THE INFERENCE PROBLEM AND UPDATES IN RELATIONAL DATABASES

Csilla Farkas
Department of Computer Science and Engineering
University of South Carolina, Columbia, SC 29208
farkas@cse.sc.edu

Tyrone S. Toland
Department of Computer Science and Engineering
University of South Carolina, Columbia, SC 29208
toland@cse.sc.edu

Caroline M. Eastman
Department of Computer Science and Engineering
University of South Carolina, Columbia, SC 29208
eastman@cse.sc.edu

Abstract   In this paper, we extend the Disclosure Monitor (DiMon) security mechanism (Brodsky et al. [1]) to prevent illegal inferences via database constraints in dynamic databases. We study updates from two perspectives: 1) updates on tuples that were previously released to a user may cause that tuple to be "outdated", thus providing greater freedom for releasing new tuples; 2) observation of changes in released tuples may create cardinality based inferences, which are not indicated by database dependencies. We present a mechanism, called *Update Consolidator* (UpCon) that propagates updates to the user's history file to ensure that no query is rejected based on outdated data. We also propose a *Cardinality Inference Detection* (CID) module, that generates all data that can be disclosed via cardinality based attacks. We show that UpCon and CID, when integrated into the DiMon architecture, guarantee *confidentiality* (completeness property of the data-dependent disclosure inference algorithm) and maximal *availability* (soundness property of the data-dependent disclosure inference algorithm) even in the presence of updates.

Keywords: disclosure inference, confidentiality, availability, updates, access control.

1.   INTRODUCTION

During the last couple of decades, our society became increasingly dependent on computerized information resources. Electronic databases contain information with sensitivity levels ranging from public (e.g., airline schedules, phone numbers, and addresses) to highly sensitive (e.g., financial and medical information, military research). The aim of information security policies is to protect the *confidentiality* (secrecy) and *integrity* of data, while ensuring data *availability*. Access control mechanisms, such as discretionary and mandatory access control, prevent unauthorized, direct accesses to data. However, they are unable to protect against indirect data accesses, when unauthorized information is obtained via inference channels.

Most of the inference channels in relational databases are created by combining *meta-data* (e.g., database constraints) with non-sensitive data to obtain sensitive information. Techniques to detect and remove inference channels can be organized into two categories. The first category includes techniques that detect inference channels during database design time [2, 3, 7, 8, 11, 14, 15, 17, 18, 20]. Inference channels are removed by modifying the database design and/or by increasing the classification levels of some of the data items. The techniques of the second category seek to eliminate inference channel violations during query processing time [5, 10, 12, 13, 16, 19]. If an inference channel is detected, the query is either refused or modified to avoid security violations. While, in general, database design time approaches are computationally less expensive and may provide better online performance than query processing time approaches, database design time approaches often result in over-classification of data items, thus reducing data availability. Query processing time approaches allow greater availability by analyzing the data released to the user to determine whether a security violation is present or not.

Updates in multilevel secure databases have been studied from the perspective of how to perform updates safely, creating the problem of polyinstantiation [6, 9]. The main problem is that users with different security levels see different versions of a multilevel secure database. Updates may create inconsistencies among these versions (e.g., inserting a new tuple with an existing but invisible primary key) or create downward signaling channels (e.g., rejection of an update because of the existence of a high security data item).

In this paper we study updates from a different perspective. Instead of addressing the problem of how to ensure secure updates, we focus on how updates can increase data availability due to outdated data. Our work extends the Disclosure Monitor (DiMon) model presented by Brodsky et al.[1]. DiMon detects and eliminates inference channels created by database constraints by using a Disclosure Inference Engine(DiIE) that generates all information that can be disclosed based on a user's previous queries (results), the current query (result)

and a set of Horn-Clause constraints. The disclosure inference algorithms by the properties of *soundness* and *completeness*. Intuitively, soundness means that only existing disclosure inferences are generated by the algorithm (data availability); completeness means that all existing disclosure inferences are generated (secrecy).

However, Brodsky et al. do not consider dynamic databases where updates may lead to cardinality based inferences or may violate the soundness property of the data-dependent disclosure inference algorithm. The following two examples show a proper execution of DiMon (first example) and an execution that violates the soundness property (second example). Consider the original *Employee* relation in Table 1, containing information about the name, rank, salary, and department of employees. The relation satisfies the functional dependency (FD) $RANK \rightarrow SALARY$. The security requirement is that the employees' salaries should be kept confidential; that is, partial tuples over attributes *NAME* and *SALARY* can only be accessed by authorized users. However, to increase data availability, unauthorized users are allowed to access values for *NAME* and *SALARY* separately.

Suppose an unauthorized user submits the following two queries:

Query 1: "List the name and rank of the employees working in the Toy department." ($\Pi_{NAME,RANK}\sigma_{DEPARTMENT=Toy}$)

Query 2: "List the salaries of all clerks." ($\Pi_{SALARY}\sigma_{RANK='Clerk}$)

The answers to these queries are:

Query 1: $\{< John, Clerk >, < Mary, Secretary >\}$

Query 2: $\{< Clerk, 38,000 >\}$

Since the *Employee* relation satisfies the FD $RANK \rightarrow SALARY$, these answers reveal that John's salary is \$38,000. These kinds of inferences are correctly detected by DiIE.

Now, consider the following series of events when the *Employee* relation is updated between the queries. **Time 1**: User submits Query 1, **Time 2**: Salaries of all clerks are raised by 4%, **Time 3**: Employee John is promoted to the rank "Manager", and **Time 4**: User submits Query 2. The answers of the two queries are:

Query 1: $\{< John, Clerk >, < Mary, Secretary >\}$, (same as in the first scenario)

Query 2: $\{< Clerk, 39,520 >\}$

The second relation of Table 1 shows the updated *Employee* relation (Query 2). In this case, DiIE would indicate that subtuple $< John, 39,520 >$ is disclosed and, since it is confidential, Query 2 should be rejected. However, this subtuple is not contained in the updated relation and has never been present in any of the earlier versions. Therefore, the second query could be safely answered. The reason for this is that the FD was applied on data values that have been modified. We call inferences that were established on outdated (i.e., modified)

Table 1.   Employee relation

ID	NAME	RANK	SALARY	DEPT.
1	John	Clerk	38,000	Toy
2	Mary	Secretary	28,000	Toy
3	Chris	Secretary	28,000	Marketing
4	Joe	Manager	45,000	Appliance
5	Sam	Clerk	38,000	Appliances
6	Eve	Manager	45,000	Marketing

Original

ID	NAME	RANK	SALARY	DEPT
1	John	Manager	45,000	Toy
2	Mary	Secretary	28,000	Toy
3	Chris	Secretary	28,000	Marketing
4	Joe	Manager	45,000	Appliances
5	Sam	Clerk	39,520	Appliances
6	Eve	Manager	45,000	Marketing

Updated

data values "wrong" inferences. We will permit queries that generate "wrong" inferences to increase availability.

In this paper we propose a conceptual framework, called Dynamic Disclosure Monitor (D^{2}Mon), that guarantees data confidentiality and maximal availability even in the presence of inferences and updates. Our work extends the DiMon [1] model by taking into consideration that the data items received by a user may have been updated and are not valid any longer. To the authors' best knowledge, this is the first work that considers updates from the perspective of increased data availability. We propose formal characterizations of the effects of updates. We also study the cardinality inference problem that is created by observing changes in query answers over a period of time. Our model addresses both inferences via generalized database dependencies (represented as Horn-Clause constraints) and cardinality based (statistical) inferences within the same framework.

We develop a security mechanism, called *Update Consolidator* (UpCon) that, given a user's history file, updates on the base relation, and the database constraints, generates a new history file that marks outdated data values and indicates only valid inferences. All updates on the base relation are recorded in an update log. The history file of a user contains all data that the user previously received or can be disclosed from the received answers. When a new query is submitted by a user, the history file is modified according to the updates that have happened since the history file was last created. Each modified data value in the history file is stamped with the new, modified value. Intuitively, a stamped value means that the value is no longer valid, and the

new value, that is unknown to the user, is the stamp. Stamping will prevent the disclosure inference algorithm from indicating a disclosure based on outdated values (soundness) while maintaining maximal equalities among modified data values, thus allowing maximal dependency application (completeness). We show that with the use of UpCon the Data-Dependent Disclosure Inference Algorithm [1] is *sound* and *complete* even in the presence of updates.

Moreover, we recommend a *Cardinality Inference Detection* (CID) mechanism. This module can detect illegal inferences based on small query set size, query overlap, complementary attacks and aggregation based attacks. Since CID may detect a disclosure of a partial secret which, when combined with database dependencies, discloses a secret (DiIE) and vice verse; CID and DiIE work together until no more changes occur.

The paper is organized as follows. In the next section we describe our Dynamic Disclosure Monitor architecture. Section 3 contains the preliminaries and our notations. In Section 4 we present our Update Consolidator and Cardinality Inference Detection mechanisms. Finally, we conclude our research in Section 5 and recommend future extensions.

2.   SECURITY ARCHITECTURE

2.1.   Dynamic Disclosure Monitor

Our model is built upon the Disclosure Monitor (DiMon) security architecture, developed by Brodsky et al. [1], to protect data confidentiality against illegal inferences in Multilevel Secure (MLS) relational databases. However, DiMon addresses the inference problem in static databases, and thus may overprotect information due to some already modified data values (see the second example in the Introduction). Our extended architecture, called Dynamic Disclosure Monitor (D^2Mon), incorporates this liberating effect of updates, while still guaranteeing that no unauthorized disclosure is possible via data requests. Note, that we assume that updates are performed in a secure way, i.e., we do not consider covert channels created by updates. We propose two extensions to DiMon: 1) an Update Consolidator (UpCon) module that incorporates the effects of the updates on the history files maintained for each user; and 2) a Cardinality Inference Detection (CID) module that detects illegal inferences based on the cardinality of the released answers. The functionality of D^2Mon is shown in Figure 1. The shaded modules represent the components of DiMon. The actual algorithm of D^2Mon is given in Figure 2. The additional features of D^2Mon over DiMon are used when Disclosure Inference Engine (DiIE) works in data-dependent mode. Clearly, in data-independent mode when the actual data items received by a user are not known by DiIE, updates made on the base relation will not effect the processing of DiIE.
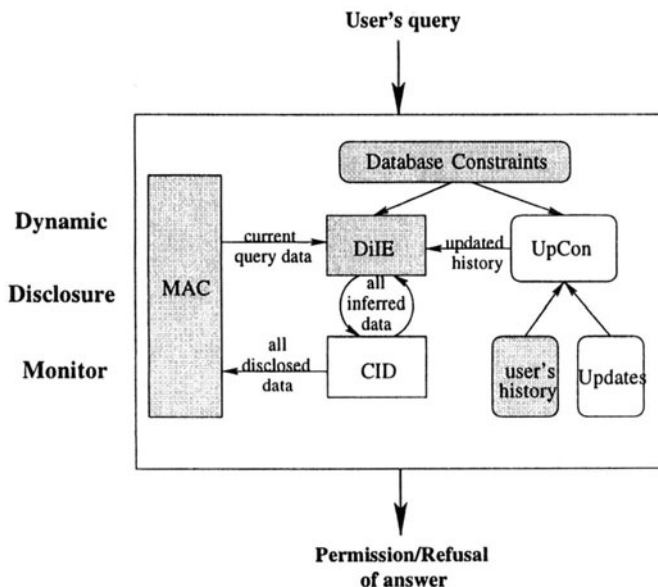
Figure 1.   Dynamic Disclosure Monitor

3.   PRELIMINARIES AND NOTATIONS

Before explaining the functionality of the *Update Consolidator* (UpCon) we need to modify the concept of data-dependent disclosure in dynamic databases and introduce some related concepts. We require that each tuple within the database has a unique tuple identifier ID that is maintained internally and not released to the user. Table 1 shows the original *Employee* relation with tuple identifiers. When a new record is added to the database, a unique ID is assigned to it. Modifications to the database are stored in the update log, which we will now define.

Definition 3.1 (Update)
Given a schema R and a relation r over R, an *update* UP on r is a 5-tuple of the form $(t, id, att, v_{old}, v_{new})$, where t is the time of the update, id is the identifier of the tuple in r that is being updated, att is the attribute name that is being updated such that $att \in R$, and v_{old}, v_{new} are the old and the new (updated) values of att, respectively such that $v_{old}, v_{new} \in dom(att) \cup \{-\}$. The special symbol $-$ is used to represent new values of the attributes when the update is a deletion.

Consider the examples presented in the Introduction. The updates at times t_2 and t_3 are recorded as follows: $UP = (t_2, 1, SALARY, 38,000, 39,520)$, $(t_2, 5, SALARY, 38,000, 39,520)$, $(t_3, 1, RANK, Clerk, Manager)$,

---

Algorithm 1: Dynamic Disclosure Monitor (D^{2}Mon)

INPUT

1. User's query (object) Q_i
2. User's id U
3. Security classification $< \mathcal{O}, \mathcal{U}, \lambda >$
4. User's history-file $U_{history}$: data which were previously retrieved by the user
5. $\mathcal{D}$, a set of database constraints

OUTPUT

Answer to Q_i and update of the user's history-file or refusal of Q_i

---

METHOD

MAC evaluates direct security violations:

IF direct security violation is detected **THEN** Q_i is rejected

(D^{2}Mon functions as the basic MAC mechanism)

ELSE (no direct security violation was detected)

BEGIN

    1 Use *Update Consolidator* (UpCon) to modify $U_{history}$ according to the relevant updates to create $U_{updated-history}$

    2 Let $U_{all-disclosed} = U_{updated-history} \cup Q_i(answers)$
       begin
       Repeat until no change occurs:

          (a) Use *Disclosure Inference Engine* (DiIE) to generate all data that can be disclosed from the $U_{all-disclosed}$ and the *database constraints* $\mathcal{D}$
             $U_{all-disclosed} = U_{all-disclosed} \cup U_{newly-disclosed}$

          (b) Use *Cardinality Inference Detection* (CID) to find all data that can be disclosed by cardinality inferences
             $U_{all-disclosed} = U_{all-disclosed} \cup U_{newly-disclosed}$

       end

    3 MAC reevaluates security violations in $U_{all-disclosed}$:
       IF illegal disclosure is detected **THEN** reject Q_i and
       $U_{history} = U_{updated-history}$

    4 **ELSE** (security is not violated) answer Q_i and
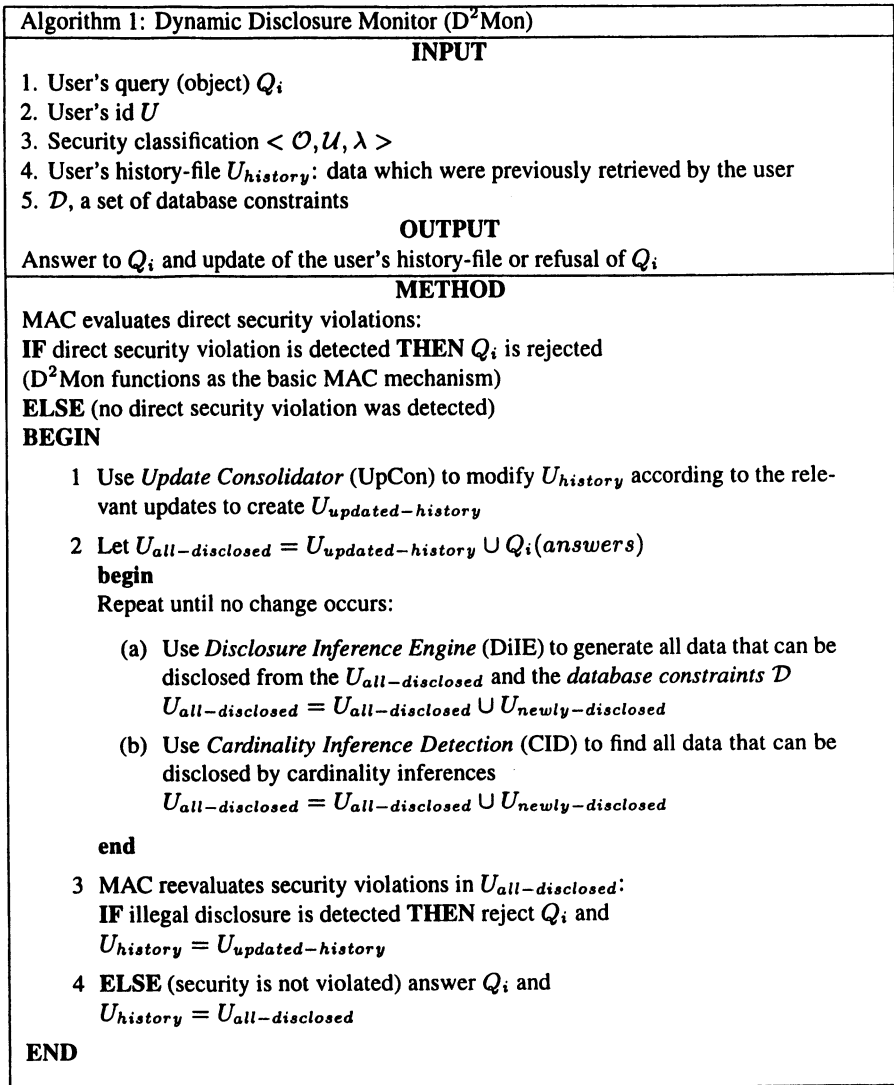 $U_{history} = U_{all-disclosed}$

END

Figure 2.   Algorithm for Dynamic Disclosure Monitoring

$(t_3, 1, SALARY, 39, 520, 45, 000)$. We represent updates on tuples which were previously released to a user by "stamping" these values with the new value. For this, we need the notion of a stamped attribute value.

Definition 3.2 (Stamped Attribute)

Let A be an attribute name and $dom(A) = a_1, \ldots, a_l$ the domain of A. A *stamped attribute* SA is an attribute such that its value sa is of the form

$a_i{}^{a_j}$ $(i, j = 1, \ldots, l)$. We call a_i the value of SA and a_j is the stamp or updated value of the stamped attribute SA.

For example, assume that the user has received the tuple $< Clerk, 38,000 >$ over the attributes *RANK* and *SALARY*. If at a later time the salaries of the clerks are modified, e.g, increased to \$39,520, the corresponding tuples in the history file are stamped as follows $< Clerk, 38,000^{39,520} >$.

Definition 3.3 (Equalities of Stamped Attributes)
Given two stamped attributes $sa = a_i{}^{a_j}$ and $sb = b_i{}^{b_j}$, we say that $sa = sb$ *iff* $a_i = b_i$ and $a_j = b_j$. Given a stamped attribute $sa = a_i{}^{a_j}$ and a non-stamped attribute b, we say that $sa = b$ *iff* $a_i = b$ and $a_i = a_j$.

Next, we need the notion of projection facts and stamped projection facts which are manipulated by the DiIE.

Definition 3.4 (Projection Fact)
A *projection fact* (PF) of type $A_1, \ldots, A_k$, where $A_1, \ldots, A_k$ are attributes in R, is a mapping m from $\{A_1, \ldots, A_k\}$ to $\bigcup_{j=1}^{k} dom(A_j) \cup \bigcup_{j=1}^{k} dom(SA_j)$ such that $m(A_j) \in dom(A_j) \cup dom(SA_j)$ for all $j = 1, \ldots, k$. A projection fact is denoted by an expression of the form $R[A_1 = v_1, \ldots, A_k = v_k]$ where R is the relation name, and $v_1, \ldots, v_k$ are values of attributes $A_1, \ldots, A_k$, respectively.

A *stamped projection fact* (SPF) is a projection fact $R[A_1 = v_1, \ldots, A_k = v_k]$, where at least one of v_j $(j = 1, \ldots, k)$ is a stamped attribute.

A *non-stamped projection fact*, or simply a projection fact, is a projection fact $R[A_1 = v_1, \ldots, A_k = v_k]$, where all v_js are constants in $dom(A_j)$.

For example, $Employee[NAME = John, Rank = Clerk]$ is a projection fact, and $Employee[NAME = John, Rank = Clerk^{Manager}]$ is a stamped projection fact.

Definition 3.5 (Un-Stamping a Stamped Projection Fact)
Given a SPF of the form $R[A_1 = a_1^{b_1}, \ldots, A_k = a_k^{b_k}]$, where some of the b_i $(i = 1, \ldots, k)$ may not exist. The un-stamped projection fact PF is generated from SPF by simply removing the stamps of SPF, i.e., $PF = R[A_1 = a_1, \ldots, A_k = a_k]$

Definition 3.6 (Query-answer pair)
An *atomic query-answer pair* (QA-pair) is an expression of the form $(P, \Pi_Y \sigma_C)$, where P is a projection fact over Y that satisfies C or P is a stamped projection fact, such that the un-stamped projection fact generated from P satisfies C. A query-answer pair is either an atomic QA-pair or an expression of the form $(\mathcal{P}, \Pi_Y \sigma_C)$, where $\mathcal{P}$ is a set of (stamped) projection facts $\{P_1, \ldots, P_l\}$ such that every P_i, $(i = 1, \ldots, l)$ is over Y and satisfies C.

Definition 3.7 (Dynamic data-dependent disclosure)

Let $\mathcal{D}$ be a set of database constraints, $\mathcal{UP} = \{UP_1, \ldots, UP_m\}$ be a set of updates, $P_1, \ldots, P_n$ be sets of projection facts over attribute sets $X_1, \ldots, X_n, PF$ be a projection fact over Y, and $t_1 \leq \ldots \leq t_n \leq t$ are times. We say that the set $\mathcal{P} = \{[(P_1, \Pi_{X_1}\sigma_{C_1}), t_1], \ldots, [(P_n, \Pi_{X_n}\sigma_{C_n}), t_n]\}$ *discloses* $[(PF, \Pi_Y\sigma_C), t]$ under database constraints $\mathcal{D}$ and in the presence of updates $\mathcal{UP}$, if

1) there is no update recorded on P_is $(i = 1, \ldots, n)$ between t_n and t, and

2) for every $r_0, r_1, \ldots, r_n, r$ over R, that all satisfy $\mathcal{D}$, $P_i \subseteq \Pi_{X_i}\sigma_{C_i}(r_i)[t_i]$ $(i = 1, \ldots, n)$ implies $PF \in \Pi_Y\sigma_C(r)[t]$, where r_0 is the original relation and r_i $(i = 1, \ldots, n), r$ are the relations generated from r_0 by the updates in $\mathcal{UP}$, respectively. Dynamic data-dependent disclosure is denoted as $\mathcal{P} \models_{\mathcal{UP},\mathcal{D}} [(PF, \Pi_Y\sigma_C), t]$.

We denote by $[(PF, \Pi_Y\sigma_C), t_1] \models_{\mathcal{UP}} [(PF', \Pi_{Y'}\sigma_{C'}), t_2]$ the disclosure $\{[(PF, \Pi_Y\sigma_C), t_1]\} \models_{\mathcal{UP},\mathcal{D}} [(PF', \Pi_{Y'}\sigma_{C'}), t_2]$, where $\mathcal{D}$ is empty. In this case, we say that $[(PF, \Pi_Y\sigma_C), t_1]$ *dominates* $[(PF', \Pi_{Y'}\sigma_{C'}), t_2]$.

The above definition requires that all updates are performed according to the database constraints; that is, if a relation r satisfies the set of constraints $\mathcal{D}$ before an update, it will also satisfy $\mathcal{D}$ after the update. Furthermore, disclosure is established based on updated tuples rather than the originally released tuples. Intuitively, this may allow a user, who is unaware of the update, to generate a fact that may have never existed in the base relation or is not valid any longer. In either case, we allow such "wrong" inferences, since our goal is to prevent disclosure of valid sensitive data.

By extending the concepts of projection fact and query-answer pairs to incorporate stamped attributes, DiIE can operate on relations containing regular attribute values (constants), stamped attribute values, and null-values. In the following section we develop the Update Consolidator (UpCon) module that creates a new input history file for DiIE by stamping those attribute values that have been modified by an update operation.

4. UPDATE AND CARDINALITY INFERENCES

In this section we introduce our remaining extensions, the Update Consolidator (UpCon) and the Cardinality Inference Detection (CID) modules.

4.1. Update Consolidator

As we illustrated in the Introduction, in dynamic databases it may happen that answers received by a user may not be valid any longer after an update. Clearly, if we refuse a new request of the user based on some outdated data value we unnecessarily limit data access. Update Consolidator (UpCon) propagates updates that happened between the current and the last answered query to the user's history file. For this we require that every tuple of the relation has a unique

Table 2. The *User's* history file after Query 2

ID	NAME	RANK	SALARY	DEPARTMENT
1	John	Clerk^{Manager}	δ_1	Toy
2	Mary	Secretary	δ_2	Toy
5	δ_3	Clerk	39,520	Toy

identifier and all original answers (after the application of selection conditions on the answers) are kept permanently. Also, we keep track of the database dependencies D that were previously applied on the history-file. Figure 4 shows the algorithm of UpCon. Table 2 shows the stamped history file (tuples 1 and 2) and the incorporated answer of Query 2. Clearly, in this case the FD $RANK \rightarrow SALARY$ cannot be applied, since $Clerk^{Manager} \neq Clerk$.

Theorem 1 (Dynamic Data Decidability)
The following problem is decidable: Given a set $\mathcal{D}$ of database constraints, a set of updates $\mathcal{UP}$, and a set $\mathcal{P}$ of QA-pairs with the time when the QA-pairs have been generated, whether $\mathcal{P}\models_{\mathcal{UP},\mathcal{D}}S$ for a given set S of atomic QA-pairs.

Theorem 1 will be a corollary to Theorem 2 that states correctness (i.e., soundness and completeness) of DiIE in the presence of updates.

Proposition 4.1 *Given a set $D = d_1, \ldots, d_k$ dependencies that can be applied by DiIE on a set of projection facts S, i.e., projection facts without stamped attributes. Then at most $D = d_1, \ldots, d_k$ can be applied on S', a set of stamped projection facts generated from S by stamping some of the attribute values.*

Proofsketch 4.1 Assume by contradiction that a dependency d_m can be applied on S' but not on S and that $d_m \notin D$. Since a dependency can be applied only if we can find a valuation from the body of the dependency to the (stamped) projection facts, such valuation must exist from the body of d_m to S'. Since stamping cannot introduce new equalities (Definition 3.3) there must also exist a valuation from the body of d_m to S, thus d_m is applicable on S, which contradicts our original assumption.

Theorem 2 The data-dependent disclosure inference algorithm is *sound* and *complete* when used with UpCon.

Proofsketch 4.2 To prove Theorem 2 we use that the Data-Dependent Disclosure Algorithm is sound and complete in static databases. This is a special case of dynamic data-dependent disclosure (Definition 3.7), when $UP = \emptyset$, thus $r = r_1 = \ldots = r_n$. Then, we need only to show that stamping a released data value will not incorrectly modify a history file, i.e., all valid equalities

---

Algorithm 2: Update Consolidator (UpCon)
INPUT
1. Time of last query $time_{last}$ 2. Time of current query $time_{current}$ 3. Update log $\mathcal{UP} = \{up_1, up_2, \ldots, up_m\}$ 4. User's history-file $U_{history}$ 5. DA, a set of database constraints previously applied on $U_{history}$
OUTPUT
Updated history-file of the user $U_{updated-history}$
METHOD

Construct $TUP = \{up_1, up_2, \ldots, up_l\}$ (temporary update file) from $\mathcal{UP}$ such that
 	
 ■ $time_{last} \leq up_i[t] \leq time_{current}$, where $up_i[t]$ is the time of the update, and
 	
 ■ there exists a tuple t_j in $U_{history}$ such that $up_i[id] = t_j[id]$, $up_i[id]$ is the identifier of the tuple that has been updated and $t_j[id]$ is the identifier of the tuple t_j in $U_{history}$.
 	
 IF $TUP \neq \emptyset$ (i.e., updates occurred on the tuples of the base relations from which some of the released tuples originated) **THEN**
 BEGIN
 	
    1 $U_{history} = U_{history} - \{t \ if \ t[id] =''\ 999''$ (drop inferred tuples)
 	
    2 For every $up_i = (t, id, att, old, new) \in TUP$ **DO**
 	
      ■ Find tuple t in $U_{history}$ such that $t_{id} = id$
 	
      ■ Stamp value old of attribute att in t with value new
 	
    3 $U_{updated-history} = U_{history}$
 	
    4 Chase $U_{updated-history}$ with DA.
 	
 END

Figure 3. Update Consolidator

exploited by the inference algorithm to apply a database dependency remain intact (completeness) and only the valid equalities are present (soundness).

Since stamping an attribute value only reduces possible equalities, and therefore, possible application of the dependencies, the proof of preserving the soundness property is straight forward.

The proof of completeness is constructed as follows: assume by contradiction that a tuple t is correctly disclosed from the originally released tuples, the current query answer, the database dependencies and the updates but t was not generated by DiIE applied on the stamped history file. Since the Data-Dependent Inference Algorithm, used by DiIE is complete and t is disclosed, DiIE must generate a tuple that dominates t if the originally released tuples, the current query answer and the database dependencies are given as input. t

clearly cannot be one of the originally released tuples, because those are stored in the stamped history and stamps are only added according to the updates. But then, t must have been generated by a sequence of dependency applications $d_1, \ldots, d_l$. But since the updates satisfied the database dependencies and t is correctly disclosed based on valid inferences then $d_1, \ldots, d_l$ must be applicable on the stamped history file. But then, $d_1, \ldots, d_l$ would have been applicable on the stamped history file, and DiIE would have created a tuple that coincides with t on the non-stamped attributes. This is a contradiction of our original assumption.

4.2.   Cardinality Inference Detection (CID)

In this section we present an additional module to detect inferences based on the cardinalities of the query answers. Similar inferences have been considered in statistical databases (see Denning [4] for an overview) and protection techniques have been recommended. CID protects against illegal inferences due to small query set size, query overlap, complementary inferences, and data aggregation. CID and DiIE share their output repeatedly, until all possible inferences are detected. Due to space limitations, CID is not described in details in this paper.

5.   CONCLUSIONS AND FUTURE WORK

In this paper we present a Dynamic Disclosure Monitor architecture that guarantees data confidentiality and maximal data availability in the presence of inferences based on database constraint and updates. We propose two extensions of DiMon to incorporate the effects of updates in order to increase data availability and decrease the risk of cardinality inferences: 1) UpCon uses the user's existing history file, updates that occurred, and the database constraints to generate a new history file that does not contain any outdated data values; 2) CID checks for cardinality based inferences, such as small query set size, query overlap, complementary inferences, and data aggregation. CID and DiIE share their output repeatedly, until all possible inferences are detected. We show that using UpCon with DiIE will guarantee data secrecy (completeness property of inference algorithm) and data availability (soundness property of inference algorithm) in the presence of inferences via database constraints.

We conclude with suggestions for further work. Currently we are developing a prototype of DiMon and its extensions to D^2Mon. We use Java to express the security requirements and implement the disclosure inference algorithms. Completion of the prototype will allow us to provide experimental results in addition to theoretical evaluation of the model. In addition, we recommend the following improvements: Currently the model allows all "wrong" inferences, regardless of whether they reveal data items that had previously existed in the

relation or not. However, in certain applications even outdated data might be sensitive, i.e., that John, as a clerk, earned $38,000. Our update-log based model can be extended to allow distinction of the different kind of inferences.

Also, this paper focuses on multiple attacks of a single user. However in a real-life situation, malicious users can share information, thus obtaining data for which they do not have the proper authorization. Users likely to share information could be grouped together to prevent security violations via collaborating users.

References

[1] A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosure. *IEEE Trans. Knowledge and Data Eng.*, November, 2000.

[2] L.J. Buczkowski. Database inference controller. In D.L. Spooner and C. Landwehr, editors, *Database Security III: Status and Prospects*, pages 311–322. North-Holland, Amsterdam, 1990.

[3] S. Dawson, S.De Capitani di Vimercati, and P. Samarati. Specification and enforcement of classification and inference constraints. In *Proc. of the 20th IEEE Symposium on Security and Privacy, Oakland, CA*, May 9–12 1999.

[4] D.E. Denning. *Cryptography and Data Security*. Addison-Wesley, Mass., 1982.

[5] D.E. Denning. Commutative filters for reducing inference threats in multilevel database systems. In *Proc. IEEE Symp. on Security and Privacy*, pages 134–146, 1985.

[6] D.E. Denning and T.F. Lunt. A multilevel relational data model. In *Proc. IEEE Symp. on Security and Privacy*, pages 220–233, 1987.

[7] J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symp. on Security and Privacy*, pages 75–86, 1984.

[8] T.H. Hinke. Inference aggregation detection in database management systems. In *Proc. IEEE Symp. on Security and Privacy*, pages 96–106, 1988.

[9] S. Jajodia, R. Sandhu, and B. T. Blaustein. Solution to the polyinstantiation problem. In M. Abrams et al., editor, *Information Security: An Integrated Collection of Essays*. IEEE Computer Society Press, 1995.

[10] T. F. Keefe, M. B. Thuraisingham, and W. T. Tsai. Secure query-processing strategies. *IEEE Computer*, pages 63–70, March 1989.

[11] D.G. Marks. Inference in MLS database systems. *IEEE Trans. Knowledge and Data Eng.*, 8(1):46–55, February 1996.

[12] D.G. Marks, A. Motro, and S. Jajodia. Enhancing the controlled disclosure of sensitive information. In *Proc. European Symp. on Research in Computer Security*, Springer-Verlag Lecture Notes in Computer Science, Vol. 1146, pages 290–303, 1996.

[13] S. Mazumdar, D. Stemple, and T. Sheard. Resolving the tension between integrity and security using a theorem prover. In *Proc. ACM Int'l Conf. Management of Data*, pages 233–242, 1988.

[14] M. Morgenstern. Controlling logical inference in multilevel database systems. In *Proc. IEEE Symp. on Security and Privacy*, pages 245–255, 1988.

[15] G.W. Smith. Modeling security-relevant data semantics. In *Proc. IEEE Symp. Research in Security and Privacy*, pages 384–391, 1990.

[16] P.D. Stachour and B. Thuraisingham. Design of LDV: A multilevel secure relational database management system. *IEEE Trans. Knowledge and Data Eng.*, 2(2):190–209, June 1990.

[17] T. Su and G. Ozsoyoglu. Inference in MLS database systems. *IEEE Trans. Knowledge and Data Eng.*, 3(4):474–485, December 1991.

[18] T.H.Hinke, Harry S. Delugach, and Asha Chandrasekhar. A fast algorithm for detecting second paths in database inference analysis. *Jour. of Computer Security*, 3(2,3):147–168, 1995.

[19] B.M. Thuraisingham. Security checking in relational database management systems augmented with inference engines. *Computers and Security*, 6:479–492, 1987.

[20] R. W. Yip and K. N. Levitt. Data level inference detection in database systems. In *Proc. of the 11th IEEE Computer Security Foundation Workshop, Rockport, MA*, pages 179–189, June 1998.