what the keys are. Thus if an auditor cannot find a public key for U from a trusted source, the audit will fail.

In the remainder of the proof, we assume that U is a plausible user.

The public field should be different. Recall that the *public* field is optional, and we have not given a precise definition of what it means for it to be "correct." One option is to view the field as a hint whose contents are irrelevant for the outcome of the audit. In this case, the plausibility of a provenance record does not depend on the contents of its *public* field. If the field is empty or garbled, or the auditor does not trust the authority who signed the certificate there, then the auditor simply looks elsewhere to find the public key of U. In practice, of course, it can be very convenient to have a public key certificate present in the provenance record. Thus the other option is to say that the contents of the field are plausible if they are signed by a trusted authority. Under this interpretation, the auditor will have to check the signature on any certificate present in the *public* field, and the audit should fail if the signature is incorrect or the authority is not trusted.

In the remainder of the proof, we assume that U is a plausible user with a known public key.

The I field should be different. Recall that W is encrypted with a session key, as described in Section 3. Only auditors who can obtain that session key can decrypt W, producing w, and determine whether w is a permitted action. Recall that we assume that U trusted at least one auditor, and that auditors (or, more realistically, auditor roles) remain available forever. Since we are performing a superaudit, there should be at least one auditor who U trusted and who is participating in the audit. Further, I should not be empty, and the audit will fail if I is empty.

There are several possible formats for a non-empty I. One option is for I to contain copies of a session key, with each copy encrypted with the public key of an auditor that U trusted. We do not have any background knowledge about which auditors U trusted, so the contents of I are plausible as long as the session key is encrypted with the public key of at least one auditor. We are performing a superaudit, which implies that we have an auditor on hand who should be able to decrypt the session key in I, and determine whether it can be used to decrypt W. If the session key was not properly placed in I, or if W is opaque and the session key does not decrypt it, then the audit will detect this and will fail. It is possible that I will contain duplicates, or additional material that no auditor can decrypt to produce the session key. We view such a provenance record as plausible.[1]

Recall that session keys may also be made available in I using broadcast encryption, threshold cryptography, and secret sharing. For each of these cases, the format of I still involves a session key (or a share of a session key), encrypted with a public key associated with one or more auditors. We do not have any background knowledge regarding the exact separation of duty requirement, broadcast encryption scheme, or secret sharing scheme that U intended to enforce, so all of the options are equally plausible to us. Thus the argument used in the previous paragraph still applies: as long as the auditors can obtain the session key needed to decrypt W, the history is plausible. As in the previous paragraph, sometimes the material in I may be rather odd, but we still consider it plausible. For example, I might contain the session key encrypted with the public key associated with the root of the broadcast encryption tree, and also include the session key encrypted with the keys for several other nodes in the tree. These extra copies of the session key are not needed, but we consider them plausible. Similarly, for the reasons discussed in the previous footnote, we consider a history plausible as long as an auditor can successfully extract a session key that allows her to carry out her tasks, regardless of what other extraneous material may be included in I.

In the remainder of the proof, we assume that the auditor has been able to extract a session key from I, if one is needed.

The action is not permitted. Recall that our interpretation of "plausible" in this paper does not involve any domain knowledge or temporal reasoning; we assume that if a particular user signed a provenance record, then he or she was able to carry out the actions given in the record. We make only the very weak assumption that auditors who are authorized to view the contents of an action field are familiar with the vocabulary that can be used to describe actions. If actions must be replayable or reversible, then the vocabulary should be

[1]If we adopt a stricter interpretation – I should not contain garbage – then audits will be unpleasantly expensive. To ensure that I contains no garbage, we will have to decrypt every entry in I during every audit. Thus every audit will require the participation of every auditor U trusted, which is undesirable. A side effect of this approach is that tampering with the contents of I will not be detected until no auditor can decrypt a session key.

appropriately restricted.

Since this is a superaudit and the audit has not yet failed, one of the auditors must have been able to obtain a session key from I. This key can be used to decrypt W, producing the plaintext action description w. If w was not permitted, or if the session key obtained from I is not the correct session key, then the superauditor will detect this, and the audit will fail.

The checksum should be different. U should have created C by signing the value $hash(U, W, hash(D_i), public, I) \mid C'$ with the appropriate private key, where C' is the checksum from the previous record in the chain. The auditor already knows U's public key, so the auditor can check the signature on C. The auditor also knows the function $hash$, and from parsing P_i, the auditor has already obtained the values for U, W, $hash(D_i)$, $public$, and I. The auditor has already checked the previous record in the chain, which contained the checksum C'. Thus the auditor can check whether U actually signed $hash(U, W, hash(D_i), public, I) \mid C'$, and the audit will fail if U signed the wrong tuple or did not sign correctly.

The $hash(D_i)$ field should be different. If P_i is the last record in the chain, then any auditor can check whether $hash(D)$ is the hash value recorded in the record. If they are different, then the audit will fail.

If P_i is not the last record in the chain, and actions are replayable, then the superauditor has already determined that the previous actions in the chain could have produced a document D_{i-1} that hashed to the value $hash(D_{i-1})$ recorded in the previous provenance record. The superauditor can decode the action w_i and simulate applying it to D_{i-1}. If the resulting document does not hash to $hash(D_i)$, then the audit will fail. For reversible actions, the auditor can first check weak plausibility of the chain, and then check plausibility in a separate pass that starts at the end of the chain and moves backward. The inductive argument is otherwise the same as that for replayable actions. □

We highlight several important issues in the remainder of this section.

Proposition 1. *Users cannot repudiate provenance records that they create (I5). If a chain is weakly plausible, then users cannot argue that the previous records in the chain were different at the time they added their own record.*

Proof. Recall that we assume that users' private keys are never compromised. Since we use a non-repudiable digital signature on the contents of each record, users cannot claim that they did not sign a record that bears their signature. Further, the collision-resistance property of cryptographic hashes means that users cannot claim that the next-to-last record in the chain had different contents at the time they added the last record. Extending this argument further back in the chain, if the chain is weakly plausible, then no user can claim that the earlier records in the chain were different at the time they added their own record. □

Proposition 2. *Auditors can only decode record details for which they are authorized.*

Proof. We argue the case where a single session key k_i is used to encrypt all sensitive details in the record, and the key itself is protected using broadcast encryption; the argument is similar if multiple session keys, a single shared key, threshold cryptography, or secret shares are used for this purpose.

A session key k for W is accessible only to principals that can retrieve it by decrypting at least one item in set I. If a principal can decrypt one of these items, then it possesses a private key in the broadcast encryption tree, and must therefore be an auditor represented by a leaf in the subtree rooted at the private key in question. Thus the principal is an auditor who should be allowed to obtain the session key, and therefore should be allowed to see all data encrypted with it, including W. □

The previous theorems do not consider the possibility that cryptographic commitments are being used for potentially sensitive plaintext, and the plaintext is subsequently removed. In such a situation, auditors should check whether the plaintext matches its commitment, whenever the plaintext is available. Otherwise, the audit will not detect tampering of the plaintext.

Theorem 5. *If we remove the plaintext for one or more committed fields from a weakly plausible chain, then the chain is still weakly plausible. If a chain is not weakly plausible, but all cryptographic commitments in the chain do match their plaintext, then the chain will still not be weakly plausible if we remove the plaintext from one or more committed fields.*

Proof. The checksum component of each record is computed using the cryptographic commitments, not the sensitive data item's plaintext. Thus the auditor computes the same set of checksums, whether or not the plaintext for the sensitive fields is available. The only difference in the audits is that the auditor cannot check whether the plaintext matches its commitment, once the plaintext is removed. □

5 Co-provenance

As discussed earlier, different applications of provenance have different threat models. For some applications, it may be important to be able to verify that the history recorded in a provenance chain its *actual* history, not just a plausible history. For example, consider an insurance claim that has been processed and denied. The claimant can always appeal the decision, but appeals are carefully scrutinized; so if the claim is fraudulent, an adversary might prefer to roll back the denial and reroute the claim to a different adjuster who is more likely to accept the claim. Or an adversary can replace the provenance chain for a document by a new chain. If all users named in the new chain collude with the adversary, they can create a plausible but false history of the document. An auditor cannot distinguish this history from the actual history.

For such applications, we can greatly increase the chance that a plausible-but-not-actual provenance chain will be detected, by including copies of some of its records in the chains associated with other documents. We call this approach *co-provenance.* As each document's co-provenance chain has records from other documents, chain truncation attacks become more difficult to launch. To be able to roll back the history of document D and create a new plausible history for it, an adversary must find enough colluders to roll back and rewrite the chains of *all* the other documents that should contain records from the old and new chains. In practice, this will often pose an insurmountable challenge.

The ability to detect plausible-but-not-actual provenance chains comes at a price. First, a document modification may require provenance records to be affixed to multiple chains, which will be slower. Second, each record must identify the document with which it is associated, so each record is larger. Third, each chain contains more records, so verifying its integrity takes longer. Fourth, those additional chains must be available when new records are written and also at audit time, so chains cannot be deleted at will. The auditor does not need to scrutinize more than one chain if a plausible history suffices for a particular audit of document D. Otherwise, the auditor must determine the set of additional chains for D, fetch them, find the records that should have been included in D's chain and vice versa, ensure that they were included at the correct points in each chain, and essentially validate every additional chain. The auditor must also decide whether the ordering of records in the set of additional chains is consistent; this can be tricky when multiple documents were modified at almost the same time. Finally, in situations where it is vital that the chain represent actual history, an auditor might want to check the original chain, the additional chains, the additional chains for those additional chains, and so on. Thus the extent of the audit procedure, and its cost, should be dictated by the application. The application must also determine whether tampering in any chain, no matter how remote from the original document being audited, means that the entire audit fails. For example, an extensive audit could detect that an additional chain was tampered with before the document being audited was created.

The application will also dictate the best approach for determining which additional provenance chains a new record should be appended to. The writer of a new provenance record must know what additional chains to affix it to, and this mapping should not depend on the particular contents of the record (e.g., the action it records). Otherwise an adversary could devise a new plausible history that had completely different additional chains, and auditors would not know to check the original set of additional chains. One approach is for the application to divide documents into groups, such that all the documents in a group have overlapping chains. For example, a Human Resources department might place all the employee records for employees hired in a particular month in the same group.

Poor choices for any of these application-dependent parameters can negate the advantages of using co-provenance. For example, if only one employee is hired in a particular month, then co-provenance reduces to ordinary provenance for that group. Or if only one employee in a particular group is still working for the company, then the other employee records in the group will be completely static. The remaining employee's employment history will continue to evolve, but the others will not. Thus for the remaining employee, it will be no harder than with ordinary provenance to roll back history and replace it with a different plausible history. The auditor will not be able to detect that tampering.

The preceding discussion assumes that an entire provenance record will be affixed to multiple chains. In practice, it suffices for the checksums of the chains to be intertwined. Under this approach, a co-provenance chain differs from an ordinary provenance chain only in how its checksums are computed. Co-provenance checksums can be defined in many different ways; for the case of non-overlapping groups, one option is:

$$C = S_U(hash(U, W, hash(D), public, I)|C^G),$$

where C^G denotes the concatenation of the most recent checksums of all the documents in the group G that D belongs to. The provenance chains of the other documents in G will contain checksums from D's own chain. For this approach to work, auditors must be able to determine *which* of the checksums in the chains in G should have been included in the checksum for another chain. One option is to append explicit marker records to the other chains. For example, when computing the checksum for the ith record for document D, we can append a record to D''s chain that states that the checksum in the previous provenance record for D' was used in computing the checksum for the ith record for D, for each document D' in G. To ensure their integrity, marker records must be included in the hash chain just like other records. Thus the marker records act as a stripped-down version of the new provenance record. Of course, appending marker records or even entire provenance records to foreign chains does not completely solve the auditor's problem of determining whether the correct records were included in the chain, and whether a set of chains provides a mutually consistent ordering of events. In other words, co-provenance increases the chances that the auditor will be able to determine that some plausible histories are not the actual history, but does not guarantee that the auditor can recognize all non-actual histories. Further, network and computational delays in chain construction could cause the audit of an actual history to fail, due to apparently inconsistent record ordering across chains.

Instead of including marker records or entire provenance records in additional chains, another option is to timestamp records and keep clocks roughly synchronized. But clock synchronization is not always an option, even in these days of GPS. Even with synchronized clocks, the auditor can only approximate the interval from which a timestamp should have been taken, due to computational and networking delays. Thus the decision as to whether a chain passes its audit will become very application-dependent under this approach.

6 Empirical Evaluation

6.1 Implementation Considerations

Different avenues are available for implementing secure provenance functionality: in the operating system kernel, at the file system layer, or in the application realm.

Kernel Layer. In this implementation approach, provenance record functions are handled by trapping kernel system calls, similar to the approach taken in the Provenance-aware Storage System (PASS) [MRHBS06]. The main advantage of this approach is its transparency to user level applications and the file system layer. One drawback is that the logic and higher level data management semantics are not naturally propagated to the kernel, thus limiting the types of provenance-related inferences that can be made. Another drawback is the limited portability of this approach, as any new deployment platform will require porting effort.

File System Layer. The file system can be made provenance-aware and augmented to transparently collect and secure provenance information. Similarly to the kernel layer implementation, one of the main advantages of such an approach is transparency. However, persisting provenance state transparently inside the file system layer will reduce the portability of the provenance assurances, e.g., when provenance-augmented files traverse non-compliant environments. Another drawback is that higher level application semantics are not naturally propagated to the file system, thus limiting the types of provenance-related inferences that can be made.

Application Layer. In this approach, the provenance mechanisms are offered through user-level libraries. This can maintain the transparency of the previous approaches while also allowing for a high degree of portability, i.e., by being independent of kernel and file system layer instances. The provenance libraries can be layered on top of any file system, making rapid prototyping and deployment very easy. Moreover, through dynamic linking and by maintaining a compatible interface, existing user applications do *not* need to be recompiled to become provenance-aware.

The application-layer approach can be used to track provenance at a relatively low level, such as file system read and write calls, or at a high level, such as workflow actions. The advantage of tracking higher-level actions is that the resulting records naturally incorporate application-level semantics, which may make them easier to query. The disadvantage is that high-level tracking is easier to subvert than lower-level tracking, because more of the activity in the system is outside the view of the provenance collection system.

6.2 The Sprov Library

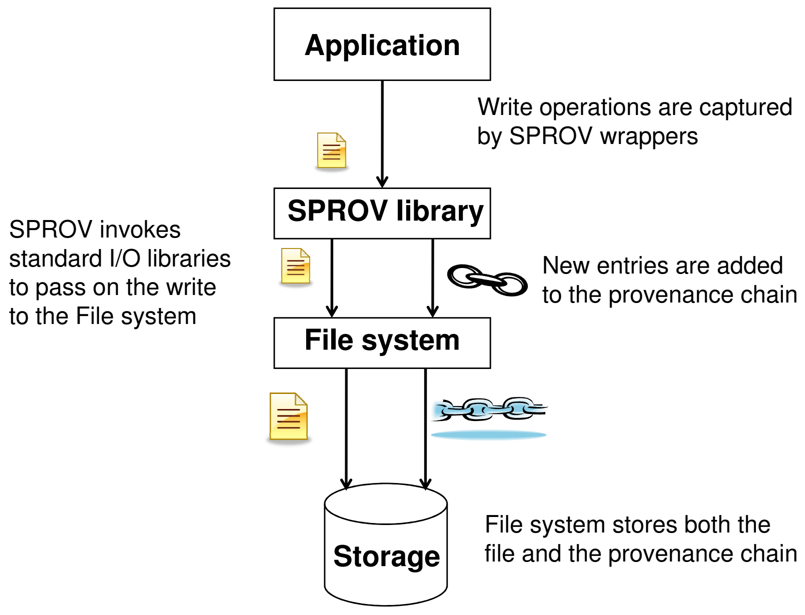
Application
Write operations are captured by SPROV wrappers
SPROV library
SPROV invokes standard I/O libraries to pass on the write to the File system
New entries are added to the provenance chain
File system
Storage
File system stores both the file and the provenance chain

Figure 4: The Sprov System Architecture.

We implemented a prototype of the secure provenance primitives as an application layer C library named Sprov, consisting of wrapper functions for the standard file I/O library `stdio.h`. The resulting library is fully compatible with `stdio` functionality, in addition to transparently handling provenance assurances. We used the basic model introduced in Sections 3.2 and 3.3 in this prototype.

Sprov tracks changes in documents at the file level. In building Sprov, our goal was to show that provenance collection can be affordable even for the most frequently-performed operations on files. To this end, in our experimental evaluation, Sprov is tracking the following file system calls:

fopen and fclose No changes to the content of the provenance chain.

fread No impact on the provenance chain if reads are not being tracked. Otherwise, a new provenance record is appended to the chain, recording the location and amount of data read.

fwrite A new provenance record is appended to the chain, recording the data being written and the write location.

We also designed and built several utilities to facilitate provenance collection and transfer, and to allow us to track certain operations in a more sophisticated manner would otherwise be possible.

plogin When a user logs into her system, the `plogin` utility is invoked, which initializes the session keys and loads the user's preferences and list of trusted auditors. From that point onward, provenance will be tracked.

pcopy The `pcopy` utility has the same effect as the UNIX *cp* command (i.e., makes a copy of a file), but it also makes a copy of the provenance chain of the source file, and then records information about the

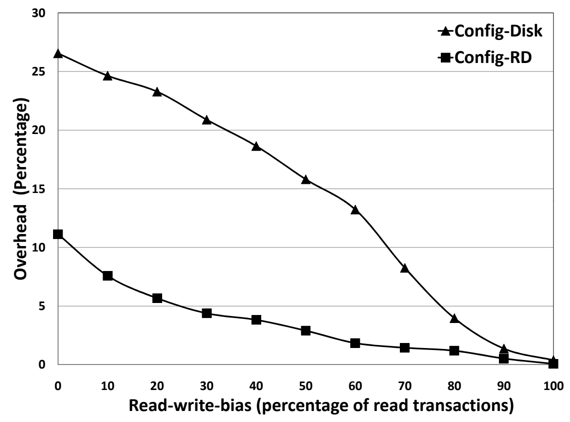

Figure 5: Run-time overheads of secure provenance with Postmark. The overheads are shown from 0% read bias (100% write transactions) to 100% read bias (no write transactions).

copy operation into a new provenance record appended to the provenance chain of the destination file. The contents and provenance chain for the original document are not altered. In contrast, a call to `cp` in our system will create a new provenance chain whose only record shows that the entire contents of the file were written.

pdelete The `pdelete` utility is used to delete a file – in this case, the delete operation is logged in a new provenance record, which is appended to the provenance chain of the deleted file. The user also has to specify the expiration time for the provenance chain, after which it will be removed.

pmonitor The `pmonitor` daemon periodically scans for and removes expired provenance chains.

Our experiments track changes to the internal contents of files only. In a deployment of Sprov, one might choose to track additional file metadata, such as the file's computer of residence, creation time, owner, permissions, last write time, or path. As discussed earlier, if the audit is to check constraints related to these new fields, then additional operations may need to be tracked as well (e.g., `chown`, `chmod`, `chgrp`). If the provenance system needs to support other application-specific constraints, such as providing special support for provenance chains when files are transferred, then additional operations (e.g., `ftp`, email attachments) will need to be tracked. To avoid duplication of effort, the provenance system can coordinate the support it provides for higher-level operations and the lower-level operations they call, as we did for `cp` and `rm`.

In Sprov, a *session* is defined as all the operations performed by a user on a file between file open and close. When a file is opened in write or append mode, Sprov initiates a new record in the provenance chain of the file. Information about the user (user ID, group ID, and login name), application (process ID), and environment (host name, timestamp) are collected. During write operations, Sprov gathers information about the writes. In this phase, we log all bytes written to the file before it is closed. Logging uses a reversible representation of document modifications; as discussed in previous sections, this allows strong verification of the relationship between current document contents and the document's provenance chain. The provenance chain can be used as a rollback log, which can form the basis of a versioning file system. Further consideration of this enhancement is beyond the scope of the current paper.

At file close, the session ends. A new record is written in the provenance chain for the changes made during this session. At this point, the cryptography associated with the chain integrity constructs is executed, as described in Section 3. We implement the primitives in these constructs (SHA-1 hashes, 1024-bit DSA signatures, 128-bit AES encryption – with random padding for semantic security) using the `openssl` library [The03a]. The provenance chain is stored in a separate meta-file for portability.

6.3 Experiments

The experimental setup for the performance evaluation employed x86 Pentium 4 3.4GHz hardware with 2GB of RAM, running Linux (Suse) at kernel version 2.6.11. In this configuration, each 1024-bit DSA signature

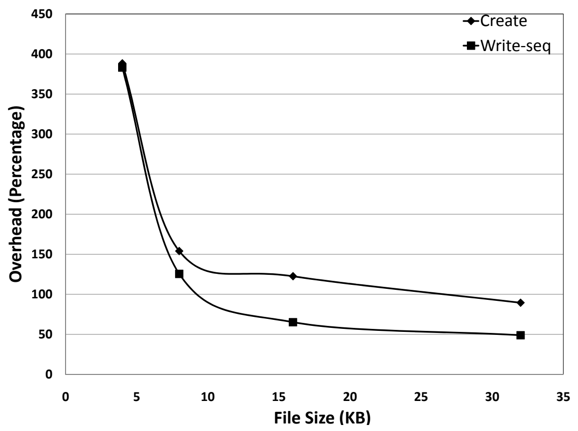

Figure 6: Small file system microbenchmark create and write performance for 2500 files (*Config-Disk*).

took 1.5ms to compute. The testing environment included a mix of four of the following drive types: Seagate Barracuda 7200.11 SATA 3Gb/s 1TB, 7200 RPM, 105MB/s sustained data rate, 4.16ms average seek latency and 32MB cache, and Western Digital Caviar SE16 3Gb/s, 320GB, 7200 RPM, 122MB/s sustained data rate, 4.2ms average latency and 16MB onboard cache. The experiments involving read operations were performed on a machine with an Intel Core Duo 2.2 GHz CPU, 2GB RAM, and a Hitachi Travelstar 5k160 SATA disk drive with the following specifications: 1.5 Gb/s peak transfer rate, 160 GB capacity, 5400 RPM, 150 MB/s sustained transfer rate, 5.5ms average seek latency, and 8MB onboard cache.

We conducted our experiments using multiple benchmarks, in a quest to match several different deployment settings of relevance. In each case, we compared the execution times for the baseline unmodified benchmark (with no provenance collection at all), with a run with secure provenance enabled. We measured performance with (i) Postmark [Kat97] – a standard benchmark for file system performance evaluation; (ii) the Small and Large file microbenchmarks that have been used to evaluate the performance of the PASS provenance collection system [MRHBS06, SSB^{+}95]; and (iii) a custom transaction-level benchmark meant to test the performance in live file systems with file sizes distributed realistically [ABDL07, DB99], and real-life file system workloads [ELMS03, LPGM08, RLA00]. Except where otherwise noted, each experiment is tracking write operations and is not tracking reads.

We also evaluated two different configurations for storing the provenance chain. In the first configuration, provenance chains were recorded on disk (**Config-Disk**), while in the second one, provenance chains were stored in a RAM disk, with a `pmonitor` chron daemon periodically flushing the chain to disk (**Config-RD**).

6.3.1 Postmark Benchmark

We measured the execution time of the Postmark benchmark with and without the Sprov library. A data set containing 20,000 Postmark-generated binary files with sizes ranging from 8KB to 64KB was subjected to Postmark workloads of 20,000 transactions. Each transaction set was a mixture of writes and reads of sizes varying between 8KB and 64KB. We sampled the performance overhead under different write loads by varying the read-write bias from 0% to 100% in 10% increments (i.e., the percentage of write transactions was varied from 100 to 0%). The overheads are illustrated in Figure 5 for both *Config-Disk* and *Config-RD*. The overheads range from 0.5% to 11% for *Config-RD*, and 0.5% to 26.5% for the *Config-Disk* configuration. As we increase the read bias (i.e. reduce the percentage of write transactions), the overhead falls considerably. At 50% read bias (i.e. equal number of read and write transactions), the overhead is 15.8% for *Config-Disk*, and 2.9% for *Config-RD*. At 90% read-bias (i.e., only 10% write transactions), the overhead goes down to 1.4% (0.5% with *Config-RD*).

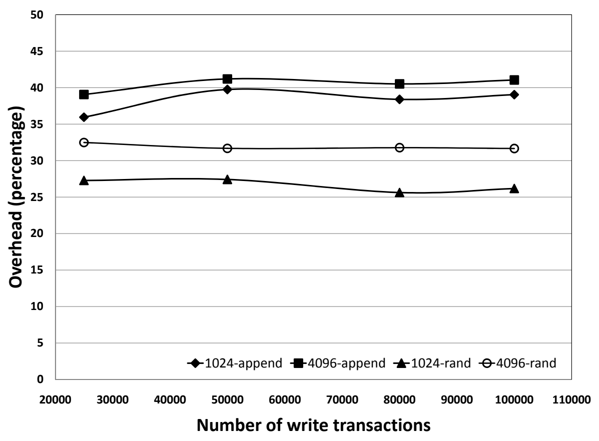

Figure 7: Overhead for different numbers of write transactions and data sizes, at 100% write load, without a RAM disk (*Config-Disk*).

6.3.2 Small and Large File Microbenchmarks

The small and large file microbenchmarks [SSB+95] have been used in the evaluation of PASS. The small file microbenchmark creates, writes to and then deletes 2500 files of sizes 4KB, 8KB, 16KB, or 32KB. We benchmarked the overhead for file creation as well as synchronous writes. The results for *Config-Disk* are displayed in Figure 6.

An interesting effect can be observed. Similar to the experiments with PASS, the overhead percentage is quite high for small files and decreases rapidly with increasing file sizes. This effect arises from disk caching. Specifically, for very small file size accesses – which go straight to the disk's onboard memory cache – the main overhead culprit (crypto signatures) dominates. As file sizes increase, additional real disk seeks are incurred in both cases and start to even out the execution times. Eventually, the overhead stabilizes at approximately 50% for 32KB files, which suggests that roughly 1-3 seek times are paid per file and the secure case adds the equivalent of another seek time (the crypto signature).

	no prov	Sprov (Config-Disk)	% Overhead	Sprov (Config-RD)	% Overhead
Seq-write	13.084	13.328	1.87%	13.308	1.71%
Rand-write	15.211	15.390	1.18%	15.285	0.48%

Table 1: Run times (sec) and overhead for sequential and random writes under the large file microbenchmark, with and without a RAM disk.

The small file microbenchmark only measures the effect of writes to many small files [MRHBS06]. Often, writes to large files can provide more representative estimates of typical overheads in file systems. Thus, next we deployed the Large file benchmark as described in [SSB+95]. We performed the sequential-write and random-write operations of the benchmark. Both unmodified and provenance-enhanced versions of the benchmark were run. This time, the disk write-caches were turned off to eliminate unwanted disk-specific caching effects, namely, the tendency for small benchmark runs to exploit the onboard disk cache without incurring the actual disk activity that accompanies a heavier workload.

The benchmark consists of creating a 100 MB file by writing to it sequentially in 256KB chunks, followed by writing 100 MB of data in 256KB units written in random locations of the file. The overheads for sequential and random writes are presented in Table 1. For both *Config-Disk* and *Config-RD*, the overheads (1.87% for sequential writes, 1.18% for random writes) are lower than those reported in [MRHBS06] (15.47% for sequential writes, 13.58% for random writes). Of course, Sprov and PASS take different approaches to provenance

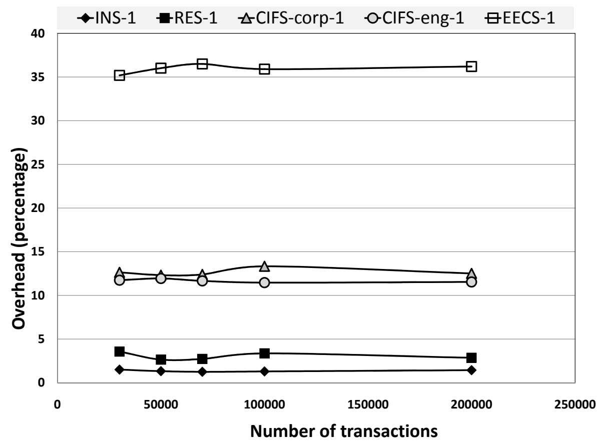

Figure 8: Overhead for five different workloads without a RAM disk (*Config-Disk*).

collection – PASS tracks the information flow and causal relationships while Sprov tracks records all the data written to the file in its provenance chain.

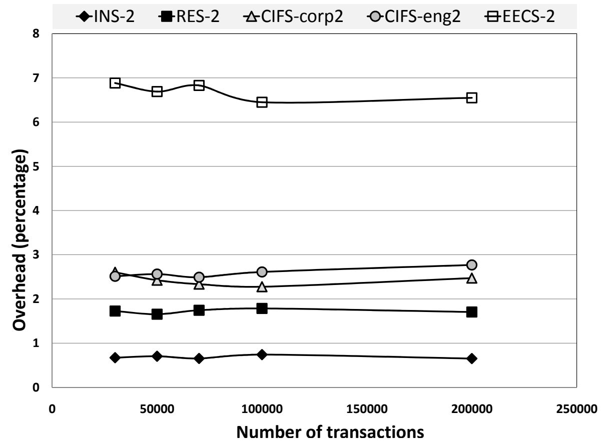

Figure 9: Overhead for five different workloads (*Config-RD*).

6.3.3 Hybrid Workload Benchmark

Benchmarks like Postmark are useful due to their standardized nature and the ease of replicating experiments in new settings. We also evaluated our overheads in a more realistic scenario, involving practical, documented workloads and file system layouts. We constructed a layout as described by Douceur et al. [DB99], which showed that file sizes can be modeled using a log-normal distribution. We used the parameters $\mu^e = 8.46$, $\sigma^e = 2.4$ to generate a distribution of 20,000 files, with a median file size of 4KB, and mean file size of 80KB, along with a small number of files with sizes exceeding 1GB to account for large data objects, as suggested in [ABDL07, DB99].

Our first workload on this dataset involved a fixed number of write transactions. Under the *Config-Disk* setting, we performed 25K, 50K, 80K, and 100K write transactions. Between each pair of experimental runs, we recreated the dataset, cold-booted the system, and flushed file system buffers to avoid variations caused by OS or disk caching. In each transaction, a file was opened at random, and a fixed amount of data (1KB and

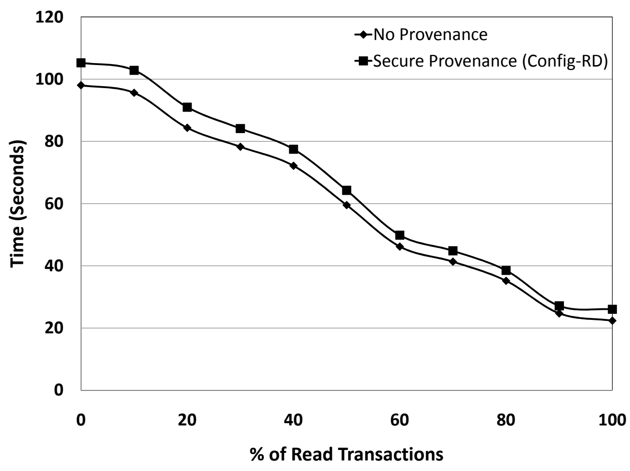

Figure 10: Run time of Postmark with no provenance and secure provenance for both reads and writes (*Config-RD* setting).

4KB) was written into it. We measured the overhead for both appends and random writes. These are shown in Figure 7. Constant overheads can be observed for each of the four configurations, with append situated between 32% and 42%, and random writes between 26% and 33%.

Next, we modeled the percentage of write and read transactions according to the data in [ELMS03, LPGM08, RLA00], which suggest this varies from 1.1% to 82.3%. To this end, we deployed information about workload behavior and used parameters for the instructional (INS), research (RES) [RLA00], a campus home directory (EECS) [ELMS03], and CIFS corporate and engineering workloads (CIFS-corp, CIFS-eng) [LPGM08]. The RES and INS workloads are read-intensive, with the percentage of write transactions less than 10%. The CIFS workloads are less read-intensive, with the read-write ratio being 2 : 1. The EECS workload has a very high write load, with more than 80% write transactions. The results are shown in Figures 8 and 9, with and without a RAM disk.

As these experiments are not tracking reads, the read-intensive workloads RES and INS show less than 5% overhead without a RAM disk, and less than 2% with a RAM disk. For write-intensive workloads, the overheads are higher, but still under 14% for the CIFS workloads (*Config-Disk*), and less than 36% for EECS (*Config-Disk*). With a RAM disk, the overheads go down to less than 3% for CIFS and around 6.5% for EECS.

Impact of recording read information. The previous experiments recorded provenance for write operations only. To evaluate the overheads imposed by recording both file reads and file writes in the provenance chain, we repeated the Postmark benchmark with a RAM disk. A data set containing 20,000 Postmark-generated binary files with sizes ranging from 8KB to 64KB was subjected to Postmark workloads of 20,000 transactions. Each transaction set was a mixture of writes and reads of sizes varying between 8KB and 64KB. We sampled the performance overhead under different write loads by varying the read-write bias from 0% to 100% in 10% increments (i.e., the percentage of write transactions was varied from 100 to 0%). For each file read request, we recorded the file offset and the size of the read request, and added the new provenance record to the provenance chain of the file. We did not record the actual bytes read. Under this approach, provenance records corresponding to a read request are much shorter than those corresponding to file writes.

Figure 10 shows the absolute run times in seconds for Postmark with no provenance and with secure provenance for both reads and writes (Sprov (Config-RD)). As shown in Figure 10, the absolute overhead for 0% reads (i.e., 100% writes) is higher than that for 100% reads (i.e. 0%) writes, because more information is captured for writes than for reads. For 100% reads, the absolute overhead for recording secure provenance is approximately half of that for 0% reads.

Figure 11 shows the relative overhead for secure provenance for reads, compared to the no provenance case. The overhead is approximately 8% for 0-80% writes. As Postmark takes almost one-fifth the time to do

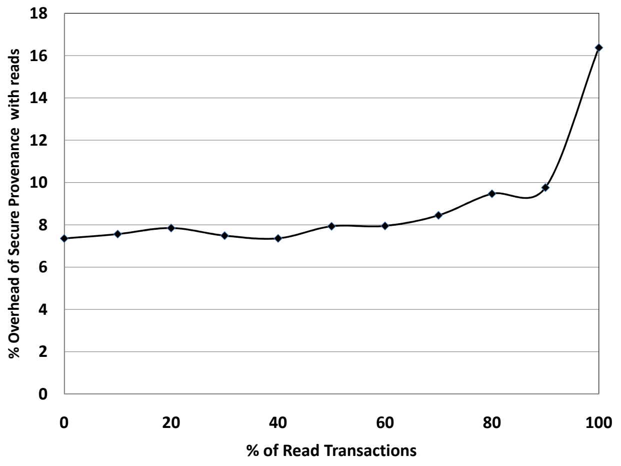

Figure 11: Relative overhead of secure provenance with Postmark (Config-RD)

100% reads as it takes to do 100% writes, the overhead for the 100% write scenario is higher even though the absolute overhead is very small.

We also evaluated overheads of collecting both read and write provenance with the real-life workloads CIFS-Corp, CIFS-eng, and EECS, under the Config-RD setting. The relative overheads range from 12-16%, as shown in Figure 12. These experiments show that limited information about file reads can be tracked with low impact on system performance.

Summary. Sprov facilitates collection of provenance with integrity and confidentiality assurances, while incurring only modest overhead. Benchmarks show that with the *Config-RD* setting, use of Sprov incurs an overhead less than 3% to record provenance information about write operations in a variety of realistic workloads.

7 Related Work

Previous research from the file system community has investigated the collection of provenance information at the operating system layer. The Provenance-aware Storage System (PASS) [BSS08, MRHBS06, MRBH$^+$09] takes this approach using a modified Linux kernel. PASS does not actually record the data written to files, but it does collect elaborate information flow and workflow descriptions at the OS level, for both reads and writes. PASS can also be integrated with network storage and applications such as web-browsers, to allow collection, querying, and storage of provenance across multiple layers [MRBH$^+$09]. Compared to the Sprov approach of collecting provenance information at the application layer, collection at the operating system layer has the advantage of being harder to circumvent and the disadvantages of being more expensive and harder to deploy. Our specific techniques for securing provenance chains can be used to augment PASS or any other such system, to provide security assurances at minimal cost.

The database community has explored a variety of aspects of provenance, including the notions of why-provenance and where-provenance and how to support provenance in database records and streams (e.g., [BCC06, BCCV06, BKT01, BKT00, VP06, Wid05]). Others have examined the applications of provenance to social networks [Gol06] and information retrieval [Lyn01]. Researchers have also categorized provenance systems for science [SPG05] and investigated the question of how to capture provenance information, typically through instrumenting workflows and recording their provenance [BD06, BGH$^+$06, MGM$^+$08, SM03, APM08, TGM$^+$06]. Other provenance management systems used in scientific computing include Chimera [FVWZ02] for physics and astronomy, myGrid [ZGSB04] for biology, CMCS [MAB$^+$04] for chemistry, and ESSW [FB01] for earth science. These systems typically do not include approaches for ensuring that tampering with provenance information can be detected. Our approaches to securing provenance chains can be applied to these systems,

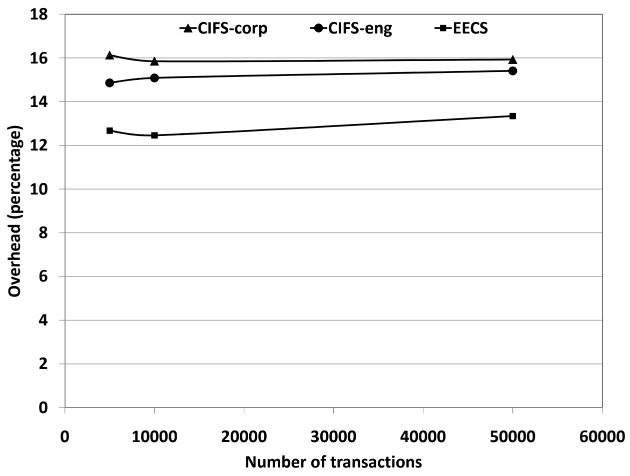

Figure 12: Relative overhead of secure provenance with the real-life workloads (CIFS-Corp, CIFS-eng, and EECS) (*Config-RD*)

to increase their security guarantees. However, in some of these domains there is a significant threat that an adversary may try to pass off someone else's work as their own – the reverse of the Prada and pharmaceutical examples that motivated our work. Additional measures must be taken to guard against such threats.

Overall, the body of research on provenance has focused on the collection, semantic analysis, and dissemination of provenance information, and little has been done to secure that information [BSS08, HSW07]. One exception is the Lineage File System [SC05], which automatically collects provenance at the file system level. It supports access control in the sense that a user can set lineage metadata access flags, and the owner of a file can read all of its lineage information. However, this does not meet the challenges of providing a plausible and confidential history discussed in this paper and elsewhere [BSS08, HSW07].

Another exception is the security guarantees provided in the Open Provenance Model (OPM) [MFF+08, TMG+06]. The OPM specification allows individual provenance entries (or *p-assertions*) to be signed, in order to provide non-repudiation. However, OPM does not specify any chaining mechanism to protect the integrity of the temporal order of the records. It also does not specify mechanisms for finer grained confidentiality and privacy of document modifications.

Outside the domain of provenance, researchers have used *entanglement* to preserve the historic states of distributed systems in a non-repudiable, tamper-evident manner [MB02, SW07]. Entanglement allows multiple mutually distrustful services to maintain a verifiable global timeline. Systems exchange authenticators, and entangle them with their own local states. This prevents any system from lying about its own history at a later time, as other systems can serve as witnesses and provide proof of malicious behavior. Our co-provenance scheme discussed in Section 5 uses this approach, but works at a much finer granularity of individual files. Also, in entanglement, history of a system is maintained locally, and unlike provenance of documents, local history does not propagate to other (potentially untrusted) users or systems. Therefore, confidentiality and privacy of local system states are not the focus of entanglement, whereas our scheme has to provide these assurances for documents that propagate among users.

Source code management systems (SCM) such as Subversion [CS02], GIT [Loe06], or CVS [Ber90] with secure audit trails can provide integrity assurances for versions in a centralized file system. Most of these systems, however, are not applicable to a situation where documents traverse multiple boundaries between potentially untrusted domains. GIT, Monotone [mon], and several other systems do provide support for a distributed infrastructure with users maintaining local histories, but they still use the notion of a virtual (centralized) repository where the users merge and synchronize their local repositories. Such an approach can be used when the participants cooperate to the extent that they act as a virtual organization. Our approach is intended for situations where a priori cooperation agreements are not in place, and documents are physically exchanged between users in different systems and domains.

Researchers have investigated the use of verifiable audit trails for versioning file systems [PBAB07], using keyed hash-chains to protect version history. This approach assumes that all auditors are fully trusted, to the extent that auditors share a symmetric key with the file system for creating the MACs. Malicious auditors can easily forge provenance records and falsify audits. The audit authenticators need to be published to a trusted third party, which must provide them accurately during audits. Our approach places much less trust in auditors. In particular, they do not have access to special keys that would allow them to tamper with provenance chains. We do trust that auditors will carry out their audit functions faithfully; in a domain where this assumption is unrealistic, one could use multiple auditors for each audit, and follow a Byzantine agreement protocol to ensure consensus on the outcome of the audit.

Similarly to audit trails, secure audit logs based on hash chains have been used in computer forensics [SK99, SYC04]. Such schemes work under different system and threat models than secure provenance. By their very nature, audit logs are stationary and protect the integrity of local state. By contrast, provenance information can be highly mobile and traverse multiple untrusted domains. Moreover, audit logs rarely require the selective confidentiality assurances needed for provenance. For example, the mechanisms proposed in [SK99] secure logs as a whole, but do not allow authentication of individual modifications. Additionally, provenance is usually associated with a digital object (e.g., file). This association introduces attacks that are not applicable to secure audit logs. For example, a malicious user might take the provenance chain of document D and claim that it belongs to a completely different document D'. In this attack, the provenance chain itself is not tampered, but the adversary makes a false assertion about the association between the chain and the document. While typical mechanisms for integrity of audit logs will provide integrity for the provenance chain, they are not useful to preserve the association between the document and its previous versions with its provenance. Finally, most secure audit log schemes assume that only one or a few parties will process the audit log and compute checksums. This assumption eliminates the problem of ensuring fine-grained control over which auditors can read which details of a provenance record.

Multiply-linked hash chains have been used for signature cost amortization in multicast source authentication [GR01, GM01, MS01, PCTS00]. Our spiral chain constructs are similar in principle. The main difference is that such hash chains all assume a single sender signing the message block containing the hashes. We can adopt these methods to amortize signature costs in consecutive provenance chain entries from the same principal, but with multiple principals, we need chaining using non-repudiable signatures. Also, many of the hash-chain schemes require the entire stream to be known a priori, an assumption not applicable to provenance deployment settings. Finally, our spiral construction allows integrity-preserving compaction, which is not possible with the other hash chain schemes.

Researchers have investigated integrity guarantees for cooperative XML updates [MFBK06], where document originators define a flow path policy before dissemination and recipients can verify whether the document updates happened according to this flow policy. For flexibility and wider applicability, our approach ensures that histories are plausible even when flow path policies are not predefined. Additional plausibility constraints specific to a particular flow path policy can be defined at a higher level, and passed down to our library for enforcement.

8 Conclusion

Tracking data provenance is an essential requirement in science, medicine, commerce, and government, for rights protection, regulatory compliance, management of intelligence and medical data, and authentication of information as it flows through workplace tasks. Yet, to be effective, provenance mechanisms require strong security assurances of integrity and confidentiality. In this paper we presented a set of efficient mechanisms, based on signed chained checksums, that can be used to provide such assurances in operating system kernels, file systems, or in the application layer. We implemented this approach for tracking file reads and writes using an application-layer library, in such a way that existing applications do not need to be recompiled to be used with provenance tracking. Experiments show that when tracking write operations, our Sprov library imposes overheads of only 1–13% on typical real-life workloads. For scenarios where both read and write operations are tracked, Sprov imposes overheads of 12–16% on typical real-life workloads.

Our work shows that with a careful implementation, it is possible to routinely collect provenance information in a minimally intrusive manner on today's workstations. This ability opens up several potential avenues for future research. First, we can investigate ways of passing information between the provenance collection system and higher levels of software, as discussed in previous sections. Such information can involve very fine-grained application-directed decisions about what provenance details to record, and whether and when to record them at all. Second, we can add specialized support for small writes to large files, so that document integrity is guaranteed using checksums computed over bounded-size regions of files – either at the application layer, as in Sprov, or in the operating system. Third, we can explore a world where provenance information is collected routinely – because collection is cheap – and queried relatively rarely. In such applications, what performance is needed for query processing, and what auxiliary support is needed to attain that level of performance?

Acknowledgement

The authors would like to thank Bill Bolosky and John Douceur for providing suggestions about the file system distributions. Kiran Kumar Muniswamy-Reddy and Prof. Margo Seltzer helped with the Small & Large File Microbenchmarks. Hasan and Winslett were supported in part by NSF awards CNS-0716532 and CNS-0803280. Sion was supported in part by the NSF through awards CNS-0627554, CNS-0716608, CNS-0708025 and IIS-0803197. Sion would also like to thank Motorola Labs, Xerox PARC, IBM Research, the IBM Software Cryptography Group, CEWIT, and the Stony Brook Office of the Vice President for Research.

References

[ABDL07] Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A five-year study of file-system metadata. In *Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST)*, Berkeley, CA, USA, 2007. USENIX Association.

[APM08] Rocio Aldeco-Perez and Luc Moreau. Provenance-based Auditing of Private Data Use. In *BCS International Academic Research Conference, Visions of Computer Science*, September 2008.

[BCC06] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 539–550, New York, NY, USA, 2006. ACM Press.

[BCCV06] Peter Buneman, Adriane Chapman, James Cheney, and Stijn Vansummeren. A provenance model for manually curated data. In Moreau and Foster [MF06], pages 162–170.

[BD06] Roger S. Barga and Luciano A. Digiampietri. Automatic generation of workflow provenance. In Moreau and Foster [MF06], pages 1–9.

[Ber90] Brian Berliner. CVS II: parallelizing software development. In *Proceedings of the Winter USENIX Conference*, pages 341–352, Berkeley, CA, USA, 1990. USENIX Assoc.

[BGH+06] Uri Braun, Simson L. Garfinkel, David A. Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I. Seltzer. Issues in automatic provenance collection. In Moreau and Foster [MF06], pages 171–183.

[BKT00] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Data provenance: Some basic issues. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS)*, pages 87–93, London, UK, 2000. Springer-Verlag.

[BKT01] Peter Buneman, Sanjeev Khanna, and Wang C. Tan. Why and where: A characterization of data provenance. *Lecture Notes in Computer Science*, 1973:316–330, 2001.

[Blu81] Manuel Blum. Coin flipping by telephone. In *Proceedings of CRYPTO*, pages 11–15, 1981.

[BSS08] Uri Braun, Avraham Shinnar, and Margo Seltzer. Securing provenance. In *The 3rd USENIX Workshop on Hot Topics in Security*, USENIX HotSec, Berkeley, CA, USA, July 2008. USENIX Association.

[Cen96] Centers for Medicare & Medicaid Services. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). Online at http://www.cms.hhs.gov/hipaa/, 1996.

[CJR08] Adriane Chapman, H.V. Jagadish, and Prakash Ramanan. Efficient provenance storage. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, Vancouver, Canada, 2008. ACM Press.

[Con99] Congress of the United States. Gramm-Leach-Bliley Financial Services Modernization Act. Public. Law No. 106-102, 113 Stat. 1338, 1999.

[CS02] Ben Collins-Sussman. The subversion project: buiding a better CVS. *Linux J.*, 2002(94):3, 2002.

[DB99] John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 59–70. ACM New York, NY, USA, 1999.

[ELMS03] Daniel Ellard, Jonathan Ledlie, Pia Malkani, and Margo Seltzer. Passive NFS tracing of email and research workloads. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, pages 203–216, Berkeley, CA, USA, 2003. USENIX Association.

[FB01] James Frew and Rajendra Bose. Earth system science workbench: A data management infrastructure for earth science products. In *Proceedings of the Thirteenth International Conference on Scientific and Statistical Database Management (SSDBM '01)*, page 180, Washington, DC, USA, 2001. IEEE Computer Society.

[FVWZ02] Ian T. Foster, Jens-S. Vockler, Michael Wilde, and Yong Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM '02)*, pages 37–46, Washington, DC, USA, 2002. IEEE Computer Society.

[GM01] Philippe Golle and Nagendra Modadugu. Authenticating streamed data in the presence of random packet loss. *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, pages 13–22, 2001.

[Gob02] Carole Goble. Position statement: Musings on provenance, workflow workflow and (semantic web) annotations for bioinformatics. In *Workshop on Data Derivation and Provenance*, 2002.

[Gol06] Jennifer Golbeck. Combining provenance with trust in social networks for semantic web content filtering. In Moreau and Foster [MF06], pages 101–108.

[GR01] Rosario Gennaro and Pankaj Rohatgi. How to Sign Digital Streams. *Information and Computation*, 165(1):100–116, 2001.

[HS02] Dani Halevy and Adi Shamir. The LSD broadcast encryption scheme. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '02)*, pages 47–60, London, UK, 2002. Springer-Verlag.

[HSW07] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability (StorageSS)*, pages 13–18, New York, NY, USA, 2007. ACM Press.

[Kat97] Jeffrey Katcher. Postmark: a new file system benchmark. Network Appliance Tech Report TR3022, October 1997.

[KSW06] Noam Kogan, Yuval Shavitt, and Avishai Wool. A practical revocation scheme for broadcast encryption using smartcards. *ACM Trans. Inf. Syst. Secur.*, 9(3):325–351, 2006.

[Loe06] Jon Loeliger. Collaborating with GIT. *Linux Magazine*, June 2006.

[LPGM08] Andrew W. Leung, Shankar Pasupathy, Garth Goodson, and Ethan L. Miller. Measurement and analysis of large-scale network file system workloads. In *Proceedings of the USENIX Annual Technical Conference*, pages 213–226, Berkeley, CA, USA, 2008. USENIX Association.

[Lyn01] Clifford A. Lynch. When documents deceive: Trust and provenance as new factors for information retrieval in a tangled web. *Journal of the American Society for Information Science and Technology*, 52(1):12–17, 2001.

[MAB$^+$04] James D. Myers, Thomas C. Allison, Sandra Bittner, Brett Didier, Michael Frenklach, Jr. William H. Green, Yen-Ling Ho, John Hewson, Wendy Koegler, Carina Lansing, David Leahy, Michael Lee, Renata McCoy, Michael Minkoff, Sandeep Nijsure, Gregor von Laszewski, David Montoya, Carmen Pancerella, Reinhardt Pinzon, William Pitz, Larry A. Rahn, Branko Ruscic, Karen Schuchardt, Eric Stephan, Al Wagner, Theresa Windus, and Christine Yang. A collaborative informatics infrastructure for multi-scale science. In *Proceedings of the 2nd International Workshop on Challenges of Large Applications in Distributed Environments (CLADE '04)*, page 24, Washington, DC, USA, 2004. IEEE Computer Society.

[MB02] Petros Maniatis and Mary Baker. Secure history preservation through timeline entanglement. In *Proceedings of the 11th USENIX Security Symposium*, pages 297–312, Berkeley, CA, USA, 2002. USENIX Association.

[MF06] Luc Moreau and Ian T. Foster, editors. volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.

[MFBK06] Giovanni Mella, Elena Ferrari, Elisa Bertino, and Yunhua Koglin. Controlled and cooperative updates of XML documents in byzantine and failure-prone distributed systems. *ACM Trans. Inf. Syst. Secur.*, 9(4):421–460, 2006.

[MFF$^+$08] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers, and Patrick Paulson. The open provenance model: An overview. In Juliana Freire, David Koop, and Luc Moreau, editors, *IPAW*, volume 5272 of *Lecture Notes in Computer Science*, pages 323–326. Springer, 2008.

[MGM$^+$08] Luc Moreau, Paul Groth, Simon Miles, Javier Vazquez-Salceda, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, and Laszlo Varga. The provenance of electronic data. *Commun. ACM*, 51(4):52–58, 2008.

[mon] Monotone Distributed Version Control. Online at `http://www.monotone.ca/`, accessed on December 22, 2008.

[MRBH$^+$09] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. Layering in provenance systems. In *Proceedings of the USENIX Annual Technical Conference*, Berkeley, CA, USA, June 2009. USENIX Association.

[MRHBS06] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo I. Seltzer. Provenance-aware storage systems. In *Proceedings of the USENIX Annual Technical Conference*, pages 43–56, Berkeley, CA, USA, 2006. USENIX Association.

[MS01] Sara K. Miner and Jessica Staddon. Graph-based authentication of digital streams. In *IEEE Symposium on Security and Privacy*, pages 232–246, 2001.

[PBAB07] Zachary N. J. Peterson, Randal Burns, Giuseppe Ateniese, and Stephen Bono. Design and implementation of verifiable audit trails for a versioning file system. In *Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST)*, pages 20–20, Berkeley, CA, USA, 2007. USENIX Association.

[PCTS00] Adrian Perrig, Ran Canetti, Doug Tygar, and Dawn Xiaodong Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, May 2000.

[Pug90] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[RLA00] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson. A comparison of file system workloads. In *Proceedings of the USENIX Annual Technical Conference*, Berkeley, CA, USA, 2000. USENIX Association.

[SC05] Can Sar and Pei Cao. Lineage file system. Online at http://crypto.stanford.edu/ cao/lineage.html, January 2005.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SK99] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, 1999.

[SM03] Martin Szomszor and Luc Moreau. Recording and reasoning over data provenance in web and grid services. In *International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE)*, volume 2888 of *Lecture Notes in Computer Science*, pages 603–620, Catania, Sicily, Italy, November 2003.

[SPG05] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[SSB$^+$95] Margo Seltzer, Keith A. Smith, Hari Balakrishnan, Jacqueline Chang, Sara McMains, and Venkata Padmanabhan. File system logging versus clustering: a performance comparison. In *Proceedings of the USENIX Annual Technical Conference*, pages 21–21, Berkeley, CA, USA, 1995. USENIX Association.

[SW07] Daniel Sandler and Dan S. Wallach. Casting votes in the auditorium. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, Berkeley, CA, USA, 2007. USENIX Association.

[SYC04] Richard T. Snodgrass, Shilong Stanley Yao, and Christian Collberg. Tamper detection in audit logs. In *Proceedings of the Thirtieth international conference on Very large data bases (VLDB)*, pages 504–515, Toronto, Canada, 2004. VLDB Endowment.

[TGM$^+$06] Victor Tan, Paul Groth, Simon Miles, Sheng Jiang, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Security issues in a SOA-based provenance system. In Moreau and Foster [MF06], pages 203–211.

[The03a] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. `www.openssl.org`, April 2003.

[The03b] The U.S. Securities and Exchange Commission. Rule 17a-3&4, 17 CFR Part 240: Electronic Storage of Broker-Dealer Records. Online at `http://edocket.access.gpo.gov/cfr_2002/aprqtr/17cfr240.17a-4.htm`, 2003.

[TMG$^+$06] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A profile for non-repudiable process documentation. Technical report, University of Southampton, `http://eprints.ecs.soton.ac.uk/13054/`, November 2006.

[U.S02] U.S. Public Law No. 107-204, 116 Stat. 745. The Public Company Accounting Reform and Investor Protection Act, 2002.

[VP06] Nithya N. Vijayakumar and Beth Plale. Towards low overhead provenance tracking in near real-time stream filtering. In Moreau and Foster [MF06], pages 46–54.

[Wid05] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR '05)*, January 2005.

[ZGSB04] Jun Zhao, Carole A. Goble, Robert Stevens, and Sean Bechhofer. Semantically linking and browsing provenance logs for e-science. In *Proceedings of the First International IFIP Conference on Semantics of a Networked World (ICSNW)*, pages 158–176, 2004.