Do You Know Where Your Data's Been? – Tamper-Evident Database Provenance

Jing Zhang[1] Adriane Chapman[2] Kristen LeFevre[1]

[1] University of Michigan, Ann Arbor, MI 48109 {jingzh, klefevre}@umich.edu
[2] The MITRE Corporation, McLean, VA 22102 achapman@mitre.org

Abstract. Database provenance chronicles the history of updates and modifications to data, and has received much attention due to its central role in scientific data management. However, the use of provenance information still requires a leap of faith. Without additional protections, provenance records are vulnerable to accidental corruption, and even malicious forgery, a problem that is most pronounced in the loosely-coupled multi-user environments often found in scientific research. This paper investigates the problem of providing integrity and tamper-detection for database provenance. We propose a checksum-based approach, which is well-suited to the unique characteristics of database provenance, including non-linear provenance objects and provenance associated with multiple fine granularities of data. We demonstrate that the proposed solution satisfies a set of desirable security properties, and that the additional time and space overhead incurred by the checksum approach is manageable, making the solution feasible in practice.

1 Introduction

Provenance describes the history of creation and modification of data. Problems of recording, storing, and querying provenance information are increasingly important in data-intensive scientific environments, where the value of scientific data is fundamentally tied to the method by which the data was created, and by whom [3, 7, 10, 11, 14, 19, 29]. In de-centralized and multi-user environments, we observe that individuals who obtain and use data (*data recipients*) often still need to make a leap of faith. They need to trust that the provenance information associated with the data accurately reflects the process by which it was created and refined. Unfortunately, provenance records can be corrupted accidentally, and they can even be vulnerable to malicious forgery.

To this point, integrity concerns have been largely ignored for database provenance. While recent work considered a similar problem in the context of file systems [22], the proposed solutions are not directly applicable to databases. In particular, Hasan et al. [22] only considered provenance that could be expressed as a totally-ordered chain of operations on an atomic object (e.g., a file). In databases, however, we observe that provenance is often expressed in terms of a partially-ordered set of operations on compound objects (e.g., records, tables, etc.). This is best illustrated with an example.

Example 1. A pharmaceutical company, TrustUsRx, wants to show that their new drug is safe and effective. TrustUsRx delivers the result of their clinical trial (with accompanying provenance information) to the FDA for approval. The provenance information indicates that the patients' ages and weights were originally collected by PCP Paul. Endocrine activity measurements were produced by the Perfect Saints Clinic, but then PCP Pamela

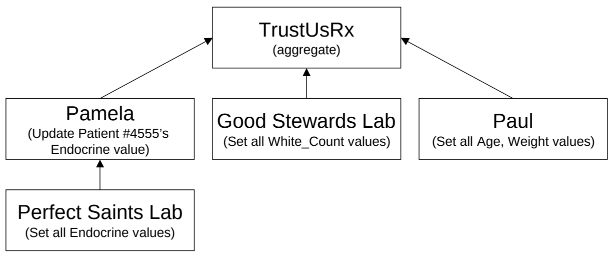
TrustUsRx
(aggregate)
Pamela
(Update Patient #4555's Endocrine value)
Good Stewards Lab
(Set all White_Count values)
Paul
(Set all Age, Weight values)
Perfect Saints Lab
(Set all Endocrine values)

Fig. 1. Sample Provenance Scenario

amended the Endocrine value for patient #4555. White blood cell counts were determined by blood samples sent to GoodStewards Labs. Finally, all of the patient data was aggregated by TrustUsRx. The provenance of this final aggregate data is shown in Figure 1. Given the company's pecuniary incentives, the FDA wants to verify that this provenance information has not been tampered with or forged.

This example highlights the two major problems that are not addressed by Hasan et al. [22]. First, each patient record is a compound object; it contains several attributes (e.g., Age, Weight, Endocrine, and White_Count), which were obtained through different methods, and have different provenance. Thus, we cannot treat records or tables as atomic; instead, a fine-grained approach is needed. Second, the modifications to the data do not form a totally-ordered (*linear*) sequence of operations (reads, writes, and updates). Instead, due to aggregation operations (e.g., the aggregation performed by TrustUsRx), the provenance associated with the final (compound) object delivered to the FDA is actually a DAG (*non-linear provenance*).

Throughout this paper, we will consider an abstract set of *participants* (users, processes, transactions, etc.) that contribute to one or more data objects through insertions, deletions, updates, and aggregations [5, 7, 9, 17, 19]. Information about these modifications is collected and stored in the form of *provenance records*. Various system architectures have been proposed for collecting and maintaining provenance records, from attaching provenance to the data itself as a form of annotation [5, 7] to depositing provenance in one or more repositories [10, 11, 14, 19, 29]. Thus, one of our chief goals is to develop a cross-platform solution for providing tamper-evident provenance. Since provenance is often collected and shared in a de-centralized and loosely-organized manner, it is impractical to use secure logging tools that rely, for example, on trusted hardware [32] or other systems-level assumptions about secure operation [37].

Occasionally, a *data recipient* will request and obtain one or more of these data objects. In keeping with the vision of provenance, each data object is accompanied by a *provenance object*. Our goal is to collect enough additional information to provide *cryptographic proof* to the data recipient that the provenance object has not been maliciously altered or forged.

1.1 Contributions & Paper Overview

This is the first in-depth study of integrity and tamper-evidence for database provenance. While related work has focused on security (integrity and confidentiality) for file system provenance [22], we extend the prior work in the following important ways:

– **Non-Linear Provenance:** Database operations often involve the integration and aggregation of objects. One might consider treating an object produced in this way as if

it were new (with no history), but this discards the history of the objects taken as input to the aggregation. Thus, in databases it is common to model provenance in terms of a DAG, or *non-linear* provenance.
– **Compound Objects:** In databases, it is critical to think of provenance associated with multiple granularities of data, rather than to simply associate provenance with atomic objects. For example, in the relational data model, each table, row, and cell has associated provenance, and the provenance of these objects is inter-related.

The remainder of this paper is organized as follows: In Section 2, we lay the groundwork by describing the database provenance model and integrity threat model. We then develop tamper-evident provenance tools for atomic and compound objects (Sections 3 and 4). Finally, an extensive performance evaluation (Section 5) indicates that the additional time and space overhead required for tamper-evidence (beyond that of standard provenance tracking) is often small enough to be feasible in practice.

2 Preliminaries

We begin with the preliminary building blocks for our work, which include the basic provenance model and integrity threat model. Throughout this paper, we will consider a database, $\mathbb{D}$, consisting of a set of data objects. Each object has a unique identifier, which we will denote using a capital letter, and a value. We will use the notation $A.val$ to refer to the current value of object A. We assume that the database supports the following common operations:

– **Insert(**A**,** val**)**: Add a new object A to $\mathbb{D}$ with initial value val.
– **Delete(**A**)**: Remove an existing object A from $\mathbb{D}$.
– **Update(**A**,** val'**)**: Update the value of A to new value val'.
– **Aggregate(**$\{A_1, ..., A_n\}$**,** B**):** Combine objects $A_1, ..., A_n$ to form new object B.

2.1 Provenance Model

With the exception of deletion, each operation is documented in the form of a *provenance record*. (For the purposes of this paper, after an object has been deleted, it's provenance object is no longer relevant[3].) We model each provenance record as a quadruple of the form $(seqID, p, \{(A_1, v_1), ..., (A_n, v_n)\}, (A, v))$. p identifies the participant who performed the operation. $\{(A_1, v_1), ..., (A_n, v_n)\}$ describes the (set of) input object(s), and their values. (A, v) describes the output object and its value.[4] $seqID$ is necessary to describe the relative order of provenance records associated with specific objects. In particular, if two provenance records $rec1$ and $rec2$ involve the same object (with the same id) as either input or output, then $rec1.seqID < rec2.seqID$ indicates that the operation described by $rec1$ occurred before the operation described by $rec2$.

[3] This is not essential, but does enable some optimizations

[4] This is certainly not the only possible way of describing an operation. We selected this model for the purposes of this work because we found it to be quite general. In contrast to provenance models that *logically* log the operation that was performed (e.g., a selection, or a sum), this simple model captures black-box operations (e.g., user-defined functions) and even non-deterministic functions. On the other hand, our proposed integrity scheme is easily translated to a provenance model that simply logs the white-box operations that have been performed.

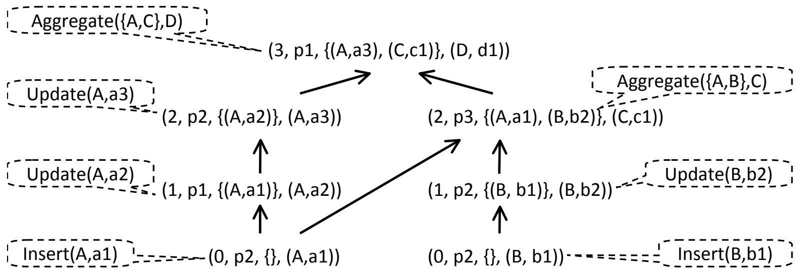
Aggregate({A,C},D)
(3, p1, {(A,a3), (C,c1)}, (D, d1))
Update(A,a3)
(2, p2, {(A,a2)}, (A,a3))
Aggregate({A,B},C)
(2, p3, {(A,a1), (B,b2)}, (C,c1))
Update(A,a2)
(1, p1, {(A,a1)}, (A,a2))
(1, p2, {(B, b1)}, (B,b2))
Update(B,b2)
Insert(A,a1)
(0, p2, {}, (A,a1))
(0, p2, {}, (B, b1))
Insert(B,b1)

Fig. 2. An Example of Non-linear Provenance

Definition 1 (Provenance Object). *The provenance of a data object, A, consists of a set of provenance records, which are partially-ordered by $seqID$. (Alternatively, it is easy to think of the provenance object as a DAG.) Each data object A always has a single* most recent *provenance record, with greatest $seqID$.*

For simplicity, we will assume that $seqID$ values are assigned in the following way: When a new object is inserted, its initial $seqID = 0$. On each subsequent update, we add one to the $seqID$. Finally, for each aggregation operation, we add 1 to the maximum $seqID$ of any input object. This is illustrated with a simple example.

Example 2. Consider the example provenance object (for data object D) shown in Figure 2. This information indicates that participant p_2 originally inserted objects A and B, with initial values a_1 and b_1, respectively. Each of these objects was updated several times. The original version of object A, and an updated version of B were aggregated together to form C. Finally, D was created by aggregating C and a later version of A. Also, notice that the DAG shown in the figure is induced by the sequence ID values associated with each provenance record.

2.2 Threat Model

In the absence of additional protections, the provenance records and objects described in the previous section are vulnerable to illegal and unauthorized modifications that can go undetected. Throughout this paper, our goal is to develop an efficient scheme for detecting such modifications. In this section, we outline our threat model and desired guarantees, which are a variation of those described by Hasan et al. [22].

In particular, consider a data object A and its associated provenance object P. Suppose that P accurately reflects the provenance of A, but that a group of one or more attackers would like to falsify history by modifying A and/or P. In the worst case, the attackers themselves are insiders (participants).

We set out the following desired guarantees with respect to a single attacker:

R1: An attacker (participant) cannot modify the contents of other participants' provenance records (input and/or output values) without being detected by a data recipient.

R2: An attacker cannot remove other participants' provenance records from any part of P without being detected by a data recipient.

R3: An attacker cannot insert provenance records (other than the most recent one) into P without being detected.[5]
R4: If an attacker modifies (updates) A without submitting a proper provenance record to P documenting the update, then this will be detected by a data recipient.
R5: An attacker cannot attribute provenance object P (for data object A) to some other data object, B, without being detected by a data recipient.

In short, we must be able to detect an attack that results from modifying any provenance record that has an immediate successor. Also, we must be able to detect any attack that causes the last provenance record in P to mismatch the current state of object A.

In addition, it may be the case that multiple participants collude to attack the provenance object. In this case, we seek to make the following guarantees:
R6: Two colluding attackers cannot insert provenance records for non-colluding participants between them without being detected by a data recipient.
R7: Two colluding attackers cannot selectively remove provenance records of non-colluding participants between them without being detected by a data recipient.

Finally,
R8: Participants cannot repudiate provenance records.

It is important to point out the distinction between these threats and two related threat models. First, notice that our goal is to *detect* tampering; we do not consider denial-of-service type attacks, in which, for example, an attacker deletes or maliciously modifies data and / or provenance objects to prevent the information from being used. Second, we do not address the related problem of *forged authorship* (piracy) in which an attacker copies a data object, and claims to be the original creator of the data object.

2.3 Cryptography Basics

We will make use of some basic cryptographic primitives. We assume a suitable public-key infrastructure, and that each participant is authenticated by a certificate authority.

– **Hash Functions:** We will use a cryptographic hash function (e.g., SHA-1 [1] or MD5 [33]), which we will denote $h()$. Generally speaking, $h()$ is considered secure if it is computationally difficult for an adversary to find a collision (i.e., messages $m_1 \neq m_2$ such that $h(m_1) = h(m_2)$).
– **Public Key Signatures:** We assume that each participant p has a public and secret key, denoted PK_p and SK_p. p can sign a message m by first hashing m, and then encrypting $h(m)$ with this secret key. We denote this as $S_{SK_p}(m)$. RSA is a common public key cryptosystem [34].

3 Provenance Integrity for Atomic Objects

We begin with the simple case in which we have a database $\mathbb{D}$ comprised entirely of atomic objects. In this case, we propose to provide tamper-evidence by adding a *provenance checksum* to each provenance record. In the case of linear provenance (operations

[5] A participant can always append a provenance record with increasing $seqID$ when the participant executes a corresponding database operation. In this case, the provenance record must properly document the operation in order to comply with requirement $R4$.

seqID	Participant	Input	Output	Checksum
0	p_2	$\{\}$	(A, a_1)	$C_1 = S_{SK_{p_2}}(0\|h(A, a_1)\|0)$
0	p_2	$\{\}$	(B, b_1)	$C_2 = S_{SK_{p_2}}(0\|h(B, b_1)\|0)$
1	p_1	$\{(A, a_1)\}$	(A, a_2)	$C_3 = S_{SK_{p_1}}(h(A, a_1)\|h(A, a_2)\|C_1)$
1	p_2	$\{(B, b_1)\}$	(B, b_2)	$C_4 = S_{SK_{p_2}}(h(B, b_1)\|h(B, b_2)\|C_2)$
2	p_2	$\{(A, a_2)\}$	(A, a_3)	$C_5 = S_{SK_{p_2}}(h(A, a_2)\|h(A, a_3)\|C_3)$
2	p_3	$\{(A, a_1), (B, b_2)\}$	(C, c_1)	$C_6 = S_{SK_{p_3}}(h(h(A, a_1)\|h(B, b_2))\|h(C, c_1)\|C_1\|C_4)$
3	p_1	$\{(A, a_3), (C, c_1)\}$	(D, d_1)	$C_7 = S_{SK_{p_1}}(h(h(A, a_3)\|h(C, c_1))\|h(D, d_1)\|C_5\|C_6)$

Fig. 3. Non-Linear Provenance Example with Integrity Checksums

consisting of only insertions, updates, and deletions), we take an approach similar to that proposed by Hasan et al. [22], and we begin by recapping this approach. Then, we extend the idea to aggregation operations (non-linear provenance).

Consider each database operation resulting in a provenance record (insert, update, and aggregate), and the additional checksum associated with the provenance record:

Insert: Suppose that participant p inserts an object A with value val. The checksum C_0 is constructed as

$$C_0 = S_{SK_p}(0|h(A, val)|0)$$

Update: Now consider the provenance record collected during an update in which participant p changes the value of object A from val to val'. Suppose that the checksum of the previous operation on A is C_{i-1}. The checksum for the update is

$$C_i = S_{SK_p}(h(A, val)|h(A, val')|C_{i-1})$$

Aggregate: Finally, consider the provenance record collected as the result of an aggregation operation that takes as input objects $A_1, ..., A_n$ (with values $val_1, ..., val_n$, respectively) and produces an object B with value val. Assume that the input objects are sorted according to a globally-defined order (e.g, numeric or lexical). We denote the checksums for the previous operations on $A_1, ..., A_n$ as $C_1, ..., C_n$. The checksum is

$$C = S_{SK_p}\Big(h\big(h(A_1, val_1)|h(A_2, val_2)|\cdots|h(A_n, val_n)\big)\Big|h(B, val)\Big|C_1|C_2|\cdots|C_n\Big)$$

Example 3. Consider again the non-linear provenance from Figure 2. Figure 3 shows (in tabular form) the provenance records augmented with checksums.

Consider the data recipient who obtains object D and the provenance object P defined by these records. She can verify that P and D have not been maliciously altered by checking that all of the following conditions hold:

1. D matches the $output$ field in the most recent provenance record.
2. Beginning with the earliest checksums (i.e., those associated with provenance records having the smallest $seqID$ values among all provenance records with the same output object), recompute the checksum using the $input$ and $output$ fields of the provenance record (and the previous checksum if applicable). Check to make sure that each stored checksum matches the computed checksum.

3.1 Checksum Security

In this section, we will briefly explain how the provenance checksums provide the integrity guarantees outlined in Section 2.2.

Property **R1** is guaranteed because each input and output is cryptographically hashed, and then signed by the acting participant. Thus, in order to modify the input / output values that are part of the provenance record, without being detected, an attacker would need to either forge another participant's signature, or find a hash collision. Also, attacks that require inserting or deleting provenance records (**R2, R3, R6, R7**) can be detected because each checksum contains the previous checksum(s) (defined by $seqID$).

Moreover, consider a data recipient who receives a data object A and associated provenance object P. By comparing A to the $output$ field of the most recent provenance record in P, in combination with the other checks, the data recipient can verify that the provenance has not been reassigned to a different data object (**R4**) and that a participant (attacker) has not modified the object without submitting proper provenance (**R5**).

Finally, non-repudiation (**R8**) is guaranteed by participants' signatures on provenance checksums.

3.2 Local vs. Global Checksum Chaining

Finally, notice that when there are multiple data objects (each with associated provenance), we chose to "chain" provenance checksums on a *per-object* basis, rather than constructing a single *global* chain. While both approaches would satisfy our integrity goals, in a (potentially distributed) multi-user environment with many data objects, there are strong practical arguments in favor of the local-chaining approach. In particular, if we elected to construct a global chain, we would have to enforce a particular global sequence on entries into the provenance table, which would become a bottleneck. Consider, for example, two participants p_1 and p_2, who are working on objects A and B. Using the global approach, the two participants would have to enforce a total order on their provenance records (e.g., using locking). In contrast, using the per-object approach, the participants can construct provenance chains (and checksums) for the two objects in parallel.

Also, we find that local chaining is more resilient to failure. If the provenance associated with object A is corrupted, this does not preclude a data recipient from verifying the provenance of another object B (provided that B did not originate from an aggregation operation that took A as input).

4 Provenance Integrity for Compound Data Objects

In the previous section, we described a checksum-based scheme for providing integrity for provenance (linear and non-linear) describing atomic objects. In this section, we expand the approach to the case where objects are compound (contain other objects).

4.1 Extended Data Model

Throughout the rest of this paper, we will expand our data model to include richer and more realistic structure. In particular, instead of modeling the database $\mathbb{D}$ as an unorganized set of objects, we will model the database abstractly in terms of a set of trees (a *forest*). This abstraction allows us to express provenance information associated with varying levels of data granularity in two common data models: relational and tree-structured

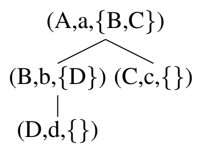

Fig. 4. Example compound object

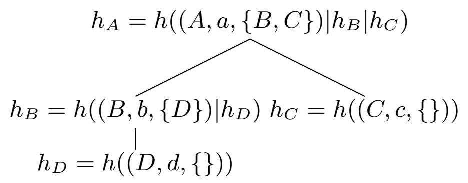
$h_A = h((A, a, \{B, C\})|h_B|h_C)$
$h_B = h((B, b, \{D\})|h_D)$
$h_C = h((C, c, \{\}))$
$h_D = h((D, d, \{\}))$

Fig. 5. Example compound hash value

XML. In the relational model, we can use a tree to express varying granularities of data (e.g., tables, rows, and cells).

Using this abstraction, we expand the idea of an *atomic data object* to be a triple of the form $(id, value, \{child\_ids\})$, where id uniquely identifies the object, $value$ is the atomic value associated with the object, and $\{child\_ids\}$ identifies the set of other objects of which this object is the parent in $\mathbb{D}$. We will also refer to any set of atomic objects such that the child relationships form a tree as a *compound object*. We will use the notation $subtree(A)$ to refer to the compound object defined by the subtree rooted at A. We assume that the database supports the following primitive operations:

– **Insert(**A**,** val**,** $\langle parent \rangle$**)**: Add a new atomic object to $\mathbb{D}$ with $value = val$. The $parent$ field is optional, and indicates the id of A's parent. (For simplicity, the primitive operation only supports insertions and deletions of leaf objects. However, more complex operations can be expressed using multiple primitive operations, as described in Section 4.4.)
– **Delete(**A**):** Remove an existing (leaf) atomic object A from $\mathbb{D}$.
– **Update(**A**,** val'**):** Update the $value$ field of object A to new value val'.
– **Aggregate(**$\{A_1, ..., A_n\}$**,** B**):** Combine $subtree(A_1), ..., subtree(A_n)$ to produce a new compound object rooted at B. For simplicity, we assume that the resulting root B has no parent in $\mathbb{D}$.

Example 4. As a simple example, consider the compound object shown in Figure 4, which contains atomic objects A, B, C, and D (with values a, b, c, d).

4.2 Extended Provenance Model

The execution of each primitive operation is documented in the form of a provenance record. In this case, we extend the provenance records slightly; specifically, the input and output of each operation can be a compound (rather than atomic) object:

$$(seqID, p, \{subtree(A_1), ..., subtree(A_n)\}, subtree(A))$$

A provenance object consists of a set of provenance records of this extended form, which are partially-ordered by $seqID$ like before.

When it comes to compound objects, a unique challenge arises because provenance among objects is naturally not independent. For example, consider a relational database, and a participant who updates a particular cell. Intuitively, if we are collecting provenance for cells, rows, and tables, a record of this update should be maintained in the provenance of the cell, but also for the row and table.

The extended provenance model captures this through the idea of *provenance inheritance*. Conceptually, when an update or insert is applied to an atomic object A, we collect the standard provenance record for A: $(seqID, p, \{subtree(A)\}, subtree(A)')$, where $subtree(A)$ denotes the subtree rooted at A before the update (in the case of insertions, this is empty), and $subtree(A)'$ denotes the subtree rooted at A after the update. In addition, when an object A is inserted, updated, or deleted, we must also collect, for each ancestor B of A the provenance record $(seqID, p, \{subtree(B)\}, subtree(B)')$.

Of course, this conceptual methodology is not an efficient means of collecting and storing inherited provenance. Efficient collection and storage of fine-grained provenance is beyond the scope of this paper; however, this problem has been studied in prior work. For example, [7, 11] describe a set of optimizations that can be used.

4.3 Extended Provenance Checksums

Finally, in order to provide provenance integrity for compound objects, we must extend the signature scheme described in Section 3. We accomplish this using an extended signature scheme related to Merkle Hash Trees [25].

Consider the provenance record $(seqID, p, \{subtree(A)\}, subtree(A)')$ collected as the result of an update (or inherited update) on compound object $subtree(A)$. Suppose also that the checksum of the previous (actual or inherited) operation on $subtree(A)$ is C_{i-1}. We will construct the following checksum for this provenance record:

$$C_i = S_{SK_p}(h(subtree(A))|h(subtree(A)')|C_{i-1})$$

Notice that this checksum includes hashes computed over full compound objects (i.e., $h(subtree(A))$). While we could use any blocked hashing function for this purpose, we elected to define the hash function recursively, which allows us to reuse hashes computed for one complex object when computing the checksums necessary for inherited provenance records.

For example, in Figure 5, h_B is the hash value for $subtree(B)$ from Figure 4. (In order to ensure that these checksums are always consistent, we again assume that there exists a pre-defined total order over atomic objects.) Notice that an update of object B would generate a provenance record for B, but also an inherited provenance record for A. We are able to reuse $h(subtree(B))$ when computing $h(subtree(A))$.

Economical Approach A Basic version of this algorithm will hash all nodes in the input $subtree(A)$, and hash all nodes in the output $subtree(A)$. Even reusing $h(subtree(B))$ when computing $h(subtree(A))$, this approach requires two walks over the entire tree rooted at A. A more economical approach is to compute the hashes of the input nodes in $subtree(A)$, and only re-compute a hash if the node has changed. In the worst case, this still could require 2 traversals of the tree. However, in the best case, it would be 1 traversal of the tree for computing the hash of the input and 1 traversal of the height of the tree to compute the hash of the output.

Checksum Guarantees These extended checksums provide the same guarantees as described earlier (Section 2.2). The analysis is essentially the same as in Section 3.1; the only important addition is to observe that the extended hash value constructed for a compound object is also difficult for an adversary to reverse.

4.4 Complex Operations

At the most basic level, provenance records (including checksums) are defined and collected at the level of primitive operations (insert, update, and aggregate), and in the case of compound objects, updates are inherited upward whenever a descendant object is inserted, updated, or deleted.

Of course, in practice, it may not be necessary to collect provenance records for every primitive operation. Instead, we can group together a sequence of insert, update, and delete operations to form a *complex operation* (which we assume produces a modified complex object). This is based on the idea of transactional storage described in [7]. In this case, for every object A, and its ancestors still present in the database after a series of operations, we collect the provenance record $(seqID, p, \{subtree(A)\}, subtree(A)')$. The checksum associated with this record is exactly the same as described in the last section.

5 Experiments

This section briefly describes our experimental evaluation, the goal of which is to better understand the time and space overhead introduced by generating and storing checksums. Our experiments reveal that these costs are often low enough to be feasible in practice.

5.1 Experimental Setup

Our experimental setup includes two databases. First, we have a back-end database, which contains the user data about which we collect provenance. Second, we have a provenance database. We will assume that both databases are relational. For the purposes of fine-grained provenance, we will view the back-end database as a tree of depth 4, with a single root node, and subsequent levels representing tables, rows, and cells.

Our main goal is to measure the additional time and space cost incurred by collecting integrity checksums, as opposed to the cost of collecting provenance itself, which has been studied extensively in prior work. Thus, for each complex operation, our experiments record: $\langle SeqID(int), Participant(int), Oid(int), Checksum(binary(128))\rangle$.

For the experiments, we generated synthetic back-end databases, each consisting of one or more synthetic data tables, as described in Table 1(a). We also constructed a set of synthetic complex operations on the back-end database, as described in Table 2.

Our hardware and software configuration consists of a Celeron 3.06GHz machine with 1.96G RAM running Windows XP and Java SE runtime environment (JRE) version 6 update 13. Our provenance collection and checksumming code is written in Java, and connected to a MySQL database (v5.1) using MySQL connector/J. For hashing, we use java.security.MessageDigest (algorithm "SHA"), which generates a 20-byte message digest; for encryption, we use java.crypto.Cipher (algorithm "RSA"), which produces a 128-byte signature (given a 1024-byte key).

For all performance experiments, we report the average across 100 runs, including 95% confidence intervals.

5.2 Experimental Results

We conducted several experiments, which illustrate the effect of database size on hashing time, the difference between basic and economical hashing, and the effect of operation types on checksum generation.

Table 1. Synthetic Tables And Databases

(a) Synthetic Tables

Table No.	Num. Attr.	Num. Row	Attr. types
1	8	4000	all integer
2	9	3000	all integer
3	10	2000	all integer
4	5	5000	all integer

(b) Synthetic Databases

Combination of tables	Num. of nodes
1	36002
1,2	66000
1,2,3	88004
1,2,3,4	118006

Table 2. Complex Operations for Each Experiment

Experimental Setup	Complex Operations
A	1 update on 1 cell
	$400n$ updates on $400n$ cells in $400n$ rows($n = 1, \cdots, 10$)
	$4000n$ updates on $4000n$ cells in 4000 rows ($n = 2, \cdots, 8$)
B	500 deletes of rows
	500 inserts of rows
	4000 updates of cells in 500 rows
	4000 updates of cells in 4000 rows
C	500 operations 96(19.2%) deletes 189(37.8%) inserts 15(43%) updates
	500 operations 183(36.6%) delete 152(30.4%) inserts 165(33%) updates
	500 operations 285(57%) deletes 106(21.2%) inserts 109(21.8%) updates
	500 operations 391(78.2%) deletes 49(9.8%) inserts 60(12%) updates

Hashing To understand the effect of the back-end database size on hashing time, we use four databases with increasing sizes as listed in Table 1(b). The time to hash each database is shown in Figure 6. It can be seen that the time grows roughly linearly with the number of nodes (thus the size of the database).

To compare the Basic and Economical hashing approaches described in Section 4.3, we use a back-end database with one synthetic table (4000 rows and 8 integer-valued attributes). We used the complex operations in Experimental Setup **A** (Table 2), which consist entirely of updates, with increasing numbers of cells updated as part of the operation. As expected, the hashing time remains approximately constant when using the basic approach; however, the economical hashing time increases with the number of updated cells.

Of course, a pressing question is whether these techniques can scale to a much larger database (i.e., larger than available memory). To do this we can read one row at a time, hashing the row and the cells in it, and updating the table's hash value with the row's hash value. When all rows are read and hashed, we get the final hash value of the table and update the database's hash value with the table's hash value. When all tables are hashed, we get the final hash value of the database. As a simple experiment, we hashed a relational database with a single table named "Title". This table had 18,962,041 rows and two fields: Document ID (integer) and Title (varchar). (The total number of nodes was thus 56,886,125.) The time to hash this database was 1226.7 seconds (excluding the time of writing the hash values to disk), i.e., the average time of hashing a node took 0.02156

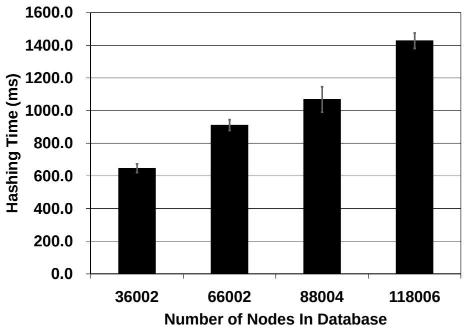

Fig. 6. Average Hashing Time For A Database

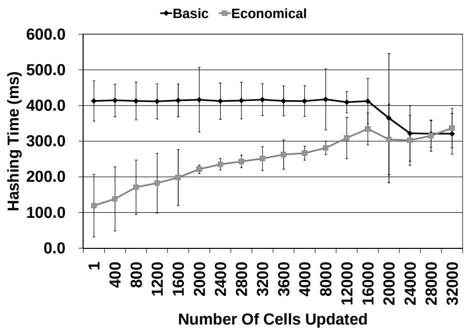

Fig. 7. Hashing The Output Tree Using Basic and Economical Approaches

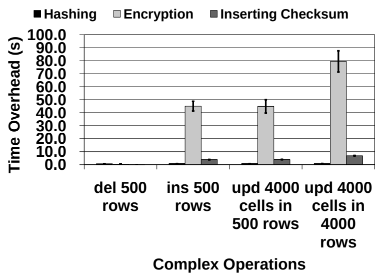

Fig. 8. Time Overheads for Complex Operations of All-Deletes, All-Inserts and All-Updates

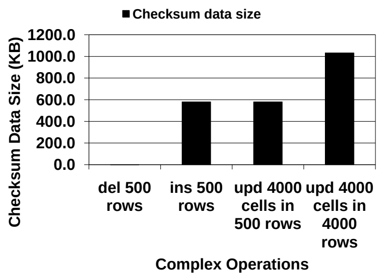

Fig. 9. Space Overheads for Complex Operations of All-Deletes, All-Inserts and All-Updates

milliseconds. Although it is not an apples-to-apples comparison, this average hashing time of a node is within one order of magnitude of that when the whole tree fits into memory.

Effects of Different Operations Recall from Section 4.2 that, in the fine-grained provenance model, if a node n has x ancestors, and we delete n, then we must produce x (inherited) checksums. Alternatively, if we inserted or updated n, this would produce a total of $x + 1$ (actual and inherited) checksums.

To analyze this relationship between operations and checksum overhead, we used the complex operations in Experimental Setup **B**, and we ran these operations on a database with one synthetic table consisting of 4000 rows and 8 integer-valued attributes. From Figure 8, we can see that the time overhead for the all-deletes operation is the smallest. The time overhead for the all-inserts and all-updates operation are similar to one another. Figure 9 shows the space overhead of storing the (actual and inherited) checksums for these four complex operations. As expected, the space overhead is much larger for inserts and updates, as these produce more total provenance records and checksums.

In addition, we conducted some experiments for complex operations containing combinations of insert, update, and delete primitives. Figure 10 shows the time of hashing trees, encrypting and inserting checksums while running Experimental Setup **C**. As expected, the time overhead decreases as the percentage of deletes increases in the complex operation. Similarly, Figure 11 shows that the space overhead is also inversely proportional to the number of deletions.

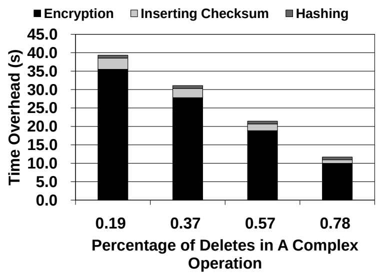

Fig. 10. Time Overhead for Complex Operations Combining Deletes, Inserts and Updates

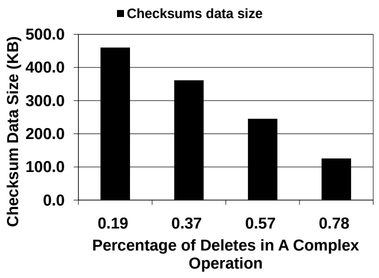

Fig. 11. Space Overhead for Complex Operations Combining Deletes, Inserts and Updates

6 Related Work

Issues surrounding provenance have been studied in database systems [4, 5, 9, 8], workflow systems [10, 14, 17, 20, 29], scientific applications [3, 7, 18, 19] and general provenance issues [11, 30]. However, this paper is the first to provide platform-independent support for verifying the integrity of provenance associated with data at multiple granularities, and through aggregation.

The closest work to ours was described by Hasan et al. [21, 22], and focused on security problems (integrity and confidentiality) that arise when tracking and storing provenance in a file system (e.g., PASS [27]). While our work utilizes a similar threat model and integrity checksum approach, we must deal with a significantly more complicated data model (compound objects) and provenance model (non-linear provenance objects) in order to apply these techniques in the database setting.

A recent vision paper by Miklau and Suciu [26] considered the problem of data authenticity on the web, and described a pair of operations (signature and citation) for tracking the authenticity of derived data. One of the main differences between that work and ours is the structure of participants' transformations. The previous work assumed that transformations were structured in a limited way (specifically, as conjunctive queries), whereas we consider arbitrary black-box transformations.

The general problem of logging and auditing for databases has become increasingly important in recent years. Research in this area has focused on developing queryable audit logs (e.g., [2]) and tamper-evident logging techniques (e.g., [32, 36, 37]). In addition, there has been considerable recent interest in developing authenticated data structures to verify the integrity of query results in dictionaries, outsourced databases, and third-party data publishing (e.g., [15, 16, 24, 28, 31]).

Finally, the provenance community has begun to think about security issues surrounding provenance records and annotations. [6, 38] motivate the need for and complications of security in provenance systems. Several systems have implemented a provenance system to protect the information from unauthorized access: [39] for provenance in a SOA environment; [23] for annotations. Meanwhile, several groups are interested in securely releasing information. First, [13] use the history of data ownership to determine if a user may access information. Second, [12], provide views of provenance information based on the satisfaction of an access control policy. Finally, [35] describe the particular requirements that provenance mandates in access control abilities, and propose an extension to attribute based access control to satisfy these requirements.

7 Conclusion

In this paper, we initiated a study of tamper-evident database provenance. Our main technical contribution is a set of simple protocols for proving the correctness and authenticity of provenance. This is the first paper dealing with the specific provenance and security issues that arise specifically in databases, including non-linear provenance resulting from aggregation and provenance expressed for data at multiple levels of granularity. Through an extensive experimental evaluation, we showed that the additional performance overhead introduced by these protocols can be small enough to be viable in practice.

References

1. Secure hash standard. *Federal Information Processing Standards Publication (FIPS PUB)*, 180(1), April 1995.
2. R. Agrawal, R. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance with a hippocratic database. In *VLDB*, 2004.
3. J. Annis, Y. Zhao, J.-S. Vöckler, M. Wilde, S. Kent, and I. Foster. Applying chimera virtual data concepts to cluster finding in the sloan sky survey. In *Proceedings of the ACM / IEEE Conference on Supercomputing*, 2002.
4. O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.
5. D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *VLDB*, 2004.
6. U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *USENIX*, July 2008.
7. P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *ACM SIGMOD*, 2006.
8. P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. In *ICDT*, 2007.
9. P. Buneman, S. Khanna, and W.-C. Tan. What and where: A characterization of data provenance. *Lecture Notes in Computer Science*, 2001.
10. S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silvaand, and H. T. Vo. VisTrails: Visualization meets data management. In *ACM SIGMOD*, 2006.
11. A. Chapman, H.V. Jagadish, and P. Ramanan. Efficient provenance storage. In *ACM SIGMOD*, 2008.
12. A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang. Scientific workflow provenance querying with security views. In *WAIM*, 2008.
13. A. Cirillo, R. Jagadeesan, C. Pitcher, and J. Riely. *Tapido: Trust and Authorization Via Provenance and Integrity in Distributed Objects*. Lecture Notes in Computer Science. Programming languages and systems edition, 2008.
14. S. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludascher, T. McPhillips, S. Bowers, and J. Freire. Provenance in scientific workflow systems. *IEEE Data Engineering Bulletin*, 32(4), 2007.
15. P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. *Journal of Computer Security*, 12(6), 2004.
16. P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. In *Proceedings of the IFIP 11.3 Workshop on Database Security*, 2000.
17. I. Foster, J. Vockler, M. Eilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM*, July 2002.

18. J. Frew, D. Metzger, and P. Slaughter. Automatic capture and reconstruction of computational provenance. *Concurr. Comput. : Pract. Exper.*, 20(5):485–496, 2008.
19. P. Groth, S. Miles, W. Fang, S. Wong, K.-P. Zauner, and L. Moreau. Recording and using provenance in a protein compressibility experiment. In *IEEE International Symposium on High Performance Distributed Computing*, 2005.
20. P. Groth, S. Miles, and L. Moreau. PReServ: Provenance recording for services. In *Proceedings of the UK OST e-Science second All Hands Meeting 2005 (AHM'05)*, 2005.
21. R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: Problems and challenges. In *International Workshop on Storage Security and Survivability*, 2007.
22. R. Hasan, R. Sion, and M. Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *FAST*, 2009.
23. I. Khan, R. Schroeter, and J. Hunter. *Implementing a Secure Annotation Service*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Provenance and Annotation of Data edition, 2006.
24. F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *ACM SIGMOD*, 2006.
25. R. Merkle. A certified digital signature. In *Proceedings of the 9th Annual International Cryptology Conference*, 1989.
26. G. Miklau and D. Suciu. Managing integrity for data exchanged on the web. In *WebDB*, 2005.
27. K. Muniswamy-Reddy, D. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *USENIX*, 2006.
28. M. Naor and K. Nissim. Certificate revocation and certificate update. In *USENIX*, 1998.
29. T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10), 2006.
30. Open provenance model. http://twiki.ipaw.info/bin/view/Challenge/OPM, 2008.
31. H. Pang, A. Jain, K. Ramamritham, and K. Tan. Verifying completeness of relational query results in data publishing. In *ACM SIGMOD*, 2005.
32. J.M. Peha. Electronic commerce with verifiable audit trails. In *Internet Society*, 1999.
33. R. Rivest. The MD5 message digest algorithm, 1992.
34. R. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2), 1978.
35. A. Rosenthal, L. Seligman, A. Chapman, and B. Blaustein. Scalable access controls for lineage. In *Workshop on the Theory and Practice of Provenance*, 2009.
36. B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2), 1999.
37. R. Snodgrass, S. Yao, and C. Collberg. Tamper detection in audit logs. In *VLDB*, 2004.
38. V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. *Security Issues in a SOA-Based Provenance System*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Provenance and Annotation of Data edition, 2006.
39. W. T. Tsai, X. Wei, Y. Chen, R. Paul, J.-Y. Chung, and D. Zhang. Data provenance in SOA: security, reliability, and integrity. *Journal Service Oriented Computing and Applications*, 2007.