Foundations of Secure Deductive Databases*

Piero Bonatti† Sarit Kraus‡ V.S. Subrahmanian§

Note: This article appears in *IEEE Transaction on Knowledge and Data Engineering*, 7(3):406-422, 1995.

Abstract

In this paper, we develop a formal logical foundation for secure deductive databases. This logical foundation is based on an extended logic involving several modal operators. We develop two models of interaction between the user and the database called "yes-no" dialogs, and "yes-no-don't know" dialogs. Both dialog frameworks allow the database to lie to the user. We develop an algorithm for answering queries using yes-no dialogs and prove that secure query processing using yes-no dialogs is NP-complete. Consequently, the degree of computational intractability of query processing with yes-no dialogs is no worse than for ordinary databases. Furthermore, the algorithm is maximally cooperative to user in the sense that lying is resorted to only when absolutely necessary. For Horn databases, we show that secure query processing can be achieved in linear time – hence, this is no more intractable than the situation in ordinary databases. Finally, we identify necessary and sufficient conditions for the database to be able to preserve security. Similar results are also obtained for yes-no-don't know dialogs.

1 Introduction

The need to maintain security in databases arises naturally in many contexts. In multi-user databases, where different users are allowed to access different pieces of data, mechanisms must be in place where all users can access data that they are allowed to access, unless doing so would violate security in one way or the other.

In previous work, Bancilhon and Spyratos [2] raised four important questions that need to be addressed when developing a foundation for database security:

(Q1) What do we protect ?

(Q2) Against whom do we protect ?

(Q3) How do we protect ?

(Q4) What does "protect" mean ?

*This work was supported by the Army Research Office under Grant Number DAAL-03-92-G-0225, Israeli Ministry of Science and the Arts under grant 4210-1-93, as well as by NSF Grants IRI-91-09755 and IRI-9123460. We are grateful to Sushil Jajodia and Moshe Vardi for useful discussions. In particular, Sushil Jajodia drew our attention to [27] and Moshe Vardi to [31]. Address correspondence to V. S. Subrahmanian.

†Dip. di Informatica, Universita' di Pisa, Corso Italia 40, 56100 Pisa, Italy. E-Mail: bonatti@di.unipi.it. This work was done while the author was visiting the University of Maryland Institute for Advanced Computer Studies.

‡Department of Computer Science Bar-Ilan University Ramat Gan, Israel. E-Mail: sarit@bimacs.cs.biu.ac.il.

§Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, College Park, Maryland 20742. E-Mail: vs@cs.umd.edu

There are no "fixed" answers to the Bancilhon-Spyratos questions. A multiplicity of answers may be obtained depending upon

- the *representation* of the database (e.g. relational, hierarchical, Horn Clause logic, propositional logic, etc.),
- the *expressiveness* of the query language that the user is allowed to use,
- the *space of admissible responses* (e.g. yes, no, don't know, refuse to tell you), and
- the initial body of knowledge the user may possess

In this paper, we consider the problem of security in *deductive databases*. A deductive database is a finite set of formulas of propositional logic. Likewise, the query language we allow is also classical propositional logic. This is a reasonable restriction from the point of view of studying declarative semantics because when we consider deductive databases generated by a finite set of predicate symbols, and constant symbols, then each ground atom $p(c_1, \ldots, c_n)$ where the c_i's are constant symbols, can be regarded as a proposition. A non-ground atom can be regarded as a set of propositions by grounding out the variables in all possible ways. An existentially quantified statement of the form $(\exists x)q(x)$ may be regarded as a disjunction $q(c_1) \vee \ldots \vee q(c_k)$ where $c_1, \ldots c_k$ are all the constants in our language. Similarly, a universal quantification can be viewed as a conjunction. Hence, from the point of view of studying *declarative* semantics, this restriction leads to no loss of generality[1].

The principal *contributions* of this paper are the following:

1. *Model Theoretic Semantics:* We provide a declarative model-theoretic semantics for secure databases that describes:
   (a) *The Space of Secrets*: We provide an axiomatization of secrecy that describes both the explicit and the implicit secrets of the system. This is done by defining a simple modal logic of secrecy.
   (b) *The Interactions with the User over Time*: This is studied by extending the modal logic of secrecy with some aspects of branching time temporal logic and adding **Ask** and **Tell** modalities. We call this the *logic of interaction.* Intuitively, at any given point of time, the database reasons using only the modal logic of secrecy. The logic of interaction is not used by the database – rather it is used by *us* to study the behavior of the database, and to prove that our query answering strategies are sound, complete, and safe (i.e. they do not lead to violations of secrecy).

2. *Query Processing Algorithms:* We introduce the concept of "dialogs" of two kinds: "yes-no" dialogs where the database answers user queries with either a yes or a no, and "yes-no-don't

[1] However, when studying *operational* semantics, this assumption may lead to inefficiencies.

know" dialogs where the database may respond with either a "yes" or a "no" or a "don't know". In particular, we allow the database to lie when doing so is the only way to preserve security. We prove the following results:

(a) Using yes-no dialogs, there is a strategy for the database to preserve security IFF the user did not initially know a disjunction of secrets.

(b) For yes-no dialogs, the above result leads to a natural algorithm for answering queries to the database. The algorithm lies only if not doing so will lead to a violation of security.

3. *Complexity Issues:* We show that secure query answering in logic databases is NP-complete. This shows that as far as computational complexity is concerned, secure query processing is not in a "more intractable" complexity class than ordinary query processing. For Horn theories augmented with the Closed World Assumption, we show that secure query processing is achievable in linear time.

We present below, a couple of motivating examples that illustrate our view of security. We use these examples throughout the paper to help fix intuitions.

Example 1.1 Suppose, for example, our database contains the following information on who the shareholders of certain banks are. The US government has rules specifying that foreign banks (like BCCI) cannot own 50% or more stock in an American bank. Thus, BCCI's database may contain information of the following nature:

1) $status(bcci, foreign)$.

2) $status(bank\_of\_maryland, usa)$.

3) $status(bank\_of\_virginia, usa)$.

4) $status(first\_american\_bank, usa)$.

5) $owns(bcci, bank\_of\_maryland, 49)$.

6) $owns(bcci, bank\_of\_virginia, 49)$.

7) $owns(bcci, first\_american\_bank, 49)$.

8) $owns(bank\_of\_maryland, first\_american\_bank, 2)$.

9) $owns(bank\_of\_virginia, first\_american\_bank, 2)$.

10) $owns(citibank, first\_american\_bank, 47)$.

In addition, we assume that there are background arithmetical operations of the usual kind. In effect, BCCI's holding in First American Bank exceeds 50%. BCCI wishes to hide the fact that it owns a controlling interest in First American Bank. For this reason, it may protect the *conjunction* of statements (5) through (9). In other words, the formula $((5) \wedge (6) \wedge (7) \wedge (8) \wedge (9))$ is a secret.

If a user asks, as his first query, how much stock BCCI owns in First American, the database could answer 49% using statement (7) *without* violating secrecy. At this point, the formula $((5)\wedge(6)\wedge(8)\wedge(9))$ becomes secret even though this formula was not secret before.

On the other hand, if the user instead, had asked how much stock BCCI owns in Bank of Maryland, then the database could have answered with 49% using sentence (5), and now the formula $((6) \wedge (7) \wedge (8) \wedge (9))$ would become secret.

Thus, we see that what needs to be kept secret depends *not only on what the kernel secrets were initially, but also on the questions asked previously by the user and the answers provided by the system.* Thus, the dialog between user and the system has a direct impact on the set of secrets that must be protected at a given time. ■

Example 1.2 Let DB be a database containing information about the members of a Secret Service. Some of the individuals in DB are saboteurs and/or assassins, and hence, their job description needs to be kept secret. The database also includes secretaries and other people who are not involved in secret activities, and these peoples' jobs need not be kept secret. Furthermore, information on people who are not involved in secret activities is useful to users, and the DB should answer queries about such individuals.

Let Q be the query "is X an assassin?", where X is one of the individuals in the database. How should the database respond if X is indeed an assassin? Of course, the database should not answer "yes".

If the answer was "I refuse to answer" then the user might reason as follows:

> *If X was not an assassin, then DB would not need to refuse to answer, for it could simply answer "no". Therefore X must be an assassin.*

If the answer was "I don't know" then the user might think:

> *This database contains complete data about all the members of the Secret Service, so the database must be lying. If X was not an assassin, then the database would not need to lie. Hence X must be an assassin.*

Therefore, the only way to preserve security in this instance, and to provide information whenever it doesn't violate security, is for the database to answer "no", which shows that sometimes lying is the only way to protect a secret. ■

The strategy of lying, has been suggested, in passing, in [26, 31], and will be discussed later. More importantly, if the initial beliefs of the user are wrong, the system should take that into account. This is not considered in [31, 26]. Our logic encompasses this possibility because secrets are defined in terms of the user's *beliefs* rather than what is true. The user's beliefs may or may not be correct.

Another approach to preserving security in such situations is to withhold answers in order to prevent the user from obtaining secrets. For example, in the Secret Service DB example above, the DB may refuse to answer any query related to the job description [11, chap. 6] or arbitrarily refuse to answer some of these queries (in addition to those queries that may directly reveal secrets) [26]. Lying may sometimes be a dangerous approach since the user may act upon the wrong information. However, not providing information that does not directly violate security may also be dangerous. For example, in any situation where the user critically needs an answer to a query Q, if the database refuses to answer, then the user is forced to make an (ill-informed) decision and s/he may well make the wrong choice. The following example illustrates this phenomenon.

Example 1.3 (Radiation Example) Consider a civilian research station in the Antarctic that is experiencing a terrible storm in the winter that makes venturing out extremely hazardous. To add to their troubles, a submarine disintegrates in the ice near their station. They are aware of the fact that some submarines carry nuclear payloads, and they access the Navy database to find out if this is the case. The Navy database refuses to answer even though providing the true answer ("there are no nuclear weapons on board") will not violate security. The researchers have to decide whether to leave the station or not. Should the submarine be carrying a nuclear payload and if they decide to stay, they are at risk. Should the submarine not be carrying a nuclear payload, and they decide to risk a dangerous crossing in the storm over treacherous ice floes, then they are unnecessarily placing themselves at risk. ■

The above example demonstrates that "refusing to answer" may not always be a good option especially when providing the correct answer does not violate security – in other words, refusing to answer may be an uncooperative strategy. Thus, the policy to be chosen in order to preserve security depends on the information in the database and its intended usage. We provide techniques that are appropriate when *maximal cooperation* is desired, i.e. the set of correct answers should be maximized. These are appropriate under some circumstances, while other techniques (such as refusing to answer) are appropriate under others.

2 Preliminaries

Before proceeding to discuss the precise details of the logics of secrecy and interaction, we must discuss exactly how the database is structured, how the user's beliefs are structured, and how "smart" the user is.

The database consists of two components:

(1) the *information* component, **DB**, which consists of a set of formulas of propositional logic. Intuitively, this is the "real" data in the database.

(2) the *user* component which is intended for internal database use. Basically, for each user u, there are three parts:

a. a set $\mathbf{KER\_SEC}(u)$ of statements of the form "a must be kept secret from user u"

b. a set $\mathbf{Init}(u)$ of wffs reflecting what the database thinks are the *initial beliefs* of the user u.

c. a *history*, $\mathbf{Hist}(u)$ of information provided by the database to the user u in previous interactions. $\mathbf{Hist}(u)$ is a set of wffs of propositional logic.

In this paper we assume that both components are based on a monotonic logic.[2] No non-monotonic inferences will be used and, in particular, no closed world assumption, which says that we can conclude $\neg c$ from DB iff $DB \not\models c$, will be made (unless explicitly stated otherwise).

In its interactions with a *specific* user, u_0, the database will only use its information component and the sets $\mathbf{KER\_SEC}(u_0), \mathbf{Init}(u_0), \mathbf{Hist}(u_0)$ to determine the course of action it should take in responding to the query. In our framework, we assume that the user is a *perfect reasoning agent*, i.e. if the user believes each formula in a set T of wffs and a is a consequence of T in propositional logic, then the user will be smart enough to conclude a. The assumption that the user is a perfect reasoner is the most cautious approach to take. We further assume that the reasoner is "sound", i.e. s/he never draws unwarranted inferences. Throughout the paper, we assume that $\mathbf{L}$ is an underlying propositional language in which the database is expressed. In the next section we define the logic $\mathbf{L_S}$. $\mathbf{L_S}$ is an extension of logic $\mathbf{L}$ which can be used to reason about the DB, about secrets that should be kept from a user, and about the user's beliefs. Formally, we assume that there is a set $\mathbf{L}_0$ of propositional variables and $\mathbf{L}$ is defined as follows:

1. Every propositional symbol $p \in \mathbf{L}_0$ is a formula of $\mathbf{L}$.
2. If a and b are formulas of $\mathbf{L}$, then $\neg a$, $(a \vee b)$, $(a \wedge b)$, $(a \rightarrow b)$ are formulas of $\mathbf{L}$.

3 The Logic of Secrecy $\mathbf{L_S}$

In this section, we extend the propositional logic $\mathbf{L}$ by appending three new modal connectives $\mathbf{S}$, $\mathbf{All}$ and $\mathbf{B_U}$. Intuitively, $\mathbf{S}a$ is intended to be read as "a is secret". $\mathbf{All}\,a$ is intended to be read as "The user is *allowed* to believe a at some point of time". Finally, as the database must reason not only about what is true, but also what the user *believes* (whether true or not), we need a modality, $\mathbf{B_U}$ standing for "the user believes"[19]. Thus, $\mathbf{B_U}\,a$ is read as "The user believes a". The following definition specifies the syntax of formulas of the logic $\mathbf{L_S}$.

Definition 3.1 (Wffs) We define the *well-formed formulas* (wffs) of $\mathbf{L_S}$ inductively as the smallest set satisfying the conditions:

[2]In monotonic logics, if a conclusion is warranted on the basis of certain premises, no additional premises will ever invalidate the conclusion.

1. Every formula of our propositional logical language $\mathbf{L}$ is a formula of $\mathbf{L_S}$.
2. If a is a wff of $\mathbf{L_S}$, then $\mathbf{S}a$, $\mathbf{All}\, a$ and $\mathbf{B_U}\, a$ are wffs of $\mathbf{L_S}$.
3. If a, b are wffs of $\mathbf{L_S}$, then $(a \wedge b)$, $(a \vee b)$, $\neg a$, $(a \rightarrow b)$ are wffs of $\mathbf{L_S}$. (Other connectives can be defined in terms of these connectives in the usual way.)

We assume that all the classical axioms and inference rules of propositional logic (cf. [25]) also hold in the extended logic. Furthermore, we provide new logical axioms that our secrecy and allowed-ness modalities are expected to satisfy.

Thus, we are now in a position to formally state what constitutes a secret.

Definition 3.2 (Secrets) A *secret* is a (true) fact, that the user is not allowed to believe. Formally,

$$\mathbf{S}a \leftrightarrow_{def} (a \wedge \neg \mathbf{B_U}\, a \wedge \neg \mathbf{All}\, a) \tag{1}$$

Intuitively, a is a secret iff a is true and the user doesn't believe a and the database is not allowed to tell the user a.

We use $\mathbf{KER\_SEC}(u)$ to denote the set of secrets that should be kept from a given user u. Thus, $\mathbf{KER\_SEC}(u) \subseteq \{Sa \mid \text{a is a propositional logic formula}\}$. $\mathbf{S}a \in \mathbf{KER\_SEC}(u)$ means that a should be kept secret from user u. We use the notation $\mathbf{KER\_SEC}^*(u)$ to denote the set $\{a \mid Sa \in \mathbf{KER\_SEC}(u)\}$.

Note that it is possible that the user believes a, and $\mathbf{S}a$ is in $\mathbf{KER\_SEC}(u)$. In this situation, it is impossible to preserve security (and in fact, the set of formulas $\{\mathbf{B_U}\, a, \mathbf{S}a\}$ is inconsistent w.r.t. the model theory we develop later in the paper).

In addition to the axiom defining the modality $\mathbf{S}$ above, the logic $\mathbf{L_S}$ also has axioms, given below, defining the properties of "the user believes" (i.e. the modality $\mathbf{B_U}$) and "the user is allowed to believe" (i.e. the modality $\mathbf{All}$). These axioms embody intuitive security-preserving properties that the above modalities should satisfy. In particular, these axioms assume that the user is a perfect reasoner, i.e. s/he can draw all logically valid conclusions from information that s/he believes. Assuming that the user is a perfect reasoner means that the database should always assume that s/he will be able to infer a secret if such a secret is logically inferrable from the information provided by the database.

Definition 3.3 (Perfect Reasoner Axioms)

$$\mathbf{B_U}\,(a), \quad \text{for all propositional tautologies } \; a \tag{2}$$

$$\mathbf{B_U}\, a \wedge \mathbf{B_U}\,(a \rightarrow b) \rightarrow \mathbf{B_U}\, b \tag{3}$$

$$\mathbf{All}\, a \wedge \mathbf{B_U}\,(a \rightarrow b) \rightarrow \mathbf{All}\, b, \quad \text{for all propositional wffs } \; a, b \tag{4}$$

The last axiom says that if the user believes $(a \rightarrow b)$, and is allowed to believe a, then s/he is implicitly allowed to believe b. Finally, we introduce a knowledge modality $\mathbf{K_U}$ in terms of the modalities already introduced. The standard definition of knowledge as "true belief" will be used.

Definition 3.4 $\mathbf{K_U}\, a \leftrightarrow_{def} a \wedge \mathbf{B_U}\, a$.

3.1 Properties of $\mathbf{L_S}$

In this section, we prove some propositions that illustrate how the operator $\mathbf{S}$ interacts with the standard logical connectives. These propositions show how implicit secrets can be derived from explicit secrets.

All the propositions of this section follow from the inference rules of propositional logic, the definition of secrecy (Axiom 1), and the perfect reasoner axioms. The following two statements can be easily derived from (2) and (3).

$$\mathbf{B_U}\,(a \leftrightarrow b) \rightarrow \mathbf{B_U}\,(a \rightarrow b) \tag{5}$$

$$\mathbf{B_U}\,(a \leftrightarrow b) \rightarrow \mathbf{B_U}\,(b \rightarrow a) \tag{6}$$

In the rest of this section, let a and b be arbitrary (modality free) propositional formulas. The following proposition asserts that if the user *knows* that a and b are equivalent w.r.t. the underlying database, then a is a secret iff b is a secret.

Proposition 3.1 [3] $\mathbf{K_U}\,(a \leftrightarrow b) \rightarrow (\mathbf{S}a \leftrightarrow \mathbf{S}b)$

A subtle point to note in (3.1) above is that we deal with *knowledge* rather than belief. The following example demonstrates the intuition behind (3.1).

Example 3.1 Let L be the law stating that no foreign bank can own more than 50% of an American bank, and let b denote the proposition "BCCI violates law L". Let a denote the proposition "BCCI holds more than 50% of an American bank". Then we assume that any competent user knows that the equivalence $a \leftrightarrow b$ follows from Example 1.1 and arithmetical axioms.

Now let's come back to Example 1.1. BCCI violates L and wants to keep that secret. If BCCI allowed a to be known, then, through the equivalence $a \leftrightarrow b$, the user could deduce b, and thus violate the secret. Therefore, if b is a secret, a should be kept secret, too.

Conversely, suppose that the goal of BCCI is to keep a secret (perhaps because of some economic strategy). If BCCI allowed b to be known, then, through the equivalence $a \leftrightarrow b$, the user could deduce a. Therefore, if a is a secret, b should also be kept secret. ■

[3] Proofs of results are can be found in [8] unless those proofs are essential to the ongoing discussion in the paper.

The more general question arising out of the previous proposition, is:

Question 3.1 *Shouldn't we then consider* $\mathbf{B_U}(a \leftrightarrow b) \rightarrow (\mathbf{S}a \leftrightarrow \mathbf{S}b)$ *to be true even if the database does not consider a and b to be logically equivalent ?*

The answer is "no". Suppose the user mistakenly believes $(a \leftrightarrow b)$, i.e. $\mathbf{B_U}(a \leftrightarrow b)$ is true, but $(a \leftrightarrow b)$ does not follow from the database. Then what should be true (and is true) is the formula: $\mathbf{B_U}(\mathbf{S}a \leftrightarrow \mathbf{S}b)$ which says that "the user believes that a is secret iff b is secret". Of course, this is not the same as saying that "a is secret iff b is secret", because this belief of the user may be false.

Question 3.2 *How about the converse ? Should the formula* $(\mathbf{S}a \leftrightarrow \mathbf{S}b) \rightarrow \mathbf{B_U}(a \leftrightarrow b)$ *be true ?*

The answer, once again, is no. The database might say that "a is a secret iff b is a secret" without the user believing that a and b are equivalent. For instance, the sentence "$salary(john, 50000)$ is secret" iff "$salary(bill, 57000)$ is secret" may be a constraint that the database wishes to enforce, but there is no need for the user to believe that the statements $salary(john, 50000)$ and $salary(bill, 57000)$ are equivalent.

The following result says that if the user believes $(a \rightarrow b)$ and b is a secret, then a should be kept secret if a is true. In particular, if a is not true, then there may be no need to keep a secret.

Proposition 3.2 $\mathbf{B_U}(a \rightarrow b) \wedge \mathbf{S}b \rightarrow (a \rightarrow \mathbf{S}a)$

The following question comes to mind immediately:

Question 3.3 *Should the relationship* $\mathbf{B_U}(a \rightarrow b) \wedge \mathbf{S}b \rightarrow \mathbf{S}a$ *be true ?*

Our answer to this is a resounding no. The reason is illustrated by the following example.

Example 3.2 Assume the database is the same as in example 1.1. Let $b =$ "BCCI violates law L" where law L is as defined in Example 3.1, and suppose b is a secret. Let $a =$ "BCCI holds 85% of an American bank". We assume that any competent user knows that the implication $a \rightarrow b$ holds.
Now, suppose that a user asks whether a holds. If BCCI answered "yes", then the user could infer the secret b from $a \rightarrow b$. However, BCCI can honestly answer "no", because a is false. Therefore a need not be a secret.
Of course, if a had been true, then the system should have lied, because the honest answer ("yes") would have entailed the secret formula b. ∎

An interesting consequence of the preceding proposition is that if a is a secret, and $(a \wedge b)$ holds, then $(a \wedge b)$ should be a secret too.

Proposition 3.3 $\mathbf{S}a \to ((a \wedge b) \to \mathbf{S}(a \wedge b))$

Proposition 3.3 implies the corollary below, which says that if a is a secret and b is true, then $(a \wedge b)$ is a secret. Intuitively, this is true because if $(a \wedge b)$ were not a secret, then a "yes" answer would tell the user that a is true, thus violating the secrecy of a.

Corollary 3.1 $(\mathbf{S}a) \wedge b \to \mathbf{S}(a \wedge b)$

The following example demonstrates the above property.

Example 3.3 Assume the database is the same as in Example 1.1. Let $a =$ "BCCI holds more than 50% of First American Bank", and let $b =$ "BCCI holds less than 50% of Bank of Maryland" (note that both a and b are true). Assume that a is a secret.
If the user asks whether $(a \wedge b)$ holds, then BCCI should lie, because the honest answer ("yes") implies the secret a.
On the other hand, if the user asks if $(a \wedge \neg b)$ holds, then BCCI can honestly answer "no" without violating any secret; knowing that $(a \wedge \neg b)$ is false does not allow the user to infer a. ■

The more general question arising out of the previous proposition is:

Question 3.4 *If $(a \wedge b)$ is a secret, then should both a and b be secrets?*

The answer, as can be seen from Example 3.3, is an unequivocal no. The reason is that we wish the database to be as cooperative as possible in answering queries posed by the user – the only requirement is that secrecy cannot be violated. The statement $\mathbf{S}(a \wedge b)$ says that the fact that *both* a and b are true is to be kept secret. Of course, if the user first asks a, then we could answer "yes" to this query, because the user cannot conclude, based on this "yes" answer alone, that $(a \wedge b)$ is true. However, if s/he later asks about b, then we are forced to keep b secret in some way (we do so by resorting to lying). Hence, given that $(a \wedge b)$ is a secret, there is, in general, no need that both a, b be secrets.

On the other hand, if a is a secret and b is a secret, then clearly $(a \wedge b)$ should be a secret.

Proposition 3.4 $\mathbf{S}a \wedge \mathbf{S}b \to \mathbf{S}(a \wedge b)$

Conversely, if $(a \vee b)$ is a secret, then either a is a secret, or b is a secret, but not necessarily both.

Proposition 3.5 $\mathbf{S}(a \vee b) \to (\mathbf{S}a \vee \mathbf{S}b)$

Example 3.4 Consider Example 1.2, and assume that X is an individual in the database, whose job is either "saboteur" or "assassin". Let $a =$ "X is a saboteur" and $b =$ "X is an assassin". Assume that the (true) formula $(a \vee b)$ is a secret.

Now, consider the only two possible cases. First suppose X is a saboteur (i.e. a holds). If a user asks whether a holds, then the database cannot answer honestly (i.e. say "yes"), because the user can then deduce the secret proposition $(a \vee b)$ from a. Therefore, in this case a should be kept secret.
On the other hand, suppose X is an assassin (i.e. b holds). In this case, by a similar argument, b should be kept secret.
As a consequence, either a or b must be kept secret. ■

The following question arises when the previous proposition is considered.

Question 3.5 *Given that either a is a secret or that b is a secret should $(a \vee b)$ be a secret?*

The answer to this question is "no". Intuitively, suppose a is a secret and the user asks us a query about $(a \vee b)$. Answering "yes" to the query will not violate the secrecy of a; it only tells the user that $(a \vee b)$ is true, not that a is true.

3.2 Meta-security

Suppose a is a secret, i.e. $\mathbf{S}a$ is true. Then it may be important that not only should a be kept secret, but the fact that a is a secret should itself be kept secret.

Example 3.5 The "Secret Service" example 1.2 can be given the following explanation: If the database refuses to respond, then the user concludes that the fact "X is an assassin" is a secret, and from this fact s/he can derive that "X is an assassin" must be true. ■

The informal concept of meta-security introduced above is made formal by the following definition.

Definition 3.5 (Meta-Security) A secret a is *meta-secure* iff $\mathbf{S}a$ is a secret.

In our logic, all secrets are also meta-secure, i.e. $\mathbf{S}a$ implies $\mathbf{SS}a$. Furthermore, the converse is true as well. This is encapsulated in the following result.

Proposition 3.6 $\mathbf{S}a \leftrightarrow \mathbf{SS}a$.

Hence, in our framework, meta-security is enforced iff no secrets are violated. Any strategy that preserves secrecy will simultaneously preserve this form of meta-security.[4]

In this section, we have studied the axioms for the logic of secrecy. Intuitively, the logic of secrecy is the logic used by the database to determine, *at a given point in time*, what formulas should be kept secret from the user. This determination is based on the *history* of information provided by the system to the user. In the next section, we introduce the concept of a dialogue and describe the *interaction* between the user and the system over an extended period of time. This includes the history (previous interactions), the current situation, and all possible future interactions.

[4]The Kripke-style possible worlds semantics that is given in later sections is not applicable to meta-security as we don't allow nested beliefs. That topic will be studied in a future work.

4 Dialogs: The Logic of Interaction

We consider an *event* to consist of the following: the user *asks* a question (in propositional logic) and the database provides a *response*.

We assume a discrete model of time where an event occurs at each time instant. In other words, at time 1, the user asks a query and the database provides a response, at time 2, the user asks another query and the database provides another response, and so on. In other words, we assume that the time taken for "internal" processing is not relevant, and that the user is patient enough to wait till a response is obtained.

The aim of this section is to study the possible sequence of events that may occur when the user and the database engage in a dialog. How exactly should the database respond so as to guarantee that security is not violated ?

4.1 Yes-No Dialogs

The logic of interaction takes the logic of secrecy described earlier, and adds *three* new modal operators to it:

- The modal operator **Ask**: Intuitively, $\mathbf{Ask}\, a$ is true (at time t) if the user asks the query a of the database (at time t).
- The modal operator **Tell**: Intuitively, $\mathbf{Tell}\, a$ is true (at time t) if the database tells the user formula a (at time t).
- The modal operator $\bigcirc$: Intuitively, $\bigcirc a$ is true at time t if a is true at time $(t+1)$. $\bigcirc$ is also known as the "next" time modality.

Below, we define the concept of a "dialog" which involves a sequence of question-answer pairs, reflecting the interaction between the user and the database. This definition makes use of the **Ask** and **Tell** modalities introduced above.

Definition 4.1 (Dialogs) A dialog is a (possibly infinite) sequence of events $(A_1, T_1), (A_2, T_2), \ldots, (A_i, T_i), \ldots$ where A_i is of the form $\mathbf{Ask}\, a_i$ for some formula a_i of propositional logic, and T_i is of the form $\mathbf{Tell}\, b_i$ for some formula b_i of propositional logic.

According to the above definition of dialog, when the database is asked a query like "Is Bush the President of the USA ?", the database may respond with "The moon is made of blue cheese." Clearly, such irrelevant responses serve no useful purpose. The following definition specifies that if asked the query a, the database must respond with a (or "yes") or $\neg a$ (corresponding to "no"). Dialogs that involve such responses are called "yes-no" dialogs, and are formalized below.

Definition 4.2 (Yes-No Dialogs) We say that a dialog is a *yes-no dialog* (yn-dialog, for short) iff for every formula a_i asked by the user, the database responds by answering either $\mathbf{Tell}\, a_i$ or $\mathbf{Tell}\, \neg a_i$. In other words, a dialog $(A_1, T_1), (A_2, T_2), \ldots, (A_i, T_i), \ldots$, where each $A_i = \mathbf{Ask}\, a_i$, is a yn-dialog iff, for each $i \geq 1$, T_i is either $\mathbf{Tell}\, a_i$ or $\mathbf{Tell}\, \neg a_i$.

The definition of yn-dialogs does not say anything about the correctness of the answer provided by the database. For this purpose, we define a concept of honesty.

Definition 4.3 (Honest Yes-No Dialogs) A yn-dialog $(A_1, T_1), (A_2, T_2), \ldots, (A_i, T_i), \ldots$ where $A_i = \mathbf{Ask}\, a_i$ is said to be *completely honest* iff for all $i \geq 1$ it is the case that:

- if $T_i = \mathbf{Tell}\, a_i$, then a_i is a logical consequence of DB and
- if $T_i = \mathbf{Tell}\, \neg a_i$, then $\neg a_i$ is a logical consequence of DB.

In yes-no dialogs, in response to any user query $\mathbf{Ask}\, a$, the database must respond with either $\mathbf{Tell}\, a$ or $\mathbf{Tell}\, \neg a$, i.e. there is no scope for the database to admit ignorance. However, for logic databases (and even for relational databases), this situation is unreasonable. For example, consider the logic database:

If x is a squirrel, then x is either red or grey	*Clyde is a squirrel.*
Denny is a squirrel.	*Clyde is grey.*

It is impossible to answer either "yes" or "no" to the query "Is Denny red in color ?" Denny may be red; he may be grey, but we don't know which. A similar situation occurs in relational databases with null values. One may, in an employee database, have a null value which says that Lisa's age lies between 35 and 40. The database simply cannot answer the query "Is Lisa's age 37 ?" with a definitive yes or a definite no. Hence, it may be appropriate to consider situations where the database is allowed to admit ignorance – these are formalized in the next subsection on "yes-no-don't know" dialogs.

Before proceeding to "yes-no-don't know" dialogs, we show, via the following theorem, that complete honesty leads to security violations as far as yn-dialogs are concerned.

Theorem 4.1 (Honesty may not be the Best Policy) Suppose DB is any logic database and there exists a user u such that $\mathbf{KER\_SEC}(u) \neq \emptyset$. Then: a completely honest, yn-dialog based query answering strategy cannot preserve security.

Proof. Suppose $a \in \mathbf{KER\_SEC}^*(u)$ and $A_1 = a$. As a is a secret, a must be true w.r.t. DB. Then, as the query answering strategy is completely honest and involves only yn-dialogs, T_1 must be the correct answer "yes" to the query a; this violates the security of a. ■

The above theorem says that if the database is to be maximally cooperative to the user (in the sense that it always answers honestly when such answers do not violate security), then it is impossible to preserve security when only honest yn-dialogs are permitted.

4.2 Security in Incomplete Databases

It may turn out that a given database does not have complete information about the domain it describes. In particular, the database may be incompletely specified and/or contain null values. Given a query Q, an incomplete database DB may either entail Q (i.e. correct answer is "yes") or may entail $\neg Q$ (i.e. correct answer is "no") or may entail neither Q nor $\neg Q$ (i.e. i.e. correct answer is "don't know"). It may turn out that the correct answer to a query is "yes", but providing this answer will violate security. Unlike the case of complete databases, where the database lies and answers "no," an incomplete database has the option of choosing to lie by saying either "no" or "don't know." Thus, incomplete databases have greater flexibility in answering queries, and hence, in order to capture this enhanced flexibility, we provide a number of new definitions and results below.

We now define the concept of a dialogue where the database may respond with either a "yes", a "no", or a "don't know". In order for the user to reason about the beliefs/ignorance of the database, the language for conducting yes-no dialogs needs to be augmented with a new modality, $\mathbf{B_D}$. Intuitively, $\mathbf{B_D}\, a$ means that the database believes a.

Definition 4.4 (Yes-No-Don't Know Dialog) We say that a dialog is a *yes-no-don't know dialog* (ynd-dialog, for short) iff for every formula a_i asked by the user, the database responds with either $\mathbf{Tell}\, a_i$ or $\mathbf{Tell}\, \neg a_i$ or $\mathbf{Tell}\, (\neg \mathbf{B_D}\, a_i \wedge \neg \mathbf{B_D}\, \neg a_i)$.
In other words, a dialog $(A_1, T_1), (A_2, T_2), \ldots, (A_i, T_i), \ldots$, where each $A_i = \mathbf{Ask}\, a_i$, is a ynd-dialog iff, for each $i \geq 1$, T_i is either $\mathbf{Tell}\, a_i$ or $\mathbf{Tell}\, \neg a_i$ or $\mathbf{Tell}\, (\neg \mathbf{B_D}\, a_i \wedge \neg \mathbf{B_D}\, \neg a_i)$.

Honest yes-no-don't know dialogs are analogous to honest yes-no dialogs.

Definition 4.5 (Honest YND-Dialogs) A ynd-dialog $(A_1, T_1), (A_2, T_2), \ldots, (A_i, T_i), \ldots$ where $A_i = \mathbf{Ask}\, a_i$ is said to be *completely honest* iff for all $i \geq 1$ it is the case that:

- if $T_i = \mathbf{Tell}\, a_i$, then a_i is a logical consequence of DB.
- if $T_i = \mathbf{Tell}\, \neg a_i$, then $\neg a_i$ is a logical consequence of DB.
- if $T_i = (\neg \mathbf{B_D}\, a_i \wedge \neg \mathbf{B_D}\, \neg a_i)$, then neither a_i nor $\neg a_i$ is a logical consequence of DB.

The following theorem shows that a completely honest query answering strategy can lead to security violations even if ynd-dialogs are allowed.

Theorem 4.2 *Suppose DB is any logic database and there exists a user u such that* $\mathbf{KER\_SEC}(u) \neq \emptyset$. *Then: a completely honest, ynd-dialog based query answering strategy cannot preserve security.*

It may be suggested that the database be occasionally dishonest, i.e. on random occasions, it may say "I don't know" even if it does indeed know the answer to the query[26]. However, as shown in Example 1.3 (the Radiation Example), this approach may sometimes lead to undesirable consequences.

4.3 Trustful Users

When discussing the interaction between a user and the database system, we need to address the issue: Does the user believe the answers s/he is getting from the database ? This question is even more crucial when the DB is lying from time to time. In the algorithm that we present, if the size of the set of secrets is relatively small in comparison to the size of the DB, the DB will lie only rarely. We all encounter people who lie from time to time. However, if our estimation is that a person lies only rarely and if it is costly to verify his/her claims, we usually believe what we are told and act on the basis of what s/he says (even though there is some probability that s/he lies or is wrong). We make similar assumption about our user; we assume that the user always believes the responses given by the database. We call users of this kind, *trustful* users, and the axiom given below formalizes this "trustfulness" property.

Definition 4.6 (Axiom of Trustfulness) The *axiom of trustfulness* is: $\mathbf{Tell}\, a \rightarrow \mathbf{B_U}\, a$

The axiom of trustfulness simply says that if the database tells the user formula a, then the user immediately believes formula a.

As a matter of fact, the user is trustful only if s/he does not detect a contradiction. More specifically, when an answer of the database is inconsistent with the user's beliefs, the perfect reasoner axioms make the user believe all the formulas of the language; as a consequence, every propositional formula is believed by the user and hence, security is violated. Intuitively, if the set of answers provided by the database is inconsistent, then it is clear that the database is lying, and the user may try to figure out which are the incorrect answers. The following axioms state that the database must always respond to a user query.

Definition 4.7 (Response Axiom) The *response axiom* for yn-dialogs is: $\mathbf{Ask}\, a \rightarrow \mathbf{Tell}\, a \vee \mathbf{Tell}\, \neg a$. The *response axiom* for ynd-dialogs is: $\mathbf{Ask}\, a \rightarrow \mathbf{Tell}\, a \vee \mathbf{Tell}\, \neg a \vee \mathbf{Tell}\, (\neg \mathbf{B_D}\, a \wedge \neg \mathbf{B_D}\, \neg a)$.

The following section formalizes how beliefs held by the user are influenced by the database's responses to previous queries.

4.4 Persistence Axioms

The information part of our database consists of wffs of the underlying propositional language $\mathbf{L}$. This is the "objective" part of the database, which is denoted by DB, and constitutes the domain knowledge that the user is trying to elicit from the database. In this paper, we assume that the user never forgets any objective (i.e. non-modal) information that s/he has learned[5]. In other words, if the

[5] Our results apply even if the user forgets information. However, for security purposes, it is best that the user be considered as intelligent as possible, and hence, the assumption that the user never forgets information means that the database must take all its previous responses into account when answering a query.

user believes a non-modal wff a at time t, then the user will continue to believe a at all times $t' > t$. This is a kind of monotonicity property – the set of non-modal beliefs of the user keeps growing as s/he interacts with the system over an extended period of time. This property can be expressed as the following axiom:

$$\mathbf{B_U}\, a \rightarrow \bigcirc \mathbf{B_U}\, a$$

where a is a formula of $\mathbf{L}$, the underlying propositional language. What this says is that if the user believes a at time t, then s/he continues to believe a at time $(t+1)$. Furthermore, the set of objective formulas that the agent does *not* believe decreases as time passes. At time t, suppose the agent doesn't believe a; i.e. $\neg\mathbf{B_U}\, a$ is true. Then at time $(t+1)$, the agent may believe a because of the most recent response of the database. The following specifies that after the occurrence of an event, the beliefs of the user are the logical consequences of his/her previous beliefs and the new information provided by the database.

$$\neg\mathbf{B_U}\,(a \rightarrow b) \rightarrow \bigcirc(\mathbf{Tell}\, a \rightarrow \neg\mathbf{B_U}\, b) \tag{7}$$

The conclusion of (7) states that the beliefs of the user don't increase magically[6]: if he did not believe that $a \rightarrow b$ was true, then, after being told a, s/he will not magically believe b.

4.5 Properties of Dialogs

We now establish some elementary properties of yn-dialogs and ynd-dialogs. These show how the queries of the user and the space of secrets constrain the behavior of the database. All the propositions of this section follow from the axioms introduced above.

The first proposition does not depend on any particular kind of dialog. It makes two observations: first that if a is a secret, then the database will not tell the user a. Secondly, if the a is a secret which the user asks about and further knows that the database is in possession of the answer to his/her query, then the database will lie and answer $\neg a$. Note that these two important properties are a *consequence* of our logical framework – they are not assumptions.

Proposition 4.1 The following two properties hold:

1. $\mathbf{S}a \rightarrow \neg\mathbf{Tell}\, a$
2. $\mathbf{Ask}\, a \wedge \mathbf{S}a \wedge \mathbf{B_U}\, \neg(\neg\mathbf{B_D}\, a \wedge \neg\mathbf{B_D}\, \neg a) \rightarrow \mathbf{Tell}\, \neg a$

In any yn-dialog, the user believes that the database is complete, and hence, that the answer to any query should be yes/no. Consequently, the first part of the following proposition which says that in yn-dialogs, if a is a secret, and the user asks about a, then the database should lie and answer

[6] Actually, as long as any magical increases of knowledge on the part of the user are known to the database, our theory still applies.

"no," follows immediately from (2). The second part of the following proposition indicates that in ynd-dialogs, the database is free to either lie or plead ignorance. We reiterate that these propositions follows from our axioms and are not assumptions.

Proposition 4.2 The following two properties hold:

1. For all yn-dialogs, $\mathbf{Ask}\, a \wedge \mathbf{S} a \rightarrow \mathbf{Tell}\, \neg a$
2. For ynd-dialogs, $\mathbf{Ask}\, a \wedge \mathbf{S} a \rightarrow (\mathbf{Tell}\, \neg a \vee \mathbf{Tell}\, (\neg \mathbf{B_D}\, a \wedge \neg \mathbf{B_D}\, \neg a))$

The above properties do not specify an algorithm that the database might use to answer queries using the logics of secrecy and the interactions. This is the focus of the next section.

5 A Security-Preserving Strategy for Complete Databases

In this section, we develop an algorithm for answering queries to logic databases. The algorithm considers the case when the database DB is complete, i.e., for all queries Q expressible in the user's query language, either $DB \models Q$ ("yes" answer) or $DB \models \neg Q$ ("no" answer). For such databases, only yn-dialogs need to be considered (and hence, the modality $\mathbf{B_D}$ is not required). The algorithm we develop for complete databases has the following properties:
(a) It provides a simple *sound and complete* method for answering queries in such a way that secrecy will not be violated unless the user believed a disjunction of secrets at the beginning.
(b) The algorithm lies only when not doing so would cause a secret to be violated, either now or in the future (i.e. regardless of any sequence of queries that the user may ask in future).
(c) The algorithm does not need to perform any "lookaheads" when answering queries.

The principal result of this section is the following: Suppose DB is a logic database, and $\mathbf{KER\_SEC}(u)$ is a finite set of secrets. Then for any sequence $Q_1, Q_2, \ldots$ of queries that the user may ask, there is a yn-dialog $(Q_1, T_1), \ldots, (Q_i, T_i), \ldots$ which preserves secrecy iff the user's initial beliefs did not entail $\bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$.

The above result provides a direct algorithm that may be used to implement the database's query answering procedure. We will prove this theorem later on (Theorem 5.1).

Algorithm 1 (Query Answering Algorithm) Suppose DB is a complete database, $\mathbf{KER\_SEC}(u)$ is the set of secrets, and $Bel(u)$ is the set of current beliefs of the user. Suppose the current user query is Q.

1. (Correct Answer = "yes", Disjunction of Secrets is not Believed) If $DB \models Q$ and $Bel(u) \cup \{Q\} \not\models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$, then answer "yes". Add Q to $Bel(u)$.
2. (Correct Answer = "no", Disjunction of Secrets is not Believed) If $DB \models \neg Q$ and $Bel(u) \cup \{\neg Q\} \not\models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$, then answer "no". Add $\neg Q$ to $Bel(u)$.

3. (Correct Answer = "yes", Disjunction of Secrets is Believed) If $DB \models Q$ and $Bel(u) \cup \{Q\} \models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$, then answer "no". Add $\neg Q$ to $Bel(u)$.
4. (Correct Answer = "no", Disjunction of Secrets is Believed) If $DB \models \neg Q$ and $Bel(u) \cup \{\neg Q\} \models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$, then answer "yes". Add Q to $Bel(u)$.

In Steps 1 and 2 of the above algorithm, we consider the case when giving the correct answer does not violate any disjunction of secrets. In this situation, security is preserved with an honest response. On the other hand, steps 3 and 4 deal with the situation where providing an honest response causes a disjunction of secrets to become known to the user. In these cases, lying is resorted to.

As can be immediately seen from the specification of the above algorithm, at any point in time when an (**Ask**,**Tell**) event is occurring, the query answering strategy determines the response to the query based solely upon what the user's current set of beliefs is, what is contained in the database, and the set of secrets. In particular, no forward look-ahead is performed. More importantly, secrecy can be preserved irrespective of any queries that may be asked in the future (as long as the user did not believe a disjunction of secrets when s/he started out). The following example shows how the algorithm works.

Example 5.1 Consider the following Horn database, DB, which contains cyclic dependencies.

$$
\begin{array}{lllllllll}
p & \leftarrow & q \,\&\, r & s & \leftarrow & t & r & \leftarrow & \\
r & \leftarrow & s \,\&\, w & q & \leftarrow & w & t & \leftarrow & \\
w & \leftarrow & r & w & \leftarrow & p & v & \leftarrow &
\end{array}
$$

The set, $Bel(u)$, of beliefs of the user, is: $Bel(u) = \{r \leftarrow v \,\&\, w\}$. The set of secrets is $\{r, s\}$. Consider the following sequence of events.

Suppose the user asks **Ask**p. Now, p follows from DB. In Step 1 of the query answering algorithm, an attempt is made to check if $Bel(u) \cup \{p\} \models (r \vee s)$. In this case, $Bel(u) \cup \{p\}$ does not entail $(r \vee s)$, and hence, by Step 1, the database will respond with **Tell**p, i.e. by saying "yes." $Bel(u)$ is augmented with p.

Suppose the user now asks **Ask**w. Again, w follows from DB. In Step 1 of the query answering algorithm, an attempt is made to check if $Bel(u) \cup \{p, w\} \models (r \vee s)$. $Bel(u) \cup \{p, w\}$ does not entail $(r \vee s)$, a the database responds with **Tell**w, i.e. by saying "yes." $Bel(u)$ is augmented with w.

The next query asked by the user is v. It is immediate that $DB \models v$. However, $Bel(u) \cup \{p, w, v\}$ does entail $(r \vee s)$. Hence, if the database honestly answered "yes" to this query, this would allow the (perfectly reasoning) user to conclude $(r \vee s)$, and hence, the database should lie and answer "no." This is done in Step 3 of the algorithm. ■

Before proceeding to discuss the proof of the Preservation of Security Theorem, it is appropriate to briefly discuss the implementation of the above algorithm. In principle, any mechanism for checking entailment of a query by a database can be used to implement the Query Processing Algorithm because each of the four steps of the algorithm only make such tests. There are a variety of methods to perform these tests for different kinds of databases (cf. magic sets [3], resolution [21], counting methods [24], to name a few).

In a joint project between Cornell University and University of Maryland [4, 23], various implementations of monotonic and non-monotonic deductive databases have been developed. These implementations are based on incremental mixed integer linear programming techniques[7] : at any given point in time, an optimized linear programming tableau corresponding to the database is maintained. Query processing corresponds to adding a new linear constraint to the the tableau and re-optimizing an objective function. Experimental results reported in [4, 23] indicate that such reoptimizations make effective use of previous computations (captured by the previously optimized simplex tableau) and are very efficient in practice. These mechanisms can be used to implement the above Query Answering Algorithm as follows: at any given point in time, we maintain an optimized simplex tableau, $Tab(DB)$, corresponding to the database, DB, and another tableau, $Tab(Bel(u))$, corresponding to the user's beliefs at that time and the constraints corresponding to the negation of $\bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$. Given a query Q, step 1 of the Query Answering Algorithm is implemented as follows:

(1a) Add a constraint corresponding to the negation of the query Q to $Tab(DB)$, and check for solvability of the resulting tableau. If there is no solution, then $DB \models Q$.

(1b) If this is the case, then add the constraint corresponding to Q to $Tab(Bel(u))$. Check for solvability of the resulting tableau. If it is solvable, then $Bel(u) \cup \{Q\} \not\models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$, and we can answer "yes" at the end of Step 1.

(1c) Delete the constraint corresponding to $\neg Q$ from $Tab(DB)$. If the answer "yes" was not given, then delete the constraint corresponding to Q from $Tab(Bel(u))$.

Steps (2), (3) and (4) of the Query Answering Algorithm may be similarly implemented.

5.1 Model Theoretic Basis for Yes-No Dialogs

We now define a model-theoretic semantics for the logics of secrecy and interaction that will enable us to prove our Preservation of Security Theorem. We start by defining the concept of interpretations. We will then prove the validity of our axioms and present results describing conditions that guarantee the preservation of security.

[7] A linear constraint is a relation of the form $a_1x_1 + \cdots + a_nx_n \Re b$, where $\Re$ is one of $\leq, =, \geq$ and the a_i's, $1 \leq i \leq n$ are integers, and the x_i's, $1 \leq i \leq n$, are integer variables [17]. An integer linear program consists of a set of linear constraints together with an objective function that reflects a quantity to be minimized or maximized. An objective function minimizes or maximizes an expression of the form $a_1x_1 + \cdots + a_nx_n$ where the a_is and x_i's are as before. A linear program is usually represented as a special kind of table called a tableau [17]. Bell et. al. [4] show how deductive databases can be represented as linear programs, and how queries (and their negations) can be represented as additional constraints. The reader interested in the technical details of this technique is referred to [4, 23] – it is beyond the scope of this paper.

Definition 5.1 (World) A *world* w is a propositional interpretation, i.e. w is a set of propositional symbols. If $F \in \mathbf{L}$, we use the notation $w \models F$ to denote the statement "w satisfies formula F" (cf. Shoenfield [25]), and define it in the usual way as follows:

1. If F is a propositional symbol, (i.e., $F \in \mathbf{L}_0$) then $w \models F$ iff $F \in w$.
2. If $F = \neg a$ then $w \models \neg a$ iff $w \not\models a$.
3. If $F = a \vee b$ then $w \models a \vee b$ iff $w \models a$ or $w \models b$.
4. If $F = a \wedge b$ then $w \models a \wedge b$ iff $w \models a$ and $w \models b$.

As is common in modal logics [9], we use the notion of a world as the basis for defining interpretations of our modal language for yn-dialogs. These interpretations characterize the possible changes in the beliefs of the user/database as an interaction between the user and the database proceeds.

Definition 5.2 (Yes-No Interpretations) A *yn-interpretation* is a pair (w_0, T) where w_0 is a world (called the *initial world*) and T is a tree defined as follows:

1. (Consistency of Interior Nodes) Each non-leaf node of the tree is labeled by a *non-empty set* of worlds and
2. (Leaf Nodes are Inconsistent) All leaf nodes are labeled with the empty set of worlds and
3. Each link in the tree is labeled with an event[8] of the form $(\mathbf{Ask}\, a, \mathbf{Tell}\, a)$ or $(\mathbf{Ask}\, a, \mathbf{Tell}\, \neg a)$. $\mathbf{Ask}\, a$ is called the *query* asked at this link, and $\mathbf{Tell}\, \psi$ for $\psi \in \{a, \neg a\}$ is called the *response* given at this link.
4. If N is a node and the label of N is non-empty, then for every event $(\mathbf{Ask}\, \psi, \mathbf{Tell}\, Response)$ expressible in the language, where *Response* is either ψ or $\neg\psi$, there is a link from N to a child of N labeled with this event.
5. If there is a link $(Query, Reponse)$ from a node labeled with the set W of worlds, to its child W', then the following conditions hold:
   - (Non-Forgetfulness) $W' \subseteq W$ and
   - (Trustfulness) If $Response = \mathbf{Tell}\, \psi$, then $w' \models \psi$ for all $w' \in W'$ and
   - (No Magic) If $Response = \mathbf{Tell}\, \psi$, then for all $w \in W$, if $w \models \psi$, then $w \in W'$.

Intuitively, if (w_0, T) is an interpretation, w_0 is an assignment of truth values to the non-modal formulas in DB. The root of T represents the set of propositional worlds that the user initially believes are possible. Each path in the tree T may be represented as a sequence of events $E_1, E_2, \ldots$ where E_1 is the label of a link from the root of T to a node at depth 1, E_2 is the label of a link from this node to one at depth 2, and so on. Figure 1 shows a diagrammatic representation of a yn-interpretation. Each

[8] This can be generalized further to allow non yes-no interactions, but is not required for our purposes.

of the $\mathbf{W}_j$'s is a set of worlds. Strictly speaking, the tree in Figure 1 should be infinitely branching. However, as all user queries are expressed in the underlying propositional language, which we assume is generated by finitely many propositional symbols, there are only finitely many formulas (upto logical equivalence). Hence, T may be considered to be a finitely branching tree. In Figure 1, for each possible query that the user may ask, there are two possible children – one corresponding to a "yes" response, and the other corresponding to a "no" response. The dotted lines in the figure express the fact that several possible events are not represented explicitly.

The following definition specifies what formulas are true in a given yn-interpretation after a sequence of events ("history") has occurred.

Definition 5.3 (Satisfiability) Suppose $K = (w_0, T)$ is an interpretation and $E_1, \ldots, E_n$ $(n \geq 1)$ is a finite sequence of events (possibly empty) K satisfies ψ w.r.t. the event sequence $E_1, \ldots, E_n$, denoted $K[E_1, \ldots, E_n] \models \psi$ (or $K[] \models \psi$ if the sequence of events is empty) if the following conditions hold:

1. if ψ is propositional, then $K[E_1, \ldots, E_n] \models \psi$ iff $w_0 \models \psi$
2. if $\psi = \phi_1 \vee \psi_2$ then $K[E_1, \ldots, E_n] \models \psi$ iff $K[E_1, \ldots, E_n] \models \phi_1$ or $K[E_1, \ldots, E_n] \models \phi_2$.
3. if $\psi = \neg\phi$ then $K[E_1, \ldots, E_n] \models \psi$ iff $K[E_1, \ldots, E_n] \not\models \phi$
4. If $\psi = \mathbf{B_U}\,\phi$ (note that ϕ is a propositional wff) then $K[E_1, \ldots, E_n] \models \psi$ iff for all $w \in W$ where W is the label of the node reached by traversing the links $E_1, \ldots, E_n$ from the root, it is the case that $w \models \phi$.
5. If $\psi = \bigcirc\phi$ then $K[E_1, \ldots, E_n] \models \psi$ iff for every event E_{n+1} such that $E_1, \ldots, E_n, E_{n+1}$ is a path in T, $K[E_1, \ldots, E_n, E_{n+1}] \models \phi$
6. If $\psi = \mathbf{All}\phi$, where ϕ is a proposition, then $K[E_1, \ldots, E_n] \models \psi$ iff there exists a sequence of events, $E_{n+1}, \ldots, E_{n+m}$ where $m \geq 0$ such that $E_1, \ldots, E_{n+m}$ is a path in T and for all $w \in W$ where W is the label of the node reached by traversing the links $E_1, \ldots, E_{n+m}$ from the root, it is the case that $w \models \phi$.

In the situation when our event sequence has length 0, instead of writing $K[] \models \phi$, we abuse notation and write $w_0 \models \phi$. The following example illustrates the notion of satisfaction.

Example 5.2 Consider the example of Figure 1. Consider the path E_1, E_2. Then:

- $K[E_1, E_2] \models \mathbf{B_U}\,\phi$ iff ϕ is true in each world in $\mathbf{W}_5$.
- $K[E_1, E_2] \models \bigcirc\mathbf{B_U}\,\phi$ iff ϕ is true in each world in $\mathbf{W}_8 \cup \mathbf{W}_9$.
- $K[E_1, E_2] \models \mathbf{All}\phi$ iff there exists a node in either the subtree rooted at $\mathbf{W}_8$ or the subtree rooted at $\mathbf{W}_9$ (including $\mathbf{W}_8$ and $\mathbf{W_9}$) having label $\mathbf{W}$ such that ϕ is true in all worlds in $\mathbf{W}$.

Given a yn-interpretation (w_0, T), we now identify certain subtrees of T as being "forcing tree." Intuitively, in a forcing tree, there is a sequence of queries that will always enable the user to violate security irrespective of the answer(s) provided by the database. Consequently, avoiding a forcing tree is very important in order to maintain security – the precise importance will become evident from Lemmas 5.1 and 5.2 later in the paper. ■

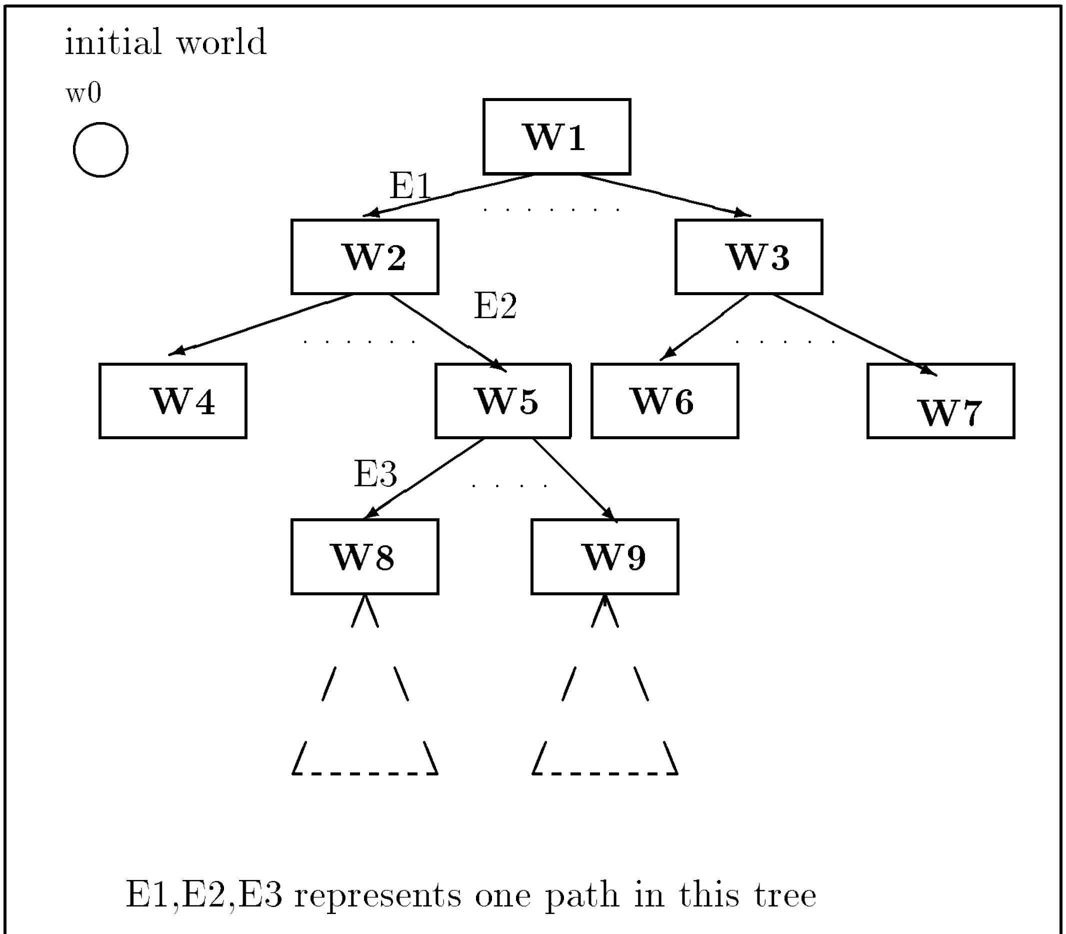

Fig. 1. Yes-No Interpretations

Definition 5.4 (**S-Forcing Tree**) Let (w_0, T) be a yn-interpretation, $Q = q_1, \ldots, q_n$ be a sequence of propositional queries, and **KER_SEC**(u) be a set of propositional secrets. A *forcing tree* for **KER_SEC**(u) w.r.t. the yn-interpretation (w_0, T) and Q, is a finite binary *subtree* T' of T such that:

1. the root of T' is the root of T and
2. every non-leaf node $N \in T'$ of level i has exactly two children N_1 and N_2 such that the link from N to N_1 is labeled by $(\mathbf{Ask}\, q_{i+1}, \mathbf{Tell}\, q_{i+1})$ and the link from N to N_2 is labeled by $(\mathbf{Ask}\, q_{i+1}, \mathbf{Tell}\, \neg q_{i+1})$.

3. For every leaf node L of T' (note that leaf nodes of T' are not necessarily leaf nodes of T), there exists a formula $\psi \in \mathbf{KER\_SEC}^*(u)$ such that for every world w in the set of worlds labeling L, it is the case that $w \models \psi$.

Essentially, every path in T' corresponds to a path in T. Though the definition of forcing trees allows dialogs to continue even after secrecy has been violated, we will assume that this never occurs, i.e. any time a node is reached that leads to a security violation, we will assume that that node is not expanded any further. Thus, T' is obtained from T by

- removing all paths that do not correspond to the sequence of queries Q and
- truncating paths at nodes that cause secrets to be violated.

The following example demonstrates an S-forcing tree.

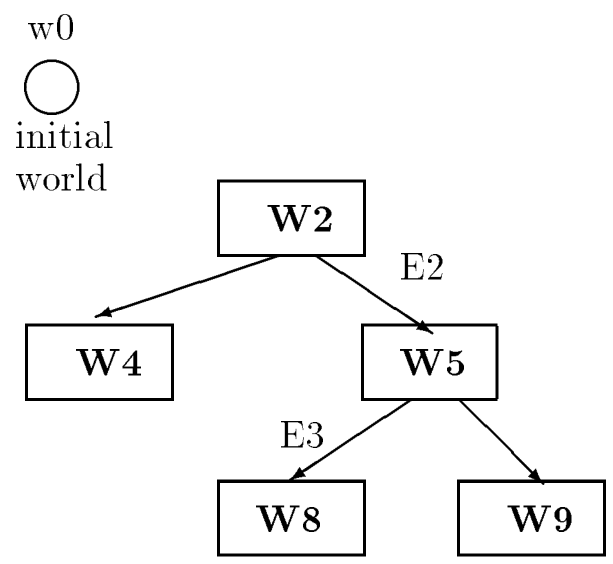

Fig. 2. S-Forcing Tree

Example 5.3 Consider the tree in Figure 1. Consider the subtree T' consisting solely of the nodes shown in Figure 2. T' is an S-forcing tree iff:

1. $\mathbf{W4} = \emptyset$ (in fact, this follows from the definition of yn-interpretation itself) and
2. there is a secret s_1 which is true in each world in $\mathbf{W8}$ and there is a secret s_2 which is true in each world in $\mathbf{W9}$. Note that one or both of $\mathbf{W8}$, $\mathbf{W9}$ may be empty, but this is not necessarily the case. ∎

Basically, if the user believes a disjunction of secrets, there exists a strategy that will enable him/her to force the database to reveal a secret. This is the essence of the following lemma.

Lemma 5.1 (Believing a Disjunction of Secrets Implies Existence of Forcing Trees) Suppose DB is a database, (w_0, T) is a yn-interpretation, and $\mathbf{KER\_SEC}(u)$ is a finite set of propositional secrets. Let $\mathbf{KER\_SEC}^*(u) = \{s_1 \ldots, s_n\}$. If the user initially believed $(s_1 \vee \cdots \vee s_n)$, i.e., $w_0 \models \mathbf{B_U}(s_1 \vee \cdots \vee s_n)$ (we use this notation when no events have occurred), then $Q = s_1, \ldots, s_n$ is a sequence of queries such that there exists a forcing tree w.r.t. (w_0, T) and Q.

Proof. It is easy to see that there is only one tree T' (upto reordering of branches) associated with T that has the same root as T, which satisfies Conditions 1 and 2 of Definition 5.4. We need only to show that this T' also satisfies condition 3 of Definition 5.4.

Suppose W is the set of worlds labeling a leaf node in T'. The path from the root of T' to this leaf node is obtained by events of the form $(\mathbf{Ask}\, s_1, \mathbf{Tell}\, Resp_1), \ldots, (\mathbf{Ask}\, s_n, \mathbf{Tell}\, Resp_n)$. As this is a yn-dialog, $Resp_i$ must be either s_i or $\neg s_i$ for all $1 \leq i \leq n$. There are two possibilities:
Case 1: [For all $1 \leq i \leq n$, $Resp_i = \neg s_i$] In this case, all worlds in W satisfy $(\neg s_1 \wedge \ldots \wedge \neg s_n)$ because of the conditions of Trustfulness and Non-Forgetfulness in the definition of yn-interpretations (Definition 5.2). However, due to the axiom of Non-Forgetfulness, as every world in the label of the root of T' satisfies the initial beliefs of the user, it follows that each world in W must also satisfy the initial beliefs of the user. In particular, $(s_1 \vee \cdots \vee s_n)$ is true in each world of W. However, the only way this could happen is if $W = \emptyset$. Consequently, condition (3) of Definition 5.4 is satisfied.
Case 2: [There exists an $1 \leq i \leq n$ such that $Resp_i = s_i$] Then secret s_i is violated immediately and due to the axiom of non-forgetfulness, all worlds $w_i \in W$ satisfy s_i. ■

Basically, an S-forcing tree causes a secret to be violated or revealed to the user. Hence, a query answering strategy must try to ensure that it never reaches a node such that the subtree rooted at that node is S-forcing. We now prove this below.

Lemma 5.2 (Existence of Forcing Trees Implies that a Disjunction of Secrets was Initially Believed) Suppose DB is a database, (w_0, T) is a yn-interpretation, and $\mathbf{KER\_SEC}(u)$ is a finite set of propositional secrets. Let $\mathbf{KER\_SEC}^*(u) = \{s_1 \ldots, s_n\}$. If there is a sequence $Q = q_1, \ldots, q_n$ of queries such that there is a forcing tree w.r.t. (w_0, T) and Q, then the user must have initially believed a disjunction of secrets, i.e. $w_0 \models \mathbf{B_U}(s_1 \vee \cdots \vee s_n)$.

Proof. Suppose T' is a forcing tree w.r.t. (w_0, T) and Q. Let h be the height of T' (we assume that a tree containing a single node has height 0). We proceed by induction on h.

$\boxed{\text{Base Case } (h = 0)}$ In this case, T' contains only one node. Let W be the set of worlds labeling T'. Since T' is a forcing tree, there is a secret s_i such that each world $w \in W$ satisfies s_i. This completes the proof of the base case.

$\boxed{\text{Inductive Case } (h = k + 1)}$ In this case, let W be the set of worlds labeling the root of T' and let W_1 and W_2 be the sets of worlds labeling the left child (denoted N_1) and right child (denoted N_2), respectively of the root of T'. Figure 3 demonstrates this situation.

Let (w_0, T_1) and (w_0, T_2) be interpretations, where T_1 and T_2 are subtrees of T, rooted in N_1 and N_2 respectively. The subtrees of T' rooted at N_1 and N_2 (denoted by T_1', T_2' respectively) are forcing trees with respect to (w_0, T_1) and (w_0, T_2) and the sequence of queries $Q' = q_2, \ldots, q_n$. By the induction hypothesis applied to (w_0, T_1), (w_0, T_2) and Q', it follows that every world $w \in W_1 \cup W_2$ satisfies a disjunction of secrets, i.e. $(\forall w \in W_1 \cup W_2) w \models (s_1 \vee \cdots \vee s_n)$.

We need to show that for all worlds $w^* \in W$, $w^* \models (s_1 \vee \cdots \vee s_n)$. We know from the definition of yn-interpretations that $W_1, W_2 \subseteq W$. If $w^* \in W$, then w^* must either satisfy query q_1 or satisfy $\neg q_1$. In the first case, $w^* \in W_1$; in the second case, $w^* \in W_2$. Thus, $W = W_1 \cup W_2$. Consequently, $(\forall w^* \in W) w^* \models (s_1 \vee \cdots \vee s_n)$. ■

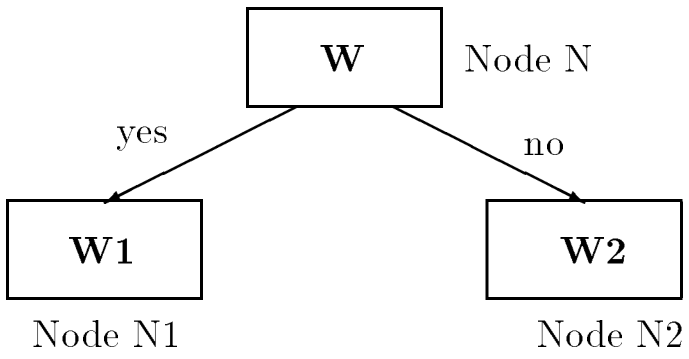

Fig. 3. Yes-No Answers

The above two lemmas tell us that if we start out in an initial situation where the user did not believe a disjunction of secrets, then it is possible to preserve security *without* any look-ahead. To do so, one simply must make sure that the answer provided to the user does not violate a disjunction of secrets. We now state the "Preservation of Security" Theorem; its proof follows from Lemmas 5.1 and 5.2.

Theorem 5.1 *[Preservation of Security] Suppose DB is a logic database, and* **KER_SEC**(u) *is a finite set of secrets. Then for any sequence $Q_1, Q_2, \ldots$ of queries that the user may ask, there is a* *yn-dialog* $(Q_1, T_1), \ldots, (Q_i, T_i), \ldots$ *which preserves secrecy iff the user's initial beliefs did not entail* $\bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$. ■

Proof. Let (w_0, T) be any yn-interpretation.

- Suppose the user's initial beliefs entail $\bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$. By Lemma 5.1 there is a forcing tree with respect to a sequence of queries $Q_1, ..., Q_n$. If the user will ask these queries, any combination of answers that will be given by the DB, will lead to revealing of a secret, since the forcing tree corresponds to any possible policy of answering the queries $Q_1, ..., Q_n$, and all possibilities lead to leaves where a secret is believed by the user. That is, there is no yn-dialog for $Q_1, ..., Q_n$ which preserves secrecy.
- Suppose the user's initial beliefs do not entail $\bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$ and suppose (w_0, T) is a yn-interpretation such that $w_0 \not\models B_U(\bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi)$. By Lemma 5.2, there is no forcing tree with respect to (w_0, T) and with respect to any sequence of queries $Q_1, ..., Q_n$. That is, for any sequence of queries, there is at least one sequence of events $E_1, ..., E_n$ such that $E_i = (\mathbf{Ask}\, Q_i, \mathbf{Tell}\, Resp_i)$ and for all $i \geq 1$ $K[E_1, ..., E_n] \not\models B_U s_i$, for all $s_i \in \mathbf{KER\_SEC}^*(u)$. The DB should respond to the user with $Resp_i$ for all queries Q_i that appear in a given sequence of queries $Q_1, ..., Q_n$. ■

Having shown that security is preservable using yn-dialogs iff the user did not initially believe a disjunction of secrets, we now proceed to study the complexity of secure query-answering.

5.2 Complexity of Secure Query-Answering

In this section, we briefly address the complexity of query processing in secure deductive databases. We prove (cf. Theorem 5.2 below) that query answering is NP-complete. Note that query processing in logic databases when the set of secrets is empty is also NP-complete. Consequently, the *need to preserve security does not increase the computation of query processing to a more intractable complexity class*, i.e. both problems are located in the same complexity class, and hence are polynomially reducible to each other.

Furthermore, when only atomic secrets (or even conjunctions of atoms) are considered, and databases are *definite*[9], we prove that query processing is achievable in linear-time even if secrets need to be preserved. Hence, here too, the complexity class of query processing does not increase to a more intractable class.

A warning is in order, though: even though ordinary query processing (without any need to preserve secrets) is located in the same complexity class as that of secure query processing, the *algorithmic complexity* of the latter is affected by a linear factor.

[9] A definite database is a set of clauses of the form $A \leftarrow B_1 \wedge \ldots \wedge B_n$ where each of $A, B_1, \ldots, B_n$ is an atom.

Theorem 5.2 (Complexity of Secure Query-Answering) Suppose DB is a logic database, and $\mathbf{KER\_SEC}(u)$ is a finite set of secrets. Let $(Q_1, T_1), \ldots, (Q_i, T_i)$ be a yn-dialog such that no secrecy is violated. Then, given as input, a query $Q_{i+1} = \mathbf{Ask}\, q_{i+1}$, the problem of answering query Q_{i+1} so that the user is not led to believe any secret, is NP-complete.

Proof. $\boxed{\text{NP-Hardness}}$ The proof is by a reduction to SAT [15]. Suppose $C = \{c_1, \ldots, c_n\}$, is a set of propositional clauses (We assume $n \geq 2$. This assumption leads to no loss of generality as a set consisting of a single clause is always satisfiable). Let $C1 = C - \{c_n\}$ and let $c_n = (\ell_1 \vee \cdots \vee \ell_k)$ where each $\ell_i, 1 \leq i \leq k$ is a propositional literal.

	C is satisfiable
iff	it is not the case that $C1 \models \neg c_n$
iff	it is not the case that $C1 \models (\neg \ell_1 \wedge \cdots \wedge \neg \ell_k)$

Take $DB = C1$, $Bel(u) = C1 - \{c_1\}$, and let $\mathbf{KER\_SEC}^*(u)$ consist of the single secret $s = (\neg \ell_1 \wedge \cdots \wedge \neg \ell_k)$, i.e. $\mathbf{KER\_SEC}^*(u) = \{s\}$. Let Q be c_1. Thus, we answer "yes" to the query $\mathbf{Ask}\, Q$ iff

	it is not the case that $Bel(u) \cup \{Q\} \models s$
iff	it is not the case that $(C1 - \{c_1\}) \cup \{c_1\} \models (\neg \ell_1 \wedge \cdots \wedge \neg \ell_k)$
iff	it is not the case that $C1 \models (\neg \ell_1 \wedge \cdots \wedge \neg \ell_k)$
iff	C is satisfiable

Consequently, as this reduction is polynomial-time, it follows that our problem is NP-hard.

$\boxed{\text{Membership in NP}}$ The problem is clearly in NP because our algorithm can be executed in polynomial time on a non-deterministic Turing machine. ∎

An immediate consequence of the proof of Theorem 5.2 is the following:

Corollary 5.1 Suppose DB is a *complete database*, i.e. for each formula ψ in the underlying propositional language, $DB \models \psi$ or $DB \models \neg \psi$. Suppose $\mathbf{KER\_SEC}(u)$ is a finite set of secrets. Let $(Q_1, T_1), \ldots, (Q_i, T_i)$ be a yn-dialog, where each Q_i is of the form $\mathbf{Ask}\, q_i$ where q_i is an atom. Furthermore, suppose no secret is violated. Then, given as input a query $Q_{i+1} = \mathbf{Ask}\, q_{i+1}$, the problem of answering query Q_{i+1} so that security is preserved" is NP-complete. ∎

A *Horn Database* is a finite set of clauses of the form $a_1 \wedge \ldots \wedge a_n \rightarrow b$ where $a_1, \ldots, a_n, b$ are all *atoms*. It is well-known[22] that if DB is a Horn database, then DB entails no negative atom, i.e. there does not exist an atom c such that $DB \models \neg c$. Consequently, in order to infer negative information, we use the "Closed World Assumption" which says that we can conclude $\neg c$ from DB iff $DB \not\models c$. Thus, the *closed world theory* $CWT(DB)$, associated with a Horn Database DB may be defined to be:

$CWT(DB) = DB \cup \{\neg c \mid c \text{ is an atom and } c \text{ not a logical consequence of } DB\}.$

As $CWT(DB)$ is simply a set of propositional formulas, we may consider $CWT(DB)$ to be a database. It is well-known [21] that the set $CWT(DB)$ is a complete database whenever DB is Horn and only atoms (and conjunctions of atoms) are considered.

Corollary 5.2 (Complexity of (Atomic) Secure Query-Answering in Horn Databases) Suppose DB is a Horn database augmented with the Closed World Assumption [21] (i.e. the theory being considered is $CWT(DB)$), and $\mathbf{KER\_SEC}(u)$ is a finite set of conjunctive atomic secrets (i.e. each secret is a conjunction of atoms). Let $(Q_1, T_1), \ldots, (Q_i, T_i)$ be a yn-dialog, where each Q_i is of the form $\mathbf{Ask}\, q_i$ where q_i is an atom. Furthermore, suppose no secrecy is violated, and that the user's belief set $Bel(u)$ consists only of atoms. Then, given as input, a query $Q_{i+1} = \mathbf{Ask}\, q_{i+1}$, the problem of answering query Q_{i+1} w.r.t. $CWT(DB)$ so that security is preserved" is solvable in linear-time. ■

Proof. The Query Answering Algorithm described earlier answers "yes" to the query $\mathbf{Ask}\, q_{i+1}$ iff either

$$(1)\quad CWT(DB) \models q_{i+1} \text{ and } Bel(u) \cup \{q_{i+1}\} \not\models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$$

is true or

$$(2)\quad CWT(DB) \models \neg q_{i+1} \text{ and } Bel(u) \cup \{\neg q_{i+1}\} \not\models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi.$$

We show below that condition (1) above can be checked in linear-time. The same reasoning applies to condition (2). Dowling and Gallier [14] (see also Vardi [28]) have given an algorithm that would check, in linear-time, if $DB \models q_{i+1}$. Under the conditions specified, it is well-known [21], that for all positive atoms A:

- $DB \models a$ iff $CWT(DB) \models a$ and
- $DB \not\models a$ iff $CWT(DB) \models \neg a$.

Consequently, if the Dowling-Gallier algorithm answered "yes", we would know that $CWT(DB) \models q_{i+1}$; if "no", we would know that $CWT(DB) \models \neg q_{i+1}$. The condition

$$Bel(u) \cup \{q_{i+1}\} \not\models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi$$

can be checked in linear time as well because it suffices to check the complementary problem

$$Bel(u) \cup \{q_{i+1}\} \models \bigvee_{\psi \in \mathbf{KER\_SEC}^*(u)} \psi.$$

As $Bel(u)$ is atomic, and as q_{i+1} is atomic, it follows by Dowling and Gallier's results that this check can be carried out in linear time as well. This completes the argument that the first case can be solved in linear time. The same reasoning applies for the second case. ■

Theorem 5.2 and Corollary 5.2 may be viewed as positive results about secure query processing. They basically state that the complexity of secure query processing is not located in a "more intractable" complexity class (they are not PSPACE-complete or EXPTIME-complete, for example) than that of regular query processing.

6 Security in Incomplete Databases: Yes-No-Don't Know Dialogs

Most databases are incomplete in the sense that there may be some queries for which the database does not know the answer. As discussed earlier, when the database does not know the answer to a query, it should be able to say "don't know". This ability is captured via yes-no-don't know dialogs. In this section, we show how security in the case of yes-no-don't know dialogs may be characterized. The key technical distinction between yn-dialogs, and ynd-dialogs is that the latter needs the extra modal connective $\mathbf{B_D}$ which denotes the beliefs of the database. In modal logic, arbitrary alternations of connectives are considered perfectly reasonable. However, in the case of secure deductive databases, this appears to be rather unnecessary. Consequently, we only consider those situations where the user reasons about the database's beliefs, i.e. we are not concerned with any interplay between the modal operators $\mathbf{B_D}$ and $\mathbf{B_U}$ except in formulas of the form $\mathbf{B_U}\,\mathbf{B_D}\,\psi$ where ψ is a propositional wff (free of modalities).

When defining model-theory to handle incomplete databases, the notion of "world" introduced earlier is inadequate because it does not specify the beliefs of the database (which may be incomplete). The notion of "epistemic world" handles this.

Definition 6.1 (Epistemic World) An *epistemic world* is any set E of the form $W \cup B$ where:

- W is a world (i.e. a set of propositional atoms) and
- B is a set of formulas of the form $\mathbf{B_D}\,\psi$ where ψ is a propositional formula. We assume that the set $\{\psi \mid \mathbf{B_D}\,\psi \in B\}$ of propositional logic formulas, is closed under ordinary propositional consequence.

Intuitively, an epistemic world E contains information about the objective, modality-free formulas (the W part) together with some information about the *beliefs* held by the database (the B part) [20]. Note that there may be propositional formulas ψ such that neither $\mathbf{B_D}\,\psi$ nor $\mathbf{B_D}\,\neg\psi$ is in an epistemic world E. Thus, epistemic worlds are considered to be two-valued w.r.t. ordinary logical formulas, but as far as formulas of the form $\mathbf{B_D}\,\psi$ and $\mathbf{B_D}\,\neg\psi$ are concerned, they adopt a three-valued character. For all practical purposes, we can think of formulas $\mathbf{B_D}\,\psi$ as being "new" propositional symbols. Epistemic worlds are used as the key component of *yes-no-don't know interpretations* which are defined below.

We say an epistemic world E satisfies the formula $\mathbf{B_D}\,\psi$ iff $\mathbf{B_D}\,\psi \in E$. As usual, this is denoted by the notation $E \models \mathbf{B_D}\,\psi$. Note that there may be propositional wffs ψ such that $E \not\models \mathbf{B_D}\,\psi$ and $E \not\models \mathbf{B_D}\,\neg\psi$. However, E must satisfy either ψ or $\neg\psi$.

The following definition extends the notion of "yes-no" interpretation introduced earlier to handle the situation where the database may answer "don't know."

Definition 6.2 (Yes-No-Don't Know Interpretations) A *ynd-interpretation* is a pair (e_0, T) where e_0 is an epistemic world (called the *initial epistemic world*) and T is a tree defined as follows:

1. (Consistency of Interior Nodes) Each non-leaf node of the tree is labeled by a *non-empty set* of epistemic worlds and
2. (Leaf Nodes are Inconsistent) All leaf nodes are labeled with the empty set of epistemic worlds and
3. Each link in the tree is labeled with a ynd-event of the form $(\mathbf{Ask}\,a, \mathbf{Tell}\,a)$ or $(\mathbf{Ask}\,a, \mathbf{Tell}\,\neg a)$ or $(\mathbf{Ask}\,a, \mathbf{Tell}\,(\neg\mathbf{B_D}\,a \wedge \neg\mathbf{B_D}\,\neg a)$. $\mathbf{Ask}\,a$ is called the *query* asked at this link, and $\mathbf{Tell}\,\psi$ for $\psi \in \{a, \neg a, (\neg\mathbf{B_D}\,a \wedge \neg\mathbf{B_D}\,\neg a)\}$ is called the *response* given at this link.
4. If N is a node and the label of N is non-empty, then for every event $(\mathbf{Ask}\,\psi, \mathbf{Tell}\,\mathit{Response})$ where *Response* is either ψ or $\neg\psi$ or $(\neg\mathbf{B_D}\,\psi \wedge \neg\mathbf{B_D}\,\neg\psi)$, there is a link from N to a child of N labeled with this event.
5. If there is a link $(\mathit{Query}, \mathit{Reponse})$ from a node labeled with the set W of epistemic worlds, to its child W', then the following conditions hold:
   - (Non-Forgetfulness) $W' \subseteq W$ and
   - (Trustfulness) If $\mathit{Response} = \mathbf{Tell}\,\psi$, then $e' \models \psi$ for all $e' \in W'$ and
   - (No Magic) If $\mathit{Response} = \mathbf{Tell}\,\psi$, then for all $e \in W$, if $e \models \psi$, then $e \in W'$.

As in the case of yn-interpretations, there is a notion of a "forcing tree" for ynd-interpretations. Existence of such ynd-forcing trees causes violation of secrets even when the database is allowed to answer "don't know" to user queries.

Definition 6.3 (YND Forcing Tree) Let (e_0, T) be a ynd-interpretation, $Q = q_1, \ldots, q_n$ be a sequence of propositional queries, and $\mathbf{KER\_SEC}(u)$ be a set of propositional secrets. A *YND forcing tree* for $\mathbf{KER\_SEC}(u)$ w.r.t. the ynd-interpretation (w_0, T) and Q, is a finite ternary *subtree* T' of T such that:

1. the root of T' is the root of T and
2. every node $N \in T'$ at level $i < n$ has three children N_1, N_2 and N_3 such that the link from N to N_1 is labeled by $(\mathbf{Ask} q_{i+1}, \mathbf{Tell}\,q_{i+1})$, the link from N to N_2 is labeled by $(\mathbf{Ask} q_{i+1}, \mathbf{Tell}\,\neg q_{i+1})$ and the link from N to N_3 is labeled by $(\mathbf{Ask}\,q_{i+1}, \mathbf{Tell}\,(\neg\mathbf{B_D}\,q_{i+1} \wedge \neg\mathbf{B_D}\,\neg q_{i+1}))$.

3. For every leaf node L, there exists a formula $\psi \in \mathbf{KER\_SEC}^*(u)$ such that for every epistemic world e in the set of epistemic worlds labeling L, it is the case that $e \models \psi$.

Lemma 6.1 below is similar to Lemma 5.2 except that it applies to *yes-no-don't know* dialogs, rather than yn-dialogs. It states that if a YND forcing tree exists (and hence the user has a way of violating secrecy), then the user must have known a disjunction of secrets when s/he started out.

Lemma 6.1 (Existence of YND Forcing Trees Implies that a Disjunction of Secrets was Initially Believed) Suppose DB is a database, (e_0, T) is a ynd-interpretation, and $\mathbf{KER\_SEC}(u)$ is a finite set of propositional secrets. Let $\mathbf{KER\_SEC}^*(u) = \{s_1 \ldots, s_n\}$. If there is a sequence $Q = q_1, \ldots, q_n$ of queries such that there is a forcing tree w.r.t. (e_0, T) and Q, then the user must have initially believed a disjunction of secrets, i.e. $e_0 \models \mathbf{B_U}\,(s_1 \vee \cdots \vee s_n)$. ■

For yn-dialogs, the converse of Lemma 5.2 is true (cf. Lemma 5.1 which states that if the user initially believed a disjunction of secrets then s/he has a way to violate security). However, for ynd-dialogs, this situation is not true. It may turn out that the user initially believed a disjunction of secrets, *but the database may still be able to prevent him from violating secrecy by using "don't know" answers.* The following example illustrates this.

Example 6.1 Suppose the user knows that either X or Y is an assassin. Suppose also that the user believes that secret information (such as being an assassin) has not been inserted into the database. Then the database can answer "I don't know" both to the question "is X an assassin?" and to the question "is Y an assassin?". The user will not be able to use these answers to conclude that X and/or Y is an assassin. ■

It turns out that a weaker version of Lemma 5.1 does hold for ynd-dialogs. It says that if the user believed a disjunction, $(a_1 \vee \cdots \vee a_n)$, of secrets when s/he started out *and* s/he believed that for each secret, a_i, in this disjunction, the database had a definite belief about a_i (i.e. the database could not say "don't know"), then the user does have a way of violating security.

Lemma 6.2 (Believing a Disjunction of Secrets and Knowing the Database is Knowledgeable About the Secrets Implies Existence of YND Forcing Trees) Suppose DB is a database, (e_0, T) is a ynd-interpretation, and $\mathbf{KER\_SEC}(u)$ is a finite set of propositional secrets. Let $\mathbf{KER\_SEC}^*(u) = \{s_1 \ldots, s_n\}$. If the user initially believed $(s_1 \vee \cdots \vee s_n)$, i.e., $e_0 \models \mathbf{B_U}\,(s_1 \vee \cdots \vee s_n)$, and the user initially believed $(\mathbf{B_D}\, s_i \vee \mathbf{B_D}\, \neg s_i)$ for all $1 \leq i \leq n$, then $Q = s_1, \ldots, s_n$ is a sequence of queries such that there exists a YND-forcing tree w.r.t. (e_0, T) and Q.

Proof. It is easy to see that there is only one tree T' associated with T that has the same root as T, which satisfies Conditions 1 and 2 of Definition 6.3. We need only to show that this T' also satisfies condition 3 of Definition 5.4.

Suppose W is the set of worlds labeling a leaf node in T'. The path from the root of T' to this leaf node is obtained by events of the form

$$(\mathbf{Ask}\, s_1, \mathbf{Tell}\, Resp_1), \ldots, (\mathbf{Ask}\, s_n, \mathbf{Tell}\, Resp_n).$$

As this is a ynd-dialog, $Resp_i$ must be either s_i or $\neg s_i$ or $(\neg \mathbf{B_D}\, s_i \wedge \neg \mathbf{B_D} \neg s_i)$ for all $1 \leq i \leq n$. There are three possibilities:

Case 1: [For all $1 \leq i \leq n$, $Resp_i = \neg s_i$] In this case, all epistemic worlds in W satisfy $(\neg s_1 \wedge \ldots \wedge \neg s_n)$ because of the conditions of Trustfulness and Non-Forgetfulness in the definition of ynd-interpretations (Definition 6.2). However, due to the axiom of Non-Forgetfulness, as every epistemic world in the label of the root of T' satisfies the initial beliefs of the user, it follows that each world in W must also satisfy the initial beliefs of the user. In particular, $(s_1 \vee \cdots \vee s_n)$ is true in each world of W. However, the only way this could happen is if $W = \emptyset$. Consequently, condition (3) of Definition 6.3 is satisfied.

Case 2: [There exists an $1 \leq i \leq n$ such that $Resp_i = s_i$] Then secret s_i is violated immediately and due to the axiom of non-forgetfulness, all worlds $w_i \in W$ satisfy s_i.

Case 3: [There exists an $1 \leq i \leq n$ such that $Resp_i = (\neg \mathbf{B_D}\, s_i \wedge \neg \mathbf{B_D} \neg s_i)$] In this case, each epistemic world in W must satisfy $(\neg \mathbf{B_D}\, s_i \wedge \neg \mathbf{B_D} \neg s_i)$. However, due to the axiom of Non-Forgetfulness in the definition of ynd-interpretations, and as the user originally believed $(\mathbf{B_D}\, s_i \vee \mathbf{B_D} \neg s_i)$ it turns out that each epistemic world $e \in W$ must satisfy $(\mathbf{B_D}\, s_i \vee \mathbf{B_D} \neg s_i)$ as well. As $(\mathbf{B_D}\, s_i \vee \mathbf{B_D} \neg s_i)$ and $(\neg \mathbf{B_D}\, s_i \wedge \neg \mathbf{B_D} \neg s_i)$ are mutually inconsistent, the only way this could happen is if $W = \emptyset$. ■

According to Lemma 6.1, the database can use the same algorithm as in the yn-dialogs (Algorithm 1) to respond to queries and to preserve security. However, by Lemma 6.2, it is clear that by using this algorithm the database will lie more than necessary and will not be maximally cooperative. An open question is how the ynd-dialogs can allow the database to preserve security "better" than yn-dialogs.

7 Related work

Consider Example 1.2. There, the response "I refuse to answer" is inadequate, because the user may use the database's refusal to respond to infer that a piece of information is secret. This case is explicitly excluded in [26]; hence, their theorems and techniques do not apply to our example. More specifically, they prove that there exist two systems that give the same answers: one where "X is an assassin" is a secret, and one where "X is not an assassin" is a secret. However, when the framework of [26] is applied to Example 1.2, the user can violate the secret because s/he can infer that the latter system should be excluded.

Our result on yes/no dialogs is easily comparable with the results of [26]. We both prove that, under suitable assumptions, the system can always answer the user's queries without violating any secret. The assumptions we make are complementary to theirs: they restrict the user's reasoning about query rejections, while we impose slightly stronger constraints on the user's initial knowledge about secret facts. We believe our approach is reasonable because we assume that the user is as smart as possible ("perfect reasoner axioms" and "non-forgetfulness"). This seems more reasonable in real life than making assumptions that limit the intelligence of the user.

As we have already pointed out, [26] suggests that the system might arbitrarily withhold answers, in order to prevent the user to make inferences like:

If Q was not a secret, then the system would have answered.

This technique introduces extra secrets, which are independent of the structure of the database and independent of the beliefs/knowledge of the user. Our technique of lying may introduce more secrets, but this happens only when these secrets are logically related to initial secrets. In fact, as can be seen from our algorithms, we only lie when not doing so would cause a secret to be potentially revealed by further questioning by the user.

Example 7.1 Let the database be composed by the following facts: (a) A, B and C are not terrorists, (b) D and E are terrorists.

If the user does not know that at least one of A, B, C, D and E is a terrorist (i.e., he cannot logically relate the five facts), then the system can honestly answer the query "Is X a terrorist?" when X either is A, B or C. The system is forced to lie only when X is D or E, i.e., only when an initial secret is involved. If the user knows that one of A, B, C, D and E is a terrorist, then the system has to lie about one of A, B and C. This shows that new secrets are introduced only when the beliefs of the user necessitate doing so. If the user doesn't know "too much", the system is very cooperative. ■

Therefore, lying is a promising technique when the set of correct answers has to be maximized. Furthermore, there are some applications that require the database to deceive some users by supplying incorrect answers (see [31] for a discussion of the effectiveness of existing database security techniques for handling such situations.) In such cases, we gives a theoretical foundation for lying, and our approach can be used to preserve the integrity of the lies and provide the database with an algorithm for lying consistently.

There are several other formalizations of security based on modal logics. In [5], a modal logic of security is defined, based on a modal knowledge operator, $\mathbf{K_A}$, and a modal operator $\mathbf{R_A}$, that captures what the user is allowed to know. The semantics of $\mathbf{R_A}$ and $\mathbf{K_A}$ are similar; both of them satisfy all the axioms of modal logic S5 (i.e. axioms *K, T, 4, 5*, see [9].)

Although $\mathbf{R_A}$ and $\mathbf{K_A}$ intuitively correspond to our operators **All** and $\mathbf{B_U}$, respectively, there are important differences. As we have already pointed out, $\mathbf{B_U}$ models the *beliefs* of the user, rather than his/her knowledge; a technical consequence of this fact is that $\mathbf{K_A}$ satisfies the well-known modal axiom T while $\mathbf{B_U}$ does not. Furthermore, our operator **All** is a "diamond"-like possibility modality, as opposed to $\mathbf{R_A}$, which is a "box"-like necessity modality. For example, $\mathbf{R_A}$ satisfies both axiom T and the schema $\mathbf{R_A}\varphi \wedge \mathbf{R_A}\psi \rightarrow \mathbf{R_A}(\varphi \wedge \psi)$ while **All** satisfies none of them.

In [10], the logical framework introduced in [5] is adapted to analyze aggregation problems. To solve this class of problems, the above schema should be invalidated, and hence, the author gives a "diamond"-like definition of $\mathbf{R_A}$. However, since the user is allowed to have only correct beliefs, $\mathbf{R_A}$ still satisfies the modal axiom T, and so it is different from **All**.

In [16], a second order modal logic is adopted, that embodies a probabilistic treatment of possible worlds. Furthermore, in this logic, $\mathbf{R_A}$ satisfies all the axioms of S5.

Su and Ozsoyoglu [27] study how to control inferences that may be drawn by users in multilevel relational DBMSs with integrity constraints. They develop a method to assign security levels to prevent inferences that lead to security violations. In contrast, our work addresses the security problem in deductive databases (as opposed to the more restricted relational model), and develops algorithms that guarantee preservation of secrets (unless the user knew a disjunction of secrets at the beginning).

In addition to the modal logic formalizations of secure databases discussed above, there has also been a good deal of formal work in the area of statistical databases and statistical methods for preserving secrets [1]. In the statistical approach, the users of a database DB, are allowed to inspect *functions* of DB rather than DB itself; typical functions are COUNT, SUM, PERCENTILE, etc. However, from the global information provided by these functions, it may be possible to infer properties of a specific individual; if this is the case, then security is violated. There are two classes of approaches to security in statistical databases (SDB's for short), based on query restrictions or data perturbation. Adam and Wortmann [1] conclude that the latter is generally more effective for SDB's. The two classes of approaches correspond to refuse-to-answers and lies, respectively. After perturbing data, some queries may change their value; the error introduced by the perturbation is called "bias". Bias can be considered acceptable if it can be controlled; the DB administrator should be able to choose the right tradeoff between precision and security.

The main difference between data perturbation and our approach is in their granularity: first, in SDB's confidential attributes cannot be directly inspected (only global functions of the attributes are available), and second, in our approach, the user has direct access to the database, including confidential attributes; they can be given *partial* protection, e.g. it is possible to make the jobs of some individuals available, while "hiding" the job of some other individuals at the same time.

8 Conclusions

During recent years, a great deal of work has been done on enforcing security in traditional databases [13, 12, 6, 7, 18, 30, 29]. A large part of this research has dealt with *multilevel* security where, intuitively, different parts of the database are assigned security "level" and each user is allowed access to certain levels only. Work on developing a formal theoretical foundation for secure databases has developed more slowly. Bancilhon and Spyratos [2] wrote an important paper identifying the key questions that must be asked when attempting to develop a foundation for secure databases. These questions are broad enough to cover not only the traditional relational databases, but also deductive Horn databases, as well as more general models. Sicherman et. al. [26] subsequently extended the work of Bancilhon and Spyratos. Recently, several formalizations of security, based on modal logics, have been explored [5, 10, 16]. The key differences between our work and [2, 26, 5, 10, 16] is that:

- In our framework, we never refuse to answer queries. As shown in the Secret Service example, refusing to answer a query may often allow the user, via meta-reasoning, to violate secrecy.
- We allow the database to provide incorrect answers when this is the only way to preserve security.
- An important distinction that must be made is between the user's *beliefs* (which may be wrong) and his/her knowledge (knowledge is defined, usually, to be "true belief" in philosophy). In other words, our work deals with the user's beliefs, rather than with his knowledge (as in [2, 26, 5, 10, 16]). The reason is that we do not wish the user to uncover a secret by using his/her incorrect beliefs.

The key contributions of this paper are:

- The development of a formal logical foundation of security, and the consequent splitting of the logic into the *logic of secrecy* which is static, and the *logic of interaction*, which describes the database-user interactions over a period of time. An important aspect of our semantics is that the degree of intractability of query processing is not increased to a new complexity level (i.e. it is only NP-complete, as opposed to being either PSPACE-complete or EXPSPACE-complete or EXPTIME-complete).
- The development of an algorithm for answering queries to databases using yn-dialogs, and the proof that secrets may be preserved indefinitely as long as the user did not initially believe a disjunction of secrets.
- The study, for the first time, of some of the issues of complexity in secure deductive databases.

A number of the ideas explored in this paper have been suggested, but not explored, in [2, 26]. These include the possibility of lying, and also the concept of meta-security which is called a "secrecy" in [26].

References

[1] N.R. Adam and J. C. Wortmann. (1989) *Security-Control Methods for Statistical Databases: A Comparative Study*, ACM Computing Surveys, 21, 4, pps 515–556.

[2] F. Bancilhon and N. Spyratos. (1977) *Protection of Information in Relational Data Bases*, Proc. Intl. Symp. on Very Large Data Bases, Tokyo, Japan, pps 494–500.

[3] F. Bancilhon, D. Maier, Y. Sagiv, and J.D. Ullman. (1986) *Magic Sets and Other Strange Ways to Implement Logic Programs*, Proc. 1986 ACM Symp. on Principles of Database Systems, pps 1–15.

[4] C. Bell, A. Nerode, R. Ng and V.S. Subrahmanian. (1992) *Implementing Deductive Databases by Linear Programming*, Proc. 1992 ACM Symp. on Principles of Database Systems.

[5] P. Bieber and F. Cuppens. (1991) *A Definition of Secure Dependencies using the Logic of Security*, Proc. of the Computer Security Foundations Workshop IV, IEEE Computer Society Press.

[6] J. Biskup and H. H. Bruggemann. (1988) *The Personal Model of Data: Towards a Privacy-Oriented Information System*, Computers and Security, 7, pps 575–597.

[7] J. Biskup and H.-W. Graf. (1988) *Analysis of the Privacy Model for the Information System Doris*, Proc. Workshop on Database Security, Ontario, Canada.

[8] P. Bonatti, S Kraus and V.S. Subrahmanian, (1992). *Declarative Foundations of Secure Deductive Databases*, University of Maryland Tech. Report, UMIACS TR 92-73 CS TR 2922.

[9] B. F. Chellas. (1980) *Modal Logic: an introduction*, Cambridge University Press, Cambridge.

[10] F. Cuppens. (1992) *A Modal Logic Framework to Solve Aggregation Problems*, S. Jajodia and C. Landwehr (eds.) *Database Security, 5: Status and Prospects.* North Holland.

[11] D.E. Denning. (1982) *Cryptography and Data Security*, Addison-Wesley.

[12] D.E. Denning and M. Morgenstern. (1986) *Military Database Technology Study: AI Techniques for Security and Reliability*, SRI International Tech. Report – Project 1644.

[13] D. Denning, T.F. Lunt, R.R. Schell, M. Heckman and W. Shockley. (1987) *A Multilevel Relational Data Model*, Proc. IEEE Symp. on Security and Privacy, Oakland, CA, pps 46–56.

[14] W.F. Dowling and J. Gallier. (1984) *Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae*, J. of Logic Programming, 1, 3, pps 267–284.

[15] M.R. Garey and D.S. Johnson. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[16] J. Gray and P. Syverson. *A Logical Approach to Multilevel Security of Probabilistic Systems*, Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 1992.

[17] F. Hillier and G. Lieberman. (1974) *Operations Research*, Holden-Day.

[18] S. Jajodia and R. Sandhu. (1990) *Polyinstantiation Integrity in Multilevel Relations* Proc. IEEE Symp. on Research in Security and Privacy, Oakland, CA, pps 104–115.

[19] S. Kraus, D. Lehmann. (1988) *Knowledge, Belief and Time*, Theoretical Computer Science, 58, pps 155–174.

[20] S. Kraus, D. Lehmann, M. Magidor. (1990) *Nonmonotonic Reasoning, Preferential Models and Cumulative Logics*, Artificial Intelligence, 44, pps 167–207.

[21] J.W. Lloyd. (1987) *Foundations of Logic Programming*, Springer.

[22] A. Nerode, W. Marek and V.S. Subrahmanian. (1991) *Logic Programming and Non-Monotonic Reasoning: Proceedings of the First International Workshop*, MIT Press.

[23] A. Nerode, R. Ng and V.S. Subrahmanian. (1992) *Computing Circumscriptive Databases*, to appear in: Information and Computation.

[24] D. Sacca and C. Zaniolo. (1986) *On the Implementation of a Simple Class of Logic Queries for Databases*, Proc. 1986 ACM Symp. on Principles of Database Systems, pps 16–23.

[25] J. Shoenfield. (1967) *Mathematical Logic*, Addison Wesley.

[26] G. Sicherman, W. de Jonge and R.P. van de Riet. (1983) *Answering Queries without Revealing Secrets*, ACM Transactions on Database Systems, 8, 1, pps 41–49.

[27] T.-A. Su and G. Ozsoyoglu. (1991) *Controlling FD and MVD Inferences in Multilevel Relational Database Systems*, IEEE Transactions on Knowledge and Data Engineering, 3, 4, pps 474-485.

[28] M. Vardi. *The Complexity of Relational Query Languages*, Proc. 14th ACM Symp. on Theory of Computing, San Francisco, 1982, pp. 137–146.

[29] I. Wilson. (1988) *Views as Security Objects in Multilevel Secure Relational DBMS* Proc. IEEE Symp. on Security and Privacy, Oakland, CA, pps 70–84.

[30] S. R. Wiseman. (1990) *Control of Confidentiality in Databases*, Computers and Security Journal, 9, 6, pps 529–537.

[31] S. Wiseman. (1991) *Lies, Dammed Lies and Database*, Royal Signals and Radar Establishment, memo 4503, England.