Protecting the Past and Present of Data, with Applications in Provenance and Regulatory-Compliant Databases

Ragib Hasan
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
rhasan@cs.uiuc.edu

ABSTRACT

While the digital nature of modern information has brought enormous benefits, it has also created new vulnerabilities. Unlike physical documents, digitally stored information can be rapidly copied, erased, or tampered. Digital data may be stored in or transmitted via untrusted systems. Even insiders can have financial or strategic motives to tamper data or violate privacy and confidentiality. Access control alone often cannot provide the necessary protection to data, when insiders with superuser access rights become adversaries. Protecting the current instance of data may not be sufficient; in a distributed computing environment, we may need to know the history of a data object – i.e., its provenance – to make informed decisions about its trustworthiness. In this context, we explore techniques for providing security for data and its history in an untrusted environment. We develop schemes for providing integrity, confidentiality, and privacy assurances for data provenance, and create a fast proof-of-concept system for secure provenance. We also design and implement techniques and architectures for databases that provide protection against tampering.

1. INTRODUCTION

Almost every aspect of our lives is impacted by data, which, in many cases, comes from external sources. When data becomes mobile and can pass through untrusted environments, securing it against outsider and insider attacks can become very important. *How can we trust data in a distributed environment?* For many applications, data security can be provided by ensuring data storage mechanisms are tamper-evident, and by providing a verifiable history of the data. In other words, the trustworthiness of data can be inferred from consideration of its *present* and its *past*. If a data item is to be considered trustworthy in its *present instance*, then we must provide guarantees that it has not been tampered with since its last authorized modification. However, it is not always sufficient to guarantee that the data item is trustworthy; it may be untampered, yet be derived from untrusted inputs. Thus we may also need to provide guarantees that the data item's past history (i.e., its life cycle through its many transformations, transmissions, and modifications) is properly documented, can be verified, and is trustworthy. That is, the *history* of the data item – its derivation, lineage, and propagation – may need to be inspected to infer and evaluate its trustworthiness. We illustrate these two aspects of data integrity using the following two scenarios:

Scenario 1. *Alice commits a transaction in a financial database. However, after some time, she regrets her actions, and wants to hide the existence of the transaction. Bob queries the database. Before he uses the query results for other purposes, he wants to check whether the data set has been modified since it was committed. Access control and/or integrity mechanisms are not enough to protect integrity since Alice might have superuser access to the database server.*

Challenge 1. We need an efficient scheme for detecting tampering of the database tuples, even when adversaries can be insiders with superuser privileges.

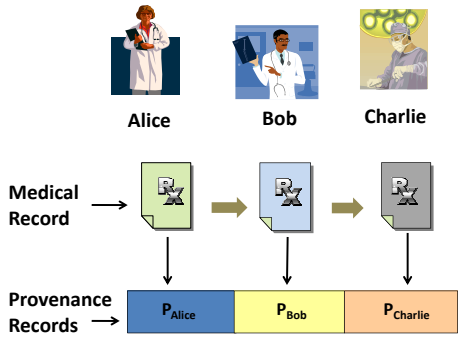

Figure 1: Provenance example scenario

Scenario 2. *Dana visits Dr. Alice for treatment. Alice refers her to the pathologist Dr. Bob, who performs some tests, and forwards the results to Dr. Charlie (Figure 1). However, Dana later suffers health problems due to misdiagnosis, and sues Dr. Charlie. To prove his innocence, Charlie wants to prove that the incorrect test results were introduced into Dana's medical records by Dr. Bob. However, Bob denies this, and claims to have provided the correct test results. To complicate matters, Bob alleges that Charlie and Alice are colluding to frame him by tampering the modification logs for Dana's medical record.*

Challenge 2. For data objects that move around in a distributed, untrustworthy environment, we need to provide a scheme to protect integrity of the history of the object, i.e. its provenance.

Research on data provenance, i.e., the lineage and derivation of data, has resulted in systems that record workflows, file modifications, and other aspects of data [13, 14, 28, 29, 37]. However, most of these systems fail to address security concerns. Unless we can

Proceedings of the Third SIGMOD PhD Workshop on Innovative Database Research (IDAR 2009) June 28, 2009, Providence, RI, USA

ensure the integrity, confidentiality, and privacy of provenance, we cannot rely on its use to evaluate trustworthiness of data.

In recent years, a series of scandals caused governments worldwide to impose new regulations on how data is stored, archived, and deleted in financial and health related institutions [7, 10, 40]. These regulations mandate protection of a data item's *present*, i.e., prevention of unauthorized modifications once it has been committed to storage. Violations can result in monetary fines and criminal prosecutions. While systems have been developed for compliance with these regulations, they do not provide the complete integrity and privacy guarantees that one may envision from reading the regulations. Most of the regulations focus on structured records, typically stored in databases, while existing solutions for regulatory compliance are coarse-grained – designed for unstructured data stored in files.

In this dissertation, we address the data security problem along these two dimensions. Our *hypothesis* can be stated as follows:

It is possible to provide integrity, confidentiality, and privacy guarantees for the past and present of a data item with minimal overhead and cost.

To test and evaluate this hypothesis, we first explore techniques for protecting *data provenance* – i.e., the derivation and lineage history of data – from illicit modifications. Second, we look into schemes for protecting the current instance of data in storage systems against tampering. In both cases, we aim at creating efficient and secure schemes for protecting past history and current state of data.

Contributions. The research contributions of my dissertation are as follows:

1. We provide an efficient scheme for securing provenance information, i.e., for providing integrity, confidentiality, and privacy assurances for provenance.
2. We design and implement a system for providing secure provenance for file systems and databases.
3. We develop and implement efficient models for databases that provide term-immutability for tuples (i.e. providing tamper evidence guarantees for data for a certain retention period, and allowing deletion after that).
4. We devise schemes for combining support for shredding of data with litigation hold support in databases that support term-immutability for tuples.

2. SECURE PROVENANCE

Provenance summarizes the history of the ownership of items and the actions performed on them. Traditionally, people have used provenance to authenticate physical objects in arts, archives, and archaeology. But life today has become increasingly dependent on digital information that originated elsewhere, was processed by other people, and was stored in untrustworthy storage. So, it is increasingly important to know the source of the information and its derivation history. In other words, to be able to trust a piece of information, we need to know and verify its provenance.

Data provenance has been defined in many ways, but all definitions share the same core concept: it is a description of the origin, derivation, and transmission history of data. Till now, scientists have been the primary users of data provenance systems. Provenance research has mainly focused on the tasks of modeling, collection, annotation, and querying. But as provenance steps into mainstream computing, new challenges arise. With its increased use in financial, medical, and other non-scientific application areas, provenance information faces a host of security threats, including active attacks from adversaries. In high-stakes business and medical applications, insiders may have significant incentives to alter data records' history. As information crosses application and organizational boundaries and passes through untrusted environments, its provenance becomes vulnerable to illicit alteration. When the trustworthiness of the provenance records themselves is in question: we need *provenance of provenance*, i.e., a model for secure provenance. Making provenance records trustworthy is challenging. Ideally, we need to guarantee *completeness* – all relevant actions pertaining to a piece of information are captured; *integrity* – adversaries cannot forge or alter provenance records; *availability* – auditors can verify the integrity of provenance information; *confidentiality* – only authorized parties can read provenance records; and *efficiency* – provenance mechanisms must have low overheads.

In this dissertation, we have taken the first step towards preventing forgery of history as stored in provenance records. In [18], we have presented a scheme for providing integrity and confidentiality assurances for provenance records, and described a proof-of-concept implementation for file systems that imposes only 1% to 13% overhead for typical real-life workloads.

2.1 Usage and Threat Model

In our model for secure provenance [17, 18], we use the term **document** to refer to the data item for which provenance information is collected, such as a file, database tuple, or network packet. 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 metadata. Each organization has one or more **auditors**, who are principals authorized to access and verify the integrity of provenance records associated with documents. Documents move from one user to another, as email attachments, file transfers, 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 some 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.

Perfect provenance tracking is impossible, as 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. 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 ability to track the origin of data by monitoring read operations is limited. Also, promulgation of provenance for read operations can result in a combinatorial explosion in overhead that can make the system unusable [29]. So, in our work, we target applications

that do not require tracking of reads.

We provide the following integrity (I1-6) and confidentiality assurances (C1-2) with respect to forgery of document history [18].

- **I1** An adversary acting alone cannot selectively add or remove other principals' records from the beginning or middle of a provenance chain without being detected at the next audit.
- **I2** Two colluding adversaries who have contributed records to a provenance chain cannot add records of non-colluding users between theirs, without being detected by the next audit.
- **I3** Once the chain contains subsequent records by non-colluding parties, two colluding adversaries who have contributed records to a provenance chain cannot remove the record of any non-colluding user between theirs, without being detected by the next audit.
- **I4** Users cannot repudiate chain records.
- **I5** An adversary cannot claim that a valid provenance chain for one document belongs to a document with different contents, without detection by the next audit.
- **I6** If an adversary alters a document without appending the appropriate provenance record to its chain, this will be detected by the next audit.
- **C1** Any auditor can verify the integrity of the chain without requiring access to any of its confidential components. Unauthorized access to confidential 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.

2.2 Our Solution

We propose a solution composed of several layered components: encryption for sensitive provenance record fields, a checksum-based approach to ensure provenance record integrity, and an incremental chained signature mechanism for securing the integrity of the chain as a whole. For confidentiality, we deploy a special keying scheme based on broadcast encryption key management to selectively regulate the access for different auditors. To provide fine-grained confidentiality, we use a cryptographic commitment based construction.

More precisely, each provenance record P_i summarizes a sequence of one or more actions taken by a user:

$$P_i \quad = \quad \langle U_i, W_i, hash(D), K_i, C_i, [public_i] \rangle,$$

where

- U_i is a plaintext identifier for the user;
- W_i is an encrypted or plaintext representation of the sequence of actions (the *modification log*) performed by the user;
- $hash(D)$ is a one-way hash of the current content of the document;
- K_i contains keying material that auditors can use to decrypt the encrypted fields, as explained below;
- C_i contains an integrity checksum (defined below) for this provenance record, signed by user U_i;
- $public_i$ is an optional encrypted or plaintext public key certificate for user U_i.

We briefly discuss each of the record's fields below. Further details and correctness proofs of this scheme can be found in [17, 18].

Confidentiality: Certain fields of a provenance record may be sensitive, such as the identity or individual steps of a proprietary process used to clean experimental data. If all users trusted all auditors, then we could just encrypt the fields with a single public key, and give the private key to the auditors. If a user trusted only certain auditors, we could make several copies of the sensitive fields, encrypt each copy with the public key of a different trusted auditor, and include all of them in the new provenance record. While secure, this wastes space. Instead, we encrypt the sensitive fields once with a new secret key, make multiple copies of the secret key, and encrypt each copy with the public key of a different trusted auditor. In this scheme, the new provenance record contains the encrypted sensitive fields plus several versions of the encrypted secret key, stored in the K_i field of the record. A trusted auditor can subsequently read the record, decrypt a copy of the secret key using the auditor's private key, and use the secret key to decrypt the sensitive fields. If there are many auditors, the record can be kept small by using a broadcast encryption tree to reduce the number of keys.

Integrity: An audit must detect whether adversaries have removed or inserted elements from the chain, and whether a chain has been switched from one document to another. To achieve this, the checksum C_i of a provenance record P_i is computed follows. First we apply a cryptographic hash function to the tuple containing the user identity U_i, the hash of the document contents $hash(D)$, the modification log W_i, the key-related information K, and (if included in the record) the user's public key $public_i$. Then we concatenate the resulting hash with the checksum C_{i-1} of the previous provenance record P_{i-1}, sign the result with the user's private key, and store the signed result in the provenance record. More formally, the integrity checksum field C_i is:

$$C_i = S_{private_i}(hash(U_i, W_i, hash(D), K_i, [public_i])|C_{i-1}),$$

where $S_{private_i}$ means that user U_i signs the hash with his or her private key.

Fine-Grained Control Over Confidentiality: Some portions of a provenance record may be quite sensitive. We cannot simply remove every record containing sensitive information, as that will break provenance integrity checks. Encrypting the entire record just to hide a small piece of sensitive information is excessive. So, we replace each sensitive field by a *cryptographic commitment* for it, e.g., by appending a random number to the contents of the field and then hashing the result. We encrypt those random numbers with a secret key and leave them in the record for trusted auditors to use. When we compute the checksum, we use the commitments in place of the sensitive fields, and include the encrypted random numbers. The official provenance record includes the sensitive *and* non-sensitive fields, the commitments for the sensitive fields, the encrypted random numbers, and the checksum. When releasing the provenance chain, we can remove the sensitive fields, and subsequent integrity checks for the chain and document will still work correctly.

2.3 Implementation and Evaluation

To evaluate the performance of our secure provenance scheme, we designed and implemented a proof-of-concept prototype that provides secure provenance for files. Our application-layer C library called SPROV, consists of wrapper functions for the standard file I/O library *stdio.h*.

We tested the performance of SPROV on a workstation with a Pentium 4 CPU at 3.4GHz, 2GB RAM, running Linux. We used two modes for storing provenance chains – in the *Config-Disk* mode, the chains were stored on the disk, while in the *Config-RD* mode, we buffered the chains in a RAM disk, and periodically flushed them to disk using a daemon.

Detailed performance results from the PostMark Benchmark [22] and realistic workload benchmarks are available in [18]. In brief, we showed that by modifying only 8 lines of PostMark source code, we could provide secure provenance with the run-time overheads ranging from 0.5% to 10% for typical workloads. To evaluate per-

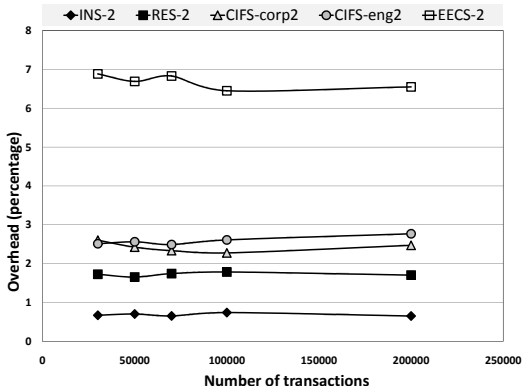

Figure 2: Effect of different workloads. with (*Config-RD*) setting.

formance in a realistic setting, we generated a distribution of 20,000 files, with a median file size of 4KB and a mean file size of 80KB, using the log-normal model for file sizes as shown by Douceur et al. [11]. Our simulations used write and read transaction percentages matching five studies of real-world file system workloads with the write percentage ranging from 1.1% to 82.3%. These workloads came from an instructional (INS) and research (RES) setting [34], a campus home directory (EECS) [12], and CIFS corporate and engineering workloads (CIFS-corp, CIFS-eng) [23]. As shown in Figure 2, for typical workloads, we achieved less than 3% overheads with the *Config-RD* setting. Further details about these simulations are available in [18].

3. PROTECTING THE PRESENT THROUGH TERM-IMMUTABLE DATABASES

While secure provenance prevents forgery of the *past* of data, there are many scenarios where we need to ensure the protection of the *present* incarnation of data. That is, we need to ensure that once data is entered into a storage system, non-authorized changes and tampering can be detected and/or prevented. This expectation of preserving the state of data once committed to storage is the focus of a number of government regulations all around the world. In this section, we look into the challenge of protecting integrity of data in storage, in the context of databases that support term-immutability.

3.1 Problem Background

Recent regulations require many corporations to ensure trustworthy long-term retention of their routine business documents during their multi-year retention periods. Examples include the Gramm-Leach-Bliley Act [9], the Sarbanes-Oxley Act [10], and Securities and Exchange Commission Rule 17a-4 [40]. To satisfy the requirements, vendors such as IBM, EMC, NetApp and others sell compliance storage servers [21, 30] that provide an approximation of write-once read-many (WORM) storage to ensure that files committed to the WORM device are read-only during their retention periods ("term-immutable"), and cannot be deleted or altered even by a system administrator inside the company or a hacker with administrative privileges. Some products also support eradication of files at the end of their retention periods and can enforce "litigation holds" to ensure that subpoenaed files are not destroyed. All these products work at the granularity of files, and their security guarantees focus on providing term-immutability for files. However, a large portion of corporate data is stored and managed at much smaller granularities, in the form of relational tuples.

Can we leverage existing compliance storage systems to support term-immutable tuples? There are several options: we can make every tuple a separate file, but that would kill performance. Next, we can treat the entire database as an ordinary document, and when a transaction commits, we can save the database to compliance storage. This would also have bad performance, because every transaction must write another term-immutable copy of the database. Yet another approach is to move the DBMS code into the WORM storage server, which we trust. Then performance will be good, but this approach is fundamentally unacceptable from a security perspective: the trusted code base needs to be as small as possible. To the maximum extent possible, a compliant DBMS must consist of untrusted code that communicates with the (trusted) storage server over as narrow an interface as possible.

To address these problems, we develop novel DBMS architectures and techniques for providing term-immutability and tuple shredding, without sacrificing high efficiency. We also engineer component interactions to ensure that term-immutability is supported throughout a tuple's life cycle.

3.2 Goals

Many regulations require that certain kinds of information be retained for a specified term, e.g., seven years. Once such information has been recorded, no one should be able to alter or delete it until its retention period is over. We refer to this as *term-immutability*. After the retention period has ended, some regulations require that the information be destroyed; others permit it to remain, though often organizations are highly motivated to go ahead and destroy it, so that they can limit the resources that they must devote to record-keeping.

Our goal here is to design database systems that provide term-immutability for tuples. In particular, we want to provide the following guarantees:

- Any tampering of data already committed to the database will be detected by an auditor during an audit.
- After its retention period has passed, all evidence that a tuple ever existed can be removed from the database, unless the tuple is subject to a litigation hold.
- A tuple will be retained in the database as long as it is subject to a litigation hold, even if its retention period is over.

3.3 Usage and Threat Model

In the compliance database model, database *users* access databases to query or add/remove tuples. The users may have superuser privileges, and/or direct access to database storage. *Adversaries* may be insiders, or outside attackers who gain superuser privileges. We assume that the data query and update process is trustworthy.

In the *regret-based threat model* [27], the main threat is that the person writing the tuples stores them appropriately, but later *regrets* her actions, and wants to surreptitiously modify the database. We assume that when an adversary (Malory) starts to regret the existence of a database tuple t, he may take over root on the platform where the DBMS runs and issue any possible command to the storage server, in an attempt to rewrite history with respect to t. These attacks can target any database file: data, indexes, logs, metadata. Malory can change the code running on the DBMS platform, assume the identity of any user, and issue any DBMS command. If we can protect the database against an adversary this powerful, the database will also be safe from less powerful foes. Malory might instead choose to strike after t has expired and been deleted from the database. At that point, he can look for any evidence of t's prior existence in the files containing the database data, indexes, logs, or metadata. We must ensure that he cannot find out more about t than he could by random guessing. Next, Malory may try to rewrite history in a way that shows that certain tuples were inserted into the

database, or certain updates performed, when in fact they were not.

Compliance data is subject to periodic audits. During audits, any clear indications of tampering will lead to automatic presumption of guilt and the imposition of hefty fines for the attackers. So, the ultimate goal of the attacker is to tamper with the records in an undetectable manner. In particular, guarding against vandalism is not a major concern; traditional techniques can be used to help address this, such as regular backups that are stored at a remote site. Another threat concerns data reaching the end of its retention period.

Removal of expired data that is not subject to a litigation hold is required by some regulations, and is a prudent corporate goal. The goal of compliance research on data shredding is to ensure that an attacker can do no better than random guessing as to the content of shredded data. Compliance data is also subject to all the other threats faced by ordinary data. Addressing ordinary threats to DBMS data is outside the scope of our work.

3.4 Research Directions

To reach our goals, we identified the following research directions.

3.4.1 Designing Efficient Term-immutable Databases

The primary research direction we explore is the design of an efficient term-immutable database. Till now, there is only one implementation of term-immutable database [27], which uses a WORM device to store transaction log tail and a special compliance log that contains record of new tuple insertions. However, as we will later discuss in Section 4, this architecture has significant overhead in terms of run-time performance and audit length. The regret interval is also high (10 minutes), which is unacceptable in many cases.

We have designed and implemented an improved architecture for term-immutable databases. Preliminary performance evaluation (details are under submission, and hence withheld here) shows that our new architecture is two orders of magnitude faster than [27] in terms of audit time, and has significantly less overhead.

3.4.2 Leveraging Trusted Hardware

A secure co-processor (SCPU) is a potential home for the integrity assurance mechanism; it can improve security guarantees (e.g., by ensuring that query answers are always correct) and even improve performance (e.g., by eliminating the need for a separate validation activity). Unfortunately, if we simply run the integrity assurance functions on the SCPU, we will ruin DBMS performance. Heat concerns limit SCPU speed and memory. So, we can use the SCPU only as a trusted aide for the main CPU. We offload parts of the integrity assurance functions to the SCPU while preserving DBMS efficiency by (i) ensuring that the SCPU is accessed sparsely, minimizing the associated overhead for expected transaction loads, and (ii) using adaptive overhead-amortized constructs to enforce term-immutability at the throughput rate of the server's ordinary processors during burst periods. Therefore, one avenue of research we explore is to leverage trusted hardware, specifically SCPUs in the design of term-immutable databases, and evaluate the performance, cost, and security implications of each of those choices.

3.4.3 Shredding and Litigation Holds

We need to shred expired tuples and be able to prove that no tuple subject to a litigation hold has been shredded. Borisov *et al.* provided a solution for a similar problem of shredding of index entries for compliance file storage [4]. Our task is also related to vacuuming, which helps transaction-time databases move old data to archival storage [39] or remove obsolete information [35]. The emphasis of much of this prior research has been on how to indicate to the user what data is missing due to vacuuming, which is exactly the opposite of our challenge: how to ensure that the user cannot detect that the data was ever stored. Our task is made more challenging by the need to support fine-grained deletion of tuples and index entries; compliance storage servers only make this task harder, because they associate a retention period with an entire file. We investigate ways of tracking retention periods at a fine granularity, and supporting litigation-holds.

3.4.4 Term-immutability without WORM

The work to date on supporting term-immutability in databases assumes the presence of a trustworthy WORM server [27]. However, the "WORM" nature of these devices as they are sold today is actually a misnomer; most of these systems are based on magnetic media, and a skillful and determined/bribed system administrator can, given sufficient time, subvert them. In Section 2, we tackled a similar problem of securing data provenance against adversaries with superuser privileges. Similar cryptographic techniques may be used to provide term immutability for databases. The main challenges will be performance – as databases typically handle hundreds of thousands of transactions per second. Any cryptographic solution will have to be fast enough to sustain performance, yet provide strong security. In this direction, we design approaches to providing term-immutability for tuples using cryptographic techniques, without relying on a WORM device.

4. RELATED WORK

4.1 Provenance

Over the years, researchers have developed provenance systems for scientific computing [37]. Researchers have explored the question of how to capture provenance information, typically relying on workflow instrumentation [28]. Provenance management systems used by scientists include Chimera for physics and astronomy, myGrid for biology, CMCS for chemistry, and ESSW for earth science [37]. Other efforts propose to collect provenance information within databases [6], social networks [15], and file systems [29]. While research efforts till now have explored the challenges of collecting, storing, representing, annotating, and querying provenance data, little has been done to secure that information [5, 17].

Researchers have proposed the use of *entanglement* to preserve distributed systems' history in a non-repudiable, tamper-evident manner [25]. Provenance-related information is supported by source code management systems such as SVN [8], GIT [24], CVS [3], and Monotone [1]; versioning file systems [33]; and *secure audit forensic logs* [36], to provide integrity assurances for subsets of system and data state in a logically centralized authority model. Our work [17, 18, 19] targets provenance information that is highly mobile and may traverse multiple untrusted domains, with no logically centralized repository or authority. Work on audit logs typically secures logs as a whole, but does not allow authentication of individual modifications. Because provenance information is associated with a digital object such as a file, this introduces attacks that are not applicable to secure audit logs. Finally, secure audit log schemes typically assume that at most a handful of parties will process the data and compute checksums, whereas multiple principals' access is required throughout the lifetime of a provenance chain.

4.2 Term-immutability in Databases

Many storage vendor offers magnetic disk or tape-based WORM archival storage products that provides term-immutable storage for files (documents) [30]. Some WORM products are strengthened

with secure coprocessors (SCPU), such as the IBM 4764 PCI-X cryptographic coprocessor [20]. Unfortunately, SCPUs are an order of magnitude slower than ordinary processors. However, the main shortcoming of WORM storage servers is that they provide protection at a file granularity, making them unsuitable for providing term-immutability of the small tuples stored in a relational database. We also cannot store B-trees on WORM, as it would be very slow and inefficient to remove a B-tree entry on WORM when the tuple that it references is deleted. We cannot gradually fill in data pages or hash tables when stored on WORM. We cannot even initialize a page that is committed, as initialization will constitute its only write. Current WORM storage servers also do not allow one to append to files that have been committed to the server, but to support term-immutability in databases, we need to be able to append to log files at a minimum.

Previous research on compliance storage has primarily focused on developing trustworthy indexes for unstructured compliance data [26, 32, 41]. These are quite efficient for keyword search of unstructured data, but not very useful for supporting term-immutability of structured relational data.

The first step toward building a compliance database was taken in [27], where term-immutability is provided by keeping the latest database log in WORM storage while keeping the actual database in regular non-WORM storage. Periodically, an auditor verifies that the database state is consistent with the transaction logs since the last audit, and certifies the state, after which the transaction logs are expired from the WORM device. However, this scheme introduces 10% run-time overhead in transaction throughput. Audits are also slow, taking about 2 weeks for yearly audits. Our ongoing work proposes to overcome these two shortcomings.

Many researchers have explored data privacy and query correctness of outsourced databases, where the database is stored on an untrusted machine [16]. Such techniques typically rely on a trusted data owner who encrypts and signs the data and index entries appropriately. However, in a compliance scenario, the *data owner* is a potential adversary who may have strong incentives to alter and re-sign data and indexes.

Fine-grained retention periods for compliance storage are discussed in [2]. Tamper proof audit logs are discussed in [31, 38]. Here, hash of database state is periodically signed by a trusted external digital notary, and is stored within the DBMS. A separate validator checks the database state against these signed hashes to detect any compromise of the audit log. However, the use of an external notary introduces large notarization intervals, during which adversaries can tamper content undetectably.

5. CONCLUSION

In this paper, we presented our research on integrity and trustworthiness of data. Our scheme for preventing history forgery with secure provenance is efficient and practical, as shown in [18]. The secure provenance library SPROV can be added to provenance-unaware applications with only minimal change. Our future research directions in provenance include extending and generalizing provenance into the general notion of remembrance, as discussed in [19]. Our ongoing work on regulatory compliant databases includes design and evaluation of efficient architectures for term-immutable databases, leveraging the already-widespread WORM devices as well as secure co-processors.

6. REFERENCES

[1] Monotone Distributed Version Control. `http://www.monotone.ca/`.

[2] A. Ataullah, A. Aboulnaga, and F. Wm.Tompa. Records retention in relational database systems. In *CIKM*, 2008.

[3] B. Berliner. CVS II: parallelizing software development. In *Winter USENIX Conference*, 1990.

[4] N. Borisov and S. Mitra. Restricted queries over an encrypted index with applications to regulatory compliance. In *ACNS*, 2008.

[5] U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *USENIX HotSec*, 2008.

[6] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD*, 2006.

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

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

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

[10] Congress of the United States. Sarbanes-Oxley Act. `http://thomas.loc.gov`, 2002.

[11] J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *SIGMETRICS*, 1999.

[12] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS tracing of email and research workloads. In *USENIX FAST*, 2003.

[13] I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM*, 2002.

[14] J. Frew and R. Bose. Earth system science workbench: A data management infrastructure for earth science products. In *SSDBM*, 2001.

[15] J. Golbeck. Combining provenance with trust in social networks for semantic web content filtering. In *IPAW*, 2006.

[16] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in database service provider model. In *SIGMOD*, 2002.

[17] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *ACM StorageSS*, 2007.

[18] R. Hasan, R. Sion, and M. Winslett. The case of the fake Picasso: Preventing history forgery with secure provenance. In *USENIX FAST*, 2009.

[19] R. Hasan, R. Sion, and M. Winslett. Remembrance: The unbearable sentience of being digital. In *CIDR*, 2009.

[20] IBM. IBM Cryptographic Hardware. `http://www-03.ibm.com/security/products/`, 2006.

[21] IBM. IBM TotalStorage Enterprise. `http://wwww-03.ibm.com/servers/storage`, 2007.

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

[23] A. Leung, S. Pasupathy, G. Goodson, and E. Miller. Measurement and analysis of large-scale network file system workloads. In *USENIX Annual Technical Conference*, 2008.

[24] J. Loeliger. Collaborating with GIT. *Linux Magazine*, June 2006.

[25] P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *USENIX Security*, 2002.

[26] S. Mitra, W. W. Hsu, and M. Winslett. Trustworthy keyword search for regulatory-compliant record retention. In *VLDB*, 2006.

[27] S. Mitra, M. Winslett, R. T. Snodgrass, S. Yaduvanshi, and S. Ambokar. An architecture for regulatory compliant databases. In *ICDE*, 2009.

[28] 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.

[29] K.-K. Muniswamy-Reddy, D. Holland, U. Braun, and M. Seltzer. Provenance aware storage systems. In *USENIX Annual Technical Conference*, 2006.

[30] Network Appliance. SnapLock. `http://www.netapp.com/us/products/protection-software/snaplock.html`, 2007.

[31] K. Pavlou and R. T. Snodgrass. Forensic analysis of database tampering. In *SIGMOD*, 2006.

[32] J. Pei, M. Lau, and P. Yu. Ts-trees: A non-alterable search tree index for trustworthy databases on write-once-read-many (worm) storage. In *AINA*, 2007.

[33] Z. Peterson, R. Burns, G. Ateniese, and S. Bono. Design and implementation of verifiable audit trails for a versioning file system. In *USENIX FAST*, 2007.

[34] D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In *USENIX Annual Technical Conference*, 2000.

[35] A. Schmidt, C. S. Jensen, and S. Saltenis. Expiration times for data management. In *ICDE*, 2006.

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

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

[38] R. Snodgrass, S. Yao, and C. Collberg. Tamper detection in audit logs. In *VLDB*, 2004.

[39] M. Stonebraker and L. A. Rowe. The design of POSTGRES. *SIGMOD Rec.*, 15(2):340–355, 1986.

[40] The U.S. Securities and Exchange Commission. Rule 17a-3&4, 17 CFR Part 240: Electronic Storage of Broker-Dealer Records.

[41] Q. Zhu and W. Hsu. Fossilized index: The linchpin of trustworthy non-alterable electronic records. In *SIGMOD*, 2005.