Confidentiality Policies for Controlled Query Evaluation

Joachim Biskup and Torben Weibert*

Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, Germany
{biskup,weibert}@ls6.cs.uni-dortmund.de

Abstract. Controlled Query Evaluation (CQE) is an approach to enforcing confidentiality in information systems at runtime. At each query, a censor checks whether the answer to that query would enable the user to infer any information he is not allowed to know according to some specified confidentiality policy. If this is the case, the answer is distorted, either by refusing to answer or by returning a modified answer. In this paper, we consider incomplete logic databases and investigate the semantic ways of protecting a piece of information. We give a formal definition of such confidentiality policies, and show how to enforce them by reusing the existing methods for CQE.

Keywords: Inference control, confidentiality policies, logic databases.

1 Introduction

Security in information systems aims at various goals, one of which is preservation of *confidentiality*: Certain information may only be disclosed to a certain subgroup of users. This is of particular importance when an information system contains both classified and public data, and is accessed by multiple users at the same time. Confidentiality can be achieved by various methods, which can be divided into two categories: *access control*, which us usually implemented by static access rights, and *information flow control*, which is often applied dynamically at query time. The latter addresses the *inference problem*: A user might combine multiple pieces of (public) information in order to infer secret information. The inference problem has been studied in a various contexts, for example statistical (see [1,2,3] for an introduction and e. g. [4,5,6] for more recent work), multi-level and relational databases (see e. g. [7,1,8,9,10,11]). See [12] for a comprehensive review of the respective approaches.

Controlled Query Evaluation (CQE) is a dynamic approach for information flow control in logic databases, which are either *complete* (i. e., they can provide an answer to each query) or *incomplete* (i. e., part of the information is missing, and some queries cannot be answered). The administrator defines a *confidentiality policy*, specifying the information to be kept secret. At runtime, before an

* This author is funded by the German Research Foundation (DFG) under Grant No. BI-311/12-1.

S. Barker, G.-J. Ahn (Eds.): Data and Applications Security 2007, LNCS 4602, pp. 1–13, 2007.

answer to a query is returned to the user, it is passed to a *censor* which investigates possible security risks. In order to identify these risks, a *log file* of past queries and answers is maintained. In case the answer would reveal any secret information (either directly or combined with previous answers), the answer is *distorted*, either by *lying* (giving a "false" answer) or by *refusal* (returning no "useful" answer at all). CQE was first proposed by Sicherman et at. [13] and Bonatti et al. [14]. A unified framework for *complete* databases was introduced by Biskup [15] and later exploited by Biskup/Bonatti to investigate CQE under various parameters [16,17,18,19]. Some of these parameters have also been investigated for incomplete databases [20,21]. This paper extends the work on incomplete databases, filling some of the gaps.

A database instance db is a consistent set of sentences of some logic (in this paper, propositional logic); a closed (yes-no) query Φ is a single sentence of that logic, and its value in a database instance db is either *true*, *false* or *undef*. A *potential secret* is a sentence Ψ. In case Ψ is *true* in db, the user may not infer this fact; otherwise, if Ψ is either *false* or *undef*, this fact may be disclosed. Thus, a potential secret protects the fact *that* some information is *true*.

For complete information systems, another type of confidentiality policies has been studied: *secrecies*. A secrecy is a pair of complementary sentences $(\Psi, \neg\Psi)$. CQE will conceal *whether* Ψ or $\neg\Psi$ holds in the database; as opposed to potential secrets, the negation is protected as well. As discussed in [18], secrecies can be protected by discretely designed enforcement methods, or by "naively" reducing them into a set of potential secrets $\{\Psi, \neg\Psi\}$, and then reusing the existing enforcement methods for potential secrets.

In this paper, we investigate whether the concept of secrecies can be adopted for incomplete information systems as well. As it turns out, incomplete information systems offer many different semantic ways of protecting a sentence Ψ. For example, it is possible to protect the partial information that "Ψ is either *true* or *false*, but not *undef*"; or one might want to keep the user from inferring *any* information about the actual value of Ψ. We show how these "generalized" confidentiality targets can be formalized, and how they can be operationally enforced by reduction to existing techniques.

In Section 2, we recall CQE for incomplete databases and potential secrets, as found in [21]. We also present an example enforcement method which can later be used for the reduction. Section 3 discusses the various ways of protecting secret information under incomplete databases, and gives a formal definition of generalized confidentiality policies. In Section 4, we demonstrate the reduction to potential secrets, and discuss the requirements for the underlying enforcement method. We finally conclude in Section 5.

2 Controlled Query Evaluation for Potential Secrets

In this section, we summarize the CQE framework for potential secrets from [21]. We first specify the abstract framework and its declarative notion of confidentiality, and then present an instantiation thereof, the *combined lying and refusal method*.

2.1 Declarative Framework

We consider (possibly) incomplete logic databases, based on propositional logic.

Definition 1. *A database schema DS is a finite set of propositions. The propositional language over DS is denoted by $\mathcal{P}_{DS}$. A database instance over the schema DS is a consistent set $db \subset \mathcal{P}_{DS}$ of propositional sentences. A query $\Phi \in \mathcal{P}_{DS}$ is a propositional sentence. The result of a query Φ in a database instance db is determined by the function*

$$eval(\Phi)(db) := \begin{cases} true & \text{if } db \models_{PL} \Phi \\ false & \text{if } db \models_{PL} \neg\Phi \\ undef & \text{otherwise} \end{cases} \quad (1)$$

(where $\models_{PL}$ denotes logical implication in propositional logic). We assume that the user does not issue a single query but a sequence of queries $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$. (The framework might be extended to a fragment of first-order logic, given that logical implication is decidable in that fragment; see [16] for a discussion.)

In previous work, CQE for incomplete databases has been studied for *potential secrets*.

Definition 2. *A confidentiality policy based on potential secrets is a set $pot\_sec = \{\Psi_1, \ldots, \Psi_m\}$ of propositional sentences, each of which we call a potential secret. The semantics of the confidentiality policy is as follows: In case Ψ_i is true in the actual database instance db, the user is not allowed to infer this information. On the other hand, if Ψ_i is either false or undef in db, this information may be disclosed.*

Potential secrets are a suitable formalization for real-life situations where the circumstance *that* a certain fact is true must be kept secret, but not the converse.

Example 3. Imagine a person applying for an employment. If that person suffers from a terminal disease, this fact must be kept secret. On the other hand, if the applicant is healthy, this information may be disclosed. The sentence "person X suffers from a terminal disease" can be formalized as a potential secret Ψ.

The confidentiality policy is declared independently from the actual database instance *db*, and may contain both potential secrets that are true in *db*, and potential secrets that are not true in *db*. This is important as we assume that the user *knows* the set of potential secrets (but of course not their respective values in *db*). CQE enforces the confidentiality policy by iteratively examining each query and the inferences the user could draw from the respective answer. In case confidentiality is threatened, a modified answer is given, in one of two possible ways:

1. *Lying*: An answer different from the actual query value is returned, for example *false* instead of *true*.
2. *Refusal*: Instead of the actual query value, the special answer *refuse* is returned.

CQE also accounts for any information known or assumed by the user prior to the first query, for example general knowledge or publicly known semantic constraints. These *a priori assumptions* are formalized as a set *prior* of propositional sentences.

All things considered, a CQE method for potential secrets can be formalized as a function $cqe(Q, db, prior, pot\_sec) := \langle ans_1, \ldots, ans_n \rangle$, where $Q = \langle \Phi_1, ..., \Phi_n \rangle$ is a query sequence, *prior* are the a priori assumptions, *db* is a database instance, and *pot_sec* is a set of potential secrets. The output is a sequence of answers ans_i. Each enforcement method *cqe* goes along with a function *precondition* that defines the admissible arguments $(prior, db, pot\_sec)$ for that method. In particular, *precondition* makes sure that *prior* does not imply any potential secret in the first place. The system will reject to start a session unless *precondition* is satisfied.

Definition 4. *Let cqe be a CQE method for potential secrets, with precondition defining the admissible arguments. cqe is defined to preserve confidentiality iff*

for all finite query sequences Q*,*
for all confidentiality policies pot_sec,
for all potential secrets $\Psi \in pot\_sec$*,*
for all a priori assumptions prior,
for all instances db_1 *so that* $precondition(db_1, prior, pot\_sec)$ *holds*
there exists an instance db_2
so that $precondition(db_2, prior, pot\_sec)$ *holds and*
(a) *[*db_1 *and* db_2 *produce the same answers]*
$cqe(Q, db_1, prior, pot\_sec) = cqe(Q, db_2, prior, pot\_sec)$
(b) *[*Ψ *is not true in* db_2*]*
$eval(\Psi)(db_2) \in \{false, undef\}$.

Condition (b) ensures that there is an instance db_2 in which Ψ is not *true*. Condition (a) guarantees that db_1 and db_2 produce the same answers; the user cannot distinguish db_1 from db_2, and thus cannot rule out that Ψ is actually *false* or *undef*. This confidentiality definition is purely declarative. In the following, we show how to operationally meet these requirements by keeping a log file of sentences in epistemic logic.

2.2 An Enforcement Method with Lying and Refusal

We outline the *combined lying and refusal* approach from [21]. In order to account for the information disclosed by previous answers, the system keeps a *log file* log_i as a set of sentences in *epistemic logic*. This logic, also known as S5 modal logic, is established by introducing the modal operator K which we read as "the database

knows that…". The resulting language, based on a set DS of propositions, is denoted by $\mathcal{L}_{DS}$. We use the common Kripke-style semantics, to be found e. g. in [22]: An $\mathcal{M}_{DS}$-structure is a triple $M = (S, \mathcal{K}, \pi)$, where S is a set of states, $\mathcal{K}$ a binary equivalence relation on S, and $\pi : S \times DS \to \{true, false\}$ assigns a truth value to each proposition from DS under each state $s \in S$. The semantics of the K operator is defined by

$$(M, s) \models K\Phi \text{ iff } (M, s') \models \Phi \text{ for all } s' \text{ such that } (s, s') \in \mathcal{K} \tag{2}$$

(where $\models$ is the ordinary model-of operator). A sentence ϕ is logically implied by a set of sentences Σ wrt. $\mathcal{M}_{DS}$ (in formulae: $\Sigma \models_{S5} \phi$) iff for every $\mathcal{M}_{DS}$-structure $M = (S, \mathcal{K}, \pi)$ and every state $s \in S$ it holds that if $(M, s) \models \Sigma$ then $(M, s) \models \phi$.

A propositional sentence ϕ and a truth value $v \in \{true, false, undef\}$ can be converted into an appropriate epistemic sentence by the function Δ with

$$\Delta(\phi, true) = K\phi,$$
$$\Delta(\phi, false) = K\neg\phi,$$
$$\Delta(\phi, undef) = \neg K\phi \wedge \neg K\neg\phi.$$

Furthermore, we define the function

$$\Delta^*(\phi, V) := \bigvee_{v \in V} \Delta(\phi, v)$$

that converts a sentence ϕ and a non-empty set of values $\emptyset \neq V \subseteq \{true, false, undef\}$ into an epistemic sentence by disjunctively connecting the single sentences $\Delta(\phi, v)$ for each $v \in V$. The set V is also called an *inference set*, as it is used to formalize (disjunctive) information about a query value. For example, $V = \{true, undef\}$ means "the query value is either *true* or *undef*, but not *false*". A unary inference set represents definitive information (exactly one value appears possible), a binary inference set disjunctive information (two values appear possible, one does not), and the inference set $\{true, false, undef\}$ represents no information (any value appears possible). In particular, note that $\Delta^*(\phi, \{true, false, undef\}) = K\phi \vee K\neg\phi \vee \neg K\phi \wedge \neg K\neg\phi$ is a tautology.

Prior to the first query, the log file is initialized with the a priori assumptions: $log_0 := prior$. Later, after each query Φ_i, log_i is established by translating the information disclosed by the i-th answer ans_i into an epistemic sentence, and adding it to log_{i-1}. In case of a regular answer $ans_i \in \{true, false, undef\}$ (being a lie or not), the translation $\Delta^*(\Phi_i, \{ans_i\})$ of the definite inference set $\{ans_i\}$ is added to the log file. In case the answer was refused ($ans_i = refuse$), it is assumed to provide no information to the user, so the tautology $\Delta^*(\Phi_i, \{true, false, undef\})$ is added.

Having formalized the previous knowledge as epistemic sentences, we can employ logical implication in order to detect confidentiality violations: A potential secret $\Psi \in pot\_sec$ is considered disclosed if it is logically implied by the

log file log_i. The goal is to prevent these violations throughout the query sequence. We formalize the combined lying and refusal approach as a function $cqe_{combined}(Q, db, prior, pot\_sec)$ with the precondition

$$precondition_{combined}(db, prior, pot\_sec) \;:=\; (\forall \Psi \in pot\_sec)[\; prior \not\models_{S5} \Psi \;],$$

which prevents that the a priori assumptions already lead to a violation. After each query Φ_i, the returned answer ans_i and the internal log file log_i are generated as follows:

1. Determine the *security configuration*, i. e., the set of definitive inferences that would lead to the disclosure of at least one potential secret:

$$C_i := \big\{\; V \in \{\{true\}, \{false\}, \{undef\}\} \mid (\exists \Psi \in pot\_sec)\; [\; log_{i-1} \cup \{\Delta^*(\Phi_i, V)\} \models_{S5} \Psi \;] \;\big\} \quad (3)$$

2. Use a *censor function* to choose the answer $ans_i \in \{true, false, undef, refuse\}$ to return, according to the security configuration C_i and the actual query value $eval(\Phi_i)(db)$. The censor function must meet certain requirements; in particular, it must make sure that $\{ans_i\} \notin C_i$. An example of an appropriate censor function is given in Table 1. Black cells indicate a modified answer.

Table 1. Censor function for the combined lying and refusal method

Security Configuration C	$eval(\Phi)(db) = ...$ *true*	*false*	*undef*
$\{\{true\}, \{false\}, \{undef\}\}$	*refuse*	*refuse*	*refuse*
$\{\{true\}, \{false\}\}$	*undef*	*undef*	*undef*
$\{\{true\}, \{undef\}\}$	*false*	*false*	*false*
$\{\{false\}, \{undef\}\}$	*true*	*true*	*true*
$\{\{true\}\}$	*undef*	*false*	*undef*
$\{\{false\}\}$	*true*	*undef*	*undef*
$\{\{undef\}\}$	*true*	*false*	*false*
$\emptyset$	*true*	*false*	*undef*

3. Update the log file by adding the answer translated into an epistemic sentence:

$$log_i := \begin{cases} log_{i-1} \cup \{\Delta^*(\Phi, \{true, false, undef\})\} & \text{if } ans_i = refuse \\ log_{i-1} \cup \{\Delta^*(\Phi, \{ans_i\})\} & \text{otherwise} \end{cases}$$

Theorem 5. *$cqe_{combined}$ preserves confidentiality according to Definition 4.*

The full proof can be found in [21], but we give a short sketch here.

First, it can be shown by induction that $log_i \not\models_{S5} \Psi$ holds for all $\Psi \in pot\_sec$ and all $1 \le i \le n$, in particular for the final log file log_n. Thus, given a potential

secret $\Psi \in pot\_sec$, there must be an $\mathcal{M}_{DS}$-structure $M = (S, \mathcal{K}, \pi)$ and a state $s \in S$ such that

$$(M, s) \models log_n \text{ but } (M, s) \not\models \Psi.$$

An alternative instance db_2 can be constructed from (M, s) by

$$db_2 := \{\ \alpha \mid \alpha \text{ is a propositional sentence and } (M, s) \models K\alpha\ \}. \tag{4}$$

db_2 is consistent, so it is a valid database instance, and it is also closed under logical implication.

As $(M, s) \not\models \Psi$, we conclude that Ψ cannot be *true* in db_2. Finally, it can be shown that the same answers are generated under both db_2 and the original instance db_1.

3 Generalized Confidentiality Policies

Controlled Query Evaluation for incomplete databases, as summarized in Section 2, protects a set of potential secrets; for each potential secret, the user may not infer that this secret is true in the actual database instance. Example 3 demonstrates that potential secrets have a useful semantics in many situations. However, there are situations in which a sentence must be protected in a different semantic way.

Example 6. For the sake of sexual equality, an applicant's gender may not have an influence on whether he or she is chosen for a particular job. Hence, a person querying a database containing applicants' data may not infer that a given person is male, and neither that this person is *not* male.

Example 7. Although being supposed to do housework, Jim secretly goes to watch his favorite team's soccer match. Talking to his wife later, Jim must keep secret whether his team won or or not. Furthermore, his wife must not even learn that Jim knows whether his team won or not, as this would disclose the fact that he went to the match.

A *confidentiality target* consists of two parts: The sentence that is to be protected, and the set of truth values the user is not allowed to infer. The latter part can be definite ("the team has won") or partial ("the team has won or has not won"[1]).

Definition 8. *A* confidentiality target *is a pair* (ψ, V)*, where* ψ *is a propositional sentence, and* $V_i \subset \{true, false, undef\}$ *with* $\emptyset \neq V_i \neq \{true, false, undef\}$ *is a non-empty inference set.*[2] *A* (generalized) confidentiality policy *is a set* $policy = \{(\psi_1, V_1), \ldots, (\psi_m, V_m)\}$ *of confidentiality targets.*

[1] This is not a tautology due to the remaining alternative "he does not know whether the team won".

[2] It is reasonable to prohibit $\{true, false, undef\}$, as this is a tautology and can never be protected. The empty set does not make sense either, as at least one value needs to be protected.

Example 9. Given two propositional sentences a and b, the confidentiality policy given by $policy = \{ (a, \{true, undef\}), (b, \{false\}) \}$ declares that (1) the user may not infer that a is either *true* or *undef*, and that (2) the user may not infer that b is *false*.

Example 10. Consider the propositional sentence a and the confidentiality policy given by $policy = \{ (a, \{true, false\}), (a, \{true, undef\}), (a, \{false, undef\}) \}$. Obviously, any disjunctive information is considered harmful, so the user may not learn any information about the value of a at all. This corresponds to the concept of *secrecies* investigated in the context of complete databases [15,18], where the user may not learn the exact value of some sentence.

We can formalize a CQE method for generalized confidentiality policies as a function

$$cqe^*(Q, db, prior, policy) := \langle ans_1, \ldots, ans_n \rangle,$$

where $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ is the query sequence, *prior* is the set of a priori assumptions, *db* is the database instance and *policy* is the confidentiality policy. Each method goes along with a function $precondition^*$ that defines the admissible arguments.

Definition 11. *Let cqe^* be a CQE method for generalized confidentiality policies with $precondition^*$ as its associated precondition for admissible arguments. cqe^* is defined to preserve confidentiality iff*

for all finite query sequences Q,
for all generalized confidentiality policies policy,
for all confidentiality targets $(\psi, V) \in policy$,
for all a priori assumptions prior,
for all instances db_1 so that $precondition^*(db_1, prior, pot\_sec)$ holds
there exists an instance db_2
so that $precondition^*(db_2, prior, pot\_sec)$ holds and

(a) [db_1 and db_2 produce the same answers]
$cqe^*(Q, db_1, prior, policy) = cqe^*(Q, db_2, prior, policy)$
(b) [ψ has a "permitted" value in db_2]
$eval(\psi)(db_2) \notin V$.

Again, this definition is purely declarative. In the following section, we will show how to operationally meet these requirements, reusing existing techniques.

One advantage of the new concept of confidentiality targets is that they have a very simple syntax, which allows easy declaration of confidentiality policies. However, we run into problems when we want to design an operational enforcement method for this kind of confidentiality policies – there is no logical implication operator defined for this language, and of course there are no proof systems available that could be used in an implementation.

In the following section, we will present a solution to this problem: Each confidentiality target can be converted into a single sentence of modal epistemic

logic. These sentences can then be regarded as (epistemic) potential secrets, and we can reuse the existing methods for potential secrets in order to enforce the converted confidentiality policy.

4 Enforcement by Reduction

In the previous section, we gave a declarative definition of confidentiality wrt. generalized policies. We will now show how to enforce these policies by reusing the methods established for potential secrets. The idea is to convert the generalized confidentiality policy into a set of potential secrets. As facts like "the value is either *true* or *false*" or "the value is *undef*" need to be protected, it is necessary to use an epistemic representation of the confidentiality targets.

Definition 12. *Let policy* $= \{(\psi_1, V_1), \ldots, (\psi_n, V_m)\}$ *be a (generalized) confidentiality policy. policy can be converted into a set of (epistemic) potential secrets by the function*

$$pot\_sec(policy) := \{\Delta^*(\psi_1, V_1), \ldots, \Delta^*(\psi_m, V_m)\} \tag{5}$$

where Δ^* *is the conversion function defined in Section 2.2.*

Remember that, according to Definition 8, the value set V_i of each confidentiality target $(\psi_i, V_i) \in policy$ is either unary or binary. Thus, all sentences in $pot\_sec(policy)$ have one of the following six syntactic forms, where ψ is a propositional sentence:

(1) $K\psi$,
(2) $K\neg\psi$,
(3) $\neg K\psi \wedge \neg K\neg\psi$,
(4) $K\psi \ \vee \ K\neg\psi$,
(5) $K\psi \ \vee \ \neg K\psi \wedge \neg K\neg\psi$,
(6) $K\neg\psi \ \vee \ \neg K\psi \wedge \neg K\neg\psi$.

Example 13. The confidentiality policy given by

$$policy \ = \ \{ \ (a, \{true, undef\}), \ (b, \{false\}) \ \}$$

is converted into

$$pot\_sec(policy) = \ \{ \ Ka \quad \vee \quad \neg Ka \wedge \neg K\neg a, \quad K\neg b \ \}.$$

The remaining problem is that the CQE methods for potential secrets, as defined in Section 2, only allow *pot_sec* to contain propositional sentences. However, as the epistemic language is a superset of the propositional language, some enforcement methods might also work for epistemic potential secrets. Given a "useful" behavior, these methods would then be exploitable for the conversion of confidentiality targets. We will first give a formal definition of these two requirements – suitable for epistemic potential secrets, and "useful" behavior – and then prove that $cqe_{combined}$ satisfies these properties.

Definition 14. *An enforcement method cqe wrt. potential secrets is* adapted for epistemic potential secrets *iff*

1. *the specific algorithm of cqe accepts a set of* epistemic sentences *(instead of propositional sentences) to be passed as the pot_sec input parameter, and*
2. *given a propositional sentence ψ and an epistemic potential secret Ψ associated with ψ, there exists an alternative database instance db_2 as demanded by Definition 4 that has the following properties wrt. the value of ψ:*

epistemic potential secret Ψ	*value of* ψ *in* db_2
$\Delta^*(\psi, \{true\}) = K\psi$	$eval(\psi)(db_2) \in \{false, undef\}$
$\Delta^*(\psi, \{false\}) = K\neg\psi$	$eval(\psi)(db_2) \in \{true, undef\}$
$\Delta^*(\psi, \{undef\}) = \neg K\psi \wedge \neg K\neg\psi$	$eval(\psi)(db_2) \in \{true, false\}$
$\Delta^*(\psi, \{true, false\}) = K\psi \ \vee \ K\neg\psi$	$eval(\psi)(db_2) = undef$
$\Delta^*(\psi, \{true, undef\}) = K\psi \ \vee \ \neg K\psi \wedge \neg K\neg\psi$	$eval(\psi)(db_2) = false$
$\Delta^*(\psi, \{false, undef\}) = K\neg\psi \ \vee \ \neg K\psi \wedge \neg K\neg\psi$	$eval(\psi)(db_2) = true$

The latter condition corresponds to condition (b) of Definition 11: Given a potential secret $\Psi = \Delta^*(\psi, V)$, there exists an indistinguishable instance db_2 with $eval(\psi)(db_2) \notin V$.

Lemma 15. *The combined lying and refusal method $cqe_{combined}$ presented in Section 2 is adapted for epistemic potential secrets.*

Proof. Condition 1: $cqe_{combined}$ only considers the potential secrets when determining the security configuration C_i (3). The implication operator $\models_{S5}$ employed allows epistemic sentences on its right hand side.

Condition 2: Consider the construction of db_2 (4) in the proof sketch of Theorem 5. Let ψ be a propositional sentence. We investigate the six ways to construct an epistemic potential secret Ψ from ψ, according to Definition 14.

Let $M = (S, \mathcal{K}, \pi)$ an $\mathcal{M}_{DS}$-structure and $s \in S$ a state such that $(M, s) \models log_n$ but $(M, s) \not\models \Psi$.

Case 1 ($\Psi = K\psi$).

Then we have $(M, s) \not\models K\psi$ and, according to (4), $\psi \notin db_2$. As db_2 is closed under logical implication, we conclude that $db_2 \not\models_{PL} \psi$ and thereby $eval(\psi)(db_2) \in \{false, undef\}$.

Case 2 ($\Psi = K\neg\psi$).

Similar to Case 1 (consider $\psi' = \neg\psi$).

Case 3 ($\Psi = \neg K\psi \wedge \neg K\neg\psi$).

Then we have $(M, s) \not\models \neg K\psi \wedge \neg K\neg\psi$, which means that $(M, s) \models K\psi \vee K\neg\psi$ and thus either $(M, s) \models K\psi$ or $(M, s) \models K\neg\psi$. Hence, it holds that either $\psi \in db_2$ or $\neg\psi \in db_2$. By the closure of db_2 and the definition of the *eval* function (1), we then have $eval(\psi)(db_2) \in \{true, false\}$.

Case 4 ($\Psi = K\psi \ \vee \ K\neg\psi$).

Then it holds that $(M, s) \not\models K\psi \vee K\neg\psi$, which means that $(M, s) \not\models K\psi$ and $(M, s) \not\models K\neg\psi$. By the results from Case 1 and Case 2, we then have $eval(\psi)(db_2) \in \{false, undef\}$ and $eval(\psi)(db_2) \in \{true, undef\}$, so it must hold that $eval(\psi)(db_2) = undef$.

Case 5 ($\Psi = K\psi \ \vee \ \neg K\psi \wedge K\neg\psi$).
Accordingly.
Case 6 ($\Psi = \neg K\psi \ \vee \ \neg K\psi \wedge K\neg\psi$).
Accordingly.

Given an enforcement method adapted for epistemic potential secrets, we can employ the reduction outlined above. The confidentiality targets are converted into potential secrets and then passed to the underlying enforcement method. The established enforcement method then satisfies our notion of confidentiality.

Theorem 16. *Let cqe be an enforcement method for potential secrets, preserving confidentiality (Definition 4) and being adapted for epistemic potential secrets (Definition 14). Let precondition be the associated precondition. Then the enforcement method for confidentiality targets given by the function*

$$cqe^*(Q, db, prior, policy) := cqe(Q, db, prior, pot\_sec(policy))$$

with the precondition

$$precondition^*(db, prior, policy) := precondition(db, prior, pot\_sec(policy))$$

preserves confidentiality in the sense of Definition 11.

Proof. Let db_1 be a database instance, *policy* a generalized confidentiality policy, *prior* the a priori assumptions so that the pertinent $precondition(db_1, log_0, pot\_sec(policy))$ is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence.

Let $(\psi, V) \in policy$ be a confidentiality target. By the definition of the *pot_sec* function (5), $pot\_sec(policy)$ contains the potential secret $\Psi = \Delta^*(\psi, V)$. By condition (2) of Definition 14, there exists a database instance db_2 under which the same answers and log files are generated as under db_1, and with $eval(\psi)(db_2) \notin V$. This satisfies conditions (a) and (b) of Definition 11.

5 Conclusion

In incomplete information systems, a sentence can be protected in various semantic ways. In particular, definite and partial information about a truth value of some sentence can be protected, or a combination thereof. We specified a generalized framework for expressing such a confidentiality target as a pair of a propositional sentence and a set of "forbidden" query values. We then gave a formal definition of confidentiality which resembles the respective notions for both potential secrets and secrecies under complete information systems and potential secrets for incomplete information systems. Instead of designing specific dedicated enforcement methods for each semantic type of confidentiality target, we picked up the idea of *naive reduction* [18] and showed how to convert a generalized confidentiality policy into a set of epistemic potential secrets, which can be used as the input to the existing enforcement methods for potential secrets. As

these existing methods are originally designed for propositional potential secrets, we had to prove that the methods are *adapted for epistemic potential secrets*.

Alternatively, a security administrator may decide to specify the confidentiality policy as a set of epistemic potential secrets in the first place, given that these epistemic sentences have one of the six syntactical forms given in Definition 14. We however believe that it is favorable to specify the confidentiality policy with the means of confidentiality targets, for the sake of easier administration.

Although confidentiality targets provide a higher expressiveness than ordinary (propositional) potential secrets, there are still some limitations, in particular when you want to protect information about two different propositional sentences at the same time. For example, the information "a is *true* and b is (at the same time) *undef*" cannot be formalized as a confidentiality target (as there is no conjunction operator for confidentiality targets, and also no disjunction or negation). It is however easy to express this information as an epistemic sentence: $Ka \wedge \neg Kb \wedge \neg K\neg b$. In order to protect such a sentence, we would need to extend our framework such that it can handle a wider variety of epistemic sentences as potential secrets (essentially those in which any propositional sub-formula is prefixed by K). This will be the topic of future work.

A prototype implementation of the work presented in this paper is available from [23]. With these results, six of the twelve scenarios for complete information systems (resulting from the three parameters: potential secrets/secrecies, known/unknown policy, lying/refusal/combined lying and refusal) have been translated to incomplete databases. Current work includes the investigation of *unknown* policies, and how to exploit the situation when the user does not know which sentences are protected. The results for complete databases [19] suggest that less answers need to be distorted then.

At the moment, our work is limited to closed (yes/no-)queries and propositional logic. A useful application might be, for example, trust negotiation [24], a technique to establish trust between two agents by subsequently presenting credentials to each other. CQE could assist the agents to protect sensitive information while exchanging their credentials. This will be covered by future work. Considerations about open queries and first-order logic can be found in [16]; we are currently working on an implementation that will act as an interface layer to the Oracle DBMS. We also plan to investigate how to handle updates to the database instance, and how to deal with the situation when the log file contains information that has become obsolete due to a modified instance.

References

1. Castano, S., Fugini, M., Martella, G., Samarati, P.: Database Security. ACM Press, New York (1995)
2. Denning, D.: Cryptography and Data Security. Addison-Wesley, London, UK (1982)
3. Leiss, E.L.: Principles of Data Security. Plenum Press, New York (1982)
4. Domingo-Ferrer, J. (ed.): Inference Control in Statistical Databases. In: Domingo-Ferrer, J. (ed.) Inference Control in Statistical Databases. LNCS, vol. 2316, Springer, Heidelberg (2002)

5. Wang, L., Jajodia, S., Wijesekera, D.: Securing OLAP data cubes against privacy breaches. In: IEEE Symposium on Security and Privacy, pp. 161–178. IEEE Computer Society, Los Alamitos (2004)
6. Wang, L., Li, Y., Wijesekera, D., Jajodia, S.: Precisely answering multi-dimensional range queries without privacy breaches. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, Springer, Heidelberg (2003)
7. Brodsky, A., Farkas, C., Jajodia, S.: Secure databases: Constraints, inference channels, and monitoring disclosures. IEEE Transactions on Knowledge and Data Engineering 12(6), 900–919 (2000)
8. Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M., Shockley, W.R.: The seaview security model. IEEE Transactions on Software Engineering 16(6), 593–607 (1990)
9. Qian, X., Lunt, T.F.: A semantic framework of the multilevel secure relational model. IEEE Transactions on Knowledge and Data Engineering 9(2), 292–301 (1997)
10. Staddon, J.: Dynamic inference control. In: 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 94–100 (2003)
11. Winslett, M., Smith, K., Qian, X.: Formal query languages for secure relational databases. ACM Transactions on Database Systems 19(4), 626–662 (1994)
12. Farkas, C., Jajodia, S.: The inference problem: A survey. SIGKDD Explorations 4(2), 6–11 (2002)
13. Sicherman, G.L., de Jonge, W., van de Riet, R.P.: Answering queries without revealing secrets. ACM Transactions on Database Systems 8(1), 41–59 (1983)
14. Bonatti, P.A., Kraus, S., Subrahmanian, V.: Foundations of secure deductive databases. IEEE Transactions on Knowledge and Data Engineering 7(3), 406–422 (1995)
15. Biskup, J.: For unknown secrecies refusal is better than lying. Data & Knowledge Engineering 33, 1–23 (2000)
16. Biskup, J., Bonatti, P.A.: Controlled query evaluation with open queries for a decidable relational submodel. In: Dix, J., Hegner, S.J. (eds.) FoIKS 2006. LNCS, vol. 3861, pp. 43–62. Springer, Heidelberg (2006)
17. Biskup, J., Bonatti, P.A.: Lying versus refusal for known potential secrets. Data & Knowledge Engineering 38, 199–222 (2001)
18. Biskup, J., Bonatti, P.A.: Controlled query evaluation for enforcing confidentiality in complete information systems. International Journal of Information Security 3, 14–27 (2004)
19. Biskup, J., Bonatti, P.A.: Controlled query evaluation for known policies by combining lying and refusal. Annals of Mathematics and Artificial Intelligence 40, 37–62 (2004)
20. Biskup, J., Weibert, T.: Refusal in incomplete databases. In: Research Directions in Data and Applications Security XVIII, pp. 143–157 Kluwer/Springer (2004)
21. Biskup, J., Weibert, T.: Keeping secrets in incomplete databases. Submitted, 2007. Extended abstract presented at the LICS'05 Affiliated Workshop on Foundations of Computer Security (FCS'05), available from `http://www.cs.chalmers.se/ andrei/FCS05/fcs05.pdf`(2005)
22. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (1995)
23. University of Dortmund, Information Systems and Security: CQE prototype implementation. `http://ls6-www.cs.uni-dortmund.de/issi/projects/cqe/`
24. Winslett, M.: An introduction to trust negotiation. In: Nixon, P., Terzis, S. (eds.) iTrust 2003. LNCS, vol. 2692, pp. 275–283. Springer, Heidelberg (2003)