Temporal Implications of Database Information Accountability

Kyriacos E. Pavlou and Richard T. Snodgrass
Department of Computer Science
University of Arizona
Tucson, USA
{kpavlou}{rts}@cs.arizona.edu

***Abstract*—Information restriction controls access and renders records immutable; information accountability requires data transparency to easily and efficiently determine when a particular use is appropriate. Information accountability in the context of relational databases is associated with time in a surprising number of ways, as is summarized in this paper. Notarization and validation of a database exploit the temporal semantics of a transaction-time database. A corruption can be associated with multiple times. Forensic analysis determines the when: bounds on the corruption time, and the where: also specified in terms of time. These bounds are depicted in a two-dimensional corruption diagram, with both axes denoting time. The various kinds of corruption events are defined in terms of time. A parameter termed the regret interval has significant security and performance implications. This paper emphasizes the deep connections between time and the definition, detection, forensic analysis, and characterized extent of a database corruption within the context of information accountability.**

***Keywords*-information accountability; temporal semantics; transaction-time databases; forensic analysis;**

The prevailing approach to achieving privacy and security for databases is *information restriction*. For example, ensuring record compliance, or information compliance in general, usually entails rendering retained records immutable and controlling access to them. We feel that the means of addressing security and compliance should be viewed as constituting a spectrum. If one asserts that information restriction lies at one end of the spectrum then the question which inevitably arises is what lies at the other end? In a recent article Weitzner et al. [1] argue that access control and cryptography are not capable of protecting information privacy and that there is a true dearth of mechanisms for effectively addressing information leaks. They propose that as an alternative information accountability "must become a primary means through which society addresses appropriate use" [1]. *Information accountability* assumes that information should be transparent so as to easily determine whether a particular use is appropriate under a given set of rules. A related concept is *continuous assurance technology*, defined as "technology-enabled auditing which produces audit results simultaneously with, or a short period of time after, the occurrence of relevant events" [2]. This concept is crucial because it can be used to achieve a meaningful operationalization of information accountability.

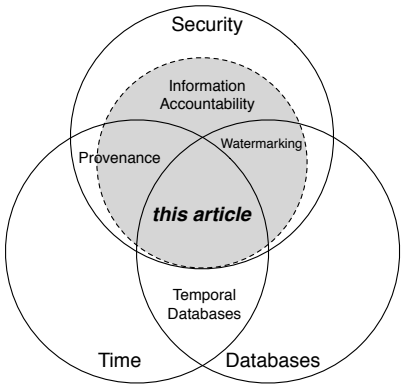

Figure 1. The context of the present paper.

This paper studies the overlap of time, databases, and security, as shown in Figure 1. We have encountered time in this context in many different places and under many different guises. This suggests a deep connection between the general topics of (i) temporal databases and (ii) information accountability. Our purpose here is to identify the many instances where time appears in the definition and implementation of information accountability and to discuss new time-security interactions we have identified recently as described in Sections VIII–X. More generally, we hope that the many tantalizing glimpses of a fundamental connection between temporal databases and information security, as seen in this abbreviated trip through the stages of database corruption, detection, and subsequent forensic analysis, will encourage further work within this community to uncover the source(s) of this connection.

I. The Audit System

In this section we describe how to audit a database and summarize the *tamper detection* approach we previously proposed and implemented [3]. We give the gist of our approach, so that its temporal implications can be understood. Table I lists the audit system execution phases, their subphases, and the actions performed during each.

The *Normal Processing* execution phase differentiates between the *Total Chain Computation* and the *Tamper Detection and Partial Chain Computation* subphases. In the first subphase transactions are hashed and cumulatively

Table I
AUDIT SYSTEM EXECUTION PHASES, SUBPHASES, AND ACTIONS.

Execution Phases	Subphases	Actions
Normal Processing	Total Chain Computation	– Hashing to create total chain – Notarization of total chain
	Tamper Detection and Partial Chain Computation	– Re-hashing of total chain – Validation of total chain *If required by forensic algorithm:* – Hashing to create partial chains – Notarization of partial chains
Forensic Analysis	Corruption Region Analysis	– Running a forensic algorithm to determine where and when
	Manual Analysis	– Determining who and why

linked (by increasing transaction commit time) using a cryptographically-strong hash function, and the resulting values are digitally notarized by an *external digital notarization service*. In the latter subphase hash values are recomputed and compared with those previously notarized. It is during validation of the total chain that tampering is detected, when the just-computed hash value doesn't match the one previously notarized.

The validator provides a vital piece of information, that tampering has taken place, but doesn't offer much else. Since the hash value is the accumulation of every transaction ever applied to the database, we don't know when the tampering occurred, or what portion of the monitored database was corrupted. We have introduced a variety of database forensic algorithms [4], [5], [6] to provide partial answers to these questions. Note that certain forensic analysis algorithms require the computation and notarization of one or more partial hash chains during the scan of the entire database that occurs during validation.

Details on performance, clarifications on the role of the external digital notarization service, and all the forensic analysis algorithms are beyond the scope of this paper and can be found elsewhere [5], [6].

II. THREAT MODEL

Time is first encountered in the underlying threat model. We assume a Trusted Computing Base (TCB) consisting of correctly booted and functioning hardware and a correctly installed operating system and DBMS. More precisely, we assume that the TCB is correctly functioning, the DBMS is created, maintained, and operates in a secure manner, and all network communication is performed through secure channels (such as SSL), ensuring the correctness of the internal state of the DBMS.

A tampering by an adversary ("Bob") occurs at time t_c. This tampering can take many forms. In general, we assume that an intruder (or an insider) who gains physical access to the DBMS server will have full freedom to corrupt any database file.

III. TAMPERING

The very definition of *tampering* can be stated in terms of time. Users and applications modify the database during normal processing, and later query that data. So how can tampering be differentiated from normal processing?

To achieve this we introduce transaction-time support to the database. A transaction-time database records the history of its content [7]. All past states are retained and can be reconstituted from the information in the database. This is ensured through the *append-only* property of a transaction-time database: modifications only add information; no information is ever deleted. Thus the database itself can serve as an audit log. It is this basic property that we exploit to validate the table.

Fortunately, the SQL:2011 standard and many commercial DBMSes now provide transaction-time support. Oracle $10g$ added support for valid-time tables, transaction-time tables, bitemporal tables, sequenced primary keys, sequenced uniqueness, sequenced referential integrity, and sequenced selection and projection, in a manner quite similar to that proposed in SQL/Temporal. Oracle $11g$ enhanced support for valid-time queries [8]. Teradata recently announced support in Teradata Database 13.10 of most of these facilities as well [9], as did IBM for DB2 10 [10].

A normal modification of a tuple can only be performed on the most recent version (that with a stop time of "until changed"). The modification changes the stop time to the current time (for deletions or updates) and inserts a new record with the current time as the start time (for insertions or updates).

A modification is considered a tampering or corruption if it (a) changes any tuple with a stop time other than "until changed", (b) inserts a tuple with a start time other than the current time and a stop time other than "until changed", (c) modifies the explicit attributes of any tuple, or (d) physically deletes any tuple. Note that the first two conditions involve timestamps stored in the database; the second condition explicitly mentions "current time." Specifically, any modification other than a *temporal upward compatible* modification [11] is considered tampering. Section VII will examine a more refined taxonomy of corruptions.

IV. TAMPER DETECTION

We now examine tamper detection in more detail. Suppose that we have just detected a *corruption event* (or *CE*), which is any event that corrupts the data and compromises the database. Table II summarizes all the time-related concepts used in this paper. Time instants are generally denoted by a subscripted t, time intervals by a subscripted I or R. *Factors* are integers. A *temporal* or a *spatial bound* occurs at a time instant, as does an *event*.

There exists a one-to-one correspondence between a CE and its *corruption time* (t_c), which is the actual time instant (in seconds) at which a CE has occurred. Figure 2 shows that t_c marks the transition from a legal database state to an illegal one.

Table II
SUMMARY OF TIME-RELATED CONCEPTS.

Symbol	Name	Definition
CE	Corruption event	An event that compromises the database
VE	Validation event	The validation of the audit log by the notarization service
NE	Notarization event	The notarization of a document (hash value) by the notarization service
t_c	Corruption time	The time instant of a CE
t_v	Validation time	The time instant of a *VE*
I_V	Validation interval	The time between two successive *VE*s
t_n	Notarization time	The time instant of a *NE*
I_N	Notarization interval	The time between two successive *NE*s
V	Validation factor	The ratio I_V/I_N
t_l	Locus time	The time instant that the corruption locus data (l_c) was stored
R_s	Spatial detection resolution	Finest interval chosen to express the spatial bounds uncertainty of a CE
N	Notarization factor	The ratio I_N/R_s
R_t	Temporal detection resolution	Finest interval chosen to express the temporal bounds uncertainty of a CE
t_{FVF}	Time of first validation failure	Time instant at which the CE is first detected
t_{RVS}	Time of most recent validation success	The time instant of the last *NE* whose revalidation yielded a true result
LTB	Lower temporal bound	Lower bound of the temporal uncertainty of the corruption region
UTB	Upper temporal bound	Upper bound of the temporal uncertainty of the corruption region
LSB	Lower spatial bound	Lower bound of the spatial uncertainty of the corruption region
USB	Upper spatial bound	Upper bound of the spatial uncertainty of the corruption region
t_b	Backdating time	The time a timestamp was backdated to
t_p	Postdating time	The time a timestamp was postdated to
I_{max_tran}	Transaction max-duration	Maximum duration of a transaction
I_R	Regret interval	Minimal time interval before an adversary can reverse a change
I_R^*	Regret interval estimate	Lower bound on the regret interval
I_{RP}	Retention interval	Length of the retention period
t_s	Shred time	The time a tuple is shredded
I_{LH}	Litigation hold interval	A duration of time specified by a court of law
I_{qv}	Query verification interval	Interval between the time a transaction reads data and the time when tampering is detected

The following discussion relates to our approach to effecting information accountability in relational databases. A CE is detected during a *validation event* (or *VE*) of the database by the notarization service. A validation can be scheduled (that is, is periodic) or could be an *ad hoc VE*. The time (instant) at which a *VE* occurs is termed the *time of validation event*, and is denoted by t_v. If validations are periodic, the time interval between two successive validation events is termed the *validation interval*, or I_V.

The validator compares the hash value it computes over the data with the hash value that was previously notarized. Tampering is indicated by a *validation failure*, in which the digital notarization service returns *false* for the particular query of a hash value and a notarization time. What is desired is a *validation success*, in which the notarization service returns *true*, stating that everything is OK: the data has not been tampered.

A *notarization event* (or *NE*) is the notarization of a document (specifically, a hash value) by the notarization service. As with validation, notarization can be scheduled (is periodic) or can be *ad hoc*. Each *NE* has an associated *notarization time* (t_n), which is a time instant. If notarizations are periodic, the time interval between two successive notarization events is termed the *notarization interval*, or I_N.

The validation interval should be equal to or longer than the notarization interval, should be an integer multiple of the notarization interval, and should also be aligned with it, that is, validation should occur immediately after notarization. This is because one can only validate what one has previously notarized. Having non-aligned notarization and validation intervals can only delay tamper detection. Thus we speak of the *validation factor* V such that $I_V = V \cdot I_N$. As long as this constraint is respected, it is possible to change V, or both I_V and I_N, as desired. This, however, will affect the size of the corruption region as emphasized in Section VI.

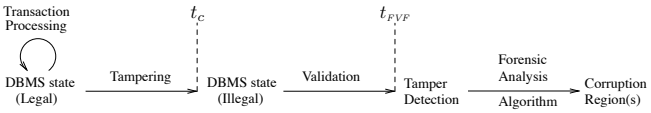

Figure 2. Tampering, Detection, and Forensic Analysis.

There are several variables associated with each corruption event. The first is the data that has been corrupted, which we term the *corruption locus data* (l_c).

Forensic analysis, as discussed in Section VI, involves *temporal detection*: the determination of the corruption time, t_c. Forensic analysis also involves *spatial detection*, the determination of "where," that is, the location in the database of the data altered in a CE. (Note that the use of the adjective "spatial" does not refer to a spatial database, but rather *where in the database* the corruption occurred.)

Interestingly, even the corruption locus is specified in terms of time. Recall that each transaction is hashed. Therefore, in the absence of other information, such as a previous dump (copy) of the database, the best a forensic analysis can do is to identify the particular transaction that stored the data that was corrupted. Instead of trying to ascertain the corruption locus data (l_c), we will instead be concerned with the *locus time* (t_l), the time instant the data was originally stored. The locus time specifically refers to the time instant when the transaction storing the corruption locus data commits. (Here we are referring to the specific *version* of the data that was corrupted. This version might be the original version inserted by the transaction, or a subsequent version created through an update operation.)

A CE can have many l_c's (and hence, many t_l's) associated with it. Such a CE is termed *multi-locus*: an intruder (hardware failure, etc.) might alter many tuples. A CE having only one l_c (such as due to an intruder hoping to remain undetected by making a single, very particular change) is termed a *single-locus CE*. Now we can formally define what a corruption event is.

Definition 1. *A* corruption event *is a two-tuple* (T_l, t_c). *The set* $T_l = \{t_{l1}, t_{l2}, \ldots, t_{ln}\}$ *is the set of all locus times associated with a particular corruption event. Each* $(t_{li}, t_c) \in \mathbb{T}^2$, *where* $\mathbb{T}$ *is a time domain.*

We define R_s as the finest interval chosen to express the uncertainty of the spatial bounds of a CE. R_s is called the *spatial detection resolution*. This is chosen by the database administrator (DBA). Similarly, the finest interval chosen by

the DBA to express the uncertainty of the temporal bounds of a CE is the *temporal detection resolution*, or R_t.

Several others works have studied tamper detection in databases. An example of a WORM-based, long-term high-integrity retention technique for fine granularity business records is the *transaction log on WORM* (TLOW) approach for supporting long-term immutability of relational tuples [12]. TLOW stores the current database instance in ordinary storage and the transaction log on Write-Once-Read-Many (WORM) storage, while dispensing with a compliance log altogether. The audit process uses hash values representing the data rather than the data themselves. An audit is successful if the hash from the old database snapshot plus the hash of all the new tuples introduced in the transaction log match the hash of the current database instance. Thus within this tamper detection framework the same notions of corruption event, auditing/validation interval, and time of first validation failure can be defined. Another time-related concept, the *query verification interval*, is specific to TLOW.

Guo, Jajodia, Li, and Liu formulated a *fragile watermarking* scheme for database tamper detection [13], [14]. Their scheme is based on a watermark that is *invisible* (watermark does not distort data) and can be *blindly verified* (original unmarked relation is not required for verification). During verification, the extracted watermark indicates the locations of alterations. This approach does not utilize a temporal definition of tampering.

V. The Corruption Diagram

To explain forensic analysis within the context of our approach, we introduce the *Corruption Diagram*, which is a graphical representation of CE(s) in terms of the temporal-spatial dimensions of a database.

Figure 3 illustrates a simple corruption event. While this figure may appear to be complex, the reader will find that it succinctly captures all the important information regarding what is stored in the database, what is notarized, and what can be determined by the *Monochromatic Forensic Analysis Algorithm*—the simplest of the algorithms we have proposed—about the corruption event.

This corruption diagram shares some aspects with commonly-encountered bitemporal diagrams [15]. In a bitemporal diagram, the axes are transaction time and valid time, with rectangular polygons indicating the bitemporal extent of facts. As we will see, the corruption diagram conveys very different information, while having the surface similarity of being a two-dimensional depiction, with time as both dimensions. First we give the definition of a corruption diagram before describing it in detail.

Definition 2. *Let* $\mathbb{T}$ *be a* time domain. *A* corruption diagram *is a plot in* $\mathbb{T}^2$ *having its ordinate associated with wall-clock time and its abscissa associated with a partition of the database according to transaction time. This diagram depicts corruption events and is annotated with hash chains and relevant notarization and validation events. At the end of forensic analysis, this diagram can be used to visualize the regions* ($\subset \mathbb{T}^2$) *where corruption has occurred.*

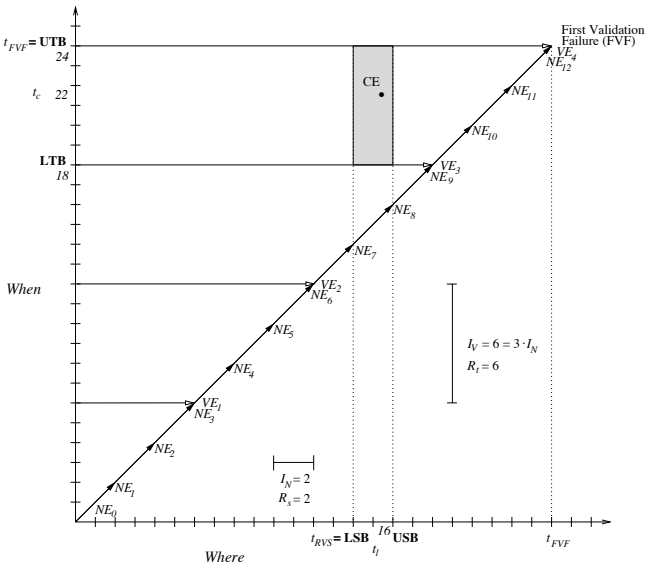

Figure 3. Corruption diagram for a data-only single-locus retroactive corruption event.

Let us first consider the simplest case. During validation, we have detected a corruption event. Though we don't know it (yet), assume that this corruption event is a single-locus CE. Furthermore, assume that the CE just altered the *data* of a tuple; no timestamps were changed.

The x-axis represents when the data are stored in the database. The database was created at time 0, and is modified by transactions whose commit time is monotonically increasing along the x-axis. (In temporal database terminology [7], the x-axis represents the transaction time of the data.) In the corruption diagram, time moves inexorably to the right.

The x-axis is labeled "*Where*." The database grows monotonically as tuples are appended (recall that the database is append-only). As previously explained, we designate "where" a tuple or attribute is in the database by the time of the transaction that inserted that tuple or attribute. We delimit the days by marking each midnight, or, more accurately, the time of the last transaction to commit before midnight.

A 45-degree line is shown and is termed the *action line*, as all the action in the database occurs on this line. The line terminates at the point labeled "FVF," which is the validation event at which we first became aware of tampering. The *time of first validation failure* (or t_{FVF}) is the time at which the corruption is first detected. (Hence the name: a corruption diagram always terminates at the *VE* that detected the corruption event.) Note that t_{FVF} is an instance of a t_v, in that t_{FVF} is a specific instance of the time of a validation event. Also note that in every corruption diagram,

t_{FVF} coincides with the current time. For example, in Figure 3 the *VE* associated with t_{FVF} occurs on the action line, at its terminus, and turns out to be the fourth such validation event, VE_4.

The actual corruption event is shown as a point labeled "CE," which always resides above or on the action line, and below the last *VE*. If we project this point onto the x-axis, we learn "where" (in terms of the corruption locus time, t_l) the corruption event occurred.

The y-axis represents the temporal dimension (actual time-line) of the database, labeled in time instants. Any point on the action line thus indicates a transaction committing at a particular transaction time (a coordinate on the x-axis) that happened at a clock time (the same coordinate on the y-axis). For this reason, the two times are totally correlated and the action line is always a 45-degree line. Projecting the CE onto the y-axis tells us when in clock time the corruption occurred, that is, the corruption time, t_c. We label the y-axis with "*When*." The diagram shows that the corruption occurred on day 22 and corrupted an attribute of a tuple stored by a transaction that committed on day 16.

Notarization event NE_1 hashes the transactions occurring during the first two days (here, the notarization interval, I_N, is two days), linking these hash values together using *linked hashing* [3]. This is illustrated with the upward-right-pointing arrow with the solid black arrowhead originating at NE_0 (since the linking starts with the hash value notarized by NE_0) and terminating at NE_1. Each transaction at commit time is hashed; here the "where" (transaction time) and "when" (wall-clock time) are synchronized; hence, hashing occurs on the diagonal. The hash value of the transaction is linked to the previous transaction, generating a linked sequence of transactions that is associated with a hash value notarized at midnight of the second day in wall-clock time and covering all the transactions up to the last one committed before midnight (hence, NE_1 resides on the action line). NE_1 sends the resulting hash value to the notarization service.

Also along the action line are points denoted with "*VE*." These are validation events for which a validation occurred. During VE_1, which occurs at midnight on the sixth day (here, the validation interval, I_V, is six days), rehashes all the data in the database in transaction commit order, denoted by the long right-pointing arrow with a white arrowhead, producing a linked hash value. In fact, the diagram shows that VE_1, VE_2, and VE_3 were successful (each scanning a successively larger portion of the database, the portion that existed at the time of validation). The diagram also shows that VE_4, immediately after NE_{12}, failed, as it is marked as FVF; its time t_{FVF} is shown on both axes.

In summary, we now know that at each of the *VE*s up to but not including FVF succeeded. When the validator scanned the database as of that time (t_v for that *VE*), the hash value matched that notarized by the *VE*. Then, at the last *VE*, the FVF, the hash value *didn't* match. The corruption event, CE, occurred before midnight of the 24^{th} day, and corrupted some data stored sometime during those twenty four days.

VI. Forensic Analysis

Once the corruption has been detected, a *forensic analysis algorithm*, like the Monochromatic Algorithm, springs into action. The task of this algorithm as shown in Figure 2, is to ascertain, as accurately as possible, the *corruption region*: the bounds on "where" and "when" of the corruption.

On validation failure we know that the corruption must lie in the upper-left triangle, delimited by the *When* and action axes, denoting that the corruption event occurred before t_{FVF} and altered data stored before t_{FVF}.

The most recent *VE* before FVF is VE_3 and it was successful. This implies that the corruption event has occurred in this time period. Thus t_c is somewhere within the last I_V, which always bounds the "when" of the CE.

To bound the "where," the Monochromatic Algorithm can validate prior portions of the database, at times that were earlier notarized.

Revisiting and revalidating the cumulative hash chains at past notarization events will yield a sequence of validation results that start out to be true and then at some point switch to false (TT. . .TF. . .FF). This single switch from true to false is a consequence of the cumulative nature of the total hash chain. We term the time of the last *NE* whose revalidation yielded a true result (before the sequence of false results starts) the *time of most recent validation success* (t_{RVS}). This t_{RVS} helps bound the "where" of the CE because the corrupted tuple belongs to a transaction which committed between t_{RVS} and the next time the database was notarized (whose validation now evaluates to false). t_{RVS} is marked on the *Where* axis of the corruption diagram in Figure 3.

Definition 3. *In light of the above observations, we define the four bounds of a CE.*

- *the* lower temporal bound*:* LTB $:= \max(t_{\text{FVF}} - I_V, t_{\text{RVS}})$,
- *the* upper temporal bound*:* UTB $:= t_{\text{FVF}}$,
- *the* lower spatial bound*:* LSB $:= t_{\text{RVS}}$, *and*
- *the* upper spatial bound*:* USB $:= t_{\text{RVS}} + I_N$.

These bounds define a corruption region, indicated in Figure 3 as a narrow rectangle, within which the CE must lie. This example shows that, when utilizing the Monochromatic Algorithm, the notarization interval, here $I_N = 2$ days, bounds the "where," and the validation interval, here $I_V = 6$ days, bounds the "when." Hence for this algorithm, $R_s = I_N$ and $R_t = I_V$. (More precisely,

$$R_t = \mathit{UTB} - \mathit{LTB} = \min(I_V, t_{FVF} - t_{RVS})$$

due to the fact that R_t can be smaller than I_V for late-breaking corruption events, such as that illustrated in Figure 5.)

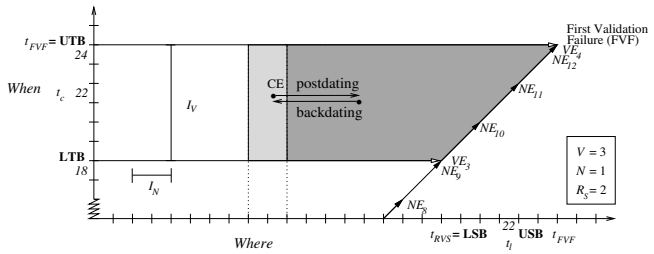

Figure 4. Postdating and backdating corruption events.

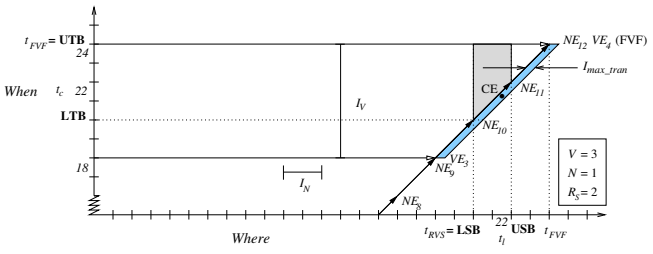

Figure 5. Corruption diagram with introactive data-only CE and envelope.

Other forensic analysis algorithms we have proposed make use of partial hash chains in addition to the total chain. These partial chains are computed and notarized during the re-hashing and validation of the total chain. The partial chains hash only parts of the data in the database and in certain cases are not cumulative. Their "placement," i.e., the parts of the database they collectively cover, creates a structure over the database that allows the forensic algorithm to prune the search space efficiently and thus correctly locate multiple CEs very quickly. It also allows the decoupling of the spatial detection resolution (R_s) and I_N. In fact, the value of R_s can be set by the DBA to be much smaller that the value of I_N.

VII. Characterization of Corruption Types

The CE shown in Figure 3 is termed a *retroactive corruption event*: a CE with locus time t_l appearing before the next to last validation event. As we will see in this section, this is but one of several corruption types, characterized by various temporal relationships.

A. Data-Only Corruptions

Figure 5 illustrates an *introactive corruption event*: a CE with a locus time t_l appearing after the next to last validation event. In this figure, the corruption event occurred on day 22, as before, but altered data on day 22 (rather than day 16 as in the diagram of Figure 3). NE_{10} is the most recent validation success. Here the corruption region is a trapezoid rather than a rectangle. This shape is implied by the definition of *LTB*.

Both retroactive and introactive corruptions are types of *data-only corruption events*.

B. Timestamp Corruptions

Data-only corruption events do not change any timestamps in the tuples. However, there are two other kinds of corruption events that arise from timestamp corruption. In a *backdating corruption event*, a timestamp is changed to indicate a previous time/date with respect to the original time in the tuple. We term the time a timestamp was backdated *to* the *backdating time*, or t_b. It is always the case that $t_b < t_l$. Similarly, a *postdating corruption event* changes a timestamp to indicate a future time/date with respect to the original commit time in the tuple, with the *postdating time* (t_p) being the time a timestamp was postdated to. It is always the case that $t_l < t_p$. Both types of timestamp corruption are illustrated in Figure 4. Observe that timestamp corruption produces two corruption regions since changing a timestamp effectively changes the order in which the tuple is hashed into the total chain.

Combined with the previously introduced distinction of retroactive and introactive, these considerations induce six specific corruption event types.

$$\left\{ \begin{array}{l} \textit{Retroactive} \\ \textit{Introactive} \end{array} \right\} \times \left\{ \begin{array}{l} \textit{Data-only} \\ \textit{Backdating} \\ \textit{Postdating} \end{array} \right\}$$

For backdating corruption events, we ask that the forensic analysis determine, to the extent possible, "when" (t_c), "where" (t_l), and "to where" (t_b). Similarly, for postdating corruption events, we want to determine t_c, t_l, and t_p. This is quite challenging given the only information we have, which is a single bit for each query on the notarization service.

We have proposed a taxonomy of many other corruption types along with a forensic analysis protocol on how to identify each type (vid. http://www.cs.arizona.edu/projects/tau/dragoon/taxonomy_protocol.pdf). Please note that the monitored database need not support valid time for forensic analysis to work but if it is a bitemporal database then more complex corruptions involving valid-time timestamps can arise, not yet considered in the taxonomy.

VIII. Very Recent Corruptions

Corruption events that occur very recently present a challenge to forensic analysis. The problem arises from the assertion that a CE cannot occur below the 45-degree line. The argument is that it is impossible for $t_c < t_l$ to occur, because that would imply that the data are corrupted before they are added to the database. This argument holds generally, but there does exist an exception, where the CE can be below the action axis: when the CE corrupts data of a currently executing transaction. Since a single transaction takes a finite non-zero amount of time, there is a window of opportunity between when the transaction starts and when it commits during which a corruption can occur. In such a case we will have $t_c < t_l$ and the CE will be below the 45-degree line.

Fortunately, the existence of such corruptions does not invalidate any of our previous analysis. Only the "at-the-time-current" transactions in the most recent validation interval are susceptible to this threat. Hence, the first change to be made is to draw a straight-line "envelope" parallel to the 45-degree action axis, whose horizontal distance from the action axis represents the maximum duration of any single transaction, denoted by I_{max_tran}. In this way the corruption region is augmented with a narrow slice, as shown in Figure 5, to account for this possibility.

This window of opportunity for each transaction varies since it depends on the duration of the transactions and that is the reason I_{max_tran} was chosen as the width of the "envelope." The only case where this will affect the result of the forensic analysis is when the CE is an introactive CE: only then can it approach and move past the 45-degree line. Thus, in such a case the upper bound on t_l will be increased by I_{max_tran}.

Note also that this weakness has a tradeoff: an introactive CE puts a tighter lower bound on t_c, meaning it is easier to find the actual time of corruption. Observe that in Figure 5 the *LTB* does not coincide with VE_3 but is instead raised from day 18 to day 21.

A different approach to solving the above problem is to introduce the notion of a *regret interval* [16]. This is a minimal time interval, I_R, before any adversary can reverse the change they have made. For example, in current legal interpretations of email compliance according to Sarbanes-Oxley [17], this time interval is zero. However, when monitoring bioscience lab results, we may be able to assume that after a new record is added, a week will pass—given that certain protocols require several days to complete—before any adversary is likely to "regret" its existence.

The true size of the regret interval is intrinsic to the semantics and social use of the application. Note that the DBA has no control over it. Furthermore, the DBA may not be able to determine its size with absolute certainty. However, the DBA can estimate it with a (possibly) tight lower bound. We call this the *regret interval estimate* and denote it by I_R^*. Observe that $I_R^* \leq I_R$.

The existence of a nonzero regret interval estimate can be leveraged to increase throughput. However, in order not to compromise the correctness of tamper detection and subsequent forensic analysis the DBA must ensure that notarization of hash values happens at time intervals which are smaller than the estimated regret interval. This forces all tamperings to transpire *after* a notarization, something that ensures tamper detection. Thus we have $I_N < I_R^* \leq I_R$. Moreover, validation events have to occur after notarizations (one cannot validate a hash value that has not already been notarized). If we set the validation interval to be smaller than the estimated regret interval then we have $I_N \leq I_V < I_R^* \leq I_R$ and this enforces the absence of introactive corruption events.

IX. Shredding and Litigation Holds

Transaction-time semantics allow us to keep the history of the entire database. This in turn enables the constant monitoring of the database state in order to detect any deviation. Keeping the totality of data that were ever stored in the database causes complications. First it has an adverse effect on performance since all the data have to be rehashed during validation. The ever-increasing cost of notarizing and validating the data will at some point become prohibitive. Moreover, companies are not wont to keep legacy data for long periods of time since such a practice poses a privacy and liability threat. In general, old data are periodically deleted while newer data must be kept for a specific *retention period* according to regulations and company policy.

Therefore, a sliding time frame, the length I_{RP} of the retention period, exists whereby records continuously become old enough to fall outside that window as time progresses. Such records are deemed safe for physical deletion (*shredding*). Shredding itself occurs at time t_s, at any time after the record exits the time frame $now - I_{RP}$. This is a serious issue because shredding breaks transaction-time semantics that requires that the monitored database is append-only.

To complicate the situation even further *litigation holds* can be issued on the data for a duration of time, I_{LH}, specified by courts of law. Such a hold overrides any retention period regulations and so none of the data can be subject to shredding until the hold is lifted. Thus it can be said that litigation holds restore transaction-time semantics. Similar concepts have been described extensively elsewhere [18].

X. Enterprise Considerations

Companies typically have many databases, a number of which are susceptible to tampering and thus fall under the purview of database information accountability. It is useful for the company's Chief Security Officer (CSO) to set general corporate policies on acceptable values for those parameters in Table II. So for example the CSO could dictate that the validation interval I_V be no longer than 2 days and that the spatial detection resolution R_s be no longer than one hour, applicable to all databases being monitored. The database administrator could then indicate, for a particular database or perhaps even particular tables, the specific values for the spatial detection resolution R_s and the regret interval estimate I_R^*. These values would subsequently dictate other values, such as I_N and I_V. As the values are related in ways summarized in previous sections, the CSO and the DBA both have flexibility in what to specify.

Note that it is important to record these various enterprise-wide and database-specific settings. These settings have a profound influence on the quality of the forensic analysis, and a cost model has been developed which incorporates these settings, in order to assess the forensic cost associated with each algorithm. The effect of the settings on the forensic cost has been experimentally studied and verified [5].

Because of their importance, all settings are stored in a separate *enterprise security database*, itself a transaction-time database (with some bitemporal portions), located in the trusted computing base. We need to know *when* each setting was specified. Time again makes its presence known.

XI. Summary and Future Work

As demonstrated throughout this paper, time arises in many guises: in the definition of tampering, the data that is tampered, the kinds of corruptions that can occur, the detection of tampering, the forensic analysis of tampering, and the "when" and even the "where" of the tampering determined by that analysis. Table II lists a full two dozen of the time instants and intervals involved throughout the definition and mechanism of database information accountability.

It would be interesting to see how defining tampering in terms of pattern recognition of complex events [19] can affect detection. Our intuition tells us that it would allow detection of tampering at a semantic level wherein modifications to the database issued through the DBMS can be identified as illegal.

In general, there is something truly fundamental going on, of which we are now seeing just the surface structure. Determining that deep structure is our challenge to this community.

Acknowledgement

We gratefully acknowledge support from NSF grants IIS-0415101, IIS-0803229, IIS-0639106, and EIA-0080123. Grants from Microsoft Corporation and from Surety, LLC also provided partial support for this work.

References

[1] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman, "Information Accountability," *Communications of the ACM (CACM)*, vol. 51, no. 6, pp. 82–87, June 2008.

[2] M. Alles, A. Kogan, and M. Vasarhelyi, "Black Box Logging and Tertiary Monitoring of Continuous Assurance Systems," *Information Systems Control Journal*, vol. 1, 2003.

[3] R. T. Snodgrass, S. S. Yao, and C. Collberg, "Tamper Detection in Audit Logs," in *Proceedings of the International Conference on Very Large Databases*, September 2004, pp. 504–515.

[4] K. E. Pavlou and R. T. Snodgrass, "Forensic Analysis of Database Tampering," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 2006, pp. 109–120.

[5] ——, "Forensic Analysis of Database Tampering," *ACM Transactions on Database Systems*, vol. 33, no. 4, pp. 30:1–30:47, November 2008.

[6] ——, "The Tiled Bitmap Forensic Analysis Algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 4, pp. 590–601, April 2010.

[7] C. S. Jensen and C. E. Dyreson (eds), "A consensus glossary of temporal database concepts—February 1998 Version," in *Temporal Databases: Research and Practice*, O. Etzion, S. Jajodia, and S. Sripada, Eds. Springer-Verlag, 1998, pp. 367–405.

[8] Oracle Corp. (2008, Aug.) Workspace Manager Developer's Guide 11*g* Release 1 (11.1). [Online]. Available: http://www.oracle.com/pls/db111/to_pdf?pathname=appdev.111/b28396.pdf

[9] Teradata Corp. (2010, Oct.) Teradata Temporal. [Online]. Available: http://www.teradata.com/database/teradata-temporal/

[10] IBM Corp. (2010, Dec.) A Matter of Time: Temporal Data Management in DB2 for z/OS. [Online]. Available: http://www14.software.ibm.com/webapp/iwm/web/signup.do?lang=en_US&source=sw-infomgt&S_PKG=db2z-temporal-tables-wp

[11] J. Bair, M. Böhlen, C. S. Jensen, and R. T. Snodgrass, "Notions of upward compatibility of temporal query languages," *Business Informatics (Wirtschafts Informatik)*, vol. 39, no. 1, pp. 25–34, 1997.

[12] R. Hasan and M. Winslett, "Efficient Audit-based Compliance for Relational Data Retention," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '11. New York, NY, USA: ACM, 2011, pp. 238–248.

[13] H. Guo, Y. Li, A. Liu, and S. Jajodia, "A fragile watermarking scheme for detecting malicious modifications of database relations," *Inf. Sci.*, vol. 176, no. 10, pp. 1350–1378, 2006.

[14] Y. Li, H. Guo, and S. Jajodia, "Tamper Detection and Localization for Categorical Data Using Fragile Watermarks," in *Proceedings of the 4th ACM Workshop on Digital Rights Management*, 2004, pp. 73–82.

[15] C. S. Jensen, M. D. Soo, and R. T. Snodgrass, "Unification of Temporal Data Models," in *International Conference on Data Engineering*, April 1993, pp. 262–271.

[16] S. Mitra, W. W. Hsu, and M. Winslett, "Trustworthy Keyword Search for Regulatory-Compliant Record Retention," in *Proceedings of the International Conference on Very Large Databases*, 2006, pp. 1001–1012.

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

[18] R. Hasan and M. Winslett, "Trustworthy Vacuuming and Litigation Holds in Long-term High-integrity Records Retention," in *Proceedings of the 13th International Conference on Extending Database Technology*, 2010, pp. 621–632.

[19] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.