The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance

Ragib Hasan
rhasan@illinois.edu
University of Illinois at Urbana-Champaign

Radu Sion
sion@cs.stonybrook.edu
Stony Brook University

Marianne Winslett
winslett@illinois.edu
University of Illinois at Urbana-Champaign

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. In this paper, we show how to provide strong integrity and confidentiality assurances for data provenance information. We describe our provenance-aware system prototype that implements provenance tracking of data writes at the application layer, which makes it extremely easy to deploy. We present empirical results that show that, for typical real-life workloads, the runtime overhead of our approach to recording provenance with confidentiality and integrity guarantees ranges from 1% – 13%.

1 Introduction

Provenance information summarizes the history of the ownership of items and the actions performed on them. 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 track the processing of DNA samples. Individuals who submit DNA samples for testing through these programs want strong assurances that no unauthorized parties will be able to see the provenance of the samples (e.g., provide it to insurance companies or anti-Semitic organizations).

Regulatory and legal considerations mandate other provenance assurances. The US Sarbanes-Oxley Act [56] sets prison terms for officers of companies that issue incorrect financial statements. As a result, officers have become very interested in tracking the path that a financial report took during its development, including both input data origins and authors. The US Gramm-Leach-Bliley Act [40] and Securities and Exchange Commission rule 17a [55] also require documentation and audit trails for financial records, as do many non-financial compliance regulations. For example, the US Health Insurance Portability and Accountability Act mandates logging of access and change histories for medical records [13].

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 record-keeping, as does US FDA testing of proposed new food additives.

To help manage the above processes, digital provenance mechanisms support the collection and persistence of information about the creation, access, and transfer of 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. But without appropriate guarantees, as data crosses application and organization boundaries and passes through untrusted environments, its associated provenance information becomes vulnerable to illicit alteration and should not be trusted.

For example, consider the repudiation incentives in the following *real-life* anonymized medical litigation scenario. Alice visited Dr. B for consultation. B referred her to Dr. Mallory for tests, and sent Alice's medical records to Mallory, who failed to analyze the test results properly, and provided incorrect information to B. B provided these reports along with other information to Dr. C, who treated Alice accordingly. When Alice subsequently suffered from health problems related to the incorrect diagnosis, she sued B and C for malpractice. To establish

Mallory's liability for the misdiagnosis, B and C hired Audrey as an expert witness. Audrey used the provenance information in Alice's medical records to establish the exact sequence of events, which in this case implicated Mallory. If Mallory had been innocent, B and C should not be able to collude and falsely implicate him. Similarly, if Mallory altered his faulty diagnosis in Alice'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 entries; *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.

In this paper, we propose and evaluate mechanisms for secure document provenance that address these properties. In particular, our *first* contribution is a cross-platform, low-overhead architecture for capturing provenance information at the application layer. This architecture captures the I/O write requests of all applications that are linked with the library, extracts the new data being written and the identity of the application writing it, and appends that information to the provenance chain for the document being written. Further, the resulting provenance chain is secure in the sense that a particular entry in the chain can only be read by the auditors specifically authorized to read it, and no one can add or remove entries from the *middle* of the chain without detection. Our *second* contribution is an implementation of our approach for file systems, along with an experimental evaluation that shows that our approach introduces little overhead at run time, only 1%–13% for typical real-life workloads.

2 Provenance Model

In this section, we define basic provenance-related concepts and discuss deployment and threat models.

2.1 Definitions and Usage Model

A **document** is an abstraction for a data item for which provenance information is collected, such as a file, database tuple, or network packet.

We define **provenance** of a document to be the record of actions taken on that particular document over its lifetime. Note that this definition differs from the information flow provenance used in PASS [38] and some other systems. Each access to a document D may generate a **provenance record** P. The types of access that should generate a provenance record and the exact contents of the record are domain-specific, 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., bytes of D or its metadata read/written); a description of the environment when the action was made, 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 real deployments, the chain is associated and transported together with a document D.

In a given security domain, **users** are principals who read and write to the documents, and/or make changes to document metadata. In a given organization, there are one or more **auditors**, who are principals authorized to access and verify the integrity of provenance chains associated with documents. Every user trusts a subset of the auditors. There can be an auditor who is trusted by everyone, and referred to as the **superauditor**.

Documents, and associated provenance chains are stored locally in the current user's machine. The local machines of the users are not trusted. Each user has complete control over the software and hardware of her local machine and storage. Documents can be transferred from one machine to another. A transfer of a document from one machine to another also causes the provenance chain to be transferred to the recipient.

Adversaries are inside or outside principals with access to the chains, who want to make undetected changes to a provenance chain for personal benefit. We do not consider denial of service attacks such as the total removal of a provenance chain.

We assume readers are familiar with semantically secure (IND-CPA) encryption and signature mechanisms [22] and *cryptographic hashes* [34]. We use ideal, collision-free hashes and strongly unforgeable signatures. We denote by $S_k(x)$ a public key signature with key k on item x. $a|b$ and a, b denote concatenating b after a.

2.2 Threat Model

In this paper, we focus on tracking document writes and securing the provenance information associated with them. We leave as future work the questions of how to ensure that provenance information is always collected, how to track document reads efficiently, and certain other technical issues discussed below. We now discuss the reasons for choosing this focus.

A provenance tracking system implemented at a particular level is oblivious to attacks that take place outside the view of that level. For example, suppose 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.

Thus, without a trusted *pervasive* hardware infrastructure and an implementation of provenance tracking at that level, we cannot prevent all potential attacks on provenance chains. Even in such an environment, a malicious user who can read a document can always memorize and replicate portions thereof later, minus the appropriate provenance information. For example, an industrial spy can memorize technical material and reproduce it later on in verbatim or edited form. Overall, it is impractical to assume that we have the ability to fully monitor the information flow channels available to attackers. Thus, our power to track the origin of data is limited.

Fortunately, in many applications of provenance-aware systems, illicit document copying and/or complete removal of provenance chains are not significant threats. For example, in cattle tracking, we are not worried that someone will try to steal a (digital record of a) cow and try to pass it off as their own. Similarly, a cow with no provenance record at all is highly suspect, and many different parties would have to collude to fabricate a convincing provenance history for that cow. Instead, the primary concern is that a farmer might want to rewrite history by omitting a record showing that a particular sick cow previously lived at his feed lot. As another application, 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. Thus, if an enterprising crook wants to sell drugs manufactured by an unlicensed company, he might want to forge a provenance chain that gives the drugs a more respectable history, in order to move them into the supply chain. 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. This is very hard to do, as a colluder with the ability to put the Prada signature on the accessories' provenance chain needs to be found. Anyone who can do that signature is a Prada insider, and Prada insiders have little incentive to endorse fake merchandise.

Of course, there are applications where there are significant incentives to track data reads and for malicious parties to sever documents from their original provenance chains. For example, one may track the flow of intelligence information so that it can be traced back to its original source. It is possible to track all reads and then use that to track information flows. However, doing that is expensive, especially in the long run as provenance chains get longer and longer. It is impossible to offer complete provenance assurances in such situations, but certain measures can be taken, such as digital watermarking and kernel-level tracking of all data read by a writing process [38] (which is expensive due to the need to log all data read). Overall, however, we believe that in the presence of any non-trivial threat model, tracking read operations for the purpose of provenance collection is impractical and necessarily insecure, because the adversary can always read data through unmonitored channels and produce verbatim or edited copies. Thus, we will not consider the tracking of read operations further in this paper.

The primary threat we guard against in this paper is **undetected rewrites of history**, which occur when malicious entities forge provenance chains to match illicit document writes and metadata modifications. Specifically, suppose that we have a provenance chain ($[A]$, $[B]$, $[C]$, $[D]$, $[E]$, $[F]$), in which, for simplicity, each entry is denoted by the identity of its corresponding principal $A, B, C, \ldots$. Then, we will provide the following integrity and confidentiality assurances:

- **I1:** An adversary acting alone cannot selectively remove other principals' entries from the start or the middle of the chain without being detected by the next audit.
- **I2:** An adversary acting alone cannot add entries in the beginning or the middle of the chain without being detected by the next audit.
- **I3:** Two colluding adversaries cannot add entries of other non-colluding users "between" them in the chain, without being detected by the next audit.

For example, colluding users B and D cannot undetectably add entries between their own, corresponding to fabricated actions by a non-colluding party E.

- **I4:** Once the chain contains subsequent entries by non-malicious parties, two colluding adversaries cannot *selectively* remove entries associated with other non-colluding users between them in the chain, without being detected by the next audit.

E.g., colluding users B and D cannot remove entries made by non-colluding user C.

An adversary in possession of a document can always eliminate *all* elements in the chain, starting from the last colluding party's entry in the chain. For example, a malicious F could remove the entry for E, if D cooperates, and claim that the chain is ($[A]$, $[B]$, $[C]$, $[D]$, $[F]$). This however, is a denial-of-service attack that cannot be prevented through technical means only. Two parties could always collude in this manner in real life through outside channels. Thus, we target chain forgeries that maliciously add new chain entries and make after-the-fact modifications.

- **I5:** Users cannot repudiate chain records.
- **I6:** An adversary cannot claim that a valid provenance chain for one document belongs to a different document (lineage forgery), without detection at the next audit by a superauditor, if not sooner.

- **I7:** If the adversary alters a document without appending appropriate provenance records to its chain, this will be detected at the next audit by a superauditor, if not sooner.
- **C1:** Any auditor can verify the integrity of the chain without requiring access to 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.

For illustration purposes, consider Bob, Charlie, and Dave editing document D, in that order. The resulting provenance chain contains chronologically ordered entries made by these users. Suppose that, Bob performed an operation on D using a proprietary algorithm, and does not want the workflow he used to be revealed to anyone except auditor Audrey. Property C1 ensures that Bob can selectively reveal the records pertaining to his actions on D to Audrey. Audrey can verify the integrity of the chain and read Bob's action record, while other auditors can only verify the integrity of the chain.

Suppose that Dave subsequently decides to release D to George, who should not learn the private information in Charlie's provenance records. Property C2 allows Dave to give George the provenance chain minus Charlie's sensitive information, while still allowing George to verify the integrity of the chain.

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 [24, 27] to selectively regulate the access for different auditors. Finally, for confidentiality (C2), we use a cryptographic commitment based construction. In the following, we detail these components.

3.1 Building Blocks

3.1.1 Chain Construction

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

$$P_i \quad = \quad \langle U_i, W_i, hash(D_i), C_i, public_i, I_i \rangle,$$

where

- U_i is an opaque or plaintext identifier for the principal;
- W_i is an opaque representation of the sequence of document modifications performed by U_i;
- $hash(D_i)$ is a cryptographic hash of the newly modified contents of D;
- C_i contains an entry integrity checksum;
- $public_i$ is an optional opaque or plaintext public key certificate for user U_i;
- I_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.

3.1.2 Confidentiality

Let w_i be a representation of the sequence of document modifications just now performed by U_i. The choice of representation for w_i is dictated by the application domain; for example, w_i could be a file diff, a log of changes, or a higher-level semantic representation of the alterations. The representation should be reversible, if we want to allow auditors to check whether the current contents of D match its declared history; otherwise the representation does not have to be reversible. Given w_i, W_i is an encrypted version of w_i: $W_i = E(w_i)$.

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

Strawman Choices of E. If all auditors should be able to read all entries, we can encrypt all w_i using a single secret key k shared with the auditors via a central keystore. I.e., $E(w_i) = e_k(w_i)$. If only a subset of the auditors should be able to read w_i, then U_i can encrypt one copy of w_i for each auditor that U_i trusts:

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

This is inefficient, as U_i has to include multiple copies of w_i, 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 entries.

To save space, U_i can encrypt w_i with a session key k_i unique to this entry, and also include the (shorter) session key k_i encrypted with the public keys of the trusted

auditors. That is,

$$E(w_i) = e_{k_i}(w_i)$$

$$I_i = \{e_{K_A}(k_i) : K_A \text{ is the public key of } A, \text{ an auditor } U_i \text{ trusts}\}$$

In the rest of this paper, we assume that each entry employs a session key. Still, I_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_i, we employ broadcast encryption [24, 27]. 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 keypair 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 an entry, we encrypt the session key k_i with the public key in the root of the tree and store it in I_i. Any auditor can then decrypt W_i using the private key in the root node (known only to auditors). Similarly, if U_i trusts a single auditor A, we encrypt k_i with the public key k_A at the leaf for A, so that $I_i = e_{k_A}(k_i)$. If U_i trusts an arbitrary subset S of auditors, we set

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

where R is a minimum-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_i, as I_i includes only $\log(n - |S|)$ copies of k_i.

If U_i should be opaque, the session key k_i can be used to hide the identity U of the user responsible for the document modifications represented in entry P_i. U should identify the principal in a manner appropriate for the application domain. For example, in a file system we can define U as:

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

In an application domain where U_i should be opaque, we can set $U_i = e_{k_i}(U)$, where k_i is the session key defined above. The same can be done for U's public key certificate $public_i$. Authorized auditors can use I_i to decrypt U_i and $public_i$.

In some application domains, we may need finer-grained control over which auditors and users can see which details of an entry, rather than using a single session key to encrypt all sensitive fields in an entry. For example, we might be willing to let the billing auditors decrypt U_i but not W_i. In this case, one option is to use additional session keys for the other sensitive fields of the entry, 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_i 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_i. For these two approaches, we employ secret sharing and threshold cryptography for I_i [49]. Under the first approach, each group has a different share of the session key i_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_i.

3.1.3 Integrity

Principle **C1** says that every auditor can verify every provenance chain, even if he or she cannot decrypt some of its W_i 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_i of an entry as:

$$C_i = S_{U_i}(hash(U_i, W_i, hash(D_i), public_i, I_i)|C_{i-1}),$$

where S_{U_i} means that user U_i signs the hash with his or her private key. We refer to this approach as **signature-based checksums**, as it creates a signature chain that enforces the integrity assurances **I1**-**I7** for each individual entry and for the chain itself. For a user Audrey to be able to verify the chain, she must be able to tell which user wrote each entry in the chain, and have access to their public keys. This can be accomplished by storing the U_i and, if present, $public_i$ fields as plaintext; if the $public_i$ field is empty, Audrey must find the public key for U_i through external means. Audrey can then verify the integrity of the provenance chain by parsing it from beginning to end and using the C_i values to verify the integrity of each entry. She can also verify that the current contents D_n of D match its hash in P_n. However, she cannot check that D_n was computed properly from D_{n-1} unless she is allowed to decrypt W_n, i.e., she is given access to the session key k_n, and a reversible representation was used for w_n. To verify that all of these transformations were performed correctly, Audrey must retrieve her keys from the broadcast encryption tree and use them to decrypt the I_i field of each entry. The I_i field holds the session key for each entry, which she can use to decrypt all of the entry's fields, thus obtaining the w_i fields. 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. If Audrey is not authorized to access all of the session keys for D, then she can only verify that the most recent j entries match the contents of D, where session key k_{n-j} is the most recent session key that she cannot access.

3.1.4 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 produce 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 [6] 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 U_i or W_i by its commitment before computing the checksum for P_i:

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

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 name of an unusual test performed in W_i can be replaced by a commitment, while leaving the other more typical tests in W_i in plaintext. When Arizona sends the chain to an internal party trusted to view the plaintext version of U_i and/or W_i, both the commitments and the original plaintext values of the unusual test s and r_s will be included as usual in the provenance entry. When 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 entries 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.

3.1.5 Augmenting Provenance Chains

While the integrity checksums of Section 3.1.3 completely satisfy the assurances I1-I7, we can introduce further optimizations for faster verification and integrity-preserving summarization of long provenance chains. Provenance chains tend to grow very fast, often becoming several magnitudes in size larger than the original data item and requiring compaction [14]. With our augmented chains, we can compact the chain by removing irrelevant entries, while preserving the validity of integrity verification mechanisms, without requiring recomputing the signatures.

The *integrity spiral*, a redundant, multiple-linked chain mechanism, is conceptually similar to skip-lists [43]. The basic idea is to compute the checksum(s) of each provenance entry by combining the (hash of the) current entry, and multiple previous checksums. We provide two constructions with different properties and usages.

Construction 1. The first construction computes the checksum C_i of the provenance entry P_i as follows

$$\begin{aligned} C_i \quad = \quad & S_{U_i}(hash(U_i, W_i, hash(D_i), public_i, I_i) \\ & |C_{prev_1}| \cdots |C_{prev_R}), \end{aligned}$$

where R is the spiral dimension, and C_{prev_k} represents a checksum chosen at random from preceding entries in the provenance chain.

Advantages. This construction allows quick detection of forgery of entries. Suppose that Mallory modifies the entry P_i and computes a new checksum C_i' based on the forged entry and the preceding part of the chain. In the singly-linked mechanism, this will be detected when the auditor checks the checksum for P_{i+1}, which Mallory is unable to forge (per I2). However, the auditor will have to verify the entire chain up to and including the entry P_i to detect this. We enable quick local verification by construction 1, in which multiple subsequent entries will be dependent on the checksum C_i of P_i. The new checksum C_i' added by Mallory will cause the checksums of these dependent entries to fail, and therefore expose

Mallory's forgery. To evade detection, Mallory will have to modify the checksums of all these entries, which in turn will affect further subsequent entries.

Construction 2. In our second construction, we construct a spiral checksum C_i as follows:

$$C_i = C_{i_1}|C_{i_2}|\cdots|C_{i_R}|C_{i_0};$$

where R is the spiral dimension, and C_{i_j} is defined as:

$$\begin{cases} S_{U_i}(hash(U_i, W_i, hash(D_i), public_i, I_i)|C_{k_j}) \text{ if } j > 0, \\ S_{U_i}(hash(U_i, W_i, hash(D_i), public_i, I_i)|C_{i_1}|C_{i_2}|\cdots|C_{i_R}) \text{ if } j = 0 \end{cases}$$

where $k < i$. Note that, unlike Construction 1, we no longer have a single checksum per entry; rather we use more than one independently computed checksum per entry.

Advantages. Using this construction, we can perform quick verification. The auditor may choose to disregard the linear checksum (i.e. the chain that links to the previous entry's checksum) and use any of the other dimensions. Based on the maximum spiral dimension R of that chain, it might reduce the cost of verification of an N entry chain to $\lfloor \frac{N}{R} \rfloor$. If the chain is constructed such that all entries belonging to a particular event type are linked by a given dimension of the checksum, then the auditor can skip irrelevant entries, but still be able to verify the integrity of the events she is looking for.

Rather than choosing previous checksums randomly, we can use a systematic approach towards building the spiral. For example, to construct C_i, we use checksums from previous entries at distance $1, 2, \ldots, 2^R$. In Construction 1, these checksums are all concatenated to the hash of the current entry, while in Construction 2, these R checksums are separately concatenated with the hash of current entry, and signed to form a collection of R checksums pertaining to the current entry.

Integrity-preserving Summarization. Using construction 2, we can compact a provenance chain while still being able to preserve integrity verification mechanism. If P_j occurs after entry P_i in the chain, and P_j's set of checksums C_j has a checksum C_{j_k} computed from a checksum C_{i_k} from the set C_i (using Construction 2), then that checksum can be used to verify the order of these two entries. If there are d entries between P_i and P_j, we can then remove these entries, while being able to prove the order. This is not contrary to I1, as the auditor can use C_{j_k} to verify the order of P_i and P_j, while detecting that d entries have been removed in between them. If P_i and P_j are not directly connected by a checksum, but have an intermediate entry P_m, such that a checksum exists in C_m that proves P_m occurs after P_i, and a checksum exists in C_j that proves P_j occurs after P_m, then we can keep P_m and remove all other entries between P_i and P_j during compaction. We can extend this technique to link any two entries using some intermediate nodes between them.

3.1.6 Chain Operations

As discussed in Section 2.2, we are concerned with tracking data write operations and document metadata changes – including changes in permissions and other document metadata. We now discuss the impact of document operations on the corresponding secured provenance chains. Although our discussion refers specifically to file system operations, the same semantics are appropriate for other scenarios, including relational databases.

read No impact on the provenance chain.

write A new provenance chain entry is created.

chmod or chown These operations change document metadata. For example, chown can be used to add a new user to the list of users with write access. The change in metadata is recorded as a provenance event.

copy A duplicate copy of the original document is created, with no change in the original document or its provenance. The original document's provenance chain is copied into the new document's provenance chain. A new entry is then added to the new chain, to record the copy operation itself.

delete The document will be removed from the file system, but its provenance chain is *not* deleted. The delete operation is recorded as a metadata operation in the provenance chain. The chain is kept until it expires (determined by its expiration timeout).

If users were guaranteed not to circumvent the provenance-aware read operation (e.g., by reading directly from disk), we could support read-related provenance entries by persisting per-principal provenance contexts containing all information ever read, similar in nature to propagated access list mechanisms [58]. However, as discussed in Section 2.2, we believe that this would give the illusion of security but not the reality, because principals have access to outside information channels. Also, promulgation of provenance for read operations tends to result in a combinatorial explosion in overhead that can ultimately render the system unusable [38].

3.2 Correctness

The mechanisms introduced above satisfy the integrity, confidentiality and privacy properties outlined in Section 2.

Theorem 1. *An adversary cannot remove entries from the beginning or the middle of the chain without detection (I1). An adversary cannot add entries in the beginning or the middle of the chain without detection (I2).*

Proof. (sketch) The proof is straightforward. For (I1) let us assume that an adversary Mallory has removed the entry P_i, $0 < i < n$. Since the integrity checksum field C_{i+1} of the subsequent entry is computed by combining

the current checksum C_i with W_{i+i} under an ideal cryptographic hash function, its verification will fail, therefore revealing the removal of P_i. Similarly, for (I2), any addition of chain entries will be detected in the verification step through the checksum components. □

Theorem 2. *Once the chain contains any subsequent entries by non-malicious users, a set of colluding adversaries cannot insert or remove entries between them in the chain (I3, I4).*

Proof. (sketch) The chained nature of the integrity checksum directly ensures this. Specifically, suppose that Eve and Mallory are two colluding adversaries who are part of the chain, with entry P_e followed by P_m later on in the chain. Moreover, let P_a be non-colluding Alice's entry following Eve's and Mallory's. The chain will thus be $\langle \ldots P_e, \ldots, P_m, \ldots, P_a, \ldots \rangle$. Due to the collision-free, one-way nature of the chained integrity checksum fields, any modification in the chain entries between P_e and P_m will naturally show up when attempting to verify P_a's checksum. □

As discussed in Section 2, if P_m is the last element in the chain, Mallory *can* always remove *all* entries between P_m and any previous entry by a colluding party, e.g., P_e. This DOS attack cannot be prevented through technical means alone.

Theorem 3. *When checksums are constructed using formula from Section 3.1.3, users cannot repudiate an entry (I5).*

Proof. (sketch) This follows by construction when integrity checksum C_i is implemented using the non-repudiable signatures of Section 3.1.3. □

Theorem 4. *An adversary cannot successfully claim that a valid provenance chain for a given document belongs to a document with different contents (I6).*

Proof. (sketch) This follows directly from the collision-free nature of hashing and the fact that a hash of the current document contents is included in each chain entry, which is then authenticated using the chained C_i checksums. Substituting the chain for a different document will be detected by a *super auditor* when a checksum fails to verify. □

Theorem 5. *If a document's contents are inconsistent with its history as recorded in a provenance chain with a reversible or plaintext representation for w_i fields, then any superauditor can detect the discrepancy (I7).*

Proof. By definition, a *superauditor* can decrypt all details of the w_i field in each entry, if the w_i fields are encrypted with session keys. Otherwise, the w_i fields are available in plaintext. After verifying the chain, the superauditor can apply the w_i entries in reverse, repeatedly verifying that the hash of D_i included in entry P_i matches the hash of its recreated contents. At the last verification, D_0 should be the empty document. □

If a non-reversible representation is used for the w_i entries, or the auditor is not a superauditor, the auditor may still be able to tell that the chain is inconsistent with the current contents of D. E.g., if a chain entry says that all document appendices were deleted, and no subsequent entry added any appendices, then application-domain-dependent reasoning lets an auditor conclude that something is wrong if the document has appendices.

Theorem 6. *Any auditor can verify the chain (C1). Auditors can only decode entry details for which they are authorized.*

Proof. (sketch) This also follows directly by construction. When deploying non-repudiable signature-based checksums as in Section 3.1.3, chain verification involves only public key signature operations and no other secret values. It can thus be performed by any party.

Now consider the question of whether unauthorized parties can access the details of entries. We argue the case where a single session key k_i is used to encrypt all sensitive details in the entry, and the key itself is protected using broadcast encryption; the argument is similar if multiple session keys or a single shared key are used for this purpose. First, a session key k_i for W_i is accessible only to principals that can retrieve it by decrypting at least one item in set I_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_i and (if encrypted) U_i and $public_i$. □

Finally, we discuss chain verifiability when cryptographic commitments are used for potentially sensitive plaintext, and the plaintext is subsequently removed.

Theorem 7. *The use of cryptographic commitments in place of potentially sensitive plaintext subfields of U_i or W_i does not affect the verifiability of the chain.*

Proof. (sketch) The checksum component of the entry is computed by using cryptographic commitments rather than the sensitive data item's plaintext. Hence, if the plaintext is removed when releasing to an untrusted principal, the chain remains verifiable, as the verification mechanism only requires the commitment. The chain integrity is also not compromised, due to unforgeability of signatures on the checksum entries for other users. □

4 Empirical Evaluation

Several 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) [38]. The main advantage of this approach is its transparency to user level applications and the file system layer. Major drawbacks include the fact 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. Yet another drawback of such an approach is its limited portability, as any new deployment platform will require porting efforts.

File System Layer. The file system can be made provenance-aware and augmented to transparently handle securing collected 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.

Application Layer. In this approach, the provenance mechanisms are offered through user-level libraries. This can still 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 for provenance-awareness.

4.1 The Sprov Library

We implemented a prototype of the secure provenance primitives as an application layer C library, 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 Section 3.1.2 and 3.1.3 in this prototype.

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 entry in the provenance chain of the file. Information about the user, application, and environment are collected. During write operations, *Sprov* gathers information about the writes. to the file before it is closed. *Sprov* uses a reversible representation of document modifications; as discussed earlier, 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; we leave this for future work.

At file close, the session ends. *Sprov* writes a new entry in the provenance chain for the changes made during this session. At this point, the cryptography (implemented using openssl [54]) associated with the chain integrity constructs is executed, as described in Section 3.1. The provenance chain is stored in a separate meta-file for portability.

We provide utilities to facilitate provenance collection and transfer. 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. Copying and deletion of a provenance-enabled file uses the *pcopy* and *pdelete* utilities, respectively, as discussed in Section 3.1.6. Finally, the *pmonitor* daemon periodically scans for and removes expired provenance chains.

4.2 Experiments

Our experiments 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 took 1.5ms to compute. The experiments used a mix of four of the following drive types: Seagate Barracuda 7200.11 SATA 3Gb/s 1TB, 7200 RPM, 105 MB/s sustained data rate, 4.16ms average seek latency and 32 MB cache, and Western Digital Caviar SE16 3 Gb/s, 320GB, 7200 RPM, 122 MB/s sustained data rate, 4.2ms average latency and 16MB 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 deployed (i) PostMark [26] – a standard benchmark for file system performance evaluation, (ii) the Small and Large file microbenchmark that has been used to evaluate the performance of PASS [38, 48], and (iii) a custom transaction-level benchmark meant to test the performance in live file systems with file sizes distributed realistically [2, 16], and real-life file system workloads [17, 28, 44].

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**).

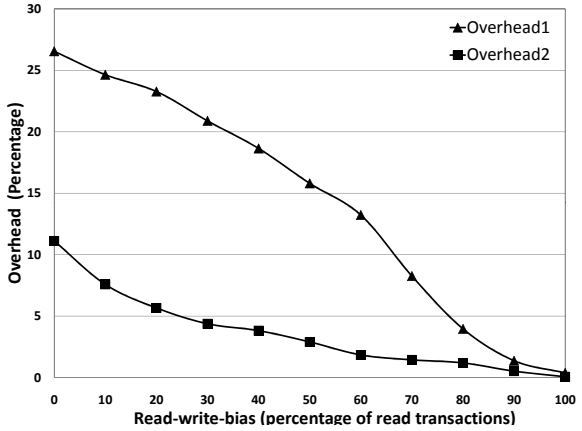

Figure 1: Overhead of secure provenance with Postmark. Overhead1 indicates the overhead using *Config-Disk*, while Overhead2 is from *Config-RD* setting. The overheads are shown from 0% read bias (100% write transactions) to 100% read bias (no write transactions).

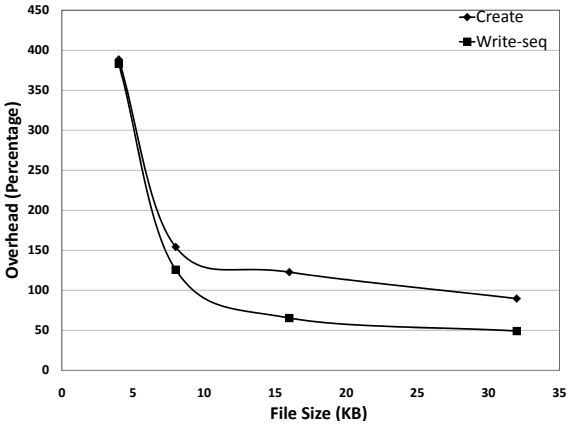

Figure 2: Small file system microbenchmark create and write performance for 2500 files.

4.2.1 Postmark Benchmark

We measured the execution time of the Postmark benchmark [26] 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 20000 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 1 for both *Config-Disk* and *Config-RD* and range from 0.5% to 11% for *Config-RD*.

4.2.2 Small and Large File Microbenchmarks

The small and large file microbenchmarks [48] have been used in the evaluation of PASS [38]. The small file microbenchmark creates, writes to and then deletes 2500 files of sizes ranging from 4KB, 8KB, 16KB, and 32KB. We benchmarked the overhead for file creation as well as synchronous writes. The results for *Config-Disk* are displayed in Figure 2.

An interesting effect can be observed. Similar to the experiments in PASS, the overhead percentage is quite high for small files and decreases rapidly with increasing file sizes. We believe this effect can be attributed to disk caching. Specifically, for very small file size accesses - which go straight to the disk 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 to under 50% for larger 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	$sprov_{CD}$	%Overhead	$sprov_{CRD}$	%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: Overhead (in seconds) for large file microbenchmark, under Config-Disk (CD) and Config-RD (CRD).

The small file microbenchmark only measures the effect of writes to many small files [38]. 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 PASS. We performed the sequential-write and random-write operations of the benchmark. Both unmodified and provenance-enhanced versions of the benchmark were run, and this time, the disk write-caches were turned off to eliminate un-wanted disk-specific caching effects.

The benchmark consists of creating a 100 MB file by writing to it sequentially in 256KB chunks, followed by writing 100MB of data in 256KB units written in random locations of the file. The overheads for sequential and random writes are presented in Table 1. In both cases (*Config-Disk* and *Config-RD*), the overheads are considerably lower than the overheads reported in [38], despite the additional costs of recording all the file writes to its provenance chain.

4.2.3 Hybrid Workload Benchmark

Benchmarks like Postmark are useful due to their standardized nature and ability to replicate the results. Additionally we decided to evaluate our overheads in a more realistic scenario, involving practical, documented work-

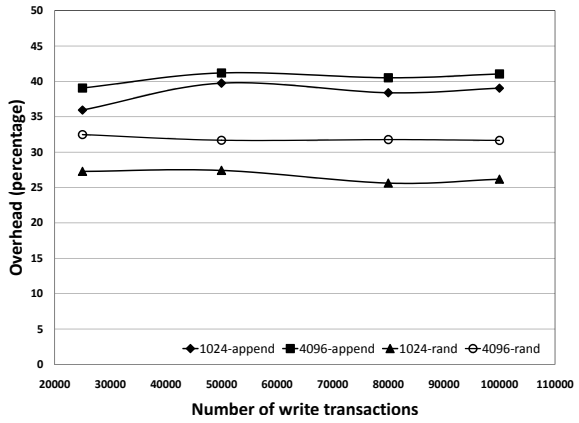

Figure 3: Overhead for different number of write transactions and data sizes, at 100% write load (*Config-Disk*).

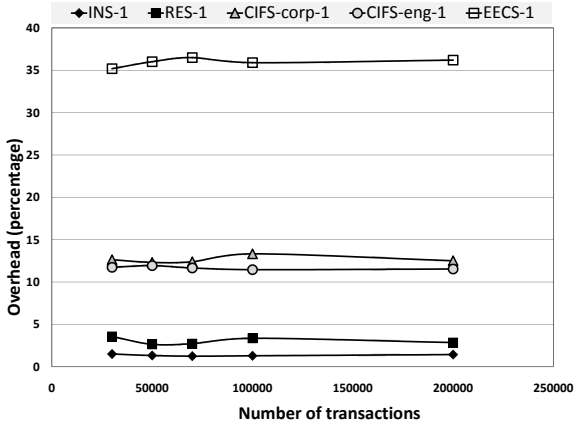

Figure 4: Overhead for various types of workloads (*Config-Disk*).

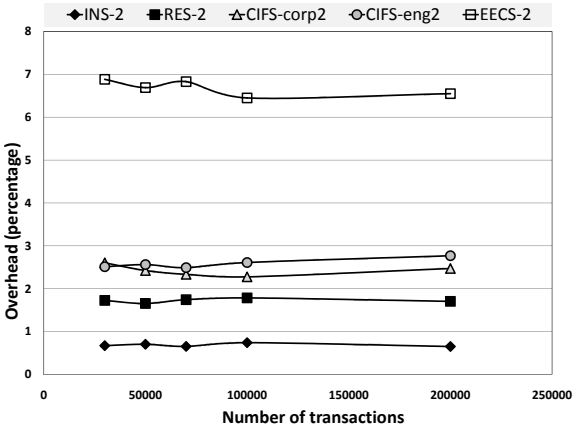

Figure 5: Overhead for various types of workloads (*Config-RD*).

loads and file system layouts. We constructed a layout as discussed by Douceur et.al [16], 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 [2, 16].

Our first workload on this dataset involved fixed number of write transactions. Under the *Config-Disk* setting, we performed 25K, 50K, 80K, and 100K write transactions. Between each 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 4KB) was written into it. We measured the overhead for both appends and random writes. These are shown in Figure 3. Constant overheads can be observed for each of the 4 configurations, with append situated between 32% and 42%, and random writes between 26% and 33%.

Next, we modeled the percentage of write to read transactions according to the data in [17, 28, 44] 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) [44], a campus home directory (EECS) [17], and CIFS corporate and engineering workloads (CIFS-corp, CIFS-eng) [28]. 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 the highest write load, with more than 80% write transactions. The results are shown in Figure 4 and 5, for both the disk-based (*Config-Disk*) and the RAM-disk optimized (*Config-RD*) modes.

Read-intensive workloads can be seen as almost overhead free, with less than 5% overheads for both RES and INS (this goes down to less than 2% for the RAMdisk-optimized mode). For write intensive workloads, the overheads are higher, but still less than 14% for the CIFS workloads (*Config-Disk*), and less than 36% for EECS (*Config-Disk*). With the RAMdisk-optimization, the overheads go down to less than 3% for CIFS and around 6.5% for EECS.

Summary. Sprov facilitates collection of provenance with integrity and confidentiality assurances, while incurring minimal overhead. Read performance is unaffected by the use of Sprov. Benchmarks show that, with the *Config-RD* setting, use of Sprov incurs an overhead less than 3% in a multitude of realistic workloads.

5 Related Work

Researchers have categorized provenance systems for science [50] and investigated the question of how to capture provenance information, typically through instrumenting workflows and recording their provenance [3, 4, 7, 37, 52, 53]. Other provenance management systems used in scientific computing include Chimera [18] for physics and astronomy, myGrid [60] for biology, CMCS [39] for chemistry, and ESSW [19] for earth science.

Another technique is to collect provenance information at the operating system layer, with the advantage of being hard to circumvent and the disadvantages of being expensive and hard to deploy. The Provenance-aware Storage System (PASS) [8, 38] takes this approach using a modified Linux Kernel. While PASS does not actually record the data written to files, it collects elaborate information flow and workflow descriptions at the OS level. Our techniques of securing provenance chains can be used to augment PASS or any such system to provide the 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., [9, 10, 11, 12, 57, 59]). Others have examined the applications of provenance to social networks [21] and information retrieval [31].

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 [8, 25]. One exception is the Lineage File System [46], 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 (I1-I7,C1-2) for confidentiality, integrity and privacy of provenance information outlined in [8, 25] and discussed in this paper.

Outside the domain of provenance, researchers have used *entanglement* – mechanisms of preserving the historic states of distributed systems in a non-repudiable, tamper-evident manner [32, 45]. This provides similar assurances to the ones sought here for the realm of systems, yet does not handle provenance for information flows and individual data records.

Source code management systems (SCM) target the provenance needs of a particular application domain. For example, Subversion [15], GIT [30], or CVS [5] with secure audit trails can provide integrity assurances for versions in a centralized file system. GIT, Monotone [1], and several other systems also provide support for a distributed infrastructure. These systems employ a logically centralized model where users maintain local histories and use a virtual (centralized) repository to merge and synchronize their local repositories. Our approach is intended for applications with a more fully decentralized model, where documents and their histories are physically passed between users in separate administrative domains that may not trust one another. In addition, as our approach is intended to meet the needs of many potential applications, we have worked to provide much higher performance than a SCM system requires

Verifiable audit trails for versioning file systems can use keyed hash-chains to protect version history [42]. Under this approach, auditors are fully trusted and share a symmetric key with the file system for creating the MACs. The audit authenticators need to be published to a trusted third party, which must provide them accurately during audits. Our approach must also handle malicious auditors who could easily falsify the audit.

Similarly to audit trails, secure audit logs based on hash chains have been used in computer forensics [47, 51]. 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. In contrast, provenance information is highly mobile and often traverses multiple un-trusted domains. Moreover, audit logs rarely require the selective confidentiality assurances needed for provenance. For example, the mechanisms proposed in [47] 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. Finally, a majority of schemes function under the assumption of single (or very few) parties processing the audit log and computing checksums – a different model from the case of provenance chains where multiple principals' access is required throughout the lifetime of a provenance chain.

Secure Untrusted Data Repository (SUNDR)[29] provides a notion of consistency for shared memory (called fork consistency) akin to the integrity property provided for provenance records in our systems. While our techniques for ensuring chain integrity are related to those used in SUNDR, the adversarial model of SUNDR is different from ours. In SUNDR, a set of trusted clients communicate with an untrusted server storing a shared filesystem. In contrast, our system does not employ a central server, and allows any number of users to be corrupted. Moreover, SUNDR does not address confidentiality issues.

Multiply-linked hash chains have been used for signature cost amortization in multicast source authentication [20, 23, 35, 41]. Our spiral chain constructs are similar in principle. One main difference however 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, the second spiral construction allows integrity-preserving compaction, which is not possible with the hash chains.

Integrity of cooperative XML updates has been discussed in [33], where document originators define a flow path policy before dissemination and recipients can verify whether the document updates happened according to this flow policy. In contrast, for flexibility and wider applicability, our model and integrity assurances do not require the existence of pre-defined flow path policies, in order to provide the integrity assurances described in Section 2.

6 Conclusion

In this paper, we introduced a cross-platform, low-overhead architecture for capturing provenance information at the application layer. Our approach provides fine-grained control over the visibility of provenance information and ensures that no one can add or remove entries in the middle of a provenance chain without detection. We implemented our approach for tracking the provenance of *data writes*, in the form of a library that can be linked with any application. Experimental results show that our approach imposes overheads of only 1–13% on typical real-life workloads.

Acknowledgments

We thank the anonymous reviewers and Alina Oprea for their valuable comments; Bill Bolosky and John Douceur for suggestions about file system distributions; and Kiran-Kumar Muniswamy-Reddy and Margo Seltzer for help with the microbenchmarks. Hasan and Winslett were supported by NSF awards CNS-0716532 and CNS-0803280. Sion was supported by NSF awards CNS-0627554, CNS-0716608, CNS-0708025 and IIS-0803197.

References

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

[2] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. In *Proc. of the 5th USENIX conference on File and Storage Technologies (FAST)*, Berkeley, CA, USA, 2007. USENIX Assoc.

[3] R. Aldeco-Perez and L. Moreau. Provenance-based Auditing of Private Data Use. In *Proc. of the BCS International Academic Research Conference, Visions of Computer Science*, Sept. 2008.

[4] R. S. Barga and L. A. Digiampietri. Automatic generation of workflow provenance. In Moreau and Foster [36], pages 1–9.

[5] B. Berliner. CVS II: parallelizing software development. In *Proc. of the Winter 1990 USENIX Conference*, pages 341–352. USENIX Assoc., 1990.

[6] M. Blum. Coin flipping by telephone. In *Proc. of Crypto*, pages 11–15, 1981.

[7] U. Braun, S. L. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer. Issues in automatic provenance collection. In Moreau and Foster [36], pages 171–183.

[8] U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *Proc. of the 3rd USENIX Workshop on Hot Topics in Security (HotSec)*, July 2008.

[9] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *Proc. of the ACM International Conference on Management of Data (SIGMOD)*, pages 539–550, New York, NY, USA, 2006. ACM Press.

[10] P. Buneman, A. Chapman, J. Cheney, and S. Vansummeren. A provenance model for manually curated data. In Moreau and Foster [36], pages 162–170.

[11] P. Buneman, S. Khanna, and W. C. Tan. Data provenance: Some basic issues. In *Proc. of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS)*, pages 87–93, London, UK, 2000. Springer-Verlag.

[12] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. *Lecture Notes in Computer Science*, 1973:316–330, 2001.

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

[14] A. Chapman, H. Jagadish, and P. Ramanan. Efficient provenance storage. In *Proc. of the ACM SIGMOD/PODS Conference*, Vancouver, Canada, 2008.

[15] B. Collins-Sussman. The subversion project: Buiding a better CVS. *Linux J.*, 2002(94):3, 2002.

[16] J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *Proc. of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 59–70. ACM Press, 1999.

[17] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS tracing of email and research workloads. In *Proc. of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, pages 203–216, Berkeley, CA, USA, 2003. USENIX Assoc.

[18] I. T. Foster, J.-S. Vockler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proc. of the 14th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 37–46, Washington, DC, USA, 2002. IEEE Computer Society.

[19] J. Frew and R. Bose. Earth system science workbench: A data management infrastructure for earth science products. In *Proc. of the 13th International Conference on Scientific and Statistical Database Management (SSDBM)*, page 180, Washington, DC, USA, 2001. IEEE Computer Society.

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

[21] J. Golbeck. Combining provenance with trust in social networks for semantic web content filtering. In Moreau and Foster [36], pages 101–108.

[22] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[23] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *Proc. of the Symposium on Network and Distributed Systems Security (NDSS)*, pages 13–22, 2001.

[24] D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *Proc. of the 22nd Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 47–60, London, UK, 2002. Springer-Verlag.

[25] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *Proc. of the ACM workshop on Storage security and survivability (StorageSS)*, pages 13–18, New York, NY, USA, 2007. ACM.

[26] J. Katcher. Postmark: a new file system benchmark. Network Appliance Tech Report TR3022, Oct 1997.

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

[28] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *Proc. of the USENIX Annual Technical Conference*, pages 213–226, Berkeley, CA, USA, 2008. USENIX Assoc.

[29] J. Li, M. N. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proc. of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 121–136, 2004.

[30] J. Loeliger. Collaborating with Git. *Linux Magazine*, June 2006.

[31] C. 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.

[32] P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *Proc. of the 11th USENIX Security Symposium*, pages 297–312, Berkeley, CA, USA, 2002. USENIX Assoc.

[33] G. Mella, E. Ferrari, E. Bertino, and Y. 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.

[34] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[35] S. K. Miner and J. Staddon. Graph-based authentication of digital streams. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 232–246, 2001.

[36] L. Moreau and I. T. Foster, editors. *Provenance and Annotation of Data, International Provenance and Annotation Workshop (IPAW)*, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.

[37] L. Moreau, P. Groth, S. Miles, J. Vazquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, A. Schreiber, V. Tan, and L. Varga. The provenance of electronic data. *Commun. ACM*, 51(4):52–58, 2008.

[38] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer. Provenance-aware storage systems. In *Proc. of the USENIX Annual Technical Conference*, pages 43–56, 2006.

[39] J. D. Myers and et al. A collaborative informatics infrastructure for multi-scale science. In *Proc. of the 2nd International Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*, page 24, Washington, DC, USA, 2004. IEEE Computer Society.

[40] National Assoc. of Insurance Commissioners. Graham-Leach-Bliley Act, 1999. `www.naic.org/GLBA`.

[41] A. Perrig, R. Canetti, D. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. of the IEEE Symposium on Security & Privacy*, pages 56–73, May 2000.

[42] Z. N. J. Peterson, R. Burns, G. Ateniese, and S. Bono. Design and implementation of verifiable audit trails for a versioning file system. In *Proc. of the 5th USENIX conference on File and Storage Technologies (FAST '07)*, pages 93–106, Berkeley, CA, USA, 2007. USENIX Assoc.

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

[44] D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In *Proc. of the USENIX Annual Technical Conference*, Berkeley, CA, USA, 2000. USENIX Assoc.

[45] D. Sandler and D. S. Wallach. Casting votes in the auditorium. In *Proc. of the USENIX Workshop on Accurate Electronic Voting Technology*, Berkeley, CA, USA, 2007. USENIX Assoc.

[46] C. Sar and P. Cao. Lineage file system. Online at http://crypto.stanford.edu/ cao/lineage.html, January 2005.

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

[48] M. Seltzer, K. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File system logging versus clustering: a performance comparison. In *Proc. of the USENIX 1995 Technical Conference*, pages 249–264, Berkeley, CA, USA, 1995. USENIX Assoc.

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

[50] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[51] R. Snodgrass, S. Yao, and C. Collberg. Tamper detection in audit logs. In *Proc. of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 504–515. VLDB Endowment, 2004.

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

[53] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. Security issues in a SOA-based provenance system. In Moreau and Foster [36], pages 203–211.

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

[55] 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.

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

[57] N. N. Vijayakumar and B. Plale. Towards low overhead provenance tracking in near real-time stream filtering. In Moreau and Foster [36], pages 46–54.

[58] D. Wichers, D. Cook, R. Olsson, J. Crossley, P. Kerchen, K. Levitt, and R. Lo. PACLs: An access control list approach to anti-viral security. In *Proc. of the 13th National Computer Security Conference*, pages 340–349, 1990.

[59] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR)*, January 2005.

[60] J. Zhao, C. A. Goble, R. Stevens, and S. Bechhofer. Semantically linking and browsing provenance logs for E-science. In *Proc. of the 1st International IFIP Conference on Semantics of a Networked World (ICSNW)*, pages 158–176, 2004.