View-Based Access Control with High Assurance*

Xiaolei Qian
Computer Science Laboratory
SRI International
qian@csl.sri.com

Abstract

View-based access control enables content-based and context-based security, as opposed to container-based security provided in operating systems. However, view-based access control in multilevel secure (MLS) databases suffers from two problems: safety and assurance. We investigate view-based access control in MLS relational databases for a large class of views expressible as project-select-join queries. We develop a polynomial-time label compilation algorithm that transforms view-level labeling to tuple-level labeling in such a way that guarantees safety and high assurance. We identify two problems related to optimal label compilation, and show that they are both NP-complete even for totally ordered security lattices of size two.

1 Introduction

Views in relational databases have long been considered ideal as the objects of access control, because they have a higher degree of logical abstraction than physical data and hence enable content-based or context-based security, as opposed to container-based security provided in operating systems.

However, view-based access control has not been in wide-spread use in multilevel secure (MLS) relational databases because of two major problems [10].

- **Safety**. The safety question asks the following. Is there a database state in which a particular user possesses a particular privilege for data in a specific view? In container-based access control, different containers do not share contents.[1] Hence, a secret label on a container ensures that data in the container are not accessible to unclassified users. In contrast, in view-based access control, views might overlap and the same data might satisfy more than one view. Hence, a secret label on a view does not guarantee that data contained in the view are not accessible to unclassified users. This is not acceptable, because it means that data might not be labeled consistently.

*This work was supported in part by the U.S. Department of Defense Advanced Research Projects Agency and the U.S. Air Force Rome Laboratory under contract F30602-94-C-0198, and in part by the National Science Foundation under grant ECS-94-22688.

[1] Even though two containers might store different copies of the same data, their contents do not overlap physically.

- **Assurance**. The assurance question asks the following. Can the safety of a particular view-based access control policy be determined with small amount of trusted code? In container-based access control, containers labeled differently can be allocated to disjoint address spaces, because they do not share contents. Hence, high assurance is achievable with a minimal amount of trusted code. In contrast, in view-based access control, the trusted computing base (TCB) is likely to be very large, including most code of a DBMS and especially the query processor that interprets views. This is not acceptable, because a small TCB is required for certification of multilevel systems above class B1.

To retain the advantages of both content-based security and container-based security, view-based access control in MLS relational databases poses the following challenge: how to bridge the gap between content-based security specification and container-based security implementation.

2 Motivating Examples

Let us consider a FLIGHT database containing three relations.

Payload (flight#, item#, weight)
Flights (flight#, date, destination, capacity)
Item (item#, itemname, type)

A secret view could be defined on the FLIGHT database to protect all flights carrying bombs to Iran.

```
CREATE VIEW Bomb_Iran (flight#, weight, date)
SELECT      Payload.flight#, Payload.weight, Flights.date
FROM        Payload, Flights, Item
WHERE       Flights.flight# = Payload.flight#
            AND Item.item# = Payload.item#
            AND Flights.destination = iran
            AND Item.itemname = bomb.
```

A confidential view could also be defined on the FLIGHT database to protect flights that carry large amounts (≥ 100) of explosives.

```
CREATE VIEW Large_Explosive (flight#, name)
SELECT      Payload.flight#, Item.itemname
FROM        Payload, Item
WHERE       Item.item# = Payload.item#
            AND Item.type = explosive
            AND Payload.weight ≥ 100.
```

Large_Explosive overlaps with Bomb_Iran in those flights that carry large amounts of bombs to Iran. A secret label on Bomb_Iran does not guarantee that these flights are not accessible to confidential users—they can access these flights through Large_Explosive.

We could define a third unclassified view could be defined on the FLIGHT database for low-capacity flights to Kuwait with not-fully-loaded item vxs606:

```
CREATE VIEW Kuwait_VXS606 (flight#)
SELECT      Payload.flight#
FROM        Payload, Flights
WHERE       Payload.flight# = Flights.flight#
            AND Payload.item# = vxs606
            AND Payload.weight < Flights.capacity
            AND Flights.destination = kuwait
            AND Flights.capacity ≤ 80.
```

Kuwait_VXS606 does not overlap with Large_Explosive, but it overlaps with Bomb_Iran in those payloads of item `vxs606`. A secret label on Bomb_Iran does not guarantee that these payloads are not accessible to unclassified users—they can access these payloads through Kuwait_VXS606.[2]

3 Solution: Label Compilation

To bridge the gap between content-based security specification and container-based security implementation, we develop an efficient label compilation algorithm that transforms view-level labeling to tuple-level labeling.

The algorithm takes a set of labeled views as input. Views are expressed as project-select-join queries where the selection condition involves comparisons of attributes to constants or other attributes. Labels on views are transformed to labels on primitive views, which are views expressed on single relations. These labeled primitive views are then used to assign labels to tuples in primitive views. The algorithm runs in polynomial time.

Label compilation using our algorithm has several desirable properties:

- **Well-Foundedness**. Compilation is deterministic. In other words, every tuple in a relational database is assigned exactly one unique label.
- **Proper Classification**. Explicit data in a view are labeled no higher than the label of the view. In other words, a secret label on a view guarantees that explicit data in the view are accessible to secret users.
- **Safety**. Explicit data in a view are labeled no lower than the label of the view. In other words, a secret label on a view guarantees that explicit data in the view are not accessible to unclassified users.

Moreover, our algorithm succeeds if and only if a well-founded, proper, and safe compilation exists. In other words, it is both sound and complete. High assurance is achieved by enforcing access

[2]Notice that payloads in Bomb_Iran and Kuwait_VXS606 could not overlap, if we knew that every payload is on a flight (i.e., Payload.flight# is a foreign key to Flights) and every flight has a unique destination (i.e., Flights.flight# is the primary key of Flights).

control not on views but on tuples. Hence, only the label compilation code needs to be trusted, and the TCB is comparable in size to those in MLS relational databases with tuple-level labeling.

We also investigate the complexity of two problems related to optimal compilation. The first is the problem of lowest classification. When there are more than one well-founded, proper, and safe compilation, we would want to choose the one in which tuples are labeled at the lowest possible levels, since doing so maximizes data sharing among users at different levels. The second is the problem of minimal upgrade. When a well-founded, proper, and safe compilation does not exist because some views are labeled inappropriately low, we would want to upgrade a minimal number of views such that compilation succeeds, since doing so maximizes the number of users eligible to access data through these views. We show that both problems are NP-complete even for totally ordered security lattices of size two.

The rest of the paper is organized as follows. In Section 4, we introduce multilevel schemas and view-level labeling. In Section 5, we present an algorithm for transforming view-level labeling to tuple-level labeling. We investigate the complexity of optimal compilation in Section 6. Finally, we compare our results with related work in Section 7, and conclude the paper in Section 8 with a brief discussion of future work.

4 Multilevel Schemas with View-Level Labeling

4.1 Relational Model

Let U be a finite collection of *attributes*. If X, Y are sequences of attributes in U, then XY denotes the concatenation of X, Y. For $A \in U$, XA denotes $X\langle A\rangle$. A *relation scheme* $R[X]$ is a sequence of attributes $X \subseteq U$ named R. A *schema* $\mathcal{B}$ is a family of relation schemes $\{R_i[X_i]\}_{1\leq i\leq n}$. For example, the schema of the FLIGHT database contains three relation schemes—Payload, Flights, and Item.

Let D be a (possibly infinite) set of values. A *tuple* over attributes X is a mapping $t[X]\colon X \mapsto D$ that assigns values from D to attributes in X. A *relation* r over relation scheme $R[X]$ is a set of tuples over X. A *database* B over schema $\mathcal{B} = \{R_i[X_i]\}_{1\leq i\leq n}$ is a family of relations $\{r_i\}_{1\leq i\leq n}$, where r_i is a relation over $R_i[X_i]$.

4.2 Views as Conjunctive Queries

We consider views that are expressible as project-select-join (PSJ) queries whose selection condition is a conjunction of comparisons of attributes to constants or other attributes.[3] These queries are equivalently expressible as conjunctive queries with built-in predicates [11].

We assume that predicates are divided into three disjoint groups: base predicates, built-in predicates, and view predicates. Given a schema $\mathcal{B}$, there is an $|X|$-ary base predicate R for every relation scheme $R[X]$ in $\mathcal{B}$. Built-in predicates are $=, \neq, <, \leq, >, \geq$. We assume that there is an unbounded supply of view predicates.

[3]Our results generalize to PSJ queries whose selection condition involves disjunction, since they can be represented equivalently by multiple PSJ queries without disjunction.

An atomic formula involving a base, built-in, or view predicate is a ***base, built-in***, or ***view literal***, respectively. Given a set C of literals, the sets of variables and constants in C are denoted as $\mathrm{var}(C)$ and $\mathrm{const}(C)$, respectively. Given a set C of built-in literals, the ***closure*** of C, denoted as C^*, is the set of built-in literals c such that $\mathrm{var}(c) \subseteq \mathrm{var}(C)$, $\mathrm{const}(c) \subseteq \mathrm{const}(C)$, and $C \models c$. Given a base literal p and a set C of built-in literals, the ***projection*** of C on p, denoted as $\pi_p(C)$, is the set of literals $c \in C^*$ such that $\mathrm{var}(c) \subseteq \mathrm{var}(p)$.

A *view* V defined on schema $\mathcal{B}$ has the form

$$h \quad :- \quad p_1, \ldots, p_m, c_1, \ldots, c_{m'} \tag{1}$$

where $h, p_1, \ldots, p_m, c_1, \ldots, c_{m'}$ are literals whose arguments are variables or constants, h is a view literal named the ***head***, $p_1, \ldots, p_m$ is a list of base literals named the ***body***, and $c_1, \ldots, c_{m'}$ is a list of built-in literals named the ***constraint***. Variables in the head are ***distinguished***. We assume that every variable in the head or the constraint also appears in the body. As a convention, we use $x, y, \ldots$ for distinguished variables and $w, u, \ldots$ for other variables. We also use $\mathrm{head}(V)$, $\mathrm{body}(V)$, and $\mathrm{constraint}(V)$ to denote the head, body, and constraint of V, respectively.

Example 1 The three views on the FLIGHT database in Section 2 can be expressed as the following conjunctive queries:

$$\begin{aligned}
V_1(x, y, z) &:- \mathrm{Payload}(x, u, y), \mathrm{Flights}(x, z, \texttt{iran}, v), \mathrm{Item}(u, \texttt{bomb}, w) \\
V_2(x, y) &:- \mathrm{Payload}(x, u_1, u_2), \mathrm{Item}(u_1, y, \texttt{explosive}), u_2 \geq 100 \\
V_3(x) &:- \mathrm{Payload}(x, \texttt{vxs606}, u), \mathrm{Flights}(x, v_1, \texttt{kuwait}, v_2), u < v_2, v_2 \leq 80.
\end{aligned}$$

□

Let V be the view (1). V is ***primitive*** if it contains one base literal, namely, $m = 1$. V is ***simple*** if it contains no built-in literals, namely, $m' = 0$. For example, view V_1 in Example 1 is simple but not primitive. Primitive view V is defined on relation scheme $R[X]$ if the base predicate in p_1 is R.

Let $\mathcal{P}$ be a set of primitive views. Two primitive views $P, P' \in \mathcal{P}$ ***overlap***, denoted as $P \simeq P'$, if

- a most-general unifier $\theta = \mathrm{mgu}(\mathrm{body}(P), \mathrm{body}(P'))$ exists, and
- $\mathrm{constraint}(P)\theta$ and $\mathrm{constraint}(P')\theta$ together are satisfiable.

Notice that the transitive closure of $\simeq$, denoted as $\simeq^*$, is an equivalence relation on $\mathcal{P}$. For primitive view $P \in \mathcal{P}$, $P/\simeq^*$ denotes the equivalence class of P under $\simeq^*$, which is the set of primitive views in $\mathcal{P}$ that are related to P through $\simeq^*$, namely, $\{P' | P' \in \mathcal{P} \wedge P \simeq^* P'\}$. $\mathcal{P}/\simeq^*$ denotes the set of equivalence classes of primitive views in $\mathcal{P}$ under $\simeq^*$, namely, $\{P/\simeq^* \mid P \in \mathcal{P}\}$.

Let P be a primitive view on $R[X]$. Given a relation r over $R[X]$, we denote the value of P in r by $P(r)$. A tuple $t \in r$ ***satisfies*** P if $P(\{t\})$ is not empty. Let V be a view on $\mathcal{B}$. Given a database B over $\mathcal{B}$, we denote the value of view V in B by $V(B)$.

Let V be the view (1). The *cover* of V, denoted as cover(V), is a family $\{P_i\}_{1 \leq i \leq m}$ of primitive views, one for every base literal in V, where P_i is

$$h_i \quad :- \quad p_i, \pi_{p_i}\text{constraint}(V)$$

where h_i is a view literal obtained by replacing the base predicate in p_i by a new view predicate. Let V' be the view

$$h \quad :- \quad p_1, \ldots, p_m, \text{constraint}(P_1), \ldots, \text{constraint}(P_m).$$

Notice that the collection of constraints in the cover of V is weaker than the constraint of V, namely, $(\bigcup_{1 \leq i \leq m} \text{constraint}(P_i))^* \subseteq \text{constraint}(V)^*$. Hence, $V(B) \subseteq V'(B)$ for every database B over schema $\mathcal{B}$.

Example 2 The covers of views in Example 1 contain the following primitive views:

$$\begin{aligned}
P_1(x, y, z) &:- \text{Payload}(x, y, z) \\
P_2(x, y, \texttt{iran}, z) &:- \text{Flights}(x, y, \texttt{iran}, z) \\
P_3(x, \texttt{bomb}, y) &:- \text{Item}(x, \texttt{bomb}, y) \\
P_4(x, y, z) &:- \text{Payload}(x, y, z), z \geq 100 \\
P_5(x, y, \texttt{explosive}) &:- \text{Item}(x, y, \texttt{explosive}) \\
P_6(x, \texttt{vxs606}, y) &:- \text{Payload}(x, \texttt{vxs606}, y), y < 80 \\
P_7(x, y, \texttt{kuwait}, z) &:- \text{Flights}(x, y, \texttt{kuwait}, z), z \leq 80.
\end{aligned}$$

P_1 overlaps with P_4 and P_6. P_4 does not overlap with P_6; neither does P_2 overlap with P_7. P_3 overlaps with P_5. □

4.3 View-Level Labeling

A *security lattice* is a lattice $(L, \preceq)$, where L is a set of *levels* and $\preceq$ is the *dominance relation*. For example, a security lattice could be $\{U, C, S\}$, where $U \preceq C \preceq S$.

We consider view-based access control in MLS relational databases, where mandatory access control policies with the simple security property and the $*$-property of the Bell-LaPadula model [7] are enforced. Given a security lattice, every user is assigned a clearance level, and every object is assigned a classification level, both of which are from the lattice.

- **The Simple Security Property**. A user is allowed a read access to an object only if the former's clearance level is identical to or higher than the latter's classification level in the lattice.
- **The $*$-Property**. A user is allowed a write access to an object only if the former's clearance level is identical to or lower than the latter's classification level in the lattice.

A *multilevel relation scheme* is a pair $(R[X], \mathcal{L})$, where $R[X]$ is a relation scheme and $\mathcal{L}$ is a lattice. A *multilevel schema* is a pair $(\mathcal{B}, \mathcal{L})$, where $\mathcal{B}$ is a schema and $\mathcal{L}$ is a lattice. For example, the schema of the FLIGHT database in Section 2 together with the lattice above form a multilevel schema of the FLIGHT database.

A multilevel schema with *view-level labeling* is a quadruple $(\mathcal{B}, \mathcal{L}, \mathcal{V}, \Delta)$, where $(\mathcal{B}, \mathcal{L})$ is a multilevel schema, $\mathcal{V}$ is a family of views on $\mathcal{B}$, and Δ is a mapping from views in $\mathcal{V}$ to levels in $\mathcal{L}$. For example, the multilevel schema above together with the three labeled views in Example 1 form a multilevel schema with view-level labeling of the FLIGHT database.

5 Compiling to Tuple-Level Labeling

5.1 Multilevel Databases with Tuple-Level Labeling

Let $\mathcal{L} = (L, \preceq)$ be a lattice, and $(R[X], \mathcal{L})$ be a multilevel relation scheme. A *multilevel relation with tuple-level labeling* over $(R[X], \mathcal{L})$ is a pair (r, λ), where r is a relation over $R[X]$ and λ is a mapping from tuples over X to levels in $\mathcal{L}$, such that $\lambda(t) = l$ if and only if $t \in r$ and t is labeled l.

Let $(\mathcal{B}, \mathcal{L})$ be a multilevel schema, where $\mathcal{B} = \{R_i[X_i]\}_{1 \le i \le n}$. A *multilevel database with tuple-level labeling* over $(\mathcal{B}, \mathcal{L})$ is a family $\{(r_i, \lambda_i)\}_{1 \le i \le n}$, where (r_i, λ_i) is a multilevel relation with tuple-level labeling over $(R_i[X_i], \mathcal{L})$. We denote it by the pair (B, Λ), where $B = \{r_i\}_{1 \le i \le n}$ is a database over $\mathcal{B}$, and $\Lambda = \{\lambda_i\}_{1 \le i \le n}$ is a family of mappings.

Let $l \in L$ be a level. The *l-slice* of multilevel relation (r, λ), denoted as $(r, \lambda)_l$, is a single-level relation containing all tuples in r that are labeled at or below l, namely, $\{t | t \in r \wedge \lambda(t) \preceq l\}$. The *l-slice* of multilevel database $(B, \Lambda) = \{(r_i, \lambda_i)\}_{1 \le i \le n}$, denoted as $(B, \Lambda)_l$, is a single-level database containing the l-slices of relations in B, namely, $\{(r_i, \lambda_i)_l\}_{1 \le i \le n}$.

Given a view V on schema $\mathcal{B}$ and a multilevel database (B, Λ) over multilevel schema $(\mathcal{B}, \mathcal{L})$, the *value* of V in (B, Λ) at l, denoted as $V_l(B, \Lambda)$, is the value of V in the l-slice of (B, Λ), namely, $V((B, \Lambda)_l)$.

5.2 Label Compilation

Let $(\mathcal{B}, \mathcal{L}, \mathcal{V}, \Delta)$ be a multilevel schema with view-level labeling, and $\mathcal{P}$ be the set of primitive views in the covers of views in $\mathcal{V}$, namely, $\bigcup_{V \in \mathcal{V}} \text{cover}(V)$. A *compilation* of labeled views in $(\mathcal{V}, \Delta)$ is a pair $(\mathcal{P}, \delta)$, where δ is a mapping from primitive views in $\mathcal{P}$ to levels in $\mathcal{L}$.

The labeled primitive views in $(\mathcal{P}, \delta)$ can serve as classification constraints in a multilevel database with tuple-level labeling over multilevel schema $(\mathcal{B}, \mathcal{L})$. In particular, for relation r over relation scheme $R[X]$, we can construct a multilevel relation with tuple-level labeling (r, λ) over multilevel relation scheme $(R[X], \mathcal{L})$ as follows. For every primitive view $P \in \mathcal{P}$ on $R[X]$ and every tuple $t \in r$ that satisfies P, $\lambda(t) = \delta(P)$. All other tuples in r are labeled at the bottom level of $\mathcal{L}$. Likewise, for database B over schema $\mathcal{B}$, we can construct a multilevel database with tuple-level labeling (B, Λ) over multilevel schema $(\mathcal{B}, \mathcal{L})$.

Example 3 The three views in Example 1 are labeled S, C, and U, respectively. A possible compilation of these views could assign labels to the primitive views in Example 2 as follows.

P_1, P_4, P_6, and P_7 are labeled U, P_3 and P_5 are labeled C, and P_2 is labeled S. Using these labeled primitive views to classify tuples in a given FLIGHT database, all Flights tuples to Iran will be labeled S, and all Item tuples that are bombs or explosives will be labeled C. All other tuples will be labeled U. □

A compilation is *well-founded* if, for every pair of primitive views $P, P' \in \mathcal{P}$, $P \simeq P'$ implies $\delta(P) = \delta(P')$. For example, the compilation in Example 3 is well-founded. Let B be a database over $\mathcal{B}$.

Theorem 1 *If a compilation is well-founded, then there is a unique multilevel database with tuple-level labeling (B, Λ) over $(\mathcal{B}, \mathcal{L})$, using the labeled primitive views in $(\mathcal{P}, \delta)$ as classification constraints.*

Proof Assume that there are two primitive views $P, P' \in \mathcal{P}$ on relation scheme $R[X]$ where $\delta(P) \neq \delta(P')$, such that there is a tuple t over attributes X satisfying both P and P'.

Assume that $P \not\simeq P'$. If body(P) does not unify with body(P'), then they must have different constants for the same argument position. Hence, there cannot be a tuple $t \in r$ that satisfies both P and P', a contradiction.

Let $\theta = \text{mgu}(\text{body}(P), \text{body}(P'))$. If constraint$(P)\theta \wedge$ constraint$(P')\theta$ is not satisfiable, then there cannot be a tuple $t \in r$ that satisfies both P and P', a contradiction.

Hence, $P \simeq P'$, and $\delta(P) = \delta(P')$, a contradiction. □

Theorem 1 states that a well-founded compilation guarantees that tuples in a database are uniquely labeled. In other words, every tuple is assigned exactly one label.

A compilation is *proper* if, for every view $V \in \mathcal{V}$, the label of V dominates the least upper bound of the labels of primitive views in the cover of V, namely, $\text{lub}(\{\delta(P) | P \in \text{cover}(V)\}) \preceq \Delta(V)$. For example, the compilation in Example 3 is proper. Let V be a view in $\mathcal{V}$ where $\Delta(V) = l$, $l' \in L$ be a level where $l \preceq l'$, and (B, Λ) be a multilevel database constructed using labeled primitive views in $(\mathcal{P}, \delta)$ as classification constraints.

Theorem 2 *If a compilation is proper, then the values of V in (B, Λ) at l and l' are identical.*

Proof For every primitive view $P \in \text{cover}(V)$, we have that $\delta(P) \preceq l$, and hence all tuples in B that satisfy P are labeled at or below l. Thus, $P_l(B, \Lambda) = P(B)$, and $V_l(B, \Lambda) = V(B)$. Since $V_l(B, \Lambda) \subseteq V_{l'}(B, \Lambda) \subseteq V(B)$, we have that $V_l(B, \Lambda) = V_{l'}(B, \Lambda)$. □

Theorem 2 states that a proper compilation guarantees that views in $\mathcal{V}$ are properly labeled. In other words, a secret label on a view guarantees that explicit data in the view are accessible to secret users through the view.

A compilation is *safe* if, for every view $V \in \mathcal{V}$, the label of V is dominated by the least upper bound of the labels of primitive views in the cover of V, namely, $\Delta(V) \preceq \text{lub}(\{\delta(P) | P \in \text{cover}(V)\})$. For example, the compilation in Example 3 is safe. Let V be a view in $\mathcal{V}$ where $\Delta(V) = l$, $l' \in L$ be a level where $l \not\preceq l'$, and (B, Λ) be a multilevel database constructed using labeled primitive views in $(\mathcal{P}, \delta)$ as classification constraints.

Theorem 3 *If a compilation is safe, then the value of V in (B, Λ) at l' is empty.*

Proof There is a primitive view $P \in \text{cover}(V)$ such that $\delta(P) \not\preceq l'$. Since every tuple that satisfies P is labeled $\delta(P)$, $P_{l'}(B, \Lambda)$ is empty. Hence, $V_{l'}(B, \Lambda)$ is empty. □

Theorem 3 states that a safe compilation guarantees that views in $\mathcal{V}$ are safely labeled. In other words, a secret label on a view guarantees that explicit data in the view are not accessible to unclassified users.

When every view in $\mathcal{V}$ contains a primitive view that is not overlapping with any other primitive views in the covers of views in $\mathcal{V}$, it is easy to verify that a well-founded, proper, and safe compilation exists.

5.3 Label Compilation Algorithm

Let $(\mathcal{B}, \mathcal{L}, \mathcal{V}, \Delta)$ be a multilevel schema with view-level labeling, and $\mathcal{P}$ be the set of primitive views in the covers of views in $\mathcal{V}$, namely, $\bigcup_{V \in \mathcal{V}} \text{cover}(V)$. Our label compilation algorithm is shown below. Initially, every primitive view in the cover of a view is assigned the label of the view. All primitive views in the same equivalence class under $\simeq^*$ are assigned the greatest lower bound of their initial labels.

Algorithm. Label Compilation
INPUT. A family of labeled views $(\mathcal{V}, \Delta)$
OUTPUT. A family of labeled primitive views $(\mathcal{P}, \delta)$
procedure compile
  $\mathcal{P} \leftarrow \bigcup_{V \in \mathcal{V}} \text{cover}(V)$;
  for $V \in \mathcal{V}, P \in \text{cover}(V)$ **do** $\delta(P) \leftarrow \Delta(V)$;
  for $V \in \mathcal{V}, P \in \text{cover}(V)$ **do** $\delta(P) \leftarrow \text{glb}(\{\delta(P') | P' \in P/\simeq^*\})$;
  for $V \in \mathcal{V}$ **do**
    if $\Delta(V) \not\preceq \text{lub}(\{\delta(P) | P \in \text{cover}(V)\})$ **then return** $(\emptyset, \emptyset)$;
  return $(\mathcal{P}, \delta)$.

Readers can verify that the algorithm applied to the labeled views in Example 1 gives rise to the labeled primitive views in Example 3.

Since all primitive views in the same equivalence class under $\simeq^*$ are labeled at the same level, compile$(\mathcal{V}, \Delta)$ is well-founded. Since every primitive view in the cover of a view is initially labeled the same as the view and could only be labeled lower through compilation, compile$(\mathcal{V}, \Delta)$ is proper. When the algorithm returns nonempty answers, compile$(\mathcal{V}, \Delta)$ is safe. Hence, the algorithm is sound. The theorem below tells us that the algorithm is also complete.

Theorem 4 *If compile$(\mathcal{V}, \Delta)$ returns empty answer, then there are no well-founded, proper, and safe compilations of $(\mathcal{V}, \Delta)$.*

Proof Suppose that compile$(\mathcal{V}, \Delta)$ returns empty answer, and there is a well-founded, proper, and safe compilation $(\mathcal{P}, \delta')$. First we show by induction that, for every pair of primitive views

$P, P' \in \mathcal{P}$, $P \simeq^* P'$ implies $\delta'(P) = \delta'(P')$. The claim holds for the base case when $P \simeq P'$, since $(\mathcal{P}, \delta')$ is well-founded. Assume that the claim holds for P_1 and P_k if $P_i \simeq P_{i+1}$ for $1 \leq i \leq k-1$, and $P_k \simeq P_{k+1}$. From the base case, $\delta'(P_k) = \delta'(P_{k+1})$. From the induction hypothesis, $\delta'(P_1) = \delta'(P_k)$. Hence $\delta'(P_1) = \delta'(P_{k+1})$.

Next we show that, throughout the algorithm, $\delta'(P) \preceq \delta(P)$ for every view $V \in \mathcal{V}$ and every primitive view $P \in \text{cover}(V)$. After the second step of the algorithm, $\delta(P) = \Delta(V)$. Since $(\mathcal{P}, \delta')$ is proper, $\delta'(P) \preceq \Delta(V)$. Hence $\delta'(P) \preceq \delta(P)$.

Consider primitive views in the set $P/\simeq^*$, all of which have the same δ'-label. After the second step of the algorithm, let l be the greatest lower bound of δ-labels of primitive views in $P/\simeq^*$. If $\delta'(P) \not\preceq l$, then there is a view $V' \in \mathcal{V}$ and a primitive view $P' \in \text{cover}(V')$ where $P \simeq^* P'$ such that $\delta'(P') \not\preceq \delta(P') = \Delta(V')$. Hence, the least upper bound of δ'-labels of primitive views in $\text{cover}(V')$ is not dominated by $\Delta(V')$, meaning that $(\mathcal{P}, \delta')$ is not proper, a contradiction.

Therefore, $\delta'(P) \preceq l$. Since $\delta(P)$ is assigned l in the third step of the algorithm, $\delta'(P) \preceq \delta(P)$ after the third step of the algorithm. Since $\text{compile}(\mathcal{V}, \Delta)$ returns empty answer, there is a view $V \in \mathcal{V}$ where $\Delta(V)$ is not dominated by the least upper bound of δ-labels (and hence δ'-labels) of primitive views in $\text{cover}(V)$, meaning that $(\mathcal{P}, \delta')$ is not safe, again a contradiction. □

Let us consider the complexity of the label compilation algorithm. First notice that the closure of a set of built-in literals is computable in time polynomial in the size of the set. Hence, the set of primitive views $\mathcal{P}$ is computable in time polynomial in the size of $\mathcal{V}$. Next notice that the unification of two base literals and the satisfiability of a set of built-in literals is computable in polynomial time. Hence, the set of equivalence classes $\mathcal{P}/\simeq^*$ is computable in time polynomial in the size of $\mathcal{V}$. Finally notice that both the greatest lower bound and the least upper bound of a set of levels are computable in time polynomial in the size of the set. Therefore, the label compilation algorithm runs in time polynomial in the size of the input.

6 Complexity of Optimal Compilation

6.1 Lowest Classification

Let $(\mathcal{B}, \mathcal{L}, \mathcal{V}, \Delta)$ be a multilevel schema with view-level labeling. If a compilation of labeled views in $(\mathcal{V}, \Delta)$ to labeled primitive views in $(\mathcal{P}, \delta)$ is well-founded, proper, and safe, then we can use the resulting labeled primitive views as classification constraints to label tuples consistently in databases.

When there are more than one well-founded, proper, and safe compilation, we would want to choose the one in which primitive views in $\mathcal{P}$ are labeled with the lowest possible levels, since doing so maximizes data sharing among users at different levels. However, the algorithm of Section 3.3 does not generate the lowest classification. In fact, the algorithm produces the highest classification. For every view $V \in \mathcal{V}$, every primitive view in $\text{cover}(V)$ is labeled by default at the highest possible level—$\Delta(V)$, unless it must be labeled lower because of safety requirements.

Example 4 It is easy to see that the compilation in Example 3 has the lowest compilation, since it is the only well-founded, proper, and safe compilation. If we did not have the labeled view V_3 in

Example 1, then we would only have primitive views P_1 through P_5 in Example 2, and hence the algorithm would produce a compilation in which P_1, P_3, P_4, and P_5 are labeled C, and P_2 is labeled S. This compilation does not have the lowest classification, since P_1 and P_4 could be safely labeled U. Notice that a compilation with the lowest classification is not always unique—P_3 and P_5 could be safely labeled U instead.[4] □

Let us consider a special case of the problem, where every relation scheme in $\mathcal{B}$ is unary, $\mathcal{L}$ is a total order with two levels $-$ and $\top$ such that $- \preceq \top$, every view in $\mathcal{V}$ is simple and unary and does not contain non-distinguished variables, and $\Delta(V) = \top$ for every view $V \in \mathcal{V}$. Let $\mathcal{P}$ be the set of primitive views in the covers of views in $\mathcal{V}$, namely, $\bigcup_{V \in \mathcal{V}} \text{cover}(V)$. The Lowest Classification problem can be stated as follows.

> Let $K \leq |\mathcal{P}|$ be a positive integer. Is there a subset of primitive views $\mathcal{P}' \subseteq \mathcal{P}$ with $|\mathcal{P}'| \leq K$, such that there is a well-founded, proper, and safe compilation of $(\mathcal{V}, \Delta)$ where the set of primitive views labeled $\top$ is $\mathcal{P}'$?

Theorem 5 *Lowest Classification is NP-complete.*

Proof The problem belongs to NP because a nondeterministic machine can randomly guess a solution and verify that it is well-formed, proper, and safe in polynomial time.

We reduce a known NP-complete problem, the Hitting Set problem [4, page 222], to the Lowest Classification problem. The Hitting Set problem states:

> Let S be a finite set, C be a collection of subsets of S, and $K \leq |S|$ be a positive integer. Is there a subset $S' \subseteq S$ with $|S'| \leq K$ such that S' contains at least one element from every subset in C?

It is easy to see that the Hitting Set problem is equivalent to the following modified Hitting Set problem:

> Let S be a finite set, C be a collection of subsets of S where $S = \bigcup_{c \in C} c$, and $K \leq |S|$ be a positive integer. Is there a subset $S' \subseteq S$ with $|S'| \leq K$ such that S' contains at least one element from every subset in C?

Our reduction of the modified Hitting Set problem is as follows. First, $\mathcal{B}$ contains an attribute A_s and a relation scheme $R_s[A_s]$ for every element $s \in S$. Second, $\mathcal{V}$ contains a view $V_c(x) : -R_{s_1}(x), \ldots, R_{s_k}(x)$ for every element $c \in C$ where $c = \{s_1, \ldots, s_k\}$. Notice that $\mathcal{P} = \{P_s(x) : -R_s(x)\}_{s \in S}$. Since primitive views overlap iff they are identical, all compilations are well-founded. Since views in $\mathcal{V}$ are labeled $\top$, all compilations are proper.

We need to show that the reduction establishes a one-to-one mapping between the two problems, in the sense that S'' is a solution of the modified Hitting Set problem if and only if it is a solution of the Lowest Classification problem.

[4] The notion of lowest classification does not take into account the sizes of primitive views. Otherwise, we would prefer to label P_1 lower since it could very likely contain many more tuples than P_3 and P_5 combined.

Suppose that S'' is a solution for the modified Hitting Set problem. We choose the compilation $(\mathcal{P}, \delta)$ where, for every primitive view $P_s \in \mathcal{P}$, $\delta(P_s) = \top$ if $s \in S''$, and $\delta(P_s) = -$ otherwise. Since S'' contains at least one element from every subset in C, every view in $\mathcal{V}$ contains at least one primitive view labeled $\top$. Hence, the compilation is safe.

Suppose that $\mathcal{P}''$ is a solution for the Lowest Classification problem. Let $S = \mathcal{P}$, and C be the set $\{\text{cover}(V) | V \in \mathcal{V}\}$. Naturally $S = \bigcup_{c \in C} c$. Since every view $V \in \mathcal{V}$ is labeled $\top$, there is at least one primitive view $P \in \text{cover}(V)$ where $\delta(P) = \top$. Hence $\mathcal{P}''$ contains at least one element from every subset in C. □

6.2 Minimal Upgrade

Let $(\mathcal{B}, \mathcal{L}, \mathcal{V}, \Delta)$ be a multilevel schema with view-level labeling. If compile$(\mathcal{V}, \Delta)$ returns empty, then there are no well-founded, proper, and safe compilations of labeled views in $(\mathcal{V}, \Delta)$.

A common cause of the problem is that some views are labeled inappropriately low, in which case we would like to ***upgrade*** these views such that the resulting schema can be safely compiled to tuple-level labeling. In other words, we want to find a new multilevel schema $(\mathcal{B}, \mathcal{L}, \mathcal{V}, \Delta')$ such that compile$(\mathcal{V}, \Delta')$ does not return empty and, for every view $V \in \mathcal{V}$, $\Delta(V) \preceq \Delta'(V)$. Naturally, we would like to minimize the number of views that must be upgraded, since doing so maximizes the number of users eligible to access data through these views.

Example 5 Consider the three labeled views below on the FLIGHT database. V_4 is labeled U, V_5 is labeled C, and V_6 is labeled C', where C and C' are not comparable.

$$\begin{aligned} V_4(x, y, z_1, z_2) &:- \text{Flights}(x, z_1, u_1, u_2), \text{Item}(y, z_2, v), \text{Payload}(x, y, w), w \leq 100 \\ V_5(x, y, z) &:- \text{Item}(y, z, v), \text{Payload}(x, y, w), w \geq 200 \\ V_6(x, y, z) &:- \text{Flights}(x, z, u_1, u_2), \text{Payload}(x, y, w), w \geq 220. \end{aligned}$$

Readers can verify that there are no well-founded, proper, and safe compilations. A minimal upgrade is to label Payload tuples U, label Item tuples C, and label Flights tuples C'. The result is that one view, namely, V_4, is upgraded to lub(C, C').[5] □

Let us consider a special case of the problem, where $\mathcal{L}$ is a total order with two levels $-$ and $\top$ such that $- \preceq \top$. Views in $\mathcal{V}$ are divided into two disjoint subsets: $\mathcal{V}_\top$ and $\mathcal{V}_\bot$, where $\Delta(V) = \top$ for every $V \in \mathcal{V}_\top$ and $\Delta(V) = -$ for every $V \in \mathcal{V}_\bot$. The Minimal Upgrade problem can be stated as follows.

> Let $K \leq |\mathcal{V}_\bot|$ be a positive integer. Is there a subset of views $\mathcal{V}' \subseteq \mathcal{V}_\bot$ with $|\mathcal{V}'| \leq K$, such that compile$(\mathcal{V}, \Delta')$ is not empty, where $\Delta'(V) = \top$ for every $V \in \mathcal{V}_\top \cup \mathcal{V}'$ and $\Delta'(V) = -$ for every $V \in \mathcal{V}_\bot - \mathcal{V}'$?

[5] Minimality in the number of upgraded views is different from minimality in the number of upgraded primitive views. For example, another upgrade is to label S those Payload tuples whose weight ≥ 200, and label Item and Flights tuples U. This upgrade is not minimal in the number of upgraded views (which is two), but it is minimal in the number of upgraded primitive views (which is one).

Let S be the set $(\bigcup_{V \in \mathcal{V}} \text{cover}(V))/\simeq^*$ and C be the set $\{\text{cover}(V)/\simeq^* \mid V \in \mathcal{V}\}$. Also let $C_\top$ and $C_\bot$ be $\{\text{cover}(V)/\simeq^* \mid V \in \mathcal{V}_\top\}$ and $\{\text{cover}(V)/\simeq^* \mid V \in \mathcal{V}_\bot\}$, respectively. The Minimal Upgrade problem can be equivalently stated as follows.

> Let $K \leq |C_\bot|$ be a positive integer. Is there a subset $C' \subseteq C_\bot$ with $|C'| \leq K$, such that there is a subset $S' \subseteq S$ in which $c \cap S' \neq \emptyset$ for every $c \in C_\top \cup C'$ and $c \cap S' = \emptyset$ for every $c \in C_\bot - C'$?

Theorem 6 *Minimal Upgrade is NP-complete.*

Proof The problem belongs to NP because a nondeterministic machine can randomly guess a solution and run the algorithm of Section 3.3 in polynomial time to determine whether a well-founded, proper, and safe compilation exists.

We transform a known NP-complete problem, the modified Hitting Set problem from Section 4, to the Minimal Upgrade problem. The modified Hitting Set problem states:

> Let S be a finite set, C be a collection of subsets of S where $S = \bigcup_{c \in C} c$, and $K \leq |S|$ be a positive integer. Is there a subset $S' \subseteq S$ with $|S'| \leq K$ such that S' contains at least one element from every subset in C?

Our transformation of the modified Hitting Set problem is as follows. Let S be a finite set, C be a collection of subsets of S where $S = \bigcup_{c \in C} c$, and $K \leq |S|$ be a positive integer. We need to construct a finite set S', a collection C' of subsets of S' where $S' = \bigcup_{c \in C'} c$, two disjoint subsets $C_\top, C_\bot \subseteq C'$ where $C' = C_\top \cup C_\bot$, and a positive integer $K' \leq |C_\bot|$, such that the modified Hitting Set problem (S, C, K) has a solution iff the Minimal Upgrade problem $(S', C', C_\top, C_\bot, K')$ has a solution.

Every element in S is an element in S', and every element in C is an element in C'. Let $C_\top = C$. For every element $s \in S$, S' contains a new element $s' \notin S$, and C' contains the element $\{s, s'\}$. Let $C_\bot = C' - C_\top$, and $K' = K$. It is easy to see that the construction can be accomplished in polynomial time.

Let S'' be a solution for (S, C, K). Also let $C'' = \{\{s, s'\} | s \in S''\}$. Obviously $C'' \subseteq C_\bot$ and $|C''| \leq K'$. Since $c \cap S'' \neq \emptyset$ for every $c \in C_\top \cup C''$ and $c \cap S'' = \emptyset$ for every $c \in C_\bot - C''$, C'' is a solution for $(S', C', C_\top, C_\bot, K')$.

Let C'' be a solution for $(S', C', C_\top, C_\bot, K')$. Also let $S'' = \{s | \{s, s'\} \in C''\}$. Obviously $S'' \subseteq S$ and $|S''| \leq K$. Since $c \cap S'' \neq \emptyset$ for every $c \in C$, S'' is a solution for (S, C, K). □

7 Related Work

View-based access control in relational databases was first introduced in IBM's System R [6], in which views expressed in SQL are the objects of authorization. It has been adopted by most commercial relational DBMSs. Because of its importance, view-based access control in MLS relational databases was recommended for long-term research by the Air Force Studies Board in 1983 [8].

One class of existing approaches to view-based access control is to restrict views to subsets of single relations. In particular, secure views in [12] are select queries where the selection condition involves comparisons of tuple labels to constants. ASD views [5] and *p*views [10] are project-select queries where the selection condition involves comparisons of attributes to constants. The safety of such views is easy to verify, and high assurance is achievable with a small amount of trusted code to interpret such views. However, the capability of content-based access control is limited.

Another class of existing approaches to view-based access control is to treat labeled views as classification constraints, which are used to compile labels on views to labels on data such as tuples or elements in tuples [2, 3]. Users can access data through arbitrarily complex views, such as with ordinary MLS relational databases. High assurance is achievable with a small amount of trusted code for label compilation. However, the safety of views is reduced to the safety of classification constraints. For easy verification of safety, classification constraints have also been restricted to project-select queries where the selection condition involves arithmetic expressions of attributes and constants [1]. Again, the capability of content-based access control is limited.

Neither class of approaches allows views involving joins, such as Bomb_Iran and Large_Explosive on the FLIGHT database. In comparison, our algorithm allows views that are project-select-join queries, which are significantly more expressive than single-relation views.

Another problem associated with the label compilation method of [1] is that data are often overclassified. When the same tuple appears in two views that are labeled differently, it is labeled at the least upper bound of the two view labels, making it inaccessible from both views. Hence, a confidential label on a view guarantees only that data contained in the view are not accessible to unclassified users, not that data contained in the view are accessible to confidential users. This is not acceptable because it might cause denial of service to users who are otherwise eligible to access data contained in a view. For example, if we have both the secret view Bomb_Iran and the confidential view Large_Explosive, then flights that carry large amounts of bombs to Iran must be labeled secret, making them inaccessible to confidential users who are otherwise entitled to access all flights that carry large amounts of explosives. In comparison, our algorithm guarantees that label compilation is proper and safe, meaning that data are neither overclassified nor underclassified.

Existing approaches do not distinguish between explicit and implicit data in a view, and treat the label on the view as applying to both explicit and implicit data. If a secret view involves the join of two relations, then it is commonly assumed that both the join and the two relations must be labeled secret, where in fact one relation can be labeled unclassified as long as the other relation is labeled secret. This confusion unnecessarily complicates the safety problem, leading to the need to overclassify data. For example, data in Payload could be safely labeled unclassified as long as data in Item are labeled confidential, even though Payload is part of the confidential view Large_Explosive. In comparison, our algorithm allows implicit data in a view to be labeled lower than explicit data in the view, so as to achieve a proper and safe label compilation.

8 Summary and Future Work

We have investigated view-based access control in MLS relational databases for a large class of views expressible as project-select-join queries. We have developed a polynomial-time label compi-

lation algorithm that transforms view-level labeling to tuple-level labeling. Compilation using our algorithm is well-founded, proper, and safe, in the sense that every tuple is assigned exactly one unique label, and data in a view are accessible precisely by users cleared at or above the level of the view. With the label compilation code in TCB, high assurance is achievable by enforcing access control not on views but on tuples.

We have also studied the complexity of two problems related to optimal label compilation: the problem of finding a well-founded, proper, and safe compilation with the lowest classification; and the problem of finding a well-founded, proper, and safe compilation with the minimal number of views upgraded. Both problems are shown to be NP-complete even for totally ordered security lattices of size two.

Several directions for future work are possible. First, label compilation may stop after generating labeled primitive views. As argued in [10], such views can be implemented with high assurance. This is especially useful for view-based discretionary access control, for it is not very practical to associate access control lists with tuples. Second, we would like to investigate the compilation of view-level labeling to tuple-level labeling where tuple labels are further constrained common integrity constraints such as referential integrity. Third, our algorithm can be generalized to the compilation of view-level labeling to element-level labeling, using the equivalence between tuple-level and element-level labeling established in [9]. Finally, we would like to consider views that are more expressive than project-select-join queries, such as views involving the difference operator or aggregation functions.

Acknowledgment

The author thanks Li Gong, Peter Neumann, Marvin Schaefer, and James O'Connor for useful comments on drafts of the paper.

References

[1] S. G. Akl and D. E. Denning. Checking classification constraints for consistency and completeness. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 196–201, 1987.

[2] B. G. Claybrook. Using views in a multilevel secure database management system. In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, pages 4–17, 1983.

[3] D. E. Denning, S. G. Akl, M. Heckman, T. F. Lunt, M. Morgenstern, P. G. Neumann, and R. R. Schell. Views for multilevel database security. *IEEE Transactions on Software Engineering*, 13(2):129–140, February 1987.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Co., 1979.

[5] C. Garvey and A. Wu. ASD views. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 85–95, 1988.

[6] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, September 1976.

[7] C. E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[8] Committee on Multilevel Data Management Security. Multilevel data management security. Technical report, Air Force Studies Board, National Research Council, National Academy Press, 1983.

[9] X. Qian and T. F. Lunt. Tuple-level vs. element-level classification. In B. M. Thuraisingham and C. E. Landwehr, editors, *Database Security, VI: Status and Prospects*, pages 301–315. North-Holland, 1993.

[10] M. Schaefer and G. Smith. Assured discretionary access control for trusted RDBMS. In *Proceedings of the Ninth IFIP WG 11.3 Working Conference on Database Security*, pages 275–289, 1995.

[11] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1. Computer Science Press, 1988.

[12] J. Wilson. Views as the security objects in a multilevel secure relational database management system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 70–84, 1988.