A Framework for Inference Control in Incomplete Logic Databases

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Torben Weibert

Dortmund

2008

Tag der mündlichen Prüfung: 7. Februar 2008

Dekan: Prof. Dr. Peter Buchholz

1. Gutachter: Prof. Dr. Joachim Biskup
2. Gutachter: Prof. Dr. Gabriele Kern-Isberner

Abstract

Security in information systems aims at various, possibly conflicting goals, two of which are availablility and confidentiality. On the one hand, as much information as possible should be provided to the user. On the other hand, certain information may be confidential and must not be disclosed. In this context, inferences are a major problem: The user might combine a priori knowledge and public information gained from the answers in order to infer secret information.

Controlled Query Evaluation (CQE) is a dynamic, policy-driven mechanism for the enforcement of confidentiality in information systems, namely by the distortion of certain answers, by means of either lying or refusal. CQE prevents harmful inferences, and tries to provide the best possible availability while still preserving confidentiality. In this thesis, we present a framework for Controlled Query Evaluation in incomplete logic databases.

In the first part of the thesis, we consider CQE from a declarative point of view. We present three different types of confidentiality policy languages with different simplicity and expressibility – propositional potential secrets, confidentiality targets, and epistemic potential secrets – and show how they relate to each other. We also give a formal, declarative definition of the requirements for a method protecting these types of policies. As it turns out, epistemic potential secrets are the most expressive policies of the three types studied, so we concentrate on these policies in the second part of the thesis.

In that second part, we show how to operationally enforce confidentiality policies based on epistemic potential secrets. We first present an abstract framework in which two parameters are left open: 1. Does the user know the elements of the confidentiality policy? 2. Do we allow only refusal, only lying, or both distortion methods? For five of the six resulting cases, we present instantiations of the framework and prove the confidentiality according to the declarative definition from the first part of the thesis. For the remaining case (combined lying and refusal under unknown policies), we show that no suitable enforcement method can be constructed using the naive heuristics.

Finally, we compare the enforcement methods to those constructed for complete databases in earlier work, and we discuss the properties of our algorithms when relaxing the assumptions about the user's computational abilities.

Contents

List of Figures

1 Introduction

1.1 Confidentiality, Availability and the Inference Problem

Security in information systems aims at various goals, two of which are *availability* and *confidentiality*. On the one hand, the information system should make its contents *available* to its users, in a way that the users can request and receive the desired information. On the other hand, part of the information might be *confidential*: It must not under any circumstances be disclosed to certain users.

Trivially, these two requirements are conflicting: An information system cannot provide full availability when, at the same time, certain information may not be passed to the user. The goal is to design the information system in a way that as much non-secret information as possible is made available to the user, while any confidential information is kept secret.

A common approach to this problem is the use of *access control*: The administrator assigns static access rights to the particular instances of the data structures within the information system, e. g., to the tables of a relational database system. The access rights are then checked whenever a user tries to access these data structures. This approach is very cheap in terms of complexity, as it can usually be implemented by means of simple table look-ups. Nevertheless, static access control bears some problems, in particular the *inference problem*.

To illustrate the inference problem, imagine an information system in which a company keeps a list of its employees and their salaries. Furthermore, assume that a certain user must not find out the salary of a particular person. However, the user could issue the following queries:

- "What is Alice's position?" — Answer: "Manager."
- "How much is a manager's salary?" — Answer: "$50,000."

Obviously, the user has never explicitly queried Alice's salary. Neither answer – "Alice is a manager.", and "A manager's salary is $50,000." – reveals any secret information. However, one can combine the two answers, and the (meta) knowledge that all managers have the same salary, and figure out that Alice's salary must in fact be $50,000.

As the example demonstrates, harmful inferences can originate from two sources: the public data retrieved from the information system, and the *meta data*, for example semantic constraints on the actual data (in the example above, the fact that all managers have the same salary).

The inference problem was first identified and studied in the context of *statistical databases*, which contain data about a group of people or objects, and allow to issue *aggregate* queries on this data, e. g., the sum or average of some data value wrt. some

group of objects. For example, a user could query the average salary of all female employees of a given department. However, if the user knows that there is exactly one female among this group, he can easily determine the exact income of that person. For an introduction to the inference problem in statistical databases, see [19, 21, 27]. More recent work can be found in e. g. [34, 35, 37].

Extensive studies have also been performed in the context of *secure multilevel databases*. In these databases, each piece of data is assigned a security level (classification), for example "public", "confidential" or "secret". Each user is assigned a similar level (clearance), and he may only read data classified at or below this level. For example, a user cleared as confidential may only access confidential and public data, but no secret data. This is also called *mandatory access control (MAC)*. Harmful inferences occur when, e. g., secret data can be deduced by combining public and confidential data. A common technique is to selectively upgrade the classification of certain pieces of information until no such inferences are possible anymore. For the inference problem in secure multilevel databases, see e. g. [18, 19, 20, 26, 29, 30, 32, 36].

A good overview of the respective approaches to inference problem in various contexts can be found in [23]. That paper also gives some hints about modern fields of research like data mining and web services.

All of the aforementioned approaches consider a particular data model. In contrast, the purpose of this thesis and the related research is to study the inference problem from a more fundamental point of view, thereby understanding the requirements and techniques applicable to *any* kind of information system. This is achieved by considering *logic databases*, in which all data and queries are modelled by means of some underlying logic. At first glance, logic databases are of rather low interest for real-world applications. However, most kinds of information systems can be regarded as a special case of a logic database. For example, the relational data model can be described as a logic database using first-order logic.

1.2 Controlled Query Evaluation

In this thesis, we study a concept named *Controlled Query Evaluation*, which was first proposed by Sicherman et al. [31] and Bonatti et al. [16], respectively, and later further investigated by Biskup [1] and Biskup and Bonatti [2, 3, 4, 5, 6]

Controlled Query Evaluation is a policy-driven, dynamic approach to the above outlined problem of confidentiality and inferences. *Policy-driven* means that the administrator specifies a *confidentiality policy* which determines the *information* to be kept secret. This is differs from traditional access control, where the access rights are assigned to the *data*. This confidentiality policy is then enforced *dynamically* by checking each query and answer for a possible violation, and distorting the answer in case it is considered dangerous. We consider two types of distortion:

- *Lying:* An answer different from the original query value is returned.
- *Refusal:* No (real) answer is returned at all.

In particular, Biskup and Bonatti consider closed (yes/no) queries to *complete* databases under three parameters:

1. Two types of confidentiality policies:
   - Potential secrets (a single sentence is protected).
   - Secrecies (the actual truth value of some sentence is protected).
2. Two possible assumptions about the awareness of the user:
   - The user knows the elements of the confidentiality policy.
   - The user does not know the elements of the confidentiality policy.
3. Three different distortion policies:
   - Uniform lying (lies may be returned, but no answer may be refused).
   - Uniform refusal (answers may be refused, but no lies may be returned).
   - Combined lying and refusal (both distortions method may be applied).

All of the resulting twelve cases were studied by the authors. For nine of these cases, algorithms were specified and proved to preserve confidentiality. For one case (uniform lying under known secrecies), it was shown that there is no algorithm capable of preserving confidentiality under these parameters. For the remaining two cases (combined lying and refusal under unknown potential secrets or secrecies, respectively), no suitable algorithms could be identified which exploit the fact that the user does not know the elements of the confidentiality policy; however, the algorithms for the "known policy" can be employed instead. The results for closed queries to complete databases are summarized in [5].

Further aspects of Controlled Query Evaluation which have been studied so far include: open queries within a decidable first-order sub-model [7, 8], pre-processing of static "inference-proof" database instances [9, 14, 15], and the correlation with traditional access control in relational databases [10, 11].

1.3 Incomplete Logic Databases

This thesis presents a framework for Controlled Query Evaluation in *incomplete* logic databases. An *incomplete database* is an information system in which some information might be missing, and which thus might not be able to provide an answer to each query.

A *logic database* is an information system in which any data and queries are expressed by the means of some underlying logic. In particular, a closed query Φ can be expressed as a sentence of that logic, and database instances can be expressed in one of three ways:

Model-theoretic approach The database instance corresponds to a structure or interpretation I of the underlying logic. The value of a closed query Φ in a database instance I is *true* if I is a model of Φ. Otherwise, the value of Φ is *false*.

Proof-theoretic approach with closed world assumption The database instance is a consistent set Σ of sentences of the underlying logic. The value of a closed query Φ in

a database instance Σ is *true* if Σ logically implies Φ. Otherwise, the value of Φ is *false*.

Proof-theoretic approach with open world assumption The database instance is a consistent set Σ of sentences of the underlying logic. The value of a closed query Φ in a database instance Σ is *true* if Σ logically implies Φ, and the value is *false* if Σ logically implies $\neg\Phi$. Otherwise, the value of Φ is undefined.

In this thesis, we rely on the latter formalization, which is a suitable approach when modelling incomplete logic databases. Moreover, we restrict our considerations to propositional logic, and we define database schemas and instances as follows:

Definition 1.1 (Database schema)**.** A *database schema* DS is a finite set of propositions.

Definition 1.2 (Database instance)**.** A *database instance* db over the database schema DS is a consistent set of propositional sentences, using only propositions from DS.

Query evaluation can then be performed by the means of logical implication:

Definition 1.3 (Queries and ordinary query evaluation)**.** A *query* Φ on the database schema DS is a propositional sentence, using only propositions from DS. A query Φ is evaluated in the database instance db by the function

$$eval(\Phi)(db) := \begin{cases} true & \text{if } db \models_{PL} \Phi, \\ false & \text{if } db \models_{PL} \neg\Phi, \\ undef & \text{otherwise,} \end{cases} \tag{1.1}$$

where $\models_{PL}$ denotes logical implication in propositional logic.

In general, we assume that the user does not issue a single query but a sequence of queries, which are answered sequentially one by one:

Definition 1.4 (Query sequences)**.** A *query sequence* Q on the database schema DS is a finite sequence of queries $\langle \Phi_i, \ldots, \Phi_n \rangle$ on DS. It is evaluated in the database instance db by the function

$$ord\_eval(Q, db) = \langle ans_1, \ldots, ans_n \rangle$$

with

$$ans_i := eval(\Phi_i, db) \in \{true, false, undef\}.$$

Previous work on complete databases remains fairly abstract and considers Controlled Query Evaluation for "some [suitable] logic" [5]. In the present thesis, however, we decided to explicitly consider propositional logic, for a number of reasons:

- *Simplicity:* This thesis aims at studying the fundamentals of inference control in incomplete databases. It is favorable to base the investigations on a simple framework, bearing in mind that it might be easily adapted to other logics.

- *Modality:* In order to specify confidentiality policies, and to keep track of the information disclosed to the user, we introduce a modal operator K and use the well-known concept of epistemic modal logic (S5). Propositional modal logic has been studied very well, but we would need to take a closer look at the desired properties if we applied modal logic to other logics, which should not be the topic of this thesis.
- *Efficiency and decidability:* Our framework relies on logical implication, which is not decidable for many logics, including first-order logic. We could not implement these concepts without restricting those logics to a decidable fragment.

Considering other logics, or determining the exact requirements for the underlying logic, could be the topic of future work. See the conclusion for further comments.

Example 1.5. We illustrate these concepts by a running example, which will be picked up several times in the remainder of this thesis. Imagine a medical information system storing information about a (single) sick person, in particular the symptoms he is suffering from, and the diagnosed diseases. To keep the example simple, imagine that we have three different possible symptoms s_1, s_2 and s_3, and the diseases *aids*, *cancer* and *flu*, formalized by the database schema

$$DS := \{ s_1, s_2, s_3, \mathit{aids}, \mathit{cancer}, \mathit{flu} \}. \tag{1.2}$$

We consider a particular situation: The person suffers from both symptoms s_1 and s_2. However, it is not known whether the person also suffers from symptom s_3, perhaps because a required examination has not yet been performed. Furthermore, assume that the person suffers from *aids* but not from *flu*, while it is not known whether he also suffers from *cancer*.

$$db := \{ s_1, s_2, \mathit{aids}, \neg \mathit{flu} \} \tag{1.3}$$

Accordingly, we have, e. g., $\mathit{eval}(\mathit{aids})(db) = \mathit{true}$, $\mathit{eval}(\mathit{flu})(db) = \mathit{false}$ and $\mathit{eval}(\mathit{cancer})(db) = \mathit{undef}$. We can also issue and evaluate more complicated queries, e. g., $\mathit{eval}(s_1 \wedge (\mathit{aids} \vee \mathit{cancer}))(db) = \mathit{true}$.

1.4 Outline of this Thesis

This thesis consists of two parts.

In Part I, "Declarative Layer", we study Controlled Query Evaluation from a declarative point of view. In particular, we present three different policy languages which can be used to formalize the confidentiality policy: propositional potential secrets (Chapter 2), confidentiality targets (Chapter 3) and epistemic potential secrets (Chapter 4). These policy language have different strength in terms of syntactical simplicity and expressiveness. Propositional potential secrets are the easiest but also the least expressive language; confidentiality targets are more powerful, and also simple to handle, but still lack certain expressiveness; epistemic potential secrets is the most sophisticated policy language, although the use of modal logic might make life harder for the administrator.

For each language, we present a declarative definition of confidentiality. We also show how propositional potential secrets can be converted into confidentiality targets, and how confidentiality targets can be converted into epistemic potential secrets, and under which circumstances a reduction is possible, i. e., which properties an enforcement method must satisfy in order to safely handle the converted policies while still preserving confidentiality wrt. the original policy language.

In Part II, "Operational Layer", we consider epistemic potential secrets (as the most expressive policy language, and basis for the above mentioned reductions) and develop an operational framework for Controlled Query Evaluation. The framework is given in Chapter 5; however, some gaps are left open which need to be filled according to two parameters: 1. Does the user know the elements of the confidentiality policy? 2. Do we only allow refusal, or only allow lying, or allow both distortion methods?

In Chapter 6, we first consider the "known policy" case, and we present four instantiations of our framework: uniform lying, uniform refusal, uniform refusal with improved availability, and combined lying and refusal. For each case, we prove that the resulting enforcement method satisfies the declarative confidentiality definition from Chapter 4.

Chapter 7 examines how the algorithms can be relaxed if we assume that the user does not know the elements of the confidentiality policy. For uniform lying and uniform refusal, it is possible to establish an adapted algorithm which provides higher availability. For combined lying and refusal, such an algorithm could not be identified.

In Chapter 8, we take a closer look at the algorithms from Chapters 6 and 7 with regard to certain special cases. In particular, we examine how our algorithms operate on database instances which happen to be complete, and we point out similarities to the algorithms for complete databases from [5]. We also consider the case that the user is not a powerful reasoner as is assumed by the confidentiality definition.

We finally conclude in Chapter 9, summarizing the results and pointing out open questions and starting points for future work.

1.5 Previous Publications

Parts of this thesis have been published as a journal article and a conference paper.

The framework for the enforcement of potential secrets was first published in [13]. Although that paper only considers propositional potential secrets, the algorithms and proof ideas are similar to those found in this thesis. That paper also presents enforcement methods for known potential secrets (found in Chapter 6 of this thesis), and a comparison of that methods with the corresponding algorithms for complete information systems, which is picked up and further elaborated in Section 8.1 of this thesis.

The concept of confidentiality targets, found in Chapter 3 of this thesis, was introduced in [12]. That paper also proposes the use of epistemic potential secrets and shows the reduction from confidentiality targets from epistemic potential secrets. The requirements for a successful reduction stated in that paper are implicitly contained in the confidentiality definition for epistemic potential secrets found in this thesis (Definition 4.27).

1.6 Contribution to Joint Work

The two publications [12] and [13], on which this thesis is based, are co-authored by my advisor Joachim Biskup. His contribution comprised joint exploration of potential approaches, ongoing discussions, proof-reading and general advisory. All elaborations in that papers, including the proofs and the writing, are my original work.

Part I

Declarative Layer

2 Policies Based on Propositional Potential Secrets

Controlled Query Evaluation dynamically protects confidentiality according to some confidentiality policy specified by the administrator. A suitable *policy language* is needed for the specification of this policy. In this chapter, and in the following two chapters, we will present three different policy languages: propositional potential secrets, confidentiality targets, and epistemic potential secrets. These languages vary in terms of expressiveness and syntactical simplicity. Basically, we have a trade-off between these two properties: the language with the most simple syntax is least expressive, and vice versa.

For each language, we formally introduce CQE methods as functions, and specify an associated notion of confidentiality wrt. these functions. However, we remain on a purely abstract, declarative level: At this stage, we do not specify a concrete implementation of these functions. This is done in the second part of this thesis.

2.1 Confidentiality Policies

The first policy language that is subject to our study *propositional potential secrets*. Formally, a propositional potential secret is nothing but a propositional sentence, and a confidentiality policy is a set of potential secrets.

Definition 2.1 (Propositional potential secret)**.** A *propositional potential secret* is a propositional sentence ψ. We also say simply *potential secret* when the context is clear and confusion with epistemic potential secrets (introduced in Chapter 4) is unlikely to occur.

Definition 2.2 (Confidentiality policy based on propositional potential secrets)**.** A *confidentiality policy based on propositional potential secrets* is a set $policy = \{\psi_1, \ldots, \psi_m\}$ of propositional potential secrets.

The semantics of a potential secret ψ, as formally given in Definition 2.8 below, can be summarized as follows: In case ψ is *true* in the actual database instance *db*, the user may not infer this fact. Otherwise, if ψ is *false* or *undef* in *db*, this information may be disclosed. In other words, it must, at any time, appear possible to the user that ψ is not *true* in the actual database instance.

A common use case for potential secrets may be sentences like "person X suffers from aids": If the person does suffer from aids, this information must be kept secret. On the other hand, if the person is healthy, this information may be disclosed.

Example 2.3. We pick up the scenario given in Example 1.5. Imagine the person under consideration applies for an employment. In case he suffers from a terminal disease – aids or cancer –, this fact must be kept secret (as it might be an obstacle for being chosen for the job). On the other hand, if the applicant is healthy, this information may be disclosed. The sentences "the person suffers from aids" and "the person suffers from cancer", respectively, can be formalized as potential secrets:

$$policy := \{ aids,\ cancer \} \tag{2.1}$$

Note that the confidentiality policy is defined independently from the actual truth values in the database instance. In the instance *db* from our running example (cf. Example 1.5), *aids* is actually *true* (which must be kept secret), while *cancer* is undefined (which may be disclosed).

This concept of potential secrets can also be found in the work on complete databases [5]. In fact, the syntactical and semantical definition is identical – except for the fact that, in complete databases, "is not *true*" implies "must be *false*", while for the case of incomplete databases, this information implies "must either be *false* or *undef*". Accordingly, the confidentiality definition for complete databases demands that the value *false* appears possible to the user, while in this thesis, we allow *false* or *undef* to appear possible.

2.2 Declarative Confidentiality

In Definition 1.4, we have introduced the ordinary query evaluation function *ord_eval* which transforms a sequence Q of queries and a database instance *db* into a corresponding sequence of (undistorted) answers. Accordingly, we can formally define a method for Controlled Query Evaluation as a function *cqe*, which also returns an answer sequence, but which has two additional parameters:

1. The confidentiality policy *policy*.
2. The user's a priori assumptions *prior*.

This latter parameter allows us to account for facts the user might know in advance, for example semantic constraints which hold in the database instance *db*. The a priori assumptions are expressed in the same language as the confidentiality policy:

Definition 2.4 (A priori assumptions)**.** The user's *a priori assumptions* is a set *prior* of propositional sentences.

Example 2.5. Assume the following information to be publicly known: Anybody with symptom s_1 suffers from *aids* or *flu*; symptom s_2 is a safe indicator that you do not have *flu*; if you have symptom s_2 but not symptom s_3, you suffer from *cancer*; symptom s_3 means that you have *aids*.

$$prior := \{ s_1 \rightarrow aids \vee flu,\ \ s_2 \rightarrow \neg flu,\ \ (s_2 \wedge \neg s_3) \rightarrow cancer,\ \ s_3 \rightarrow aids \} \tag{2.2}$$

Given these considerations, we can now specify a formal definition for CQE methods.

Definition 2.6 (CQE method for propositional potential secrets)**.** A CQE method for propositional potential secrets is a function

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

with

$$ans_i \in \{true, false, undef, refuse\},$$

where Q is a query sequence, *db* a database instance, *prior* the user's a priori assumptions, and *policy* a confidentiality policy, given as a set of propositional potential secrets.

Each method *cqe* goes along with a function

$$precondition(db, prior, policy) \in \{true, false\}$$

which defines the admissible arguments for *cqe*. The precondition allows the CQE method to reject certain arguments, for example when one of the potential secrets already follows from the a priori assumptions and thus cannot be protected at all.

Unlike ordinary query evaluation, a CQE method may choose to return an answer ans_i which does not correspond to the original query value $eval(\Phi_i)(db)$. When $ans_i = refuse$, we say that the answer was refused. When $ans_i \in \{true, false, undef\}$ but $ans_i \neq eval(\Phi_i)(db)$, we say that a lie was given. This leads to the following classification of CQE methods which we use throughout this thesis (also for CQE methods concerning the other kinds of policy languages studied in the following chapters).

Definition 2.7 (Classification of CQE methods)**.** CQE methods can be classified wrt. the distortion methods employed:

- A CQE method which possibly issues lies but does never refuse an answer is called a *uniform lying method*.
- A CQE method which possibly issues refusals but does never lie is called a *uniform refusal method*.
- A CQE method which possibly issues both lies and refusals is called a *combined lying and refusal method*.

In the previous section, we have already outlined the idea of our declarative notion of confidentiality: Given a potential secret ψ, it must appear possible to the user that ψ is *false* or *undef* in the actual database instance *db*. With the means of the *cqe* function, we can specify a formal definition for this property:

Definition 2.8 (Confidentiality for propositional potential secrets)**.** Let *cqe* be a CQE method for propositional potential secrets with *precondition* as its associated precondition for admissible arguments. *cqe* is defined to preserve confidentiality iff

for all finite query sequences Q,
for all instances db,
for all sets of potential secrets $policy$,
for all potential secrets $\psi \in policy$,
for all a priori assumptions $prior$
so that $(db, prior, policy)$ satisfies $precondition$,
there exists an instance db' and a set $policy'$ of potential secrets
so that $(db', prior, policy')$ satisfies $precondition$
and the following two conditions hold:

(a) [$(db, policy)$ and $(db', policy')$ produce the same answers]
$cqe(Q, db, prior, policy) = cqe(Q, db', prior, policy')$

(b) [ψ is not $true$ in db']
$eval(\psi)(db') \in \{false, undef\}$

In case the user is assumed to know the elements of $policy$, we additionally demand that

(c) [same policy]
$policy = policy'$.

Remember that this definition is purely declarative. It does not give any hint on how to implement this kind of function.

2.3 Benefits and Limitations

With regard to syntax, the language of propositional potential secrets is the most simple policy language presented in this thesis. In particular, using propositional logic for $policy$ and $prior$ bears two advantages:

1. The database instance is given in propositional language as well. Thus, the same language is used for db, $policy$ and $prior$.

2. Policies based on potential secrets designed for complete databases as specified in [5] can be reused for incomplete databases without modification – however, a slight difference in the confidentiality definition must be noted: we do not consider the value $undef$ dangerous.

In the following, we investigate a number of disadvantages going along with the use of propositional potential secrets.

undef cannot be protected

Due to the lack of expressiveness, it is not possible to protect information like "ψ is $undef$ in the actual database instance".

Disjunctions cannot be protected

Likewise, it is not possible to protect information like "ψ is either *true* or *false*, but not *undef*". Naively, one would try to construct a potential secret $\psi \vee \neg\psi$, however, this is a tautology in propositional logic.

Limited emulation of "secrecies"

For complete information systems, a second type of confidentiality policies has been investigated: secrecies [5]. Syntactically, a secrecy is a pair $(\psi, \neg\psi)$ of propositional sentences. Semantically, the user may not infer which of the alternatives actually holds in the current database instance. In other words, the user must regard both *true* and *false* a possible query value.

For complete databases, secrecies can be reduced to potential secrets by converting each secrecy $(\psi, \neg\psi)$ into two potential secrets ψ and $\neg\psi$ [3]. The same technique is possible in our framework for incomplete databases: By adding both ψ and $\neg\psi$ to *policy*, the CQE methods makes sure that there is

- at least one "suitable" database instance db' in which ψ is *false* or *undef*, and
- at least one "suitable" database instance db'' in which $\neg\psi$ is *false* or *undef*, which means that ψ is *true* or *undef*.

Thus, the user can neither conclude that ψ must be *true*, nor that ψ must be *false*. However, it might be still possible to infer that ψ is *undef* (if $eval(\psi)(db') = undef$ for any suitable instance db'). Hence, it is not possible to enforce that the user does not learn *any* information about the query value.

3 Policies Based on Confidentiality Targets

In the previous chapter, we introduced propositional potential secrets. Although this policy language has a simple, familiar syntax, it contains a number of drawbacks, in particular: You cannot protect *undef*, you cannot protect the disjunction of two values, and it's impossible to emulate full-featured "secrecies" which protect any information about the value of a sentence ψ. See Section 2.3 for details.

In this chapter, we present an alternative policy language: *confidentiality targets*. Again, we give formal definitions of both CQE methods and a corresponding notion of confidentiality, however still on a declarative level. Additionally, we investigate the reduction from propositional potential secrets to confidentiality targets, i. e., we show how to use a (suitable) CQE method for confidentiality targets in order to protect propositional potential secrets.

3.1 Confidentiality Policies

Confidentiality targets consist of two parts: 1. The (propositional) sentence under consideration, and 2. a set of query values. We first give a formal definition of this concept and then discuss its semantics.

Definition 3.1 (Value set)**.** A *value set* V is a non-empty subset of $\{\mathit{true}, \mathit{false}, \mathit{undef}\}$.

Definition 3.2 (Confidentiality target)**.** Let DS be a database schema. A *confidentiality target* is a pair (ψ, V), where ψ is a propositional sentence over DS, and $V \subset \{\mathit{true}, \mathit{false}, \mathit{undef}\}$ with $\emptyset \neq V \neq \{\mathit{true}, \mathit{false}, \mathit{undef}\}$ is a value set.

Definition 3.3 (Confidentiality policy based on confidentiality targets)**.** A *confidentiality policy based on confidentiality targets* is a set

$$\mathit{policy}_{ct} = \{(\psi_1, V_1), \ldots, (\psi_m, V_m)\}$$

of confidentiality targets.

We read "(ψ, V)" as "ψ has one of the values from V". For example, "$(\psi, \{\mathit{true}, \mathit{false}\})$" is to be read as "$\psi$ is either *true* or *false* (but not *undef*)". This can be formalized by an evaluation function eval_{ct}.

Definition 3.4 (Truth value of a confidentiality target, function eval_{ct})**.** Let db be a database instance, and (ψ, V) a confidentiality target. The truth value of (ψ, V) in db is determined by the function

$$\mathit{eval}_{ct}((\psi, V))(db) := \begin{cases} \mathit{true} & \text{if } \mathit{eval}(\psi)(db) \in V, \\ \mathit{false} & \text{otherwise.} \end{cases} \tag{3.1}$$

Depending on the cardinality of V, confidentiality targets can be divided into two distinct classes:

- If $V = \{v_1\}$ is unary, (ψ, V) represents *definite* information about the truth value ψ (only one value v_1 appears possible).
- If $V = \{v_1, v_2\}$ is binary, (ψ, V) represents *disjunctive* information about the truth value ψ (two values – v_1 and v_2 – appear possible).

Being used as an element of the confidentiality policy, a confidentiality target (ψ, V) has the following semantics: The user may not infer that ψ has one of the values from V. However, it is safe to learn that ψ has a value from $\overline{V} := \{\mathit{true}, \mathit{false}, \mathit{undef}\} \setminus V$. We can use unary sets in order to protect the definitive value of a sentence ψ.

Example 3.5. The confidentiality policy (based on propositional potential secrets) given in Example 2.3 can be equivalently expressed as the following set of confidentiality targets:

$$\mathit{policy}_{ct} \;:=\; \{\ (\mathit{aids}, \{\mathit{true}\}),\ (\mathit{cancer}, \{\mathit{true}\})\ \}$$

We can also use binary sets in order to protect disjunctive information about a sentence.

Example 3.6. The following confidentiality policy declares that the user may not learn *any* information about the value of *aids*:

$$\mathit{policy}'_{ct} \;:=\; \{\ (\mathit{aids}, \{\mathit{true}, \mathit{false}\}),\ (\mathit{aids}, \{\mathit{true}, \mathit{undef}\}),\ (\mathit{aids}, \{\mathit{false}, \mathit{undef}\})\ \}$$

Any disjunctive information about the value of *aids* is forbidden (and implicitly also any information about the definitive value).

3.2 Declarative Confidentiality

Based on the notions above, we now give a formal definition of a CQE method for confidentiality targets, again formalized as a function cqe_{ct} with four parameters: the query sequence Q, the actual database instance db, the a priori assumptions prior_{ct}, and the confidentiality policy policy_{ct}. The a priori assumptions are expressed in the same language as the confidentiality policy:

Definition 3.7 (A priori assumptions)**.** The user's *a priori assumptions* prior_{ct} is a set of confidentiality targets.

Example 3.8. The a priori assumptions (for propositional potential secrets) from Example 2.5 can be equivalently expressed as follows:

$$\begin{aligned}\mathit{prior}_{ct} \;:=\; \{\ & (s_1 \rightarrow \mathit{aids} \vee \mathit{flu},\ \{\mathit{true}\}),\\ & (s_2 \rightarrow \neg \mathit{flu},\ \{\mathit{true}\}),\\ & ((s_2 \wedge \neg s_3) \rightarrow \mathit{cancer},\ \{\mathit{true}\}),\\ & (s_3 \rightarrow \mathit{aids},\ \{\mathit{true}\})\ \}\end{aligned}$$

Due to the extended expressiveness of confidentiality targets, we are now able to formalize additional assumptions. For example, imagine that each person undergoes an HIV test before being treated in the hospital, so it must be known whether the person has aids or not:

$$(aids, \{true, false\})$$

Given these considerations, a CQE method for confidentiality targets can be formalized as follows:

Definition 3.9 (CQE method for confidentiality targets)**.** A CQE method for confidentiality targets is a function

$$cqe_{ct}(Q, db, prior_{ct}, policy_{ct}) = \langle ans_1, \ldots, ans_n \rangle$$

with

$$ans_i \in \{true, false, undef, refuse\},$$

where Q is a query sequence, db a database instance, $prior_{ct}$ the user's a priori assumptions, and $policy_{ct}$ a confidentiality policy, given as a set of confidentiality targets.

Each method cqe_{ct} goes along with a function

$$precondition_{ct}(db, prior_{ct}, policy_{ct}) \in \{true, false\}$$

which defines the admissible arguments for cqe_{ct}. For example, a common precondition could be that the a priori assumptions do not reveal any secret information in the first place.

Our notion of confidentiality is very similar to the one for propositional potential secrets given in Definition 2.8. We now demand that $eval_{ct}((\psi, V))(db') = false$ in the alternative database instance db', which means that ψ has a value different from those in V. This meets the intended semantics described above.

Definition 3.10 (Confidentiality for Confidentiality Targets)**.** Let cqe_{ct} be a CQE method for confidentiality targets with $precondition_{ct}$ as its associated precondition for admissible arguments. cqe_{ct} is defined to preserve confidentiality iff

for all finite query sequences Q,
for all instances db,
for all sets of confidentiality targets $policy_{ct}$,
for all confidentiality targets $(\psi, V) \in policy_{ct}$,
for all a priori assumptions $prior_{ct}$
so that $(db, prior_{ct}, policy_{ct})$ satisfies $precondition_{ct}$,
there exists an instance db' and a set of confidentiality targets $policy'_{ct}$
so that $(db', prior_{ct}, policy'_{ct})$ satisfies $precondition_{ct}$
and the following two conditions hold:

(a) [$(db, policy_{ct})$ and $(db', policy'_{ct})$ produce the same answers]
$cqe_{ct}(Q, db, prior_{ct}, policy_{ct}) = cqe_{ct}(Q, db', prior_{ct}, policy'_{ct})$
(b) [(ψ, V) is false in db']
$eval_{ct}((\psi, V))(db') = false$

In case the user is assumed to know the elements of the confidentiality policy $policy'_{ct}$, we additionally demand that

(c) [same policy]
$policy_{ct} = policy'_{ct}$.

Again, this is a purely declarative definition and does not state which algorithm to use in order to meet these requirements.

3.3 Reduction from Propositional Potential Secrets

So far, we have introduced two policy languages: propositional potential secrets and confidentiality targets. Obviously, the latter is the more expressive one. The question arises whether it is possible to convert propositional potential secrets to confidentiality targets, and then use a CQE method for confidentiality targets in order to operationally protect these derived confidentiality targets, while still meeting the confidentiality requirements of the original propositional potential secrets. In this section, we show that this kind of reduction is possible if the target CQE method cqe_{ct} meets a certain requirement, called *ct-normality-preservation*.

Recall the semantics of $eval_{ct}$ from Definition 3.4. Obviously, $eval_{ct}((\psi, \{true\}))(db) = true$ holds exactly if $eval(\psi)(db) = true$. It is therefore obvious to define a conversion from propositional sentences to confidentiality targets as follows:

Definition 3.11 (Conversion from propositional sentences to confidentiality targets)**.** Let Σ be a set of propositional sentences. We define the function

$$conv_{ct}(\Sigma) \;:=\; \{\; (\alpha, \{true\}) \;\mid\; \alpha \in \Sigma \;\}$$

which converts Σ into a set of confidentiality targets.

An important property of the $conv_{ct}$ function is invertibility, i. e., given a generated set $conv_{ct}(\Sigma)$ of confidentiality targets, we can restore the original set Σ of propositional sentences. We will later need this property to prove the reduction theorem.

Lemma 3.12. $conv_{ct}$ is invertible.

Proof. By construction, $conv_{ct}$ is injective. □

Each confidentiality target generated by $conv_{ct}$ has the form $(\psi, \{true\})$, where ψ is a propositional sentence. We give a formal definition of this property and call these confidentiality targets *ct-normal*.

Definition 3.13 (ct-normality)**.** A confidentiality target (ψ, V) is called *ct-normal* iff $V = \{true\}$. A set of confidentiality targets $policy_{ct}$ is called ct-normal if each $(\psi, V) \in policy_{ct}$ is ct-normal.

Lemma 3.14. Let Σ be a set of propositional sentences. Then $conv_{ct}(\Sigma)$ is ct-normal.

Proof. Follows immediately from the definition of $conv_{ct}$. □

We can then use $conv_{ct}$ in order to convert a set of propositional sentences *policy* and a priori assumptions *prior* into the corresponding sets $policy_{ct} := conv_{ct}(policy)$ and $prior_{ct} := conv_{ct}(prior)$ of confidentiality targets. These sets can then – combined with the query sequence Q and the database instance db – be used as the input to a suitable, confidentiality-preserving CQE method cqe_{ct} for confidentiality targets. This method will ensure that there is an alternative database instance db' and an alternative set of confidentiality targets $policy'_{ct}$ in a way that the conditions mentioned in Definition 3.10 are met. We can now argue that there is also a suitable instance db' and a suitable set of propositional potential secrets $policy'$ so that the conditions of the confidentiality Definition 2.8 are satisfied.

Now a major problem arises: Assume that the user does not know the elements of the confidentiality policy, which means that condition (c) of Definition 3.10 is ineffective. This allows $policy'_{ct}$ to be different from $policy_{ct}$. In particular, it might happen that $policy'_{ct}$ is not ct-normal. Then, there is no set of propositional potential secrets $policy'$ such that $policy'_{ct} = conv_{ct}(policy')$. The reduction has failed.

We overcome this problem by defining the notion of *ct-normality-preservation*, which can be expressed by means of the original confidentiality definition, but with a stronger precondition.

Definition 3.15 (ct-normality-preservation)**.** A method cqe_{ct} with the associated precondition $precondition_{ct}$ is called *ct-normality-preserving* iff it preserves confidentiality in the sense of Definition 3.10 under the stronger precondition

$$precondition_{ct}^{\text{ct-normal}}(db, prior_{ct}, policy_{ct}) := \\ precondition_{ct}(db, prior_{ct}, policy_{ct}) \ \wedge \ \text{“}policy_{ct} \text{ is ct-normal”}.$$

In other words: If the original set of confidentiality targets $policy_{ct}$ is ct-normal, a ct-normality-preserving method guarantees that there is also an alternative database instance db' and an alternative set of **ct-normal** confidentiality targets $policy'_{ct}$ which satisfy the conditions from Definition 3.10.

Note that, despite the similar definitions, the concepts of confidentiality (according to Definition 3.10) and ct-normality (according to Definition 3.15) are totally independent from each other. A method which preserves confidentiality does not necessarily need to preserve ct-normality as well, and vice versa. Nevertheless, it is of course favorable to construct enforcement methods which satisfy both definitions, meaning that they can be employed for arbitrary confidentiality targets as well as for converted potential secrets.

However, we can state this type of inclusion property for the case that the user is assumed to know the confidentiality policy:

Lemma 3.16. Assume that the user knows the confidentiality policy, and let cqe_{ct} preserve confidentiality in the sense of Definition 3.10. Then cqe_{ct} also preserves ct-normality.

Proof. Follows trivially from condition (c) of Definition 3.10, which demands that $policy_{ct} = policy'_{ct}$. □

We can now formally define the reduction outlined above, and prove that the CQE method for propositional potential secrets constructed in this manner does in fact preserve confidentiality if the underlying method cqe_{ct} preserves ct-normality.

Theorem 3.17 (Reduction from propositional potential secrets to confidentiality targets)**.** Let cqe_{ct} be a CQE method for confidentiality targets, preserving ct-normality in the sense of Definition 3.15. Let $precondition_{ct}$ be the associated precondition. Then the function

$$cqe(Q, db, prior, policy) := cqe_{ct}(Q, db, conv_{ct}(prior), conv_{ct}(policy))$$

with the precondition

$$precondition(db, prior, policy) := precondition_{ct}(db, conv_{ct}(prior), conv_{ct}(policy)) \quad (3.2)$$

preserves confidentiality in the sense of Definition 2.8.

Proof. Let db be a database instance, $policy$ a set of propositional potential secrets, $prior$ the a priori assumptions so that the pertinent $precondition$ is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence. Let $\psi \in policy$ be a potential secret.

As $precondition(db, prior, policy)$ is satisfied, the equivalent condition

$$precondition_{ct}(db, conv_{ct}(prior), conv_{ct}(policy)) \quad (3.3)$$

is satisfied as well. Furthermore, $conv_{ct}(policy)$ is ct-normal. So also

$$precondition_{ct}^{\text{ct-normal}}(db, conv_{ct}(prior), conv_{ct}(policy)) \quad (3.4)$$

is satisfied. Then, following Definition 3.15, a database instance db' and a set of confidentiality targets $policy'_{ct}$ exist, so that – in particular – the precondition

$$precondition_{ct}^{\text{ct-normal}}(db, conv_{ct}(prior), policy'_{ct}) \quad (3.5)$$

is satisfied, which implies that $policy'_{ct}$ must be ct-normal. Thus, we can construct a set $policy$ of propositional potential secrets by inverting the $conv_{ct}$ function:

$$policy' := \{\ \psi \ \mid\ (\psi, \{true\}) \in policy'_{ct}\ \}. \quad (3.6)$$

We then have

$$conv_{ct}(policy') = policy'_{ct}. \quad (3.7)$$

We show that db' and $policy'$ satisfy all of the conditions from Definition 2.8.

- By $precondition_{ct}^{\text{ct-normal}}(db, conv_{ct}(prior), policy'_{ct})$, we have

  $$precondition_{ct}(db, conv_{ct}(prior), policy'_{ct}), \tag{3.8}$$

  which is by (3.7) equivalent to

  $$precondition_{ct}(db, conv_{ct}(prior), conv_{ct}(policy')), \tag{3.9}$$

  and by (3.2) also equivalent to

  $$precondition(db', prior, policy'). \tag{3.10}$$

  So the precondition is satisfied for db' and $policy'$.

- By condition (a) of Definition 3.10, $(db, conv_{ct}(policy))$ and $(db', policy'_{ct})$ produce the same answers, so we have

  $$\begin{aligned} & cqe(Q, db, prior, policy) && (3.11)\\ = \ & cqe_{ct}(Q, db, conv_{ct}(prior), conv_{ct}(policy)) && (3.12)\\ = \ & cqe_{ct}(Q, db', conv_{ct}(prior), policy'_{ct}) && (3.13)\\ = \ & cqe_{ct}(Q, db', conv_{ct}(prior), conv_{ct}(policy')) && (3.14)\\ = \ & cqe(Q, db', prior, policy'). && (3.15) \end{aligned}$$

  This satisfies condition (a) of Definition 2.8.

- By condition (b) of Definition 3.10, we have

  $$eval_{ct}((\psi, \{true\}))(db') = false. \tag{3.16}$$

  By the definition of $eval_{ct}$ (3.1), this is equivalent to

  $$eval(\psi)(db') \in \{false, undef\}. \tag{3.17}$$

  This satisfies condition (b) of Definition 2.8.

- In case we assume that the user knows the confidentiality policy, condition (c) of Definition 3.10 guarantees that $policy'_{ct} = policy_{ct}$. Thus, we also have

  $$conv_{ct}(policy') = conv_{ct}(policy) \tag{3.18}$$

  and also

  $$policy' = policy. \tag{3.19}$$

  This satisfies condition (c) of Definition 2.8.

Thus, all of the conditions of Definition 2.8 are satisfied for db' and $policy'$. □

This result releases us from constructing specific enforcement methods for propositional potential secrets, in case we have suitable, ct-normality-preserving methods for confidentiality targets. In Chapter 4, we will go one step further and define a reduction from confidentiality targets to epistemic potential secrets, resulting in a ct-normality-preserving method for confidentiality targets. Then it is even possible to handle propositional potential secrets in the framework of epistemic potential secrets (for which case we will construct specific methods in the second part of this thesis).

3.4 Benefits and Limitations

The motivation to establish another kind of policy language was to overcome the limitations of propositional potential secrets, as listed in Section 2.3. In this section, we will reconsider these issues. We also show that confidentiality targets still entail some limitations.

Protection of undef

It is now possible to protect the fact that a sentence ψ has the value *undef* in the actual database instance, by the means of the confidentiality target $(\psi, \{undef\})$.

Protection of disjunctions

Disjunctive information can be protected by using binary value sets V. For example, $(\psi, \{true, false\})$ protects the information that the query value is either *true* or *false* (but not *undef*). In other words, this confidentiality target disguises the fact that the database actually "knows" the value of ψ.

Full-featured "secrecies"

It is now possible to define a set of confidentiality targets which enforce that the user does not learn anything about the value of some sentence ψ:

$$policy_{ct} = \{(\psi, \{true, false\}), (\psi, \{true, undef\}), (\psi, \{false, undef\})\}$$

Applying the semantics of cqe_{ct}, there must be

- at least one "suitable" db' in which ψ is neither *true* nor *false*,
- at least one "suitable" db'' in which ψ is neither *true* nor *undef*, and
- at least one "suitable" db''' in which ψ is neither *false* nor *undef*.

As a result, any query value appears possible to the user.

We can also define weaker versions of secrecies. For example, the policy

$$policy_{ct} = \{(\psi, \{true\}), (\psi, \{false\}), (\psi, \{undef\})\}$$

ensures that the user does not learn any *definite* information about the query value of ψ; however, it is still possible to exclude one value and thus learn *disjunction* information.

Unfortunately, there are still two limitations when using confidentiality targets:

Combinations of different sentences cannot be protected

Imagine you wanted to protect the information that

> "ψ is *true*, and ϕ is either *false* or *undef*".

Formalizing this information as a confidentiality target is not possible, as it involves two different propositional sentences with different value sets. A policy containing the two sub-sentences expressed as distinct confidentiality targets

$$policy_{ct} = \{(\psi, \{true\}), (\phi, \{true, false\})\}$$

wouldn't have the same semantics; according to Definition 3.10, there are alternative database instances db' in which at least one given confidentiality target is *false*. But you cannot enforce both to be *false* at the same time. In essence, the problem is that our policy language does not provide any logical operators between confidentiality targets.

Implementation issues

The confidentiality and ct-normality-preservation properties specified in this section are solely declarative. There is no hint on how to implement a method meeting these requirements. The methods for complete databases from [5] suggest that it is favorable to employ logical implication. However, as confidentiality targets are our own, proprietary concept, there is no notion of logical implication. And even if there was, we lack a formal proof system, and an implementation thereof. Thus, confidentiality targets are a descriptive concept for theoretical considerations, but a less suitable foundation for an implementable, concrete CQE system. It is favorable to find an expressive policy language based on a well known logic with an established notion of logical implication, and for which implementations of proof systems are available. In the following chapter, we will show that modal epistemic logic meets these requirements.

4 Policies Based on Epistemic Potential Secrets

While confidentiality targets, presented in the previous chapter, offer a useful expressiveness when defining confidentiality policies, they lack the requirements for the practical design and implementation of concrete enforcement methods: there is no notion of logical implication, and there are no proof systems available either.

In this chapter, we finally present the full-featured policy language $\mathcal{L}_{PS}$, which is a subset of epistemic modal logic. This allows us to employ the notion of logical implication known for modal logic for the construction of enforcement methods. We can also use the various available theorem proving systems when implementing these enforcement methods. CQE methods for the various parameters (known vs. unknown policies, lying vs. refusal vs. combined lying and refusal) are presented in Chapters 6 and 7. Some of them have already been implemented [33] using the Logics Workbench proving system [25].

As in the previous chapters, we start with the syntactical and semantical definition of our policy language (Section 4.1), and then give a formal definition of a CQE method using this language, as well as of the corresponding notion of confidentiality (Section 4.2).

Sections 4.3 and 4.4 investigate the reduction from confidentiality targets and propositional potential secrets, respectively, to epistemic potential secrets. We formally investigate the requirements for the underlying CQE method which make these reductions possible. In Chapters 6 and 7, you will see that most of the methods presented there meet these requirements.

4.1 Epistemic Logic and the Policy Language

We first introduce the language $\mathcal{L}_{DS}$ of epistemic logic (S5), and then define its subset $\mathcal{L}_{PS}$ which we employ as a policy language.

4.1.1 Epistemic logic $\mathcal{L}_{DS}$

An excellent discussion of epistemic logic can be found in [22]. In the following, we will briefly summarize the syntax of modal logic in general, and the particular semantics of epistemic logic.

Modal logic is established by introducing the modal operator $\Box$ of *necessity* (and the dual operator $\diamond$ of *possibility*). In the context of knowledge, $\Box$ is usually written as K, which is to be read as "it is known that ...". Given a database schema (set of propositions) DS, we denote the resulting language by $\mathcal{L}_{DS}$.

Definition 4.1 (Language $\mathcal{L}_{DS}$)**.** Let DS be a database schema, i. e., a set of propositions. The language $\mathcal{L}_{DS}$ is inductively defined along the following five rules:

1. Each propositional sentence over DS is an $\mathcal{L}_{DS}$-sentence.
2. If ϕ is an $\mathcal{L}_{DS}$-sentence, so is $\neg\phi$.
3. If ϕ and ψ are $\mathcal{L}_{DS}$-sentences, so is $\phi \wedge \psi$.
4. If ϕ and ψ are $\mathcal{L}_{DS}$-sentences, so is $\phi \vee \psi$.
5. If ϕ is an $\mathcal{L}_{DS}$-sentence, so is $K\phi$.

In order to evaluate an $\mathcal{L}_{DS}$-sentence, we use the common Kripke-style semantics, which considers sets of states (or "worlds") and a possibility relation between these states, determining which states are considered "possible" from the perspective of a given state.

Definition 4.2 ($\mathcal{M}_{DS}$-structure)**.** 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 $\pi : S \times DS \to \{true, false\}$ a function which assigns a truth value to each proposition from DS under each state $s \in S$.

Note that we demand $\mathcal{K}$ to be an equivalence relation, i. e., reflexive, symmetric and transitive. This kind of logic is called *epistemic logic*, and this is the most common way to formalize knowledge by the means of modal logic (although philosophers have argued that it might not be an appropriate formalization of "real world knowledge").

Another way to characterize a certain modal logic is using an axiom schema; in the most commonly used schema, epistemic logic is identified by satisfying the following four axioms:

- Distribution axiom (K): $(K\phi \wedge K(\phi \to \psi)) \to K\psi$
- Knowledge axiom (T): $K\phi \to \phi$
- Positive introspection (4): $K\phi \to KK\phi$
- Negative introspection (5): $\neg K\phi \to K\neg K\phi$

Epistemic logic is thus called *KT45*, or simply *S5*, a denomination introduced in a very early publication by Lewis [28].

The semantics of $\mathcal{L}_{DS}$ is defined with the means of these $\mathcal{M}_{DS}$-structures:

Definition 4.3 (Truth value of $\mathcal{L}_{DS}$-sentences)**.** Let $M = (S, \mathcal{K}, \pi)$ be an $\mathcal{M}_{DS}$-structure and $s \in S$ a state. The truth value of an $\mathcal{L}_{DS}$-sentence in (M, s) is inductively defined as follows (where $p \in DS$ denotes a proposition and ϕ and ψ denote $\mathcal{L}_{DS}$-sentences):

$$(M, s) \models p \quad \text{iff } \pi(s)(p) = true \tag{4.1}$$
$$(M, s) \models \neg\phi \quad \text{iff } (M, s) \not\models \phi \tag{4.2}$$
$$(M, s) \models \phi \wedge \psi \quad \text{iff } (M, s) \models \phi \text{ and } (M, s) \models \psi \tag{4.3}$$
$$(M, s) \models \phi \vee \psi \quad \text{iff } (M, s) \models \phi \text{ or } (M, s) \models \psi \tag{4.4}$$
$$(M, s) \models K\phi \quad \text{iff } (M, s') \models \phi \text{ for all } s' \in S \text{ such that } (s, s') \in \mathcal{K} \tag{4.5}$$

Logical implication is defined in the usual way:

Definition 4.4 (Logical implication in epistemic logic)**.** Let Σ be a set of $\mathcal{L}_{DS}$-sentences, and ϕ a single $\mathcal{L}_{DS}$-sentence. We say that Σ logically implies ϕ 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

$$\text{if } (M, s) \models \Sigma \text{ then } (M, s) \models \phi.$$

4.1.2 Policy language $\mathcal{L}_{PS}$

Based on $\mathcal{L}_{DS}$, we can now define the subset we are going to use as a policy language. In particular, we are interested in a sub-language in which all propositional sentences are prefixed by K. We call this language $\mathcal{L}_{PS}$.

Definition 4.5 (Language $\mathcal{L}_{PS}$)**.** The language $\mathcal{L}_{PS} \subset \mathcal{L}_{DS}$ is defined as follows:

1. If α is a propositional sentence, then $K\alpha$ is an $\mathcal{L}_{PS}$-sentence.
2. If ϕ is an $\mathcal{L}_{PS}$-sentence, so is $\neg\phi$.
3. If ϕ and ψ are $\mathcal{L}_{PS}$-sentences, so is $\phi \wedge \psi$.
4. If ϕ and ψ are $\mathcal{L}_{PS}$-sentences, so is $\phi \vee \psi$.

In the following, we will demonstrate some properties of $\mathcal{L}_{PS}$. First, we define two functions that convert a propositional sentence α and a value set V (or a single value v, respectively) into an $\mathcal{L}_{PS}$-sentence, and we also present an alternative evaluation function.

Definition 4.6 (Functions Δ, Δ^*)**.** Let α be a propositional sentence and $\emptyset \neq V \subseteq \{\mathit{true}, \mathit{false}, \mathit{undef}\}$ a non-empty value set. We define the function

$$\Delta^*(\alpha, V) := \bigvee_{v \in V} \Delta(\alpha, v) \tag{4.6}$$

with

$$\begin{aligned} \Delta(\alpha, \mathit{true}) &:= K\alpha, \\ \Delta(\alpha, \mathit{false}) &:= K\neg\alpha, \text{ and} \\ \Delta(\alpha, \mathit{undef}) &:= \neg K\alpha \wedge \neg K\neg\alpha. \end{aligned}$$

which converts (α, V) into an epistemic sentence.

Example 4.7. Let α be a propositional sentence. Then we have:

$$\begin{aligned} \Delta^*(\alpha, \{\mathit{true}\}) &= K\alpha \\ \Delta^*(\alpha, \{\mathit{true}, \mathit{false}\}) &= K\alpha \quad \vee \quad K\neg\alpha \\ \Delta^*(\alpha, \{\mathit{false}, \mathit{undef}\}) &= K\neg\alpha \quad \vee \quad \neg K\alpha \wedge \neg K\neg\alpha \end{aligned}$$

Definition 4.8 (Function $eval^*$)**.** The evaluation function $eval^*$, which returns the value of some sentence α in a database instance db, converted into an $\mathcal{L}_{PS}$-sentence, is defined as follows (similar to the $eval^*$ function for complete databases from [5]):

$$eval^*(\alpha)(db) := \Delta(\alpha, eval(\alpha)(db)) \tag{4.7}$$

Example 4.9. Considering the database instance $db = \{a, b \vee c, \neg d\}$, we have:

$$\begin{aligned} eval^*(a)(db) &= Ka \\ eval^*(d)(db) &= K\neg d \\ eval^*(a \wedge b)(db) &= \neg K(a \wedge b) \wedge \neg K\neg(a \wedge b) \end{aligned}$$

Lemma 4.10. Let α be a propositional sentence. Then $\Delta(\alpha, v)$ (for any $v \in \{true, false, undef\}$) and $\Delta^*(\alpha, V)$ (for any $\emptyset \neq V \subseteq \{true, false, undef\}$) are $\mathcal{L}_{PS}$-sentences.

Proof. Follows from the definition of Δ, which only allows K-prefixed propositional sentences, and Δ^*, which only uses the $\vee$ operator to connect these kinds of sentences. □

The following property follows from the fact that any propositional sentence occurring in an $\mathcal{L}_{PS}$-sentence is prefixed by K. It states that, given an $\mathcal{M}_{DS}$-structure M, any $\mathcal{L}_{PS}$-sentence has the same truth value in all states s of the same equivalence class.

Lemma 4.11. Let $M = (S, \mathcal{K}, \pi)$ be an $\mathcal{M}_{DS}$-structure and $s_1, s_2 \in S$ two states such that $(s_1, s_2) \in \mathcal{K}$. Let ϕ be an $\mathcal{L}_{PS}$-sentence. Then we have

$$(M, s_1) \models \phi \;\Leftrightarrow\; (M, s_2) \models \phi.$$

Proof. We only show one direction. The other direction then follows easily from the symmetry of $\mathcal{K}$.

Due to the inductive definition of $\mathcal{L}_{PS}$ and the semantics of epistemic logic, it is sufficient to show the proposition for $\phi = K\alpha$, such that α is propositional. The left hand side is then equivalent to

$$(M, s_1) \models K\alpha \tag{4.8}$$

and equivalent to

$$(M, s') \models \alpha \text{ for all } s' \text{ such that } (s_1, s') \in \mathcal{K}. \tag{4.9}$$

As $(s_1, s_2) \in \mathcal{K}$, and $\mathcal{K}$ is an equivalence relation, we have

$$(s_1, s') \in \mathcal{K} \;\Leftrightarrow\; (s_2, s') \in \mathcal{K}. \tag{4.10}$$

Thus, (4.9) is equivalent to

$$(M, s') \models \alpha \text{ for all } s' \text{ such that } (s_2, s') \in \mathcal{K}, \tag{4.11}$$

and also to

$$(M, s_2) \models K\alpha, \tag{4.12}$$

which is finally equivalent to

$$(M, s_2) \models \phi. \tag{4.13}$$

□

4.1.3 Evaluation of $\mathcal{L}_{PS}$-sentences

We want to consider the truth value of an $\mathcal{L}_{PS}$-sentence ϕ within a database instance db. However, the truth value of an $\mathcal{L}_{PS}$-sentence is only defined by the Kripke-style semantics given above, wrt. an $\mathcal{M}_{DS}$-structure $M = (s, \mathcal{K}, \pi)$ and a state $s \in S$ (cf. Definition 4.3). In order to evaluate an $\mathcal{L}_{PS}$-sentence ϕ within a database instance db, we first need to convert db into an appropriate structure M and state s, and then check whether (M, s) is a model of ϕ.

This kind of conversion is given in Definition 4.12 below: We choose S as the set of all propositional interpretations over the associated database schema DS. The relation $\mathcal{K}$ connects all states (interpretations) which make db true; any other interpretation is only connected to itself to ensure the reflexivity property demanded by S5. Finally, the π function evaluates a proposition p within a state s according to the usual propositional semantics.

Definition 4.12 (Structure M_{db})**.** Let DS be a database schema and db a database instance over DS. The Kripke-structure $M_{db} = (S^+_{db} \cup S^-_{db}, \mathcal{K}_{db}, \pi_{DS})$ is defined as follows (where DS^* is the set of interpretations over DS, $p \in DS$ a proposition, $s \in DS^*$ an interpretation, and $\models$ the model operator of propositional logic):

$$\begin{aligned} S^+_{db} &:= \{\ s \in DS^* \ \mid\ s \models db\ \} \\ S^-_{db} &:= \{\ s \in DS^* \ \mid\ s \not\models db\ \} \\ \mathcal{K}_{db} &:= \{\ (s, s') \mid s, s' \in S^+_{db}\ \} \cup \{\ (s, s) \mid s \in S^-_{db}\ \} \\ \pi_{DS}(s)(p) &:= s \models p \end{aligned}$$

It is easy to see that $\mathcal{K}_{db}$ is an equivalence relation: All states (interpretations) $s \in S^+_{db}$ which are a model of db are joined in one equivalence class; each other state $s \in S^-_{db}$ forms a class of its own. Thus, M_{db} is a valid $\mathcal{M}_{DS}$-structure.

Lemma 4.13. Let $M_{db} = (S^+_{db} \cup S^-_{db}, \mathcal{K}_{db}, \pi_{DS})$ be constructed as specified in Definition 4.12. Let $s \in DS^*$ be a database instance (and thus also a state from $S^+_{db} \cup S^-_{db}$). Let α be a propositional sentence. Then we have

$$s \models \alpha \quad \Leftrightarrow \quad (M_{db}, s) \models \alpha.$$

Proof. By structural induction.

Case 1 (α is a proposition p)**.** By the definition of the π_{DS} function from Definition 4.12, the left hand side is then equivalent to

$$\pi_{DS}(s)(\alpha) = \mathit{true}, \tag{4.14}$$

which is by Definition 4.3 equivalent to

$$(M_{db}, s) \models \alpha. \tag{4.15}$$

Case 2 ($\alpha = \neg\phi$)**.** Then we have

$$s \not\models \alpha, \tag{4.16}$$

which is by the induction hypothesis equivalent to

$$(M_{db}, s) \not\models \alpha, \tag{4.17}$$

and by Definition 4.3 also equivalent to

$$(M_{db}, s) \models \neg\alpha. \tag{4.18}$$

Case 3 ($\alpha = \phi \wedge \psi$)**.** Then we have

$$s \models \phi \quad \text{and} \quad s \models \psi, \tag{4.19}$$

which is by the induction hypothesis equivalent to

$$(M_{db}, s) \models \phi \quad \text{and} \quad (M_{db}, s) \models \psi, \tag{4.20}$$

and by Definition 4.3 also equivalent to

$$(M_{db}, s) \models \phi \wedge \psi. \tag{4.21}$$

Case 4 ($\alpha = \phi \vee \psi$)**.** Then we have

$$s \models \phi \quad \text{or} \quad s \models \psi, \tag{4.22}$$

which is by the induction hypothesis equivalent to

$$(M_{db}, s) \models \phi \quad \text{or} \quad (M_{db}, s) \models \psi, \tag{4.23}$$

and by Definition 4.3 also equivalent to

$$(M_{db}, s) \models \phi \vee \psi. \tag{4.24}$$

□

Based on M_{db}, we can define an evaluation function $eval_{ps}$.

Definition 4.14 (Function $eval_{ps}$)**.** Let DS be a database schema, and db an instance over DS. Let $M_{db} = (S^+_{db} \cup S^-_{db}, \mathcal{K}_{db}, \pi_{DS})$ be the $\mathcal{M}_{DS}$-structure derived from db. We define the evaluation function

$$eval_{ps} : \mathcal{L}_{PS} \times DS^* \to \{true, false\}$$

with

$$eval_{ps}(\phi)(db) \;:=\; (M_{db}, s) \models \phi \text{ for all } s \in S^+_{db}.$$

According to Lemma 4.11, this can be equivalently expressed as

$$eval_{ps}(\phi)(db) \;:=\; (M_{db}, s) \models \phi \text{ for at least one } s \in S^+_{db}.$$

We first show that the common logic connectors $\neg$, $\wedge$ and $\vee$ have their usual semantics when applied to $eval_{ps}$.

Lemma 4.15. The following equivalences hold for all database instances db and $\mathcal{L}_{PS}$-sentences ϕ and ψ:

(1) $eval_{ps}(\neg\phi)(db) = true \quad \Leftrightarrow \quad eval_{ps}(\phi)(db) = false$

(2) $eval_{ps}(\phi \wedge \psi)(db) = true \quad \Leftrightarrow \quad eval_{ps}(\phi)(db) = true$ and $eval_{ps}(\psi)(db) = true$

(3) $eval_{ps}(\phi \vee \psi)(db) = true \quad \Leftrightarrow \quad eval_{ps}(\phi)(db) = true$ or $eval_{ps}(\psi)(db) = true$

Proof. Follows from Definition 4.3 and Definition 4.14. □

We will now show a number of properties of $eval_{ps}$ which demonstrate the semantic relationship between $eval_{ps}$ and the original evaluation function $eval$. First, we can prove that a propositional sentence α is implied by a database instance db exactly if $eval_{ps}(K\alpha)(db) = true$.

Lemma 4.16. Let db be a database instance and α a propositional sentence. Let $M_{db} = (S^+_{db} \cup S^-_{db}, \mathcal{K}_{db}, \pi_{DS})$ be the $\mathcal{M}_{DS}$-structure derived from db. Then we have

$$db \models_{PL} \alpha \quad \Leftrightarrow \quad eval_{ps}(K\alpha)(db) = true,$$

or, according to Definition 4.14, equivalently

$$db \models_{PL} \alpha \quad \Leftrightarrow \quad (M_{db}, s) \models K\alpha \text{ for all } s \in S^+_{db}.$$

Proof. By Lemma 4.11, the proposition is also equivalent to

$$db \models_{PL} \alpha \quad \Leftrightarrow \quad (M_{db}, s) \models K\alpha \text{ for at least one } s \in S^+_{db}. \tag{4.25}$$

We show the latter equivalence.

Let $s \in S^+_{db}$. As db is consistent, S^+_{db} is non-empty, so there must be at least one such s. The right hand side of (4.25) is then equivalent to

$$(M_{db}, s') \models \alpha \quad \text{for all } s' \text{ such that } (s, s') \in \mathcal{K}, \tag{4.26}$$

which is by Lemma 4.13 equivalent to

$$s' \models \alpha \quad \text{for all } s' \text{ such that } (s, s') \in \mathcal{K}. \tag{4.27}$$

By the construction of $\mathcal{M}_{DS}$, the states which are related to s are exactly those from S^+_{db} (as s is also contained in S^+_{db}). So we can equivalently rewrite (4.27) as

$$s' \models \alpha \quad \text{for all } s' \in S^+_{db}. \tag{4.28}$$

Remember that S^+_{db} exactly contains all interpretations which make db true, so we equivalently have

$$s' \models \alpha \quad \text{for all } s' \text{ such that } db \models s, \tag{4.29}$$

which is by the generic definition of logical implication equivalent to

$$db \models_{PL} \alpha. \tag{4.30}$$

□

We now know that $eval(\alpha)(db) = true$ (which corresponds to $db \models_{PL} \alpha$) is equivalent to $eval_{ps}(K\alpha)(db) = true$. We can extend these considerations to the other two query values, and show that, in general, $eval(\alpha)(db) = v$ (with $v \in \{true, false, undef\}$) is equivalent to $eval_{ps}(\Delta(\alpha, v))(db) = true$, where Δ is the conversion function introduced in Definition 4.6.

Lemma 4.17. Let α be a propositional sentence, $v \in \{true, false, undef\}$ a query value, and db a database instance. Then we have

$$eval(\alpha)(db) = v \;\Leftrightarrow\; eval_{ps}(\Delta(\alpha, v))(db) = true.$$

Proof. Follows from the semantics of $eval_{ps}$ given in Definition 4.14:

Case 1 ($v = true$, $\Delta(\alpha, v) = K\alpha$)**.** Then we have by the definition (1.1) of *eval*

$$db \models_{PL} \alpha, \tag{4.31}$$

and the proposition follows from Lemma 4.16.

Case 2 ($v = false$, $\Delta(\alpha, v) = K\neg\alpha$)**.** Then we have by the definition (1.1) of *eval*

$$db \models_{PL} \neg\alpha, \tag{4.32}$$

and the proposition follows from Lemma 4.16.

Case 3 ($v = undef$, $\Delta(\alpha, v) = \neg K\alpha \wedge K\neg\alpha$)**.** Then we have by the definition (1.1) of *eval*

$$db \not\models_{PL} \alpha \text{ and} \tag{4.33}$$
$$db \not\models_{PL} \neg\alpha, \tag{4.34}$$

which is by Lemma 4.16 equivalent to

$$eval_{ps}(K\alpha)(db) = false \text{ and} \tag{4.35}$$
$$eval_{ps}(K\neg\alpha)(db) = false, \tag{4.36}$$

and by Definition 4.14 to

$$(M_{db}, s') \not\models K\alpha \text{ for some } s' \in S^+_{db} \text{ and} \tag{4.37}$$
$$(M_{db}, s'') \not\models K\neg\alpha \text{ for some } s'' \in S^+_{db}. \tag{4.38}$$

According to the semantics definition, this is equivalent to

$$(M_{db}, s') \models \neg K\alpha \text{ for some } s' \in S^+_{db} \text{ and} \tag{4.39}$$
$$(M_{db}, s'') \models \neg K\neg\alpha \text{ for some } s'' \in S^+_{db}. \tag{4.40}$$

By Lemma 4.11, any sentence satisfied in M_{db} and some state $s_1 \in S^+_{db}$ is also satisfied in M_{db} and any other state $s_2 \in S^+_{db}$, so we have for all $s \in S^+_{db}$:

$$(M_{db}, s) \models \neg K\alpha \text{ and} \tag{4.41}$$
$$(M_{db}, s) \models \neg K\neg\alpha. \tag{4.42}$$

This is equivalent to

$$(M_{db}, s) \models \neg K\alpha \wedge \neg K\neg\alpha \tag{4.43}$$

for all $s \in S^+_{db}$. By Definition 4.14, we finally get the equivalent property

$$eval_{ps}(\neg K\alpha \wedge \neg K\neg\alpha)(db) = true. \tag{4.44}$$

□

Corollary 4.18. Let α be a propositional sentence, $v \in \{true, false, undef\}$ a query value and db a database instance. Then we have

$$eval(\alpha)(db) = v \;\Leftrightarrow\; eval_{ps}(\Delta^*(\alpha, \{v\}))(db) = true.$$

Proof. Direct consequence of Lemma 4.17. Note that $\Delta^*(\alpha, \{v\}) = \Delta(\alpha, v)$. □

Corollary 4.19. Let α be a propositional sentence, and db a database instance. Then we have

$$eval_{ps}(eval^*(\alpha)(db))(db) = true.$$

Proof. Let $v := eval(\alpha)(db)$. Then the proposition follows from the definition of $eval^*$ (4.7) and Corollary 4.18. □

Corollary 4.20. Let α be a propositional sentence and db and db' two database instances such that

$$eval_{ps}(eval^*(\alpha)(db'))(db) = true.$$

Then we have $eval(\alpha)(db) = eval(\alpha)(db')$.

Proof. The proposition can be rewritten as

$$eval_{ps}(\Delta(\alpha, eval(\alpha)(db'))) = true, \tag{4.45}$$

which is by Lemma 4.17 equivalent to

$$eval(\alpha)(db) = eval(\alpha)(db'). \tag{4.46}$$

□

4.1.4 Alternative Semantics

The formal definition of $eval_{ps}$ given in Definition 4.14 – based on Kripke-semantics and the $\mathcal{M}_{DS}$-structure M_{db} derived from db – is a feasible approach when formally investigating the properties of Controlled Query Evaluation. On the other hand, it might make things difficult when implementing a CQE method based on epistemic potential secrets. For example, the enforcement methods for unknown policies presented in Chapter 7 need to consider which of the potential secrets are actually *true* in the given database instance. The necessity to generate M_{db} could be a major drawback for the implementation then.

Fortunately, $eval_{ps}$ can be equivalently replaced by an alternative, inductively defined evaluation function.

Lemma 4.21. The function $eval_{ps}^{ind}$, inductively defined as follows, is equivalent to $eval_{ps}$, i. e., for each $\mathcal{L}_{PS}$-sentence ϕ and database instance db, $eval_{ps}^{ind}(\phi)(db) = true$ if and only if $eval_{ps}(\phi)(db) = true$.

(1) $$eval_{ps}^{ind}(K\alpha)(db) := \begin{cases} true & \text{if } eval(\alpha)(db) = true \\ false & \text{otherwise} \end{cases}$$

(2) $$eval_{ps}^{ind}(\neg\phi)(db) := \begin{cases} true & \text{if } eval_{ps}^{ind}(\phi)(db) = false \\ false & \text{otherwise} \end{cases}$$

(3) $$eval_{ps}^{ind}(\phi \wedge \psi)(db) := \begin{cases} true & \text{if } eval_{ps}^{ind}(\phi)(db) = true \text{ and } eval_{ps}^{ind}(\psi)(db) = true \\ false & \text{otherwise} \end{cases}$$

(4) $$eval_{ps}^{ind}(\phi \vee \psi)(db) := \begin{cases} true & \text{if } eval_{ps}^{ind}(\phi)(db) = true \text{ or } eval_{ps}^{ind}(\psi)(db) = true \\ false & \text{otherwise} \end{cases}$$

Proof. By structural induction.

Assume that $\phi = K\alpha$, where α is propositional, and $eval_{ps}^{ind}(K)(\alpha) = true$. This is by rule (1) above equivalent to

$$eval(\alpha)(db) = true, \tag{4.47}$$

and by Lemma 4.17 equivalent to

$$eval_{ps}(K\alpha)(db) = true. \tag{4.48}$$

For the composed sentences $\neg\phi$, $\phi \wedge \psi$, and $\phi \vee \psi$, note that rules (2), (3) and (4) of this lemma correspond to the properties (1), (2) and (3) of Lemma 4.15. Thus, these sentences have the same truth value wrt. $eval_{ps}$ and $eval_{ps}^{ind}$, given that the particular components have the same truth value, which is ensured by the induction hypothesis. □

Using $eval_{ps}^{ind}$ instead of $eval_{ps}$ might be easier for a specific implementation of CQE: Given an $\mathcal{L}_{PS}$-sentence ϕ and a suitable parser for $\mathcal{L}_{PS}$, you can evaluate the atomic (K-prefixed) sub-formulas of ϕ with the means of $eval$ (i. e., logical implication in propositional logic), and then inductively determine the truth value of ϕ.

4.1.5 Epistemic potential secrets

The policy language $\mathcal{L}_{PS}$ is used to define the confidentiality policy. The elements of the confidentiality policy are called *epistemic potential secrets*. Basically, epistemic potential secrets have the same semantics as the propositional potential secrets introduced in Chapter 2: Given a potential secret Ψ, the user may not learn that Ψ is *true* in the actual database instance *db*; on the other hand, if Ψ is not *true* in *db*, this fact may be disclosed. The truth value of Ψ is determined by the means of the $eval_{ps}$ function. The modal operator K and the semantics of $\mathcal{L}_{PS}$ allow us to express and protect any combination of facts about truth values in *db*. For example, $(\neg Ka \wedge \neg K\neg a) \vee (K\neg b \wedge K\neg c)$ means that "a is *undef* or both b and c are *false* in *db*". These kinds of sentences couldn't be expressed with propositional potential secrets, and neither with confidentiality targets.

Definition 4.22 (Epistemic potential secret)**.** An *epistemic potential secret* is an $\mathcal{L}_{PS}$-sentence Ψ. We also say simply *potential secret* when the context is clear and confusion with propositional potential secrets is unlikely to occur.

Definition 4.23 (Confidentiality policy based on epistemic potential secrets)**.** A *confidentiality policy based on epistemic potential secrets* is a set $policy_{ps} = \{\Psi_1, \ldots, \Psi_m\}$ of $\mathcal{L}_{PS}$-sentences (potential secrets).

Example 4.24. The confidentiality policies (based on confidentiality targets) from Example 3.5 and Example 3.6 can be equivalently expressed by epistemic potential secrets as follows. The first policy states that the user may not learn that *aids* is true in the actual database instance, and may also not learn that *cancer* is true:

$$policy_{ps} \;:=\; \{\; K\,aids,\; K\,cancer \;\}$$

The second policy prohibits any disjunctive information about the truth value of *aids*:

$$\begin{aligned} policy'_{ps} \;:=\; \{\; & K\,aids \;\vee\; K\neg aids, \\ & K\,aids \;\vee\; \neg K\,aids \wedge \neg K\neg aids, \\ & K\neg aids \;\vee\; \neg K\,aids \wedge \neg K\neg aids \;\} \end{aligned}$$

As usual, we assume that the a priori assumptions are expressed in the same language as the confidentiality policy.

Definition 4.25 (A priori assumptions)**.** The user's *a priori assumptions* are a set $prior_{ps}$ of $\mathcal{L}_{PS}$-sentences.

4.2 Declarative Confidentiality

Given the concept of epistemic potential secrets, we can now formally define a CQE method for epistemic potential secrets as a function cqe_{ps}, similar to *cqe* (for propositional potential secrets) and cqe_{ct} (for confidentiality targets).

Definition 4.26 (CQE method for epistemic potential secrets)**.** A CQE method for epistemic potential secrets is a function

$$cqe_{ps}(Q, db, prior_{ps}, policy_{ps}) = \langle ans_1, \ldots, ans_n \rangle$$

with

$$ans_i \in \{true, false, undef, refuse\},$$

where Q is a query sequence, $prior_{ps}$ the user's a priori assumptions, db a database instance, and $policy_{ps}$ a confidentiality policy, given as a set of epistemic potential secrets.

Each method cqe_{ps} goes along with a function

$$precondition_{ps}(db, prior_{ps}, policy_{ps}) \in \{true, false\}$$

which defines the admissible arguments for cqe_{ps}.

We also define the notion of confidentiality in the usual way.

Definition 4.27 (Confidentiality for epistemic potential secrets)**.** Let cqe_{ps} be a CQE method for epistemic potential secrets with $precondition_{ps}$ as its associated precondition for admissible arguments. cqe_{ps} is defined to preserve confidentiality iff

for all finite query sequences Q,
for all instances db,
for all sets of potential secrets $policy_{ps}$,
for all potential secrets $\Psi \in policy_{ps}$,
for all a priori assumptions $prior_{ps}$
so that $(db, prior_{ps}, policy_{ps})$ satisfies $precondition_{ps}$,
there exists an instance db' and a set $policy'_{ps}$ of potential secrets
so that $(db', prior_{ps}, policy'_{ps})$ satisfies $precondition_{ps}$
and the following two conditions hold:

(a) [$(db, policy_{ps})$ and $(db', policy'_{ps})$ produce the same answers]
$cqe_{ps}(Q, db, prior_{ps}, policy_{ps}) = cqe_{ps}(Q, db', prior_{ps}, policy'_{ps})$

(b) [Ψ is false in db']
$eval_{ps}(\Psi)(db') = false$

In case the user is assumed to know the elements of $policy_{ps}$, we additionally demand that

(c) [same policy]
$policy_{ps} = policy'_{ps}$.

Just like the respective definitions for propositional potential secrets (cf. Definition 2.8) and confidentiality targets (cf. Definition 3.10), this notion of confidentiality is purely declarative. In the second part of this thesis, we will present a number of operational enforcement methods. In the remainder of this chapter, we show that confidentiality targets and propositional potential secrets can be handled by those methods as well, in case the methods meet certain requirements.

4.3 Reduction from Confidentiality Targets

First, we will show how a confidentiality policy $policy_{ct}$ based on confidentiality targets can be enforced by the means of a method cqe_{ps} for epistemic potential secrets. In particular, we will demonstrate how to convert a set of confidentiality targets into a set of $\mathcal{L}_{PS}$-sentences, and specify the requirements for cqe_{ps} in a way that the reduction preserves confidentiality.

We start with a basic property of the respective evaluation functions $eval_{ct}$ and $eval_{ps}$: They are equivalent when the Δ^* function is used to convert a confidentiality target (ψ, V) into an $\mathcal{L}_{PS}$-sentence.

Lemma 4.28. Let (ψ, V) be a confidentiality target and db a database instance. Then we have

$$eval_{ct}((\psi, V))(db) = eval_{ps}(\Delta^*(\psi, V))(db).$$

Proof. As both $eval_{ct}$ and $eval_{ps}$ return either *true* or *false*, the proposition can be rewritten as

$$eval_{ct}((\psi, V))(db) = true \;\Leftrightarrow\; eval_{ps}(\Delta^*(\psi, V))(db) = true. \tag{4.49}$$

According to Definition 3.4, the left hand side is equivalent to

$$eval(\psi)(db) \in V \tag{4.50}$$

or, equivalently,

$$(\exists v \in V)[\ eval(\psi)(db) = v\ ]. \tag{4.51}$$

According to Lemma 4.17, this is equivalent to

$$(\exists v \in V)[\ eval_{ps}(\Delta(\psi, v))(db) = true\ ]. \tag{4.52}$$

Since $\Delta^*(\psi, V)$ is the disjunction of $\Delta(\psi, v)$ for all $v \in V$, we can apply rule (4) of Definition 4.14, which results in the equivalent condition

$$eval_{ps}(\Delta^*(\psi, V))(db) = true. \tag{4.53}$$

□

The above stated relationship between $eval_{ct}$ and $eval_{ps}$ allows us to convert a set of confidentiality targets into a set of epistemic sentences, while still preserving the semantics wrt. the respective evaluation function.

Definition 4.29 (Function $conv_{ps}$)**.** We define a function $conv_{ps}$ which transforms a set of confidentiality targets into the corresponding epistemic sentences:

$$conv_{ps}(\{(\psi_1, V_1), \ldots, (\psi_k, V_k)\}) := \{\Delta^*(\psi_1, V_1), \ldots, \Delta^*(\psi_k, V_k)\} \tag{4.54}$$

All of the sentences generated by $conv_{ps}$ have a certain syntactical form, as they are generated by the Δ^* function. We call these sentences *ps-normal*.

Definition 4.30 (ps-normal $\mathcal{L}_{PS}$-sentences)**.** Let Ψ be an $\mathcal{L}_{PS}$-sentence. Ψ is defined as *ps-normal* iff there is a propositional sentence α and a non-empty value set $\emptyset \neq V \neq \{true, false, undef\}$ so that $\Psi = \Delta^*(\alpha, V)$. In other words, Ψ is ps-normal iff it has one of the following syntactical forms, where α is a propositional sentence:

(1) $K\alpha$

(2) $K\neg\alpha$

(3) $\neg K\alpha \wedge \neg K\neg\alpha$

(4) $K\alpha \quad \vee \quad K\neg\alpha$

(5) $K\alpha \quad \vee \quad \neg K\alpha \wedge \neg K\neg\alpha$

(6) $K\neg\alpha \quad \vee \quad \neg K\alpha \wedge \neg K\neg\alpha$

(Note that sentences of the form (2) are also sentences of the form (1) when we consider $\alpha' := \neg\alpha$.)

A set of $\mathcal{L}_{PS}$-sentences $policy_{ps} = \{\Psi_1, \ldots, \Psi_m\}$ is called ps-normal iff each $\Psi_i \in policy_{ps}$ is ps-normal.

Lemma 4.31. Let $policy_{ct}$ be a set of confidentiality targets. Then $conv_{ps}(policy_{ct})$ is ps-normal.

Proof. Follows immediately from the definition of the $conv_{ps}$ function (4.54): each element of the generated set of potential secret is established by the Δ^* function. □

Unfortunately, $conv_{ps}$ lacks a useful property: It is not invertible, i. e., when converting a set of confidentiality targets into a set of epistemic sentences, there is no function which converts these epistemic sentences back into confidentiality targets while ensuring that the result corresponds to the original set of confidentiality targets.

Lemma 4.32. $conv_{ps}$ is not invertible.

Proof. $conv_{ps}$ is not injective:

$$conv_{ps}(\{(\alpha, \{false\})\}) = \{K\neg\alpha\} = conv_{ps}(\{(\neg\alpha, \{true\})\}) \tag{4.55}$$

□

However, as $conv_{ps}$ is surjective (with the set of ps-normal $\mathcal{L}_{PS}$-sentences as the codomain), we can specify a right inverse:

Definition 4.33 (Function $\overline{conv_{ps}}$)**.** Let $\Sigma = \{\Psi_1, \ldots, \Psi_m\}$ be a set of ps-normal $\mathcal{L}_{PS}$-sentences. We define the function $\overline{conv_{ps}}$ that converts Σ into a set of confidentiality targets as follows:

$$\overline{conv_{ps}}(\Sigma) := \{(\alpha_1, V_1), \ldots, (\alpha_m, V_m)\}$$

with

$$(\alpha_i, V_i) := \begin{cases} (\alpha, \{true\}) & \text{if } \Psi_i = K\alpha \\ (\alpha, \{undef\}) & \text{if } \Psi_i = \neg K\alpha \wedge \neg K\neg\alpha \\ (\alpha, \{true, false\}) & \text{if } \Psi_i = K\alpha \quad \vee \quad K\neg\alpha \\ (\alpha, \{true, undef\}) & \text{if } \Psi_i = K\alpha \quad \vee \quad \neg K\alpha \wedge \neg K\neg\alpha \\ (\alpha, \{false, undef\}) & \text{if } \Psi_i = K\neg\alpha \quad \vee \quad \neg K\alpha \wedge \neg K\neg\alpha \end{cases}$$

where α is a propositional sentence. (Note that, by Definition 4.30, each ps-normal $\mathcal{L}_{PS}$-sentence has one of these syntactical forms – however, the first case captures both form (1) and form (2) specified in that definition.)

Lemma 4.34. Let $\Sigma = \{\Psi_i, \ldots, \Psi_n\}$ be a set of ps-normal $\mathcal{L}_{PS}$-sentences. Then we have

$$conv_{ps}(\overline{conv_{ps}}(\Sigma)) = \Sigma.$$

Proof. Follows directly from Definition 4.30 and Definition 4.33. □

The idea of our reduction is to convert a confidentiality policy $policy_{ct}$ into a ps-normal set of epistemic potential secrets $policy_{ps}$, which is then used as the input to a suitable enforcement method cqe_{ps} for epistemic potential secrets. The confidentiality property of this enforcement method will then ensure that there is an alternative database instance db' and an alternative set of epistemic potential secrets $policy'_{ps}$ such that the usual conditions (precondition satisfied, same answers returned, given potential secret is false, possibly the same policy used) hold. If $policy'_{ps}$ is ps-normal, we can use $\overline{conv_{ps}}$ in order to convert $policy'_{ps}$ into a set of confidentiality targets $policy'_{ct}$. db' and $policy'_{ct}$ will then satisfy the conditions of the confidentiality definition for confidentiality targets. However, it must be guaranteed that $policy'_{ps}$ is in fact ps-normal. We call this property *ps-normality-preserving*: If the original set of epistemic potential secrets $policy_{ps}$ is ps-normal, then there is also a suitable alternative set $policy'_{ps}$ which is also ps-normal. Enforcement methods which preserve ps-normality can be used for the reduction. We can formally define ps-normality-preservation by means of a stronger precondition.

Definition 4.35 (ps-normality-preserving)**.** A method cqe_{ps} with the associated precondition $precondition_{ps}$ is called *ps-normality-preserving* iff it preserves confidentiality in the sense of Definition 4.27 under the stronger precondition

$$precondition_{ps}^{\text{ps-normal}}(db, prior_{ps}, policy_{ps}) := \\ precondition_{ps}(db, prior_{ps}, policy_{ps}) \ \wedge \ \text{“}policy_{ps} \text{ is ps-normal”}.$$

Note that preservation of ps-normality does not imply that the respective method also preserves "ordinary" confidentiality in the sense of Definition 4.27, as Definition 4.35 does not state anything about non-ps-normal policies. Thus, both properties must be shown individually for each method. However, in case we assume that the user knows the confidentiality policy, the converse is true: Each confidentiality preserving method does also preserve ps-normality.

Lemma 4.36. Assuming that the user knows the set of potential secrets, each confidentiality-preserving method cqe_{ps} is also ps-normality-preserving.

Proof. Follows trivially from condition (c) of Definition 4.27, which demands that the alternative set of potential secrets $policy'_{ps}$ corresponds to the original set $policy_{ps}$. □

We can now specify the main theorem of this section, which formally defines the reduction and proves its confidentiality.

Theorem 4.37 (Reduction from confidentiality targets to epistemic potential secrets)**.** Let cqe_{ps} be an enforcement method for Controlled Query Evaluation based on epistemic potential secrets, preserving ps-normality in the sense of Definition 4.35. Let $precondition_{ps}$ be the associated precondition. Then the function

$$cqe_{ct}(Q, db, prior_{ct}, policy_{ct}) := cqe_{ps}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

with the precondition

$$precondition_{ct}(db, prior_{ct}, policy_{ct}) := \\ precondition_{ps}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct})) \quad (4.56)$$

preserves confidentiality wrt. confidentiality targets in the sense of Definition 3.10.

Proof. Let db be a database instance, $policy_{ct}$ a set of confidentiality targets, $prior_{ct}$ the a priori assumptions so that the pertinent $precondition_{ct}$ is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence. Let $(\psi, V) \in policy_{ct}$ be a confidentiality target.

As $precondition_{ct}(db, prior_{ct}, policy_{ct})$ is satisfied, the equivalent condition

$$precondition_{ps}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct})) \quad (4.57)$$

is satisfied as well. Furthermore, $conv_{ps}(policy_{ct})$ is ct-normal. So

$$precondition_{ps}^{\text{ps-normal}}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct})) \quad (4.58)$$

is also satisfied. Then, by Definition 4.35, there exists a database instance db' and a set of epistemic potential secrets $policy'_{ps}$, so that – in particular – $precondition_{ps}^{\text{ps-normal}}(db, conv_{ps}(prior_{ct}), policy'_{ps})$ is satisfied, and $policy'_{ps}$ must be ps-normal.

The construction of $policy'_{ct}$ depends on the question whether the user is assumed to know the confidentiality policy or not.

Case 1 (The user knows the confidentiality policy)**.** In this case, condition (c) of Definition 4.27 guarantees that $policy'_{ps} = policy_{ps}$. We can then choose $policy'_{ct} = policy_{ct}$, which satisfies condition (c) of Definition 3.10. Furthermore, it is trivial that

$$conv_{ps}(policy'_{ct}) = conv_{ps}(policy_{ct}) = policy_{ps} = policy'_{ps}. \tag{4.59}$$

Case 2 (The user does not know the confidentiality policy)**.** Then we construct a set $policy'_{ct}$ of confidentiality targets as follows:

$$policy'_{ct} := \overline{conv_{ps}}(policy'_{ps}). \tag{4.60}$$

By Lemma 4.34, we then have

$$conv_{ps}(policy'_{ct}) = policy'_{ps}. \tag{4.61}$$

Finally, we show that db' and $policy'_{ct}$ also satisfy the precondition and conditions (a) and (b) of Definition 3.10.

- By $precondition_{ps}^{\text{ps-normal}}(db, conv_{ps}(prior_{ct}), policy'_{ps})$, we have

  $$precondition_{ps}(db, conv_{ps}(prior_{ct}), policy'_{ps}), \tag{4.62}$$

  which is by (4.59) (or (4.61), respectively) equivalent to

  $$precondition_{ps}(db, conv_{ct}(prior_{ct}), conv_{ct}(policy'_{ct})), \tag{4.63}$$

  and by (4.56) also equivalent to

  $$precondition_{ct}(db', prior_{ct}, policy'_{ct}). \tag{4.64}$$

  So the precondition is satisfied for db' and $policy'_{ct}$.

- By condition (a) of Definition 4.27, $(db, conv_{ps}(policy_{ct}))$ and $(db', policy'_{ps})$ produce the same answers, so we have

  $$\begin{aligned} & cqe_{ct}(Q, db, prior_{ct}, policy_{ct}) && (4.65)\\ = \; & cqe_{ps}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct})) && (4.66)\\ = \; & cqe_{ps}(Q, db', conv_{ps}(prior_{ct}), policy'_{ps}) && (4.67)\\ = \; & cqe_{ps}(Q, db', conv_{ps}(prior_{ct}), conv_{ps}(policy'_{ct})) && (4.68)\\ = \; & cqe_{ct}(Q, db', prior_{ct}, policy'_{ct}). && (4.69) \end{aligned}$$

  This satisfies condition (a) of Definition 3.10.

- By condition (b) of Definition 4.27, we have

  $$eval_{ps}(\Delta^*(\psi, \{true\}))(db') = false. \tag{4.70}$$

  By Lemma 4.28, this is equivalent to

  $$eval_{ct}((\psi, V))(db') = false. \tag{4.71}$$

  This satisfies condition (b) of Definition 3.10.

Thus, all of the conditions of Definition 3.10 are satisfied for db' and $policy'_{ct}$. □

4.4 Reduction from Propositional Potential Secrets

In the previous section, we showed how to employ an enforcement method for epistemic potential secrets for the protection of a confidentiality policy based on confidentiality targets. In this section, we will investigate yet another reduction: from propositional potential secrets (introduced in Chapter 2) to epistemic potential secrets. This can be achieved by linking the two reductions presented above: First from propositional potential secrets to confidentiality targets (Section 3.3), and then from confidentiality targets to epistemic potential secrets (Section 4.3).

For simplicity and easier reading, we use three abbreviating notations:

- A CQE method for propositional potential secrets is called a *pps-method*.
- A CQE method for confidentiality targets is called a *ct-method*.
- A CQE method for epistemic potential secrets is called a *eps-method*.

Theorem 3.17 states that a ct-method can be used for the reduction from propositional potential secrets if it is *ct-normality-preserving*, i. e., there is a ct-normal alternative confidentiality policy. Likewise, Theorem 4.37 states that an eps-method can be used for the reduction from confidentiality targets if it is *ps-normality-preserving*, i. e., there is a ps-normal alternative confidentiality policy, which means that it can be converted into a set of confidentiality targets $policy'_{ct} := \overline{conv_{ps}}(policy'_{ps})$. However, in order to concatenate these two reductions as

$$\begin{aligned} cqe(Q, db, prior, policy) &:= cqe_{ct}(Q, db, conv_{ct}(prior), conv_{ct}(policy)) \\ &:= cqe_{ps}(Q, db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy))) \end{aligned}$$

we need to make sure that $policy'_{ct} = \overline{conv_{ps}}(policy'_{ps})$ can be further converted into a set of propositional sentences, i. e., $policy'_{ct}$ is ct-normal. This is possible iff $policy'_{ps}$ only contains epistemic sentences of the form $K\psi$, where ψ is a propositional sentence. $policy'_{ps}$ is then called *ps/ct-normal*.

Definition 4.38 (ps/ct-normal $\mathcal{L}_{PS}$-sentences)**.** Let Ψ be an $\mathcal{L}_{PS}$-sentence. Ψ is defined as *ps/ct-normal* iff there is a propositional sentence ψ such that $\Psi = \Delta^*(\psi, \{true\})$. In other words, Ψ is ps/ct-normal iff it has the syntactical form $K\psi$, where ψ is a propositional sentence.

A set of $\mathcal{L}_{PS}$-sentences $policy_{ps} = \{\Psi_1, \ldots, \Psi_m\}$ is called ps/ct-normal iff each $\Psi_i \in policy_{ps}$ is ps/ct-normal.

Lemma 4.39. Let $policy_{ct}$ be a ct-normal set of confidentiality targets. Then $conv_{ps}(policy_{ct})$ is ps/ct-normal.

Proof. By the definition of the $conv_{ps}$ function (4.54), each element of $conv_{ps}(policy_{ct})$ is established by the Δ^* function. Additionally, as the V-component of each confidentiality target (ψ, V) is either $\{true\}$ or $\{false\}$, the resulting potential secret is either $\Delta^*(\psi, \{true\})$ or $\Delta^*(\psi, \{false\})$. In the latter case, the potential secret is equivalent to $\Delta^*(\neg\psi, \{true\})$. □

Corollary 4.40. Let *policy* be a set of propositional sentences. Then $conv_{ps}(conv_{ct}(policy)))$ is ps/ct-normal.

Proof. Follows from Lemma 3.14 and Lemma 4.39. □

Lemma 4.41. Let $policy'_{ps}$ be ps/ct-normal. Then $\overline{conv_{ps}}(policy'_{ps})$ is ct-normal.

Proof. By the ps/ct-normality, each sentence in $policy'_{ps}$ has the syntactical form $K\psi$, where ψ is a propositional sentence. Thus, by Definition 4.33, each confidentiality target in $\overline{conv_{ps}}(policy'_{ps})$ has the form $(\psi, \{true\})$. □

As usual, we can now define the notion of *preservation of ps/ct-normality* by specifying a stronger precondition.

Definition 4.42 (ps/ct-normality-preserving)**.** An eps-method cqe_{ps} with the associated precondition $precondition_{ps}$ is called *ps/ct-normality-preserving* iff it preserves confidentiality in the sense of Definition 4.27 under the stronger precondition

$$\begin{aligned} precondition_{ps}^{\text{ps/ct-normal}}(db, prior_{ps}, policy_{ps}) := \\ precondition_{ps}(db, prior_{ps}, policy_{ps}) \ \wedge \ \text{“}policy_{ps} \text{ is ps/ct-normal”}. \end{aligned}$$

Again, this definition is independent from the preservation of “ordinary” confidentiality according to Definition 4.27, but we can state an inclusion property for the case of a known policy:

Lemma 4.43. Assuming that the user knows the set of potential secrets, each confidentiality-preserving method cqe_{ps} is also ps/ct-normality-preserving.

Proof. Follows immediately from condition (c) of Definition 4.27, which demands that the alternative set of potential secrets $policy'_{ps}$ corresponds to the original set $policy_{ps}$. □

We now construct the reduction from propositional potential secrets to epistemic potential secrets. The first step is to show that an eps-method which preserves ps/ct-normality can be used to derive a ct-normality-preserving ct-method.

Theorem 4.44 (Reduction from ct-normal confidentiality targets to epistemic potential secrets)**.** Let cqe_{ps} be an eps-method, preserving ps/ct-normality in the sense of Definition 4.42. Let $precondition_{ps}$ be the associated precondition. Then the function

$$cqe_{ct}(Q, db, prior_{ct}, policy_{ct}) := cqe_{ps}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

with the precondition

$$\begin{aligned} precondition_{ct}(db, prior_{ct}, policy_{ct}) := \\ precondition_{ps}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct})) \end{aligned} \tag{4.72}$$

preserves ct-normality confidentiality wrt. confidentiality targets in the sense of Definition 3.15.

Proof. We follow the outline of the proof of Theorem 4.37 and only state the changed assumptions and effects.

We now assume that $policy_{ct}$ is ct-normal (due to the stronger precondition stated in Definition 3.15). According to Lemma 4.39, $conv_{ps}(policy_{ps})$ is then ps/ct-normal. As cqe_{ps} is ps/ct-normality-preserving, $policy'_{ps}$ is ps/ct-normal was well. Thus, by Lemma 4.41, $\overline{conv_{ps}}(policy'_{ps})$ is ct-normal. This means that $precondition_{ct}^{\text{ct-normal}}(db', prior_{ct}, policy'_{ct})$ is satisfied.

The remainder of the proof corresponds to the one from Theorem 4.37. □

Corollary 4.45 (Reduction from propositional to epistemic potential secrets)**.** Let cqe_{ps} be an enforcement method for Controlled Query Evaluation based on epistemic potential secrets, preserving both confidentiality in the sense of Definition 4.27 and ps/ct-normality in the sense of Definition 4.42. Let $precondition_{ps}$ be the associated precondition. Then the function

$$cqe(Q, db, prior, policy) := cqe_{ps}(Q, db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

with the precondition

$$precondition(db, prior, policy) := \\ precondition_{ps}(db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

preserves confidentiality wrt. propositional potential secrets in the sense of Definition 2.8.

Proof. Follows from Theorem 4.44 and Theorem 3.17. □

As we have shown, a ps/ct-normal eps-method can be used to handle any kind of confidentiality policy: propositional potential secrets, confidentiality targets and of course epistemic potential secrets. We are now ready for the second part of this thesis, where we introduce an operational framework for eps-methods and construct several specific enforcement methods, most of which are in fact ps/ct-normal.

Part II

Operational Layer

5 An Operational Framework for Epistemic Potential Secrets

In the previous chapter, we defined confidentiality policies based on epistemic potential secrets, gave a formal but abstract definition of an enforcement method cqe_{ps} for these policies, and introduced the notion of confidentiality in Definition 4.27. Furthermore, we proved a method for epistemic potential secrets to be employable for the protection of both policies based on propositional potential secrets and policies based on confidentiality targets, given a certain property, which we call *normality-preservation*.

In the second part of this thesis, we show how to construct enforcement methods for epistemic potential secrets that operationally meet the requirements stated in Definition 4.27. The presentation consists of two steps: First, we introduce a general framework acting as the foundation of all enforcement methods, while at the same time leaving a few parameters open, which have to be filled in according to the requirements of the target method. In particular, these parameters are the functions *inference*, *violates*, *censor*, and possibly *precondition*. This framework is given in the current chapter. We also present different versions of some of the above mentioned functions. In Chapter 6 and Chapter 7, we will then instantiate the framework by selecting or specifying the missing functions, and prove the confidentiality property of the resulting enforcement methods.

5.1 Basic Architecture

The architecture of our framework is visualized in Figure 5.1. It can be summarized as follows:

- The system keeps a *log file* log_i which represents the information disclosed to the user at a given time i, i. e., up to the point where the i-th query Φ_i has been answered. The log file is initialized with the a priori assumptions $log_0 = prior_{ps}$. After each query Φ_i, the information provided by the answer ans_i is added to the log file. This information is called the *inference* from that answer. It is left to the enforcement method to decide what information to keep from an answer. Some will only store the (encoded) answer, some will also account for additional *meta inferences*.
- When the user issues a query Φ_i, the system tentatively adds any possible inference to the log file.
- It then checks which of these inferences would generate a log file that *violates* the confidentiality policy. The exact definition of a security violation depends on the enforcement method again.

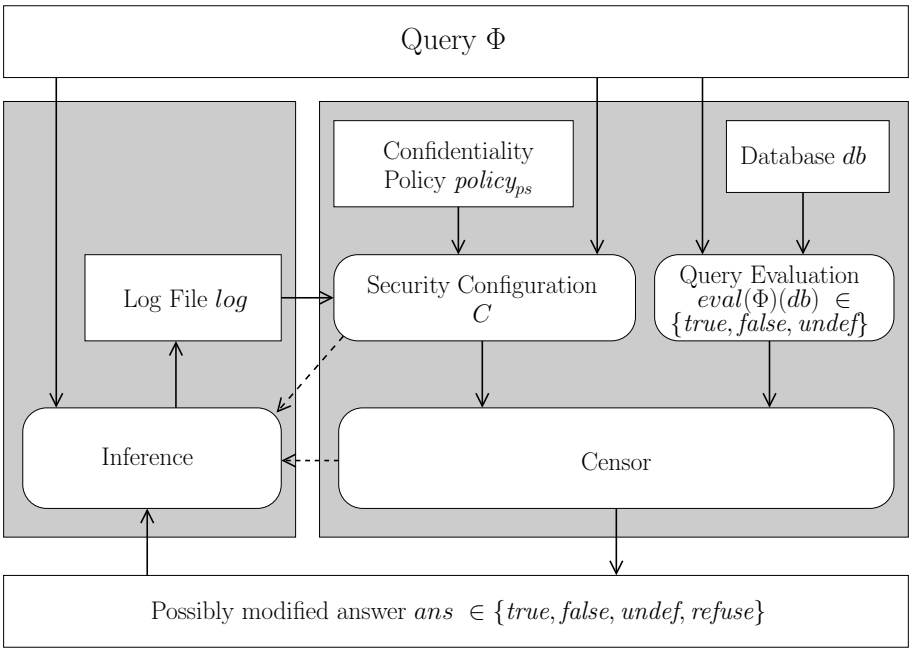

Figure 5.1: The operational framework for epistemic potential secrets

- All of these "violating" inferences are gathered in a set which is called the *security configuration* C of the query in the given situation.
- The security configuration C and the actual query value $eval(\Phi_i)(db)$ are passed to a *censor function* which chooses an appropriate answer ans_i.
- The answer is passed to the user, and the inference from the answer is stored in the log file.

In the following sections, we give a formal specification of this framework. We define a number of functions for the various steps, and show how they interact with each other. Some of the functions have a different implementation depending on the enforcement method. In that case, we present all possible versions in this section, and each enforcement method presented in Chapters 6 and 7 will later pick one of these.

5.2 Inferences and the Log File

The main advantage of Controlled Query Evaluation over existing static approaches is the fact that it addresses the *inference problem*: The user may not combine an answer given by the system with information gained from previous answers in order to infer any of the potential secrets. Obviously, this goal can only be achieved if the systems keeps track of the information already disclosed to the user throughout the query sequence. In

this section, we introduce a log file which contains this information, and show how it is encoded using *epistemic logic*.

We denote the log file by log and consider its state log_i at a given time i. Prior to the first query, we have the initial log file log_0, containing the (possibly empty) set of a priori assumptions $prior_{ps}$. Then, after each query Φ_i, the subsequent log file log_i is established by logging the query and a representation of the corresponding answer ans_i or, generally speaking, the information provided by this answer ans_i. We denote the latter as the *inference* from that answer, generally defined as the set of values that appear possible wrt. the actual query value, and formalized as a *value set* $\emptyset \neq V \subseteq \{true, false, undef\}$ (which we have introduced in the context of confidentiality targets, cf. Definition 3.1). In this context, we also speak of *inference sets* and classify them by their cardinality:

1. Unary inference sets ($\{true\}$, $\{false\}$, and $\{undef\}$): These represent *definite information* about the query value. For example, the inference set $\{true\}$ represents the information "the actual query value is *true*, but not *false* or *undef*".

2. Binary inference sets ($\{true,false\}$, $\{true,undef\}$, and $\{false,undef\}$). These represent *disjunctive information*: For example, $\{true,false\}$ means that "the actual query value is *true* or *false*, but not *undef*".

3. The set $\{true, false, undef\}$ represents *no information* about the actual query value: any value appears possible.

The methods described in Chapters 6 and 7 employ two different strategies when determining the inference set from an answer ans: *answer inferences* and *meta inferences*. In the remainder of this section, we will briefly discuss the main ideas of these two strategies, and provide a formalization as a function

$$inference(censor, C, ans)$$

where ans is the answer, $censor$ the algorithm of the censor function employed, and C the *security configuration*, introduced in Section 5.3 below, which describes the threats to confidentiality in the given situation. The censor function is formally introduced in Section 5.4 below; in a nutshell, it chooses a suitable answer according to the current security configuration and the actual query value. It is also important to be aware of the range of a particular *inference* function, so that the censor function can enumerate and analyze any possible inference. We denote the range of *inference* by $\mathcal{I}$.

Answer inferences It is assumed that each non-refused answer $ans \in \{true, false, undef\}$ provides the information that the query value is exactly ans (although this might be a lie), so the inference set would be $V := \{ans\}$. A refused answer is assumed to carry no "useful" information, which we express by $V := \{true, false, undef\}$. This is formalized by the function

$$inference^{ans}(censor, C, ans) := \begin{cases} \{ans\} & \text{if } ans \in \{true, false, undef\} \\ \{true, false, undef\} & \text{otherwise} \end{cases}$$

(5.1)

with the range

$$\mathcal{I}^{ans} = \{\{true\}, \{false\}, \{undef\}, \{true, false, undef\}\}.$$

This is the most simple way to keep a log file of the past queries and answers, though it might discard certain facts (for example, that a certain answer was refused). Thus, we have to make sure that any enforcement method making use of this approach does not rely on the possibly missing information. Most of the enforcement methods presented in Chapter 6 and Chapter 7 rely on this approach: the uniform lying method and the combined lying and refusal method for known policies, as well as the uniform lying method and the uniform refusal method for unknown policies.

Meta inferences Instead of simply "consuming" an answer ans_i, a highly sophisticated user might reflect on the origin of that answer and consider the following question:

> What is the set of actual query values $v \in \{true, false, undef\}$ that would have resulted in this answer ans_i?

In other words, the user might compute the pre-image of the answer ans_i wrt. the possible query values in the actual database instance *db*. This is formalized by the function

$$inference^{meta}(censor, C, ans) := \{ v \mid v \in \{true, false, undef\} \text{ and } censor(C, v) = ans \} \quad (5.2)$$

with the range

$$\mathcal{I}^{meta} = \mathfrak{P}^{+}(\{true, false, undef\})$$

where $\mathfrak{P}^{+}$ denotes the power set operator, excluding the empty set (the empty set cannot occur as there is always at least one value that would result in the observed answer).

Of course, calculating this pre-image requires certain knowledge: First, the user needs to be aware of the algorithm of the enforcement method, which we assume, as we follow the principle of open design. Second, the user needs knowledge about the other input parameters of the cqe_{ps} function, in particular the set of potential secrets $policy_{ps}$. If we assume that this confidentiality policy is known to the user, it is possible to draw this kind of meta inferences. This is demonstrated in Section 6.2.

The meta inferences approach is employed by the uniform refusal methods for known potential secrets. In particular, it allows us to account for meta inference from refused answers. However, it involves a certain overhead, as we need to consider a larger number of inference sets when calculating the security configuration (note that $|\mathcal{I}^{meta}| = 7$, while $|\mathcal{I}^{ans}| = 4$).

Having determined the inference V from an answer ans to query Φ, we need to find a way to store it in the log file log, along with the query Φ that caused this inference. The log file is a set of $\mathcal{L}_{PS}$-sentences. In Section 4.1, we have already demonstrated how to convert a pair (Φ, V) (in that context, a confidentiality target) into an $\mathcal{L}_{PS}$-sentence: by the means of the function Δ^* (4.6). We reuse this function here, and the update of the log file can be formalized as

$$log_i := log_{i-1} \cup \{\Delta^*(\Phi_i, V_i)\}$$

where Φ_i is the i-th query, and $V_i := inference(censor, C_i, ans_i)$ is the inference from the answer ans_i.

5.3 Security Violations

In the previous section, we showed how the information gained from the answers are stored in the log file: At any time i, the log file log_i is a representation of the information that has already been disclosed to the user, plus his initial assumptions $prior_{ps}$. The goal of Controlled Query Evaluation is to keep the user from inferring any of the potential secrets from the information disclosed to him. By means of the log file, we state that log may not *violate* the confidentiality policy at any time.

In general, having encoded both the log file and the potential secrets as $\mathcal{L}_{PS}$-sentences, the violation of the confidentiality policy can be expressed by the means of *logical implication*: a log file log violates a confidentiality policy (given as a set of potential secrets) if at least one potential secret is logically implied by log. However, this condition must be hardened or can be weakened for some of the methods presented in Chapters 6 and 7. Thus, we abstractly define a function

$$violates(db, log, policy_{ps}) \in \{true, false\}$$

where db is the database instance, log the log file and $policy_{ps}$ the set of potential secrets. In total, there are four different versions of this function. The first two versions protect *all* potential secrets, and are therefore independent from the database instance db:

Violation wrt. a single potential secret The log file log is defined to violate the confidentiality policy $policy_{ps}$ iff at least one potential secret $\Psi \in policy_{ps}$ is logically implied by log:

$$violates^{single}(db, log, policy_{ps}) := (\exists \Psi \in policy_{ps})[\ log \models_{S5} \Psi\ ] \tag{5.3}$$

This is the basic condition, employed by the uniform refusal method for known potential secrets and the combined lying and refusal method for known potential secrets.

Violation wrt. the disjunction of all potential secrets The log file log is defined to violate the confidentiality policy $policy_{ps}$ iff it logically implies the *disjunction* of all potential secrets from $policy_{ps}$:

$$violates^{disj}(db, log, policy_{ps}) \ := \ log \models_{S5} pot\_sec\_disj \tag{5.4}$$

with

$$pot\_sec\_disj := \bigvee_{\Psi \in policy_{ps}} \Psi \tag{5.5}$$

This is a stricter condition than $violates^{single}$, as the log file is even considered violating if only the fact is disclosed that at least one secret must be *true*, without revealing which secret it is. This function is employed by the uniform lying censor for known policies in order to prevent a "hopeless situation" in which no answer may be given without disclosing at least one potential secret. Details are given in Section 6.1.2 below.

When the user is assumed *not* to know the elements of the confidentiality policy, we can use a weaker version of either function: We only consider those potential secrets which are actually *true* in the given database instance *db*. The two enforcement methods for unknown policies presented in Chapter 7 are constructed according to this heuristics.

Violation wrt. a single true potential secret The log file *log* is defined to violate the confidentiality policy $policy_{ps}$ iff at least one *true* potential secret $\Psi \in policy_{ps}$ is logically implied by *log*:

$$violates^{truesingle}(db, log, policy_{ps}) := \\ (\exists \Psi \in policy_{ps})[\ eval_{ps}(\Psi)(db) = true \text{ and } log \models_{S5} \Psi\ ] \tag{5.6}$$

This function is employed by the uniform refusal method for unknown potential secrets.

Violation wrt. the disjunction of all true potential secrets The log file *log* is defined to violate the confidentiality policy $policy_{ps}$ iff it logically implies the *disjunction* of all *true* potential secrets from $policy_{ps}$, formalized as

$$violates^{truedisj}(db, log, policy_{ps}) \ := \ log \models_{S5} true\_pot\_sec\_disj \tag{5.7}$$

with

$$true\_pot\_sec\_disj \ := \ \begin{cases} \bigvee true\_pot\_sec & \text{if } true\_pot\_sec \neq \emptyset \\ \bot & \text{if } true\_pot\_sec = \emptyset \end{cases} \tag{5.8}$$

and

$$true\_pot\_sec \ := \ \{\ \Psi \in policy_{ps} \ \mid \ eval_{ps}(\Psi)(db) = true\ \}. \tag{5.9}$$

This function is employed by the uniform lying censor for unknown potential secrets, again to avoid the "hopeless situation" in which any answer would be harmful.

We have now formally defined the notion of a security violation, by means of the function *violates*. Before returning an answer ans_i to a query Φ_i, we have to make sure that the resulting log file

$$log_{i-1} \cup \{\Delta^*(\Phi_i, inference(censor, C_i, ans_i))\}$$

will not result in this kind of violation. The censors we construct avoid these security violations as follows: First, all inference sets which would (combined with the current log file) lead to such a violation are identified. We call this set of all violating inference sets the *security configuration* of the given situation. Then, an answer is chosen which does not result in one of the inference sets from the security configuration.

Definition 5.1 (Security configuration)**.** Let $policy_{ps}$ be a set of potential secrets, *log* a log file and Φ a query. Further, let $\mathcal{I}$ be the range of the *inference* function associated with the respective enforcement method, i. e., the set of all inference sets that may occur under this version of *inference*. The set

$$sec\_conf(db, log, policy_{ps}, \Phi) := \{\ V \mid V \in \mathcal{I} \text{ and } violates(db,\ log \cup \{\Delta^*(\Phi, V)\},\ policy_{ps})\ \} \quad (5.10)$$

of all inference sets that would lead to a violation of the confidentiality policy is called the *security configuration* of the query Φ wrt. $policy_{ps}$ and *log* under the database instance *db*.

Note that the range of *sec_conf* is limited by four constraints:

1. The range $\mathcal{I}$ of the underlying *inference* function. Only inference sets produced by this function may occur as elements of a security configuration C.

2. The exact definition of the underlying *violates* function. As mentioned earlier and demonstrated in Section 6.1.2 and below, the uniform lying methods protect the disjunction of all (or all true) potential secrets in order to avoid a "hopeless situation" in which neither answer may be given without revealing at least one potential secret. By avoiding this situation, the inferences $\{true\}$, $\{false\}$ and $\{undef\}$ will never be harmful at the same time, and thus $\{\{true\}, \{false\}, \{undef\}\}$ cannot occur as a security configuration.

3. The fact that disjunctive information can only be harmful if the more precise information is: If it is harmful to learn that the value of a query is either v_1 or v_2, it is also harmful to learn that the value of a query is exactly v_1 (or exactly v_2). Thus, an inference set $\{v_1, v_2\}$ can only occur as an element of a security configuration C if both of its non-empty subsets $\{v_1\}$ and $\{v_2\}$ are elements of C as well.

4. The fact that the inference set $\{true, false, undef\}$ can only be harmful if also the preceding log file has been violating the confidentiality policy, which we avoid by appropriate preconditions, and which we keep as an invariant later on. (Note that

$$\Delta^*(\Phi_i, \{true, false, undef\}) = K\Phi_i \vee K\neg\Phi_i \vee (\neg K\Phi_i \wedge \neg K\neg\Phi_i)$$

is a tautology and has no impact on the logical consequences when added to the log file.) Thus, this inference set will never occur as an element of a security configuration.

Note that, in principle, each method could choose an arbitrary *inference* function and thus produce arbitrary log files which might neglect certain pieces of information disclosed to the user. For example, remember that the answer inference mechanism presented above discards the information about refused answers. This will only work if the corresponding censor functions do not rely on the missing information. As a result, the fact that the log file, according to *violates*, does not violate the confidentiality policy does not necessarily guarantee that confidentiality is preserved. Thus, for each method, we have to prove the formal conditions required by Definition 4.27, considering the other components of the method as well.

5.4 The Censor

The censor is the core of Controlled Query Evaluation. Based on the components presented above, it decides whether an answer to a query needs to be distorted in order to preserve confidentiality, and, if so, in which manner. In order to come to a decision, the censor considers two questions:

1. What is the security configuration of the query, i. e., what are the inferences to avoid in order to preserve confidentiality?
2. What is the actual value of the query?

Thus, we can formally define a censor as a function with two parameters.

Definition 5.2 (Censor function)**.** A censor is a function

$$censor: \ \mathfrak{P}(\mathfrak{P}^+(\{true, false, undef\})) \times \{true, false, undef\} \rightarrow \{true, false, undef, refuse\}$$

which assigns an answer to each combination of a (valid) security configuration and an actual query value. $\mathfrak{P}$ and $\mathfrak{P}^+$ denote the power set operator, including and excluding the empty set, respectively.

The actual definition of the censor function depends on the constraints of the given enforcement method, and is presented in the various sections of Chapter 6 and Chapter 7. Moreover, some elements of $\mathfrak{P}(\mathfrak{P}^+(\{true, false, undef\}))$ will never occur as a security configuration under a specific method, according to the constraints mentioned in Section 5.3, in particular depending on the exact definition of the functions *violates* and *inference*. The set of valid security configurations under a specific method – i. e., the range of *sec_conf* considering its underlying functions *violates* and *inference* – is denoted by C^*, and we implicitly allow the function value of a specific *censor* function to be undefined for any other input (X, v) with $X \notin C^*$ and $v \in \{true, false, undef\}$.

5.5 Putting it All Together

As introduced in Chapter 4, a CQE method is a function

$$cqe_{ps}(Q, db, prior_{ps}, policy_{ps}) := \langle ans_1, \ldots, ans_n \rangle, \tag{5.11}$$

where $Q = \langle \Phi_1, ..., \Phi_n \rangle$ is a query sequence, $prior_{ps} = log_0$ the initial log file, db a database instance, and $policy_{ps}$ a set of potential secrets. Unless otherwise noted, we demand that the a priori assumptions $prior_{ps}$ do not violate the confidentiality policy in the first place:

$$precondition_{ps}(db, prior_{ps}, policy_{ps}) \ := \ \neg violates(db, prior_{ps}, policy_{ps}) \tag{5.12}$$

Note that this precondition does *not* require $prior_{ps}$ to be true in db. The system is also able to deal with situations where the user believes in "false" information. The answers will be adjusted accordingly, which means that true, harmless answers might be distorted in order to match the false assumptions. This design is debatable – one could argue that the a priori assumptions will never be inconsistent with the actual database instance, in particular if they only reflect semantic constraints. We however believe that this strict kind of precondition is not necessary, especially for the lying methods; as soon as the first lie is returned, the log file will become inconsistent with the actual database instance anyway. Due to the way the alternative database instance db' is constructed, we also ensure that db' makes all sentences from log_n (and thus also $prior_{ps}$) true (cf. Definition 5.3 and Lemma 5.5 below). Thus, the user will be able to imagine a database instance which does not only satisfy the conditions mentioned in Definition 4.27, but which is also consistent with his a priori assumptions.

After each query Φ_i, the answer ans_i and the subsequent log file log_i are generated as follows:

1. Determine the security configuration (implicitly using the function *violates*):

   $$C_i := sec\_conf(db, log_{i-1}, policy_{ps}, \Phi_i)$$

2. Let the censor choose the answer:

   $$ans_i := censor(C_i, eval(\Phi_i)(db))$$

3. Add the inference to the log file, encoded as an epistemic sentence:

   $$log_i := log_{i-1} \cup \{\Delta^*(\Phi_i, inference(censor, C_i, ans_i))\} \tag{5.13}$$

An enforcement method is completely identified if the functions *inference*, *violates* and *censor* are specified. Although *censor* is only one of them, we will often refer to these functions as "the algorithm of the censor".

5.6 Proving Confidentiality

According to Definition 4.27, a method for Controlled Query Evaluation preserves confidentiality if – under any "appropriate" input parameters, and for an arbitrary potential secret $\Psi \in policy_{ps}$ – there is always an indistinguishable database instance db' in which Ψ is *false*, under which the same answers are returned, and both even under the same confidentiality policy, in case it is assumed to be known by the user. We will have to prove this confidentiality property individually for each of the enforcement methods presented in Chapter 6 and Chapter 7. However, the proofs are similar to a certain extend, and share some common ideas. The purpose of this section is to present the main common proof idea, and provide some lemmas which will later help to streamline the proofs.

In particular, we have to construct the "alternative" database instance db'. This is achieved by means of the log file. For each method, it will be shown that the final log file log_n does not imply any potential secret Ψ (or any true potential secret Ψ, in case of a method for unknown policies):

$$log_n \not\models_{S5} \Psi \tag{5.14}$$

Then, by the definition of the logical implication operator $\models_{S5}$, there must be at least one $\mathcal{M}_{DS}$-structure $M = (S, \mathcal{K}, \pi)$ and a state $s \in S$ so that (M, s) is a model of log_n but not a model of Ψ. We can then construct a database instance $db_{M,s}$ from M and s which will serve as the alternative instance db' demanded by Definition 4.27.

Definition 5.3 (Alternative database instance)**.** Let $M = (S, \mathcal{K}, \pi)$ be an $\mathcal{M}_{DS}$-structure and $s \in S$ a state. The "alternative" database instance $db_{M,s}$ is constructed as follows:

$$db_{M,s} := \{\ \alpha \mid \alpha \text{ is a propositional sentence and } (M, s) \models K\alpha\ \} \tag{5.15}$$

In order to qualify as a valid database instance, $db_{M,s}$ needs to be consistent (cf. Definition 1.2). It is also important to note that $db_{M,s}$ is closed under logical implication.

Lemma 5.4. For any structure $M = (S, \mathcal{K}, \pi)$ and state $s \in S$, $db_{M,s}$ is both consistent and closed under logical implication $\models_{PL}$.

Proof. By (5.15), we have

$$(M, s) \models K\alpha \quad \text{for all } \alpha \in db_{M,s}, \tag{5.16}$$

and by the reflexivity of $\mathcal{K}$ also

$$(M, s) \models \alpha \quad \text{for all } \alpha \in db_{M,s}. \tag{5.17}$$

Hence, (M, s) can be regarded as a propositional interpretation which is a model of $db_{M,s}$. So $db_{M,s}$ must be consistent.

For the closure under implication, choose an arbitrary propositional sentence α so that $db_{M,s} \models_{PL} \alpha$. Recall (5.17) and consider that (M, s') is a model of $db_{M,s}$ for each s' with

$(s, s') \in \mathcal{K}$. By $db_{M,s} \models_{PL} \alpha$ and the definition of the logical implication operator $\models_{PL}$, if (M, s') is a model of $db_{M,s}$, it is a model of α as well, so we have

$$(M, s') \models \alpha \text{ for all } s' \text{ such that } (s, s') \in \mathcal{K}, \tag{5.18}$$

and thus

$$(M, s) \models K\alpha, \tag{5.19}$$

which means by (5.15) that $\alpha \in db_{M,s}$. Thus, $db_{M,s}$ is closed under logical implication. □

One of the properties of $db' = db_{M,s}$ to show in the confidentiality proof is that the potential secret Ψ is *false* in that alternative database instance. The following lemma states an even more general property, i. e., the equivalence of the value of some $\mathcal{L}_{PS}$-sentence in the instance and state (M, s), and the value defined by $eval_{ps}$ wrt. $db_{M,s}$. As a corollary, we can then see that Ψ must be *false* in $db_{M,s}$.

Lemma 5.5. Let log_n be a set of $\mathcal{L}_{PS}$-sentences, and Ψ an $\mathcal{L}_{PS}$-sentence so that $log_n \not\models_{S5} \Psi$. Let (M, s) be a structure and a state that witness this relationship, i. e.,

$$(M, s) \models log_n,$$
$$(M, s) \not\models \Psi.$$

Then it holds for each $\mathcal{L}_{PS}$-sentence ϕ:

$$(M, s) \models \phi \;\Leftrightarrow\; eval_{ps}(\phi)(db_{M,s}) = true.$$

Proof. We prove this property by structural induction, according to the syntactical definition of $\mathcal{L}_{PS}$ given in Definition 4.5.

Case 1 ($\phi = K\alpha$, where α is propositional)**.** The left hand side

$$(M, s) \models K\alpha \tag{5.20}$$

is by Definition 5.3 equivalent to

$$\alpha \in db_{M,s}, \tag{5.21}$$

and, as $db_{M,s}$ is consistent and closed under logical implication, also equivalent to

$$db_{M,s} \models_{PL} \alpha. \tag{5.22}$$

By the definition of the *eval* function (1.1), this is equivalent to

$$eval(\alpha)(db_{M,s}) = true \tag{5.23}$$

and by Lemma 4.17 also equivalent to

$$eval_{ps}(K\alpha)(db_{M,s}) = true. \tag{5.24}$$

For the following cases, assume as induction hypothesis that the property has already been proved for ψ and ρ.

Case 2 ($\phi = \neg\psi$)**.** Then we have

$$(M, s) \models \neg\psi \tag{5.25}$$

which is equivalent to

$$(M, s) \not\models \psi, \tag{5.26}$$

by the induction hypothesis equivalent to

$$eval_{ps}(\psi)(db_{M,s}) = \mathit{false}, \tag{5.27}$$

and by Lemma 4.15 finally to

$$eval_{ps}(\neg\psi)(db_{M,s}) = \mathit{true}. \tag{5.28}$$

Case 3 ($\phi = \psi \wedge \rho$)**.** Then we have

$$(M, s) \models \psi \wedge \rho \tag{5.29}$$

which is equivalent to

$$(M, s) \models \psi \text{ and} \tag{5.30}$$
$$(M, s) \models \rho, \tag{5.31}$$

by the induction hypothesis equivalent to

$$eval_{ps}(\psi)(db_{M,s}) = \mathit{true} \text{ and} \tag{5.32}$$
$$eval_{ps}(\rho)(db_{M,s}) = \mathit{true}, \tag{5.33}$$

and by Lemma 4.15 finally to

$$eval_{ps}(\psi \wedge \rho)(db_{M,s}) = \mathit{true}. \tag{5.34}$$

Case 4 ($\phi = \psi \vee \rho$)**.** Then we have

$$(M, s) \models \psi \vee \rho \tag{5.35}$$

which is equivalent to

$$(M, s) \models \psi \text{ or} \tag{5.36}$$
$$(M, s) \models \rho, \tag{5.37}$$

by the induction hypothesis equivalent to

$$eval_{ps}(\psi)(db_{M,s}) = \mathit{true} \text{ or} \tag{5.38}$$
$$eval_{ps}(\rho)(db_{M,s}) = \mathit{true}, \tag{5.39}$$

and by Lemma 4.15 finally to

$$eval_{ps}(\psi \vee \rho)(db_{M,s}) = \mathit{true}. \tag{5.40}$$

□

Corollary 5.6. Let log_n be a log file and Ψ a potential secret so that $log_n \not\models_{S5} \Psi$. Let (M, s) be a witness for the non-implication. Then we have $eval_{ps}(\Psi)(db_{M,s}) = \mathit{false}$.

Proof. Immediate consequence of Lemma 5.5. □

A second corollary will be useful when showing that the answers coincide under db and db', as demanded by condition (a) of Definition 4.27: When we have a query Φ and add a non-refused answer $ans \in \{\mathit{true}, \mathit{false}, \mathit{undef}\}$, encoded as $\Delta^*(\Phi, \{ans\})$, to the log file, Φ will have exactly the value ans in the resulting instance $db_{M,s}$.

Corollary 5.7. Let db be a database instance, Φ a query, $ans \in \{\mathit{true}, \mathit{false}, \mathit{undef}\}$ an answer and log_n a set of $\mathcal{L}_{PS}$-sentences so that $\Delta^*(\Phi, \{ans\}) \in log_n$. Let Ψ be an $\mathcal{L}_{PS}$-sentence with $log_n \not\models_{S5} \Psi$, $M = (\mathcal{K}, s, \pi)$ an $\mathcal{M}_{DS}$-structure and $s \in \mathcal{S}$ a state that witness this relationship, and $db_{M,s}$ the alternative database instance wrt. (M, s). constructed as specified in Definition 5.3. Then we have $eval(\Phi)(db_{M,s}) = ans$.

Proof. Due to the assumption

$$\Delta^*(\Phi, \{ans\}) \in log_n \tag{5.41}$$

we have

$$log_n \models_{S5} \Delta^*(\Phi, \{ans\}) \tag{5.42}$$

and as given in Lemma 5.5 also

$$(M, s) \models \Delta^*(\Phi, \{ans\}). \tag{5.43}$$

By Lemma 5.5, it then holds that

$$eval_{ps}(\Delta^*(\Phi, \{ans)\}))(db_{M,s}) = \mathit{true}, \tag{5.44}$$

which is by Corollary 4.18 equivalent to

$$eval(\Phi)(db_{M,s}) = ans. \tag{5.45}$$

□

6 Methods for Known Potential Secrets

In this chapter, we assume that the user knows the elements of the confidentiality policy, and we present four enforcement methods: Uniform lying, uniform refusal, uniform refusal with improved availability, and combined lying and refusal. The user's knowledge of the confidentiality policy has an impact on the confidentiality proofs: We need to ensure the existence of an alternative, "harmless" database instance which returns the same sequence of answers even under the original set of potential secrets. This is captured by condition (c) of Definition 4.27.

As it turns out, the uniform methods have special requirements with regard to the notion of inferences and confidentiality violations. The two uniform refusal methods need to consider *meta inferences* drawn from refused answers; the uniform lying method needs to protect the *disjunction of the potential secrets* in order to protect a "hopeless situation" in which no suitable lie can be returned without violating the confidentiality. The combined lying and refusal method, however, overcomes these disadvantages. It does neither need to consider meta inferences, nor protect the disjunction of the potential secrets.

Note that, according to Lemma 4.43, all methods presented in this chapter also preserve ps/ct-normality, which means that they can be employed for the construction of enforcement methods for confidentiality targets and propositional potential secrets, by the reductions given in Section 4.3 and Section 4.4, respectively.

6.1 Uniform Lying

The first enforcement method we present is *uniform lying*. As the name suggests, this method preserves confidentiality only by lying, i. e., choosing an answer *true*, *false* or *undef* possibly different from the actual query value. Answering *refuse* is not allowed. As specified in Chapter 5, we need to pick a specific version of the *inference* function, a specific version of the *violates* function, and we also need to specify a *censor* function.

6.1.1 Inferences

We use answer inferences, i. e., we employ the version of *inference* that simply adds the answer (as a definite, unary value set) to the log file, while discarding refusals (which cannot occur under this method anyway):

$$inference^{ans}(censor, C, ans) := \begin{cases} \{true, false, undef\} & \text{if } ans = refuse \\ \{ans\} & \text{otherwise} \end{cases} \tag{6.1}$$

As refusals are not allowed in this method, only unary inference sets will occur and need to be considered when determining the security configuration.

6.1.2 Security Violations

Generally, a log file is considered violating the confidentiality policy if it logically implies at least one potential secret. In Section 5.3, we already suggested the necessity to tighten this condition for the uniform lying method. We now motivate this by an example.

Example 6.1. Imagine the following situation:

$$\begin{aligned} policy_{ps} &:= \{\ Ks_1,\ Ks_2,\ Ks_3\ \} \\ log_{i-1} &:= \{\ K\alpha \rightarrow Ks_1, \quad K\neg\alpha \rightarrow Ks_2, \quad (\neg K\alpha \wedge \neg K\neg\alpha) \rightarrow Ks_3\ \} \end{aligned}$$

The user issues the query $\Phi_i = \alpha$. As refusal is not allowed, the system may only return one of the answers *true*, *false*, or *undef*. The epistemic sentences added to the log files are:

- For $ans_i = true$: $\Delta^*(\alpha, \{true\}) = K\alpha$
- For $ans_i = false$: $\Delta^*(\alpha, \{false\}) = K\neg\alpha$
- For $ans_i = undef$: $\Delta^*(\alpha, \{undef\}) = \neg K\alpha \wedge \neg K\neg\alpha$

Whatever answer ans_i the system gives – *true*, *false* or *undef* – the resulting log file

$$log_{i-1} \cup \{\Delta^*(\alpha, ans_i)\}$$

would imply one of the potential secrets. Obviously, we have a "hopeless situation", identified by the security configuration $C = \{\{true\}, \{false\}, \{undef\}\}$. What answer will the censor give?

Our solution to this problem is to avoid this "hopeless situation" in the first place. Taking a closer look at the example, one can see that the log file log_{i-1} implies the fact that *at least one* of the potentials secrets s_1, s_2 and s_3 must be *true*, depending on the value of α. More formally, we have

$$log_{i-1} \models_{S5} Ks_1 \vee Ks_2 \vee Ks_3$$

Obviously, this knowledge enables the user to maneuver the censor into the above mentioned hopeless situation. We overcome this problem by tightening our definition of security violations: We say that a log file *log* even violates a security policy $policy_{ps}$ if *log* implies only the *disjunction* of all potential secrets, formalized by

$$violates^{disj}(db, log, policy_{ps}) \ := \ log \models_{S5} pot\_sec\_disj \tag{6.2}$$

with

$$pot\_sec\_disj := \bigvee_{\Psi \in policy_{ps}} \Psi.$$

For log_0, this condition is assured by the precondition. For the subsequent log files, it is kept as an invariant. Now that the "hopeless situation" cannot occur, the only security configurations to be considered by the uniform lying censor are

$$\begin{aligned} C^*_{ans,disj} \ = \ \{\ &\{\{true\}, \{false\}\},\ \{\{true\}, \{undef\}\},\ \{\{false\}, \{undef\}\}, \\ &\{\{true\}\},\ \{\{false\}\},\ \{\{undef\}\}, \quad \emptyset\ \}. \end{aligned}$$

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

Figure 6.1: A safe uniform lying censor

6.1.3 The Censor

We can now construct a censor function for uniform lying. As described in Section 5.4, the censor function takes two parameters: the security configuration $C \in C^*$ (where only $C^*_{ans,disj}$ is relevant here) and the actual query value $v \in \{true, false, undef\}$. We construct the censor function according to the following heuristics:

- If two answers lead to a violating log file (binary security configuration), always return the remaining safe answer.
- If only one answer is harmful (unary security configuration), and if this harmful answer corresponds to the actual query value, return any of the two remaining safe answers.
- In all other cases, return the actual query value.

Censors constructed accordingly are called *safe*.

Definition 6.2 (Safe uniform lying censor)**.** A censor function *censor* is called *safe uniform lying censor* iff for each security configuration $C \in C^*_{ans,disj}$ and each value $v \in \{true, false, undef\}$ the following two conditions hold:

(a) [safe answers]
$\{censor(C, v)\} \notin C$

(b) [only lie if necessary]
if $\{v\} \notin C$ then $censor(C, v) = v$

An example of a safe uniform lying censor is given in Figure 6.1. The black cells denote the situations in which a distorted answer is given. Note that the second constraint from our heuristics leaves a certain freedom to the design of the censor: For each of the three unary security configurations, there are two safe answers to choose from. So there is a total of eight safe uniform lying censors.

6.1.4 Confidentiality

Let $cqe_{ps}^{k,L}$ be an enforcement method for uniform lying under known potential secrets, i. e., built up of $inference^{ans}$, $violates^{disj}$, and an arbitrary safe uniform lying censor (cf. Definition 6.2). (As mentioned above, there are eight of these safe censors, so there are eight different $cqe_{ps}^{k,L}$ methods as well.)

In the following, we will prove $cqe_{ps}^{k,L}$ to preserve confidentiality in the sense of Definition 4.27. First, we show the expanded precondition

$$\begin{aligned} precondition_{ps}^{k,L}(db, prior_{ps}, policy_{ps}) &:= \neg violates^{disj}(db, prior_{ps}, policy_{ps}) \\ &\equiv prior_{ps} \not\models_{S5} pot\_sec\_disj \end{aligned} \tag{6.3}$$

to be kept as an invariant for all log files log_i, $0 \leq i \leq n$.

Lemma 6.3. Let db be a database instance, $policy_{ps}$ a set of potential secrets and $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,L}(db, prior_{ps}, policy_{ps})$ is satisfied. Let $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ be a query sequence. Then we have for each $0 \leq i \leq n$: $log_i \not\models_{S5} pot\_sec\_disj$.

Proof. By induction on i. Let $i = 0$. As the precondition (6.3) is satisfied for $(db, prior_{ps}, policy_{ps})$, and $log_0 = prior_{ps}$, we have

$$log_0 \not\models_{S5} pot\_sec\_disj. \tag{6.4}$$

Now, let $i > 0$. Let Φ_i be the i-th query, C_i the associated security configuration and

$$ans_i := censor(C_i, eval(\Phi_i)(db)) \in \{true, false, undef\} \tag{6.5}$$

the answer given by the censor. By Definition 6.2, condition (a), we have for all cases

$$\{censor(C_i, ans_i)\} \notin C_i. \tag{6.6}$$

By (6.6) and according to condition (b) of Definition 6.2, we also have

$$censor(C_i, ans_i) = ans_i \tag{6.7}$$

and by (6.6) also

$$\{ans_i\} \notin C_i. \tag{6.8}$$

Thus, according to the generic construction of the security configuration (5.10), the inference set $\{ans_i\}$ cannot lead to a security violation:

$$\neg violates^{disj}(db, log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\})\}, policy_{ps}) \tag{6.9}$$

By the definition of the specific $violates^{disj}$ function (5.4), we then have

$$log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\})\} \not\models_{S5} pot\_sec\_disj, \tag{6.10}$$

and finally, by the generic construction of the subsequent log file (5.13), together with the specific $inference^{ans}$ function (6.1),

$$log_i \not\models_{S5} pot\_sec\_disj. \tag{6.11}$$

□

Now we can prove that the uniform lying method preserves confidentiality.

Theorem 6.4 (Confidentiality of the uniform lying method for known policies)**.** $cqe_{ps}^{k,L}$ preserves confidentiality in the sense of Definition 4.27.

Proof. Let db be a database instance, $policy_{ps}$ a set of potential secrets, pot_sec_disj the corresponding disjunction of the potential secrets, $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,L}(db, prior_{ps}, policy_{ps})$ (6.3) is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence.

By Lemma 6.3, we have

$$log_n \not\models_{S5} pot\_sec\_disj \tag{6.12}$$

and also

$$log_n \not\models_{S5} \Psi. \tag{6.13}$$

Let (M, s) be an $\mathcal{M}_{DS}$-structure and state that witness this relationship, i. e.,

$$(M, s) \models log_n \text{ and} \tag{6.14}$$
$$(M, s) \not\models \Psi. \tag{6.15}$$

Let $db' := db_{M,s}$ the alternative database instance as specified in Definition 5.3. According to Lemma 5.4, db' is consistent and deductively closed wrt. $\models_{PL}$. Furthermore, by Corollary 5.6, we have

$$eval_{ps}(\Psi)(db') = false \tag{6.16}$$

which satisfies condition (b) of Definition 4.27.

Next, we show that $precondition_{ps}^{k,L}(db', prior_{ps}, policy_{ps})$ is satisfied. Apparently, the definition of the precondition (6.3) is independent from the database instance. So, if it is satisfied for $(db, prior_{ps}, policy_{ps})$, it is satisfied for $(db', prior_{ps}, policy_{ps})$ as well.

Finally, we prove by induction on the query number i that the same answers are generated under both db and db'. Let ans_i (ans_i') be the answer to the i-th query Φ_i given under db (db'). We assume that the answers (and thereby the log files) up to the i-th query are identical, which is trivially true for $i = 0$.

For $i > 0$, ans_i is either *true*, *false* or *undef*, as we do not allow the answer to be refused. By Corollary 5.7, we have

$$eval(\Phi_i)(db') = ans_i. \tag{6.17}$$

We show that the censor does not distort this query value. Let $C_i := \mathit{sec\_conf}(db, log_{i-1}, policy_{ps}, \Phi_i)$ be the security configuration in the given situation. Condition (a) of Definition 6.2 guarantees that the answer $ans_i := \mathit{censor}(C_i, \mathit{eval}(\Phi_i)(db))$ given under db is safe:

$$\{ans_i\} \notin C_i \tag{6.18}$$

By condition (b) of Definition 6.2, we then have

$$\mathit{censor}(C_i, ans_i) = ans_i, \tag{6.19}$$

and by (6.17) finally

$$ans_i' := \mathit{censor}(C_i, \mathit{eval}(\Phi_i)(db')) = \mathit{censor}(C_i, ans_i) = ans_i. \tag{6.20}$$

Thus, condition (a) of Definition 4.27 is satisfied. □

Theorem 6.5 (Normality-preservation of the uniform lying method for known policies)**.** $cqe_{ps}^{k,L}$ is both ps-normality-preserving in the sense of Definition 4.35 and ps/ct-normality-preserving in the sense of Definition 4.42.

Proof. Follows immediately from Lemmas 4.36 and 4.43, respectively. □

Corollary 6.6. The function

$$cqe_{ct}^{k,L}(Q, db, prior_{ct}, policy_{ct}) := cqe_{ps}^{k,L}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

with the precondition

$$precondition_{ct}^{k,L}(db, prior_{ct}, policy_{ct}) := precondition_{ps}^{k,L}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

preserves confidentiality wrt. confidentiality targets in the sense of of Definition 3.10.

Corollary 6.7. The function

$$cqe^{k,L}(Q, db, prior, policy) := cqe_{ps}^{k,L}(Q, db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

with the precondition

$$precondition^{k,L}(db, prior, policy) := precondition_{ps}^{k,L}(db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

preserves confidentiality wrt. propositional potential secrets in the sense of of Definition 2.8.

6.2 Uniform Refusal

The next enforcement method we are going to present is uniform refusal. It follows the exact opposite idea of uniform lying: In case returning the actual query value is regarded harmful, the answer may be refused (which is indicated by returning the special answer *refuse*), but no lies may be returned. Again, we start with the definition of the *inference* function, considering the problem of *meta inferences* from refused answers. Then, we define our notion of security violations, specify a class of censor functions and prove their confidentiality.

6.2.1 Meta Inferences

When using refusal as a distortion method, *meta inferences* evolve as a major problem: The user may combine an answer with knowledge about the environment, i. e., the algorithm and the state of the inference control system. Interestingly, even a refused answer may disclose information, because of two basic assumptions made here:

1. As assumed in this chapter, the user knows the elements of the confidentiality policy $policy_{ps}$ (but of course not their respective values in the actual database instance).
2. Following the principle of open design, we also assume that the user knows the algorithm of our enforcement method, i. e., the exact definition of the function *censor* and its accompanying functions.

The censor function takes two parameters: The actual query value $eval(\Phi)(db)$ (which is invisible to the user) and the security configuration C which is determined by

$$\begin{aligned} C &:= sec\_conf(db, log, policy_{ps}, \Phi) \\ &= \{\ V \mid V \in \mathcal{I} \text{ and } violates(db, log \cup \{\Delta^*(\Phi, V)\}, policy_{ps})\ \}. \end{aligned}$$

As it turns out, the latter can be computed by the user: He knows the security policy $policy_{ps}$, the query Φ, and also the current log file log, which only depends on the initial assumptions and the past queries and answers. He does not know the database instance db, indeed – however, the $violates^{single}$ function given below, to which db is passed, works independently from this parameter, and so does *sec_conf*.

As a result, having received an answer ans, the user can determine the pre-image of the censor function wrt. ans and C, i. e., the set of query values $v \in \{true, false, undef\}$ such that $censor(C, v) = ans$, formalized by the function

$$\begin{aligned} inference^{meta}(censor, C, ans) &:= \\ &\{\ v \mid v \in \{true, false, undef\} \text{ and } censor(C, v) = ans\ \} \end{aligned} \quad (6.21)$$

with the range

$$\mathcal{I}^{meta} = \mathfrak{P}^+(\{true, false, undef\}).$$

(The empty set cannot occur as, in any case, at least one value results in the received answer.) Depending on the algorithm of the censor, even refused answers may allow the user to draw this kind of meta inferences.

Example 6.8. The user issues the query Φ and receives the answer *refuse*. He can determine the corresponding security configuration $C := \mathit{sec\_conf}(db, log, \mathit{policy}_{ps}, \Phi)$. Suppose the censor under consideration provides the following answers under this security configuration:

$$\begin{aligned} \mathit{censor}(C, \mathit{true}) &= \mathit{refuse} \\ \mathit{censor}(C, \mathit{false}) &= \mathit{false} \\ \mathit{censor}(C, \mathit{undef}) &= \mathit{refuse} \end{aligned}$$

So the user knows that the value $\mathit{eval}(\Phi)(db)$ must either be *true* or *undef*, as the value *false* would have produced a different answer. Thus, the meta inference from this answer is $\mathit{inference}^{meta}(\mathit{censor}, C, \mathit{refuse}) = \{\mathit{true}, \mathit{undef}\}$.

The uniform refusal method uses $\mathit{inference}^{meta}$ in order to calculate the meta inferences and store them in the log file. Obviously, compared to the simple version $\mathit{inference}^{ans}$ (cf. Section 6.1.1), this involves computational overhead. On the other hand, we are able to identify the information disclosed by refused answers and account for it later.

6.2.2 Security Violations

Unlike the uniform lying method, it is not necessary to protect the disjunction of the potential secrets here. Instead, we employ the original semantics of a security violation and say that a log file *log* violates a confidentiality policy policy_{ps} iff at least one of the potential secrets from policy_{ps} is logically implied by *log*:

$$\mathit{violates}^{single}(db, log, \mathit{policy}_{ps}) := (\exists \Psi \in \mathit{policy}_{ps})[log \models_{S5} \Psi] \tag{6.22}$$

What is the set of the possible security configurations $C^*_{meta,single}$ under $\mathit{inference}^{meta}$ and $\mathit{violates}^{single}$? There are $|\mathcal{I}^{meta}| = 7$ different meta inference sets and $|\mathfrak{P}(\mathcal{I}^{meta})| = 128$ different combinations of these. But due to the constraints mentioned in Section 5.3, only 18 sets qualify as a valid security configuration and have to be considered by the censor function.

6.2.3 The Censor

Again, we will specify a heuristics from which we derive a class of safe censors. As we consider meta inferences, the goal is to prevent that any answer results in a harmful meta inference, i. e., an inference set which is included in the security configuration. How can this be achieved?

Example 6.9. Remember the censor from Example 6.8 which returns the following answers under a certain security configuration C: $\mathit{censor}(C, \mathit{true}) = \mathit{refuse}$, $\mathit{censor}(C, \mathit{false}) =$

false, $censor(C, undef) = refuse$. The meta inference from $ans = refuse$ is $V = \{true, undef\}$.

Imagine that $C = \{true, undef\}$. Then the meta inference violates the confidentiality policy, which must be avoided.

We modify the censor function so that *refuse* is returned as well if the actual query value is false: $censor(C, false) = refuse$. The meta inference from $ans = refuse$ is now $\{true, false, undef\}$. This set represents no information about the query value, which is considered harmless, and thus this set is never an element of the security configuration (cf. Section 5.3).

Obviously, confidentiality can be achieved by introducing *additional refuse-conditions*, constructing censors according to the following heuristics:

- For each security configuration C and value v, if $\{v\} \in C$, set $censor(C, v) = refuse$. This is a *basic* refusal in case an answer is harmful.
- For each security configuration C, check whether the meta inference resulting from the answer *refuse* is an element of the confidentiality policy. If so, add an *additional* refusal so that this condition does not hold any longer.

Censors constructed accordingly have two important properties: 1. They only use refusal as a distortion method, i. e., $censor(C, v) \in \{v, refuse\}$ for each C and v, as required by the uniform refusal method. 2. They ensure that neither answer will result in a harmful meta inference, which is the foundation of the confidentiality proof given below. The latter property can be formalized as follows:

Definition 6.10 (Safe uniform refusal censor)**.** A censor function *censor* is called a *safe uniform refusal censor* iff for each security configuration $C \in C^*_{meta,single}$, each value $v \in \{true, false, undef\}$ and each answer $ans := censor(C, v) \in \{true, false, undef, refuse\}$ it holds that

$$inference^{meta}(censor, C, ans) \notin C$$

and

$$censor(C, v) \in \{v, refuse\}.$$

The second condition ensures that no lying is employed. However, it is not necessary for the actual protection of confidentiality; the security proofs below do not depend on it. One could easily omit this condition and construct a lying censor based on meta inferences. But this entails a number of risks when dealing with plain users. See Chapter 8.2 for a discussion.

An example of a safe uniform refusal censor is given in Figure 6.2. The black cells indicate the original *refuse*-cases. Each gray cell denotes an additional *refuse* introduced in order to prevent harmful meta inferences. For each of the three unary security configurations, there are two possibilities to choose from when placing the additional *refuse*. So, in total, there are eight different censors which can be constructed by the above mentioned heuristics.

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

Figure 6.2: A safe uniform refusal censor

6.2.4 Confidentiality

Let $cqe_{ps}^{k,R}$ be an enforcement method for uniform refusal under known potential secrets, i. e., built up of $inference^{meta}$, $violates^{single}$, and an arbitrary safe uniform refusal censor (cf. Definition 6.10). (Note that there are eight safe refusal censors, so there are eight different $cqe_{ps}^{k,R}$ methods as well.)

In the following, we will prove that $cqe_{ps}^{k,R}$ preserves confidentiality in the sense of Definition 4.27. The security proof is very similar to the one of uniform lying, as presented in Section 6.1.4. First, we show that the the expanded precondition

$$\begin{aligned} precondition_{ps}^{k,R}(db, log, policy_{ps}) &:= \neg violates^{single}(db, log, policy_{ps}) \\ &\equiv log \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \end{aligned} \tag{6.23}$$

is kept as an invariant for all log files log_i, $0 \leq i \leq n$.

Lemma 6.11. Let db be a database instance, $policy_{ps}$ a set of potential secrets and $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,R}(db, prior_{ps}, policy_{ps})$ is satisfied. Let $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ be a query sequence. Then we have for each $\Psi \in policy_{ps}$ and $0 \leq i \leq n$: $log_i \not\models_{S5} \Psi$.

Proof. By induction on i. Let $i = 0$. As the precondition (6.23) is satisfied for $(db, prior_{ps}, policy_{ps})$, and $log_0 = prior_{ps}$, we have

$$log_0 \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \tag{6.24}$$

Now, let $i > 0$. Let Φ_i be the i-th query, ans_i the answer returned by the censor and

$$C_i := sec\_conf(db, log_{i-1}, policy_{ps}, \Phi_i) \tag{6.25}$$

the associated security configuration. By Definition 6.10, we have

$$inference^{meta}(censor, C_i, ans_i) \notin C_i, \tag{6.26}$$

which is – by the generic construction of the security configuration (5.10) – equivalent to

$$\neg violates^{single}(db, log_{i-1} \cup \{\Delta^*(\Phi_i, inference^{meta}(censor, C_i, \Phi_i), ans_i))\}, policy_{ps}) \quad (6.27)$$

and – by the generic construction of the subsequent log file (5.13) – also to

$$\neg violates^{single}(db, log_i, policy_{ps}). \quad (6.28)$$

By the definition of $violates^{single}$ (5.3), this means that

$$log_i \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \quad (6.29)$$

□

Based on this observation, we can prove the confidentiality of $cqe_{ps}^{k,R}$.

Theorem 6.12 (Confidentiality of the uniform refusal method for known policies)**.** $cqe_{ps}^{k,R}$ preserves confidentiality in the sense of Definition 4.27.

Proof. Let db be a database instance, $policy_{ps}$ a set of potential secrets, $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,R}(db, prior_{ps}, policy_{ps})$ (6.23) is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence. Let log_n be the log file at the end of the query sequence.

Let $\Psi \in policy_{ps}$ be a potential secret. By Lemma 6.11, we have

$$log_n \not\models_{S5} \Psi. \quad (6.30)$$

Let (M, s) be an $\mathcal{M}_{DS}$-structure and state that witness this relationship, i. e.,

$$(M, s) \models log_n \text{ and} \quad (6.31)$$
$$(M, s) \not\models \Psi. \quad (6.32)$$

Let $db' := db_{M,s}$ the alternative database instance as specified in Definition 5.3. According to Lemma 5.4, db' is consistent and deductively closed wrt. $\models_{PL}$. Furthermore, by Corollary 5.6, we have

$$eval_{ps}(\Psi)(db') = false \quad (6.33)$$

which satisfies condition (b) of Definition 4.27.

Next, we show that $precondition_{ps}^{k,R}(db', prior_{ps}, policy_{ps})$ is satisfied. Apparently, the definition of the precondition (6.23) is independent from the database instance. So, if it is satisfied for $(db, prior_{ps}, policy_{ps})$, it is satisfied for $(db', prior_{ps}, policy_{ps})$ as well.

Finally, we prove by induction on the query number i that the same answers are generated under both db and db'. Let ans_i (ans_i') be the answer to the i-th query Φ_i given under db (db'). We assume that the answers (and thereby the log files) up to the i-th query are identical, which is trivially true for $i = 0$.

Let ans_i (ans'_i) be the answer to the i-th query given under db (db') and

$$C_i = sec\_conf(db, log_{i-1}, policy_{ps}, \Phi_i) \tag{6.34}$$

the associated security configuration. (Note that the security configuration is independent from the database instance and thereby identical under both db and db'.)

The meta inference from the answer ans_i is

$$V_i := inference^{meta}(censor, C_i, ans_i) \tag{6.35}$$

In particular, note that the actual query value $eval(\Phi_i)(db)$ is one of the values which lead to the answer ans_i, so we have $eval(\Phi_i)(db) \in V_i$.

The inference set V_i is added to the log file log_{i-1} by the means of the Δ^* function, so we have

$$\Delta^*(\Phi_i, V_i) \in log_n \tag{6.36}$$

and thus also

$$log_n \models_{S5} \Delta^*(\Phi_i, V_i) \tag{6.37}$$

and

$$(M, s) \models \Delta^*(\Phi_i, V_i). \tag{6.38}$$

Expanding the definition of Δ^* (4.6), we get

$$(M, s) \models \bigvee_{v \in V_i} \Delta(\Phi_i, v). \tag{6.39}$$

Thus, there must be at least one $v \in V_i$ so that

$$(M, s) \models \Delta(\Phi_i, v). \tag{6.40}$$

By Lemma 5.5, this is equivalent to

$$eval_{ps}(\Delta(\Phi_i, v))(db') = true \tag{6.41}$$

and by Corollary 4.18 also to

$$eval(\Phi_i)(db') = v. \tag{6.42}$$

By the definition of $inference^{meta}$ (6.21), all values from the meta inference set V_i – in particular, v and $eval(\Phi_i)(db)$ – lead to the same answer, so we have

$$\begin{aligned} ans'_i &:= censor(C_i, eval(\Phi_i)(db')) \\ &= censor(C_i, v) \\ &= censor(C_i, eval(\Phi_i)(db)) \\ &= ans_i. \end{aligned} \tag{6.43}$$

Thus, the same answers are returned under db and db'. □

Theorem 6.13 (Normality-preservation of the uniform refusal method for known policies)**.** $cqe_{ps}^{k,R}$ is both ps-normality-preserving in the sense of Definition 4.35 and ps/ct-normality-preserving in the sense of Definition 4.42.

Proof. Follows trivially from Lemmas 4.36 and 4.43, respectively. □

Corollary 6.14. The function

$$cqe_{ct}^{k,R}(Q, db, prior_{ct}, policy_{ct}) := cqe_{ps}^{k,R}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

with the precondition

$$precondition_{ct}^{k,R}(db, prior_{ct}, policy_{ct}) := precondition_{ps}^{k,R}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

preserves confidentiality wrt. confidentiality targets in the sense of of Definition 3.10.

Corollary 6.15. The function

$$cqe^{k,R}(Q, db, prior, policy) := cqe_{ps}^{k,R}(Q, db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

with the precondition

$$precondition^{k,R}(db, prior, policy) := precondition_{ps}^{k,R}(db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

preserves confidentiality wrt. propositional potential secrets in the sense of of Definition 2.8.

6.3 Uniform Refusal with Improved Availability

In Section 6.2, we have presented the uniform refusal method for known potential secrets, and we have shown that it preserves confidentiality. However, the additional *refuse*-conditions reduce the availability of the system. In this section, we show that some of these additional *refuse*-conditions can be omitted when we take another question into account: Does the user already know the value of the query, or is he at least able to exclude one possible query value? If so, it might not be necessary to refuse a certain answer.

6.3.1 Motivation

Sometimes, an answer is refused even though it would not lead to the disclosure of a potential secret. In particular, we have introduced *additional refuse-conditions* which protect other refusals that would lead to such a disclosure, or sometimes would only make the log file inconsistent (which, by the definition of logical implication, leads to the disclosure of all potential secrets).

Example 6.16. Consider the following input to the uniform refusal method $cqe^{k,R}_{ps}$:

$$\begin{aligned} Q &:= \langle\ s\ \rangle \\ db &:= \{\ \neg s\ \} \\ prior_{ps} &:= \{\ Ks\ \vee\ K\neg s\ \} \\ policy_{ps} &:= \{\ Kt\ \} \end{aligned}$$

We consider the query $\Phi_1 = s$. The inference $\{undef\}$ would make log inconsistent, any other inference is harmless, so the security configuration is

$$C_1 := sec\_conf(db, log_0, policy_{ps}, \Phi_1) = \{\{undef\}\}.$$

Furthermore, the ordinary query value is

$$eval(s)(db) = false.$$

According to the censor table from Figure 6.2, the answer returned is

$$censor(\{\{undef\}\}, false) = refuse.$$

This refusal arises from an additional *refuse*-condition which protects the ordinary *refuse* for $censor(\{\{undef\}\}, undef)$. We argue that this refusal is not necessary, and we establish an optimized method $cqe^{k,R*}_{ps}$ which provides a higher availability in this, and in similar situations. In particular, we rely on the following observations from the above example:

1. The actual truth value of s cannot be *undef*, so the "real" *refuse* will not occur.
2. The user can tell this fact from his knowledge, i. e., from the log file, which implies that s must be *true* or *false*, but not *undef*.

The idea is to account for this disjunctive "a priori" knowledge about the query value, and use a special censor function for this situation, which does not need the additional *refuse*-condition.

6.3.2 Construction

We want to take into account the user's knowledge about the possible values of a certain query. First of all, it is important to ensure that this knowledge is consistent with the actual database instance: the user may not believe that a query has a certain value different from the actual value. We achieve this by introducing a stronger precondition which states that the log file must not imply any of the potential secrets, and additionally must only contain sentences which are *true* in db:

$$\begin{aligned} precondition^{k,R*}_{ps}(db, prior_{ps}, policy_{ps})\ :=& \\ & \neg violates^{single}(db, prior_{ps}, policy_{ps})\ \wedge \\ & (\forall \phi \in prior_{ps})[\ eval_{ps}(\phi)(db) = true\ ] \end{aligned} \tag{6.44}$$

When the user issues a query Φ, we can determine those possible query values of Φ which are consistent with the current log file *log*, i. e., the values the user considers possible, and the values he can rule out in the first place. We call this set of possible values the *pre-inference* of Φ wrt. *log*, determined by the function

$$\mathit{pre\_inf}(\mathit{log}, \Phi) \;:=\; \{\; v \in \{\mathit{true}, \mathit{false}, \mathit{undef}\} \;\mid\; \mathit{log} \cup \{\Delta(\Phi, v)\} \text{ is consistent } \}. \quad (6.45)$$

Similar to our general notion of inferences (cf. Section 5.2), we have three different types of pre-inferences: unary pre-inferences (when the user already knows the value of Φ), binary pre-inferences (when the user can exclude one value, but the other two values appear possible), and $\{\mathit{true}, \mathit{false}, \mathit{undef}\}$ (all values appear possible to the user). The empty set cannot occur, as *log* is consistent at any time.

We denote the pre-inference of a query by P. Based on P, we can define a family of $\mathit{inference}^{\mathit{meta}}$ functions that restrict the meta inference to the values of P:

$$\mathit{inference}_P^{\mathit{meta}}(\mathit{censor}, C, \mathit{ans}) \;:=\; \{\; v \;\mid\; v \in P \text{ and } \mathit{censor}(C, v) = \mathit{ans} \;\}. \quad (6.46)$$

This function has the range

$$\mathcal{I}_P^{\mathit{meta}} = \mathfrak{P}^+(P).$$

Likewise, we denote the set of possible security configurations wrt. $\mathit{violates}^{\mathit{single}}$ and a particular version of $\mathit{inference}_P^{\mathit{meta}}$ by $C_P^* \subset \mathfrak{P}(\mathcal{I}_P^{\mathit{meta}})$. Again, this set is restricted by the constraints mentioned in Section 5.3. In particular, we have $P \notin C_P^*$ (it's never harmful to learn that the query has one of the values of P). As a result, we have

- under the pre-inference $\{\mathit{true}, \mathit{false}, \mathit{undef}\}$, the same set of security configurations as for the original refusal method from Section 6.2;
- under a binary pre-inference $P = \{v_1, v_2\}$, four security configurations: $\{v_1, v_2\}$, $\{v_1\}$, $\{v_2\}$ and $\emptyset$;
- under a unary pre-inference, only one security configuration: $\emptyset$.

Finally, we define a family of *censor* functions, each of which operates on a particular pre-inference P, i. e., it only considers C_P^* as possible security configurations, it only considers P as possible query values, and it only returns answers from P or *refuse*:

$$\mathit{censor}_P \;:\; C_P^* \;\times\; P \;\rightarrow\; P \cup \{\mathit{refuse}\}$$

The fact that only query values from P need to be considered by a particular censor is justified by the following lemma. Given that the log file contains only sentences which are true in the actual database instance, it states that the actual value of a query is among those the user considers possible according to the log file.

Lemma 6.17. Let *log* be a log file and *db* a database instance so that $\mathit{eval}_{ps}(\phi)(db) = \mathit{true}$ for each $\phi \in \mathit{log}$. Then we have for each propositional sentence α:

$$\mathit{eval}(\alpha)(db) \in \mathit{pre\_inf}(\mathit{log}, \alpha).$$

Proof. By Corollary 4.18, we have $eval_{ps}(\Delta(\Phi, eval(\alpha, db)))(db) = true$. So $log \cup \Delta(\Phi, eval(\alpha, db))$ must be consistent, and thus $eval(\alpha)(db)$ must be included in $pre\_inf(log, \alpha)$. □

For each of these censor functions, we can define a safeness property:

Definition 6.18 (Safe uniform refusal censor wrt. a particular pre-inference)**.** A censor function $censor_P$ is called a *safe uniform refusal censor wrt. the pre-inference* P iff for each security configuration $C \in C_P^*$, each value $v \in P$ and each answer $ans := censor_P(C, v) \in P \cup \{refuse\}$ it holds that

$$inference_P^{meta}(censor_P, C, ans) \notin C$$

and

$$censor_P(C, v) \in \{v, refuse\}.$$

This safeness property is similar to the one found in Definition 6.10. In fact, the original refusal censor described in the previous section serves as the censor for the weakest pre-inference $\{true, false, undef\}$. For the other six possible pre-inferences, we need to specify a valid safe censor function. For the three unary pre-inferences $\{true\}$, $\{false\}$ and $\{undef\}$, the respective censor function is trivially given by

$$censor_P(C, v) := v, \tag{6.47}$$

so it just returns the value which is already known by the user. For the three binary pre-inferences $\{true, false\}$, $\{true, undef\}$ and $\{false, undef\}$, a censor can easily be constructed according to the following heuristics: If there is at least one $v \in P$ so that $\{v\} \in C$, answer *refuse*. Otherwise, return the ordinary answer.[1]

$$censor_P(C, v) := \begin{cases} refuse & \text{if } (\exists v \in P)[\ \{v\} \in C\ ] \\ v & \text{otherwise} \end{cases} \tag{6.48}$$

Note that this involves one additional *refuse*-condition for each of the two unary security configurations $\{v_1\}$ and $\{v_2\}$ (assuming that $P = \{v_1, v_2\}$).

A family of safe censor functions is given in Figure 6.3.

We can now formally specify the algorithm of $cqe_{ps}^{k,R*}$. For each pre-inference $P \subseteq^+ \{true, false, undef\}$, we need a safe censor function. For each query Φ_i of the query sequence Q, the answer ans_i and the subsequent log file log_i are generated by the following steps:

1. Determine the pre-inference:

   $$P_i := pre\_inf(log_{i-1}, \Phi)$$

[1] Not surprisingly, this heuristics coincides with the uniform refusal method for known potential secrets under complete databases, a fact which we will address in Section 8.1, when we compare our methods to those for complete databases.

Pre-Inference	Censor Table
$\{true, false, undef\}$	Security Configuration C | $eval(\Phi)(db) = ...$ $true$ / $false$ / $undef$ $\{\{true\}, \{false\}, \{undef\}, ...\}$ | refuse / refuse / refuse $\{\{true\}, \{false\}, \{true, false\}\}$ | refuse / refuse / refuse $\{\{true\}, \{false\}\}$ | refuse / refuse / undef $\{\{true\}, \{undef\}, \{true, undef\}\}$ | refuse / refuse / refuse $\{\{true\}, \{undef\}\}$ | refuse / false / refuse $\{\{true\}\}$ | refuse / false / refuse $\{\{false\}, \{undef\}, \{false, undef\}\}$ | refuse / refuse / refuse $\{\{false\}, \{undef\}\}$ | true / refuse / refuse $\{\{false\}\}$ | true / refuse / refuse $\{\{undef\}\}$ | true / refuse / refuse $\emptyset$ | true / false / undef
$\{true, false\}$	Security Configuration C | $eval(\Phi)(db) = ...$ $true$ / $false$ $\{\{true\}, \{false\}\}$ | refuse / refuse $\{\{true\}\}$ | refuse / refuse $\{\{false\}\}$ | refuse / refuse $\emptyset$ | true / false
$\{true, undef\}$	Security Configuration C | $eval(\Phi)(db) = ...$ $true$ / $undef$ $\{\{true\}, \{undef\}\}$ | refuse / refuse $\{\{true\}\}$ | refuse / refuse $\{\{undef\}\}$ | refuse / refuse $\emptyset$ | true / undef
$\{false, undef\}$	Security Configuration C | $eval(\Phi)(db) = ...$ $false$ / $undef$ $\{\{false\}, \{undef\}\}$ | refuse / refuse $\{\{false\}\}$ | refuse / refuse $\{\{undef\}\}$ | refuse / refuse $\emptyset$ | false / undef
$\{true\}$	Security Configuration | $eval(\Phi)(db) = ...$ $\emptyset$ | $true$
$\{false\}$	Security Configuration | $eval(\Phi)(db) = ...$ $\emptyset$ | $false$
$\{undef\}$	Security Configuration | $eval(\Phi)(db) = ...$ $\emptyset$ | $undef$

Figure 6.3: A family of censor functions, safe wrt. their respective pre-inference

2. Determine the security configuration wrt. P_i:

$$C_i := \{\ V \mid V \in \mathcal{I}_{P_i}^{meta} \text{and } violates^{single}(db, log \cup \{\Delta^*(\Phi, V)\}, policy_{ps})\ \} \quad (6.49)$$

3. Let the appropriate censor function choose the answer:

$$ans_i := censor_{P_i}(C_i, eval(\Phi_i)(db))$$

4. Add the meta inference (restricted to P_i) to the log file:

$$log_i := log_{i-1} \cup \{\Delta^*(\Phi_i, inference_{P_i}^{meta}(censor_{P_i}, C_i, ans_i))\}$$

6.3.3 Confidentiality

We prove the confidentiality of $cqe_{ps}^{k,R*}$ as usual. First, we show that the log file log_i does not imply any of the potential secrets at any time.

Lemma 6.19. Let db be a database instance, $policy_{ps}$ a set of potential secrets and $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,R*}(db, prior_{ps}, policy_{ps})$ is satisfied. Let $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ be a query sequence. Then we have for each $\Psi \in policy_{ps}$ and $0 \leq i \leq n$:

$$log_i \not\models_{S5} \Psi$$

Proof. By induction on i. Let $i = 0$. As the precondition (6.23) is satisfied for $(db, prior_{ps}, policy_{ps})$, and $log_0 = prior_{ps}$, we have

$$log_0 \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \quad (6.50)$$

Now, let $i > 0$. Let Φ_i be the i-th query, P_i the pre-inference of the query, C_i the security configuration and ans_i the answer returned by the censor function $censor_{P_i}$. The inference added to the user log is then

$$V_i := inference_{P_i}^{meta}(censor_{P_i}, C_i, ans_i) \subseteq P. \quad (6.51)$$

for the pre-inference P_i. From the global safeness of the censor (cf. Definition 6.18), we know that

$$V_i \notin C_i \quad (6.52)$$

which is by the construction of C_i (6.49) equivalent to

$$V_i \notin \{\ V \mid V \in \mathcal{I}_{P_i}^{meta} \text{and } violates^{single}(db, log_{i-1} \cup \{\Delta^*(\Phi, V)\}, policy_{ps})\ \}. \quad (6.53)$$

As $V_i \in \mathcal{I}_{P_i}^{meta}$, we can rewrite this condition as

$$\neg violates^{single}(db, log \cup \{\Delta^*(\Phi_i, V_i)\}, policy_{ps}). \quad (6.54)$$

or equivalently as

$$\neg violates^{single}(db, log_i, policy_{ps}). \tag{6.55}$$

By the definition of $violates^{single}$ (5.3), this means that

$$log_i \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \tag{6.56}$$

□

We can now prove that $cqe_{ps}^{k,R*}$ satisfies our notion of confidentiality.

Theorem 6.20 (Confidentiality of the improved uniform refusal method for known policies)**.** $cqe_{ps}^{k,R*}$ preserves confidentiality in the sense of Definition 4.27.

Proof. Let db be a database instance, $policy_{ps}$ a set of potential secrets, $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,R*}(db, prior_{ps}, policy_{ps})$ is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence. Let log_n be the log file at the end of the query sequence. Let $\Psi \in policy_{ps}$ be a potential secret.

By Lemma 6.19, we have

$$log_n \not\models_{S5} \Psi. \tag{6.57}$$

Let (M, s) an $\mathcal{M}_{DS}$-structure and state that witness this relationship, i. e.,

$$(M, s) \models log_n \text{ and} \tag{6.58}$$
$$(M, s) \not\models \Psi. \tag{6.59}$$

Let $db' := db_{M,s}$ the alternative database instance as specified in Definition 5.3. According to Lemma 5.4, db' is consistent and deductively closed wrt. $\models_{PL}$. Furthermore, by Corollary 5.6, we have

$$eval_{ps}(\Psi)(db') = false \tag{6.60}$$

which satisfies condition (b) of Definition 4.27.

Next, we show that $precondition_{ps}^{k,R*}(db', prior_{ps}, policy_{ps})$ is satisfied. The first part ($prior_{ps}$ does not violate $policy_{ps}$) follows from $precondition_{ps}^{k,R*}(db, prior_{ps}, policy_{ps})$. For the second part (elements of $prior_{ps}$ are *true* in db'), choose an arbitrary $\phi \in prior_{ps}$. By $prior_{ps} \subseteq log_n$, we have

$$log_n \models_{S5} \phi \tag{6.61}$$

and thus also

$$(M, s) \models \phi. \tag{6.62}$$

By Lemma 5.5, we then have

$$eval_{ps}(\phi)(db') = true. \tag{6.63}$$

Thus, $precondition_{ps}^{k,R*}(db', prior_{ps}, policy_{ps})$ is satisfied.

Finally, we prove by induction on the query number i that the same answers are generated under both db and db'. Let ans_i (ans_i') be the answer to the i-th query given under db (db'). Let P_i the pre-inference of the query and C_i the associated security configuration. (Note that both P_i and C_i are independent from the database instance and thereby identical under both db and db'.)

We assume that the answers ans_i and ans_i' (and thereby the user logs and security configurations) up to query i are identical, which is obviously true for $i = 0$.

For $i > 0$, the meta inference set is

$$V_i := inference_{P_i}^{meta}(censor_{P_i}, C_i, ans_i) \tag{6.64}$$

In particular, we have

$$eval(\Phi_i)(db) \in V_i. \tag{6.65}$$

The inference set V_i is added to the log file log_{i-1} by the means of the Δ^* function, so we have

$$\Delta^*(\Phi_i, V_i) \in log_n \tag{6.66}$$

and thus also

$$log_n \models_{S5} \Delta^*(\Phi_i, V_i) \tag{6.67}$$

and

$$(M, s) \models \Delta^*(\Phi_i, V_i). \tag{6.68}$$

Expanding the definition of Δ^* (4.6), we get

$$(M, s) \models \bigvee_{v \in V_i} \Delta(\Phi_i, v). \tag{6.69}$$

Thus, there must be at least one $v \in V_i$ so that

$$(M, s) \models \Delta(\Phi_i, v). \tag{6.70}$$

By Lemma 5.5, this is equivalent to

$$eval_{ps}(\Delta(\Phi_i, v))(db') = true \tag{6.71}$$

and by Corollary 4.18 also to

$$eval(\Phi_i)(db') = v. \tag{6.72}$$

By the definition of $inference_{P_i}^{meta}$ (6.46), all values from the meta inference set V_i – in particular, v and $eval(\Phi_i)(db)$ – lead to the same answer, so we have

$$\begin{aligned} ans_i' &:= censor_{P_i}(C_i, eval(\Phi_i)(db')) \\ &= censor_{P_i}(C_i, v) \\ &= censor_{P_i}(C_i, eval(\Phi_i)(db)) \\ &= ans_i. \end{aligned} \tag{6.73}$$

Thus, the same answers are returned under db and db'. □

Theorem 6.21 (Normality-preservation of the improved uniform refusal method for known policies)**.** $cqe_{ps}^{k,R*}$ is both ps-normality-preserving in the sense of Definition 4.35 and ps/ct-normality-preserving in the sense of Definition 4.42.

Proof. Follows trivially from Lemmas 4.36 and 4.43, respectively. □

Corollary 6.22. The function

$$cqe_{ct}^{k,R*}(Q, db, prior_{ct}, policy_{ct}) := cqe_{ps}^{k,R*}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

with the precondition

$$precondition_{ct}^{k,R*}(db, prior_{ct}, policy_{ct}) := \\ precondition_{ps}^{k,R*}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

preserves confidentiality wrt. confidentiality targets in the sense of of Definition 3.10.

Corollary 6.23. The function

$$cqe^{k,R*}(Q, db, prior, policy) := \\ cqe_{ps}^{k,R*}(Q, db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

with the precondition

$$precondition^{k,R*}(db, prior, policy) := \\ precondition_{ps}^{k,R*}(db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

preserves confidentiality wrt. propositional potential secrets in the sense of of Definition 2.8.

6.3.4 Impact on the Availability

Having improved the uniform refusal method, we are now going to reconsider the situation from Example 6.16.

Example 6.24. Consider the following input to the uniform refusal method $cqe_{ps}^{k,R*}$:

$$Q = \langle\ s\ \rangle$$
$$db = \{\ \neg s\ \}$$
$$log_0 = \{\ Ks\ \vee\ K\neg s\ \}$$
$$policy_{ps} = \{\ Kt\ \}$$

We consider the query $\Phi_1 = s$. The pre-inference is

$$P_i = \{true, false\}$$

(as the user can tell from the log file that the value must be *true* or *false*). We now determine the security configuration C_i wrt. the pre-inference P_i: Neither $\{true\}$ nor $\{false\}$ is a harmful inference, so the security configuration is empty:

$$C_i = \emptyset$$

The ordinary query value is

$$eval(s)(db) = false,$$

so the answer returned by the $\{true, false\}$-censor (cf. Figure 6.3) is

$$censor_{\{true,false\}}(\emptyset, false) = false.$$

Obviously, we successfully improved the refusal method in terms of availability, while, in general, still preserving confidentiality.

6.4 Combined Lying and Refusal

In the previous sections, we presented the uniform lying and the uniform refusal method (in two different variants), which use different versions of the functions *inference* and *violates*:

- The uniform refusal methods rely on meta inferences, calculated by $inference^{meta}$. Thus, also disjunctive inferences have to be considered when determining the security configuration. This results in computational overhead compared to the uniform lying method, where the simple version $inference^{ans}$ is employed. Furthermore, we had to introduce additional *refuse*-conditions which prevent harmful meta inferences. This leads to a lower availability.
- The uniform lying method protects the disjunction of the potential secrets, formalized by $violates^{disj}$, in order to prevent the "hopeless situation" in which neither answer may be given. This is a stronger condition compared to the protection of each single secret, as formalized by $violates^{single}$, and it results in a loss of availability, as more answers might be distorted.

In this section, we establish another enforcement method, involving both lying and refusal. It combines the advantages of the respective uniform methods while eliminating their disadvantages. In particular, we make use of the two favorable functions $inference^{ans}$ and $violates^{single}$.

6.4.1 The Censor

The combined lying and refusal censor adopts its basic functioning from the uniform lying censor: In case an original value is harmful, a lie is returned as an answer. As opposed to uniform lying, we do not protect the disjunction of the potential secrets. Instead,

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*

Figure 6.4: A safe combined lying and refusal censor, derived from the uniform lying censor from Figure 6.1

we protect each potential secret independently, thus allowing the "hopeless situation" to occur again in which each value $v \in \{true, false, undef\}$ would lead to a violating log file, identified by the security configuration $C_{hopeless} = \{\{true\},\{false\},\{undef\}\}$. In this situation, the combined censor refuses the answer, regardless of the query value, thus preventing any useful meta inference from this refusal (as $censor(C_{hopeless}, v) = refuse$ for each $v \in \{true, false, undef\}$). Figure 6.4 demonstrates how a combined lying and refusal censor is established based on the uniform lying censor by adding the newly introduced security configuration $C_{hopeless} = \{\{true\},\{false\},\{undef\}\}$ under which the answer is refused. We then have the set of possible security configurations

$$C^*_{ans,single} = C^*_{ans,disj} \cup \{\{\{true\},\{false\},\{undef\}\}\}.$$

Again, we define the notion of safeness.

Definition 6.25 (Safe combined lying and refusal censor)**.** A censor function *censor* is called a *safe combined lying and refusal censor* iff for each security configuration $C \in C^*_{ans,single}$ and each value $v \in \{true, false, undef\}$ the following three conditions hold:

(a) [safe answers]
for each $v \in \{true, false, undef\}$: $\{censor(C, v)\} \notin C$,

(b) [only lie if necessary]
for each $v \in \{true, false, undef\}$: if $\{v\} \notin C$ then $censor(C, v) = v$,

(c) [safe refusals]
if $censor(C, v) = refuse$ for any $v \in \{true, false, undef\}$,
then $censor(C, v) = refuse$ for all $v \in \{true, false, undef\}$.

Conditions (a) and (b) coincide with the respective conditions for safe uniform lying censors from Definition 6.2. Condition (c) guarantees that the user cannot draw any non-empty meta inferences from refusals. This justifies the fact that we use the answer inference approach (which logs refusals as the trivial inference set $\{true, false, undef\}$, and thus disposes any information about a refused answer and possible meta inferences from that refusal).

6.4.2 Confidentiality

Let $cqe_{ps}^{k,C}$ be an enforcement method for combined lying and refusal under known potential secrets, i. e., built up of $inference^{ans}$, $violates^{single}$, and an arbitrary safe combined lying and refusal censor (cf. Definition 6.25). (Note that there are eight safe censors, so there are eight different $cqe_{ps}^{k,C}$ methods as well.)

In the following, we will prove that $cqe_{ps}^{k,C}$ preserves confidentiality in the sense of Definition 4.27. The security proof corresponds to the one for uniform lying from Section 6.1.4. But now we also have to consider refusals. First, we show that the the expanded precondition

$$\begin{aligned} precondition_{ps}^{k,R}(db, log, policy_{ps}) &:= \neg violates^{single}(db, log, policy_{ps}) \\ &\equiv log \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \end{aligned} \tag{6.74}$$

is kept as an invariant for all log files log_i, $0 \leq i \leq n$.

Lemma 6.26. Let db be a database instance, $policy_{ps}$ a set of potential secrets and log_0 a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,C}(db, prior_{ps}, policy_{ps})$ is satisfied. Let $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ be a query sequence. Then we have for each $\Psi \in policy_{ps}$ and $0 \leq i \leq n$:

$$log_i \not\models_{S5} \Psi.$$

Proof. By induction on i. Let $i = 0$. As the precondition (6.74) is satisfied for $(db, prior_{ps}, policy_{ps})$, and $log_0 = prior_{ps}$, we have

$$log_0 \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \tag{6.75}$$

Now, let $i > 0$. Let Φ_i be the i-th query, C_i the associated security configuration and

$$ans_i := censor(C_i, eval(\Phi_i)(db)) \in \{true, false, undef, refuse\} \tag{6.76}$$

the answer given by the censor.

If $ans_i = refuse$, the sentence

$$\Delta^*(\Phi_i, \{true, false, undef\}) = K\Phi_i \vee K\neg\Phi_i \vee (\neg K\Phi_i \wedge \neg K\neg\Phi_i) \tag{6.77}$$

is added to log_{i-1}. This sentence is a tautology. Thus, log_i implies the exact same sentences as does log_{i-1}, so the proposition is satisfied.

If $ans_i \in \{true, false, undef\}$, we have by Definition 6.25, condition (a)

$$\{censor(C_i, ans_i)\} \notin C_i. \tag{6.78}$$

By (6.78) and according to condition (b) of Definition 6.25, we also have

$$censor(C_i, ans_i) = ans_i \tag{6.79}$$

and by (6.78) also

$$\{ans_i\} \notin C_i. \tag{6.80}$$

Thus, according to the generic construction of the security configuration (5.10), the inference set $\{ans_i\}$ cannot lead to a security violation:

$$\neg violates^{single}(db, log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\})\}, policy_{ps}) \tag{6.81}$$

By the definition of the specific $violates^{single}$ function (5.3), we then have

$$log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\})\} \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}, \tag{6.82}$$

and finally, by the generic construction of the subsequent log file (5.13), together with the specific $inference^{ans}$ function (5.1),

$$log_i \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps}. \tag{6.83}$$

□

Theorem 6.27 (Confidentiality of the combined lying and refusal method for known policies)**.** $cqe_{ps}^{k,C}$ preserves confidentiality in the sense of Definition 4.27.

Proof. Let db be a database instance, $policy_{ps}$ a set of potential secrets, $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{k,C}(db, prior_{ps}, policy_{ps})$ is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence. Let $\Psi \in policy_{ps}$ be a potential secret.

By Lemma 6.26, we have

$$log_n \not\models_{S5} \Psi. \tag{6.84}$$

Let (M, s) an $\mathcal{M}_{DS}$-structure and state that witness this relationship, i. e.,

$$(M, s) \models log_n \text{ and} \tag{6.85}$$
$$(M, s) \not\models \Psi. \tag{6.86}$$

Let $db' := db_{M,s}$ the alternative database instance as specified in Definition 5.3. According to Lemma 5.4, db' is consistent and deductively closed wrt. $\models_{PL}$. Furthermore, by Corollary 5.6, we have

$$eval_{ps}(\Psi)(db') = false \tag{6.87}$$

which satisfies condition (b) of Definition 4.27.

Next, we show that $precondition_{ps}^{k,C}(db', prior_{ps}, policy_{ps})$ is satisfied. Apparently, the definition of the precondition (6.74) is independent from the database instance. So, if it is satisfied for $(db, prior_{ps}, policy_{ps})$, it is satisfied for $(db', prior_{ps}, policy_{ps})$ as well.

Finally, we prove by induction on the query number i that the same answers are generated under both db and db'. Let ans_i (ans_i') be the answer to the i-th query Φ_i given under db (db'). We assume that the answers (and thereby the log files) up to the i-th query are identical, which is trivially true for $i = 0$.

For $i > 0$, ans_i is either a regular answer (*true*, *false*, *undef*) or *refuse*.

Case 1 ($ans_i \in \{true, false, undef\}$)**.** The sentence added to the log file is

$$\Delta^*(\Phi_i, \{ans_i\}). \tag{6.88}$$

By Corollary 5.7, we have

$$eval(\Phi_i)(db') = ans_i. \tag{6.89}$$

We show that the censor does not distort this query value. Let $C_i := sec\_conf(db, log_{i-1}, policy_{ps}, \Phi_i)$ be the security configuration in the given situation. Condition (a) of Definition 6.2 guarantees that the answer $ans_i := censor(C_i, eval(\Phi_i)(db))$ given under db is safe:

$$\{ans_i\} \notin C_i \tag{6.90}$$

By condition (b) of Definition 6.2, we then have

$$censor(C_i, ans_i) = ans_i, \tag{6.91}$$

and by (6.89) finally

$$ans_i' := censor(C_i, eval(\Phi_i)(db')) = censor(C_i, ans_i) = ans_i. \tag{6.92}$$

Case 2 ($ans_i = refuse$)**.** Condition (c) of Definition 6.25 states that, under a particular security configuration, if the answer is refused for at least one query value, it must be refused for any other query value as well, so we have

$$ans_i' := censor(C_i, eval(\Phi_i)(db')) = refuse. \tag{6.93}$$

So condition (a) of Definition 4.27 is satisfied. □

Theorem 6.28 (Normality-preservation of the combined lying and refusal method for known policies)**.** $cqe_{ps}^{k,C}$ is both ps-normality-preserving in the sense of Definition 4.35 and ps/ct-normality-preserving in the sense of Definition 4.42.

Proof. Follows trivially from Lemmas 4.36 and 4.43, respectively. □

Corollary 6.29. The function

$$cqe_{ct}^{k,C}(Q, db, prior_{ct}, policy_{ct}) := cqe_{ps}^{k,C}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

with the precondition

$$precondition_{ct}^{k,C}(db, prior_{ct}, policy_{ct}) := precondition_{ps}^{k,C}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

preserves confidentiality wrt. confidentiality targets in the sense of of Definition 3.10.

Corollary 6.30. The function

$$cqe^{k,C}(Q, db, prior, policy) := cqe^{k,C}_{ps}(Q, db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

with the precondition

$$precondition^{k,C}(db, prior, policy) := precondition^{k,C}_{ps}(db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

preserves confidentiality wrt. propositional potential secrets in the sense of of Definition 2.8.

7 Methods for Unknown Potential Secrets

In this chapter, we study another set of enforcement methods. This time, we assume that the user does *not* know the elements of the confidentiality policy $policy_{ps}$. With regard to confidentiality, Definition 4.27 states that confidentiality is preserved when the same answers are returned even under a different confidentiality policy $policy'_{ps}$ (and an alternative database instance db'). In other words: The user cannot distinguish $(db, policy_{ps})$ from $(db', policy'_{ps})$.

In the context of complete information systems [1, 5], it was shown that this additional freedom might be exploited as follows: Instead of protecting *all* potential secrets, only the *true* potential secrets are considered by the censor. This results in higher availability, because the implication of a (false) potential secret will not lead to a refusal or lie anymore.

This heuristics can be successfully applied to the uniform lying method and the uniform refusal method from Chapter 6, which is demonstrated in Section 7.1 and Section 7.2, respectively. Additionally, the uniform refusal method does not need to consider any meta inferences anymore. Therefore, it is also not necessary to construct an improved refusal method, as we did for the known policy case.

Unfortunately, the heuristics does not work for the combined lying and refusal method. This is shown in Section 7.3 by means of a counter-example. Of course, the combined method for known policies from Section 6.4 can also be used when the policy is unknown. However, the question remains whether it can be improved in any way when we assume that the user is not aware of the confidentiality policy. This coincides with the results for complete information systems from [5].

7.1 Uniform Lying

The uniform lying method for unknown potential secrets is designed exactly according to the heuristics mentioned in the introduction of this chapter: We reuse the algorithm for known potential secrets, however, we now only consider those potential secrets which are true in the actual database instance. This can easily be achieved by adapting the originally used function $violates^{single}$ and modifying it as follows: We say that a log file log violates a confidentiality policy $policy_{ps}$ if and only if the disjunction of all true potential secrets $\Psi \in policy_{ps}$ is logically implied by log, formalized by the function

$$violates^{truedisj}(db, log, policy_{ps}) \;:=\; log \models_{S5} true\_pot\_sec\_disj \tag{7.1}$$

with

$$true\_pot\_sec\_disj \;:=\; \begin{cases} \bigvee true\_pot\_sec & \text{if } true\_pot\_sec \neq \emptyset \\ \bot & \text{if } true\_pot\_sec = \emptyset \end{cases} \tag{7.2}$$

and

$$true\_pot\_sec \;:=\; \{\; \Psi \in policy_{ps} \;\mid\; eval_{ps}(\Psi)(db) = true \;\}. \tag{7.3}$$

Just like the method for known potential secrets, we use answer log update

$$inference^{ans}(censor, C, ans) \;:=\; \begin{cases} \{true, false, undef\} & \text{if } ans = refuse \\ \{ans\} & \text{otherwise} \end{cases} \tag{7.4}$$

with the range

$$\mathcal{I}^{ans} \;=\; \{\{true\}, \{false\}, \{undef\}, \{true, false, undef\}\}.$$

The resulting set of possible security configurations is

$$\begin{aligned} C^*_{ans,truedisj} \;=\; \{\; &\{\{true\}, \{false\}\},\; \{\{true\}, \{undef\}\},\; \{\{false\}, \{undef\}\}, \\ &\{\{true\}\},\; \{\{false\}\},\; \{\{undef\}\},\; \emptyset \;\}, \end{aligned}$$

and coincides with $C^*_{ans,disj}$. This allows us to use the same class of safe censors as for the "known policy" case (cf. Definition 6.2).

We denote an enforcement method for uniform lying under unknown potential secrets (built up of $inference^{ans}$, $violates^{truedisj}$, and an arbitrary safe uniform lying censor) by $cqe^{u,L}_{ps}$. Note that there are eight safe censors, so there are eight different $cqe^{u,L}_{ps}$ methods as well. In the following, we will prove that $cqe^{u,L}_{ps}$ preserves confidentiality in the sense of Definition 4.27. First, we show that the precondition (5.12), expanded as

$$\begin{aligned} precondition^{u,L}_{ps}(db, prior_{ps}, policy_{ps}) &:= \neg violates^{truedisj}(db, prior_{ps}, policy_{ps}) \\ &\equiv prior_{ps} \not\models_{S5} true\_pot\_sec\_disj, \end{aligned} \tag{7.5}$$

is kept as an invariant for all log files log_i, $0 \le i \le n$.

Lemma 7.1. Let db be a database instance, $policy_{ps}$ a set of potential secrets and $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition^{u,L}_{ps}(db, prior_{ps}, policy_{ps})$ is satisfied. Let $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ be a query sequence. Then we have for each $0 \le i \le n$: $log_i \not\models_{S5} true\_pot\_sec\_disj$.

Proof. By induction on i. Let $i = 0$. Then, by the precondition (7.5) and $log_0 = prior_{ps}$, we have

$$log_0 \not\models_{S5} true\_pot\_sec\_disj. \tag{7.6}$$

Now, let $i > 0$. Let Φ_i be the i-th query, C_i the associated security configuration and

$$ans_i := censor(C_i, eval(\Phi_i)(db)) \in \{true, false, undef\} \tag{7.7}$$

the answer given by the censor. By Definition 6.2, condition (a), we have for all cases

$$\{censor(C_i, ans_i)\} \notin C_i. \tag{7.8}$$

By (7.8) and according to condition (b) of Definition 6.2, we also have

$$censor(C_i, ans_i) = ans_i, \tag{7.9}$$

and by (7.8) also

$$\{ans_i\} \notin C_i. \tag{7.10}$$

Thus, according to the generic construction of the security configuration (5.10), the inference set $\{ans_i\}$ cannot lead to a security violation:

$$\neg violates^{truedisj}(db, log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\})\}, policy_{ps}) \tag{7.11}$$

By the definition of the specific $violates^{truedisj}$ function (5.7), we then have

$$log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\})\} \not\models_{S5} true\_pot\_sec\_disj, \tag{7.12}$$

and finally, by the generic construction of the subsequent log file (5.13), together with the specific $inference^{ans}$ function, (7.8),

$$log_i \not\models_{S5} true\_pot\_sec\_disj. \tag{7.13}$$

□

Theorem 7.2 (Confidentiality of the uniform lying method for unknown policies)**.** $cqe^{u,L}_{ps}$ preserves confidentiality in the sense of Definition 4.27.

Proof. Let db be a database instance, $policy_{ps}$ a set of potential secrets, $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition^{u,L}_{ps}(db, prior_{ps}, policy_{ps})$ is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence. Let $\Psi \in policy_{ps}$ be a potential secret.

Case 1 ($eval_{ps}(\Psi)(db) = false$)**.** Then we choose $db' := db$ and $policy'_{ps} := policy_{ps}$, and it is trivial that $(db', prior_{ps}, policy'_{ps})$ satisfies $precondition^{u,L}_{ps}$, and that conditions (a), (b) and (c) of Definition 4.27 hold.

Case 2. $eval_{ps}(\Psi)(db) = true$ Then we have by Lemma 7.1

$$log_n \not\models_{S5} true\_pot\_sec\_disj \tag{7.14}$$

and, as Ψ is *true* in db, also

$$log_n \not\models_{S5} \Psi. \tag{7.15}$$

Thus, there must be a structure $M = (S, \mathcal{K}, \pi)$ and a state $s \in S$ such that

$$(M, s) \models log_n \text{ and} \tag{7.16}$$
$$(M, s) \not\models \Psi. \tag{7.17}$$

We generate a second database instance $db' := db_{M,s}$, as specified in Definition 5.3. According to Lemma 5.4, db' is consistent and deductively closed wrt. $\models_{PL}$. Furthermore, by Corollary 5.6, we have

$$eval_{ps}(\Psi)(db') = false \tag{7.18}$$

which satisfies condition (b) of Definition 4.27.

Furthermore, we consider the empty confidentiality policy $policy'_{ps} := \emptyset$ with its disjunction $true\_pot\_sec\_disj' = \bot$. Then we have

$$\neg violates^{truedisj}(db', prior_{ps}, policy'_{ps}) \tag{7.19}$$

which satisfies the precondition.

Finally, we prove by induction on the query number i that the same answers are generated under both $(db, policy_{ps})$ and $(db', policy'_{ps})$.

Let ans_i (ans'_i) be the answer to the i-th query Φ_i given under $(db, policy_{ps})$ $((db', policy'_{ps}))$. We assume that the answers (and thereby the log files) up to the i-th query are identical, which is obviously true for $i = 0$.

For $i > 1$, ans_i is either *true*, *false* or *undef*, as we do not allow the answer to be refused. As we use $inference^{ans}$, the sentence added to the log file is

$$\Delta^*(\Phi_i, \{ans_i\}). \tag{7.20}$$

This sentence is also contained in the final log file log_n, and as (M, s) is a model of log_n, we have

$$(M, s) \models \Delta^*(\Phi_i, \{ans_i\}). \tag{7.21}$$

By Lemma 5.5, this means that

$$eval_{ps}(\Delta^*(\Phi_i, \{ans_i\}))(db') = true, \tag{7.22}$$

and by Corollary 4.18 that

$$eval(\Phi_i)(db') = ans_i. \tag{7.23}$$

As $policy'_{ps}$ is empty, there are no harmful answers under $(db', policy'_{ps})$, so the security configuration under $(db', policy'_{ps})$ is $C'_i = \emptyset$. In other words, the censor will never lie under $(db', policy'_{ps})$, but will always return the true answer. Thus, we have

$$ans'_i = eval(\Phi_i)(db') = ans_i. \tag{7.24}$$

□

The following theorem states that $cqe^{u,L}_{ps}$ preserves both ps-normality and ps/ct-normality. This means that we can employ this method for the reduction from confidentiality targets to epistemic potential secrets (cf. Section 4.3) and from propositional potential secrets to epistemic potential secrets (cf. Section 4.4).

Theorem 7.3 (Normality-preservation of the uniform lying method for unknown policies)**.** $cqe^{u,L}_{ps}$ is ps-normality-preserving in the sense of Definition 4.35 and ps/ct-normality-preserving in the sense of Definition 4.42.

Proof. $policy'_{ps}$ is empty and thus trivially both ps-normal and ps/ct-normal. □

Corollary 7.4. The function

$$cqe^{u,L}_{ct}(Q, db, prior_{ct}, policy_{ct}) := cqe^{u,L}_{ps}(Q, db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

with the precondition

$$precondition^{u,L}_{ct}(db, prior_{ct}, policy_{ct}) := \\ precondition^{u,L}_{ps}(db, conv_{ps}(prior_{ct}), conv_{ps}(policy_{ct}))$$

preserves confidentiality wrt. confidentiality targets in the sense of of Definition 3.10.

Corollary 7.5. The function

$$cqe^{u,L}(Q, db, prior, policy) := \\ cqe^{u,L}_{ps}(Q, db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

with the precondition

$$precondition^{u,L}(db, prior, policy) := \\ precondition^{u,L}_{ps}(db, conv_{ps}(conv_{ct}(prior)), conv_{ps}(conv_{ct}(policy)))$$

preserves confidentiality wrt. propositional potential secrets in the sense of of Definition 2.8.

7.2 Uniform Refusal

In this section, we construct an enforcement method for uniform refusal under unknown potential secrets. The fact that the user does not know the confidentiality policy has a threefold positive effect on the resulting algorithm:

1. We only need to protect the true potential secrets, according to the basic heuristics outlined in the introduction of this chapter. We say that a log file *log* violates a confidentiality policy $policy_{ps}$ if and only if at least one potential secret $\Psi \in policy_{ps}$, which is *true* in *db*, is logically implied by *log*:

$$violates^{truesingle}(db, log, policy_{ps}) := \\ (\exists \Psi \in policy_{ps})[\ eval_{ps}(\Psi)(db) = true \text{ and } log \models_{S5} \Psi\ ] \quad (7.25)$$

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

Figure 7.1: $censor^{u,R}$, the refusal censor for unknown potential secrets

2. Meta inferences (cf. Section 6.2.1) cannot occur: As the user does not know the elements of $policy_{ps}$, he is not able to calculate the pre-image of a given answer. Therefore, we do not need to formally consider meta inferences anymore, and we can use the simple answer inference approach, namely the function

$$inference^{ans}(censor, C, ans) := \begin{cases} \{true, false, undef\} & \text{if } ans = refuse \\ \{ans\} & \text{otherwise} \end{cases} \tag{7.26}$$

with the range

$$\mathcal{I}^{ans} = \{\{true\},\{false\},\{undef\},\{true, false, undef\}\}. \tag{7.27}$$

The resulting set of possible security configurations is

$$\begin{aligned} C^*_{ans,truesingle} = \{ & \{\{true\},\{false\},\{undef\}\}, \\ & \{\{true\},\{false\}\},\ \{\{true\},\{undef\}\},\{\{false\},\{undef\}\}, \\ & \{\{true\}\},\ \{\{false\}\},\ \{\{undef\}\},\ \emptyset\ \}. \end{aligned} \tag{7.28}$$

3. As meta inferences cannot occur, the censor for unknown policies does not require any additional *refuse*-conditions, unlike the censor for known policies (cf. Section 6.2.3). Instead, the censor needs to refuse the answer if and only if the actual query result v would violate the confidentiality policy, i. e., $\{v\} \in C$, where C is the security configuration. There is exactly one censor function which meets these requirements, so it is not necessary to define a class of safe censors. Instead, we formally define the censor as follows:

$$censor^{u,R}(C, v) := \begin{cases} refuse & \text{if } \{v\} \in C \\ v & \text{otherwise} \end{cases} \tag{7.29}$$

The decision table of this censor is given in Figure 7.1.

We denote the resulting enforcement method (built up of $inference^{meta}$, $violates^{single}$, and $censor^{u,R}$) as $cqe_{ps}^{u,R}$. Expanding $violates^{truesingle}$, $inference^{ans}$, *sec_conf*, $censor^{u,R}$ and Δ^*, the answer generation can be rewritten as:

$$ans_i := \begin{cases} refuse & \text{if } \exists \Psi \in policy_{ps} \text{ such that } eval_{ps}(\Psi)(db) = true \text{ and} \\ & \qquad log_{i-1} \cup \{eval^*(\Phi_i)(db)\} \models_{S5} \Psi \\ eval(\Phi_i)(db) & \text{otherwise} \end{cases} \tag{7.30}$$

Furthermore, we need a stronger precondition which guarantees that $prior_{ps} = log_0$ does only contain true information:

$$\begin{aligned} &precondition_{ps}^{u,R}(db, prior_{ps}, policy_{ps}) \ := \\ &\qquad \neg violates^{truesingle}(db, prior_{ps}, policy_{ps}) \wedge \\ &\qquad (\forall \phi \in prior_{ps})[\ eval_{ps}(\phi)(db) = true\ ] \end{aligned} \tag{7.31}$$

The security proof follows the usual outline: First, we show that the precondition is kept as an invariant throughout the query sequence. Then, we prove $cqe_{ps}^{u,R}$ to preserve confidentiality.

Lemma 7.6. Let db be a database instance, $policy_{ps}$ a set of potential secrets and $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{u,R}(db, prior_{ps}, policy_{ps})$ is satisfied. Let $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ be a query sequence. Then we have for each $\Psi \in policy_{ps}$ with $eval_{ps}(\Psi)(db) = true$ and each $0 \leq i \leq n$: $log_i \not\models_{S5} \Psi$.

Proof. By induction on i. By $precondition_{ps}^{u,R}(db, prior_{ps}, policy_{ps})$ and $log_0 = prior_{ps}$, we have

$$\neg violates^{truesingle}(db, log_0, policy_{ps}), \tag{7.32}$$

which is by (7.25) equivalent to

$$log_0 \not\models_{S5} \Psi \text{ for all } \Psi \in policy_{ps} \text{ with } eval_{ps}(\Psi)(db) = true. \tag{7.33}$$

Now, let $i > 0$. Let Φ_i be the i-th query, ans_i the answer returned by the censor.

Case 1 (ans_i=*refuse*)**.** Then, by the definition of $inference^{ans}$ (7.26), the sentence added to the log file is

$$\Delta^*(\Phi_i, \{true, false, undef\}), \tag{7.34}$$

which is a tautology. Thus, log_i implies exactly the same set of sentences as log_{i-1} does, and we have

$$log_i \not\models_{S5} \Psi. \tag{7.35}$$

Case 2 ($ans_i \in \{true, false, undef\}$)**.** By the definition of $censor^{u,R}$ (7.29), we have

$$\{ans_i\} \notin C_i. \tag{7.36}$$

By the definition of *sec_conf* (5.10), this means that

$$\neg violates^{truesingle}(db, log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\})\}, policy_{ps}), \tag{7.37}$$

which is, by the definition of $violates^{truesingle}$ (7.25), equivalent to

$$log_{i-1} \cup \{\Delta^*(\Phi_i, \{ans_i\}\} \not\models_{S5} \Psi \tag{7.38}$$

for all $\Psi \in policy_{ps}$ with $eval_{ps}(\Psi)(db) = true$. As we use the answer inference approach (7.26) in order to construct the subsequent log file log_i, this coincides with

$$log_i \not\models_{S5} \Psi \tag{7.39}$$

for all $\Psi \in policy_{ps}$ with $eval_{ps}(\Psi)(db) = true$.

□

We are now able to prove that the uniform refusal method for unknown potential secrets preserves confidentiality. In the proof, we construct an alternative confidentiality policy $policy'_{ps}$ by collecting the actual query values in db' of all queries which were refused under db, and encoding these as epistemic sentences in the usual ways. The following example demonstrates this:

Example 7.7. We pick up on the scenario from Example 1.5. The confidentiality policy demands that the user may not learn that *aids* is true, and may also not learn that *cancer* is true:

$$policy_{ps} \ := \ \{ \ K\,aids, \ K\,cancer \ \}$$

Imagine a user with no a priori assumptions ($prior_{ps} = \emptyset$), and a situation in which the person under consideration suffers from aids:

$$db_1 := \{aids\}$$

When issuing the query $\Phi = aids$, the answer is refused: If we had given the actual answer *true*, the sentence $K\,aids$ would have been added to the log file, so the log file would have implied the potential secret $K\,aids$ which happens to be true in db_1.

On the other hand, if the actual database instance is

$$db_2 := \emptyset,$$

the correct answer *undef* is returned, because the new log file entry $\neg K\,aids \wedge \neg K \neg aids$ does not lead to the implication of any *true* potential secret. This demonstrates the gain of availability when only protecting *true* potential secrets — under the approach for

known policies from Section 6.2, the answer would have been refused as well, due to an additional *refuse*-condition which protects the *refuse* for the actual query value *true* under the security configuration $C = \{\{true\}\}$ (see the censor table in Figure 6.2 for reference).

Let's take a closer look at the meta inference caused by the refusal under db_1. As the answer was refused, the user knows that there must be a potential secret, which is true in the actual database instance, and which is implied by the resulting log file. However, the user does neither know the actual query value of $\Phi = aids$, nor the elements of the confidentiality policy. So, according to his observations, it could also be the case that, for example, *aids* is actually *false*,

$$db_1' := \{\neg aids\},$$

and that the confidentiality policy actually prohibits to learn that *aids* is false,

$$policy_{ps}' := \{K\neg aids\}.$$

Being able to provide an alternative confidentiality policy, the enforcement method for unknown potential secrets is always able to provide an "alibi" for a refusal which would otherwise have led to a harmful meta inference.

Theorem 7.8 (Confidentiality of the refusal method for unknown policies)**.** $cqe_{ps}^{u,R}$ preserves confidentiality in the sense of Definition 4.27.

Proof. Let db be a database instance, $policy_{ps}$ a set of potential secrets, $prior_{ps}$ a set of a priori assumptions so that the pertinent $precondition_{ps}^{u,R}(db, prior_{ps}, policy_{ps})$ is satisfied, and $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$ a query sequence. Let $\Psi \in policy_{ps}$ be a potential secret.

In case $eval_{ps}(\Psi)(db) = false$, choose $db' := db$ and $policy_{ps}' := policy_{ps}$. Then, it is trivial that $(db', prior_{ps}, policy_{ps}')$ satisfies $precondition_{ps}^{u,R}$, and that conditions (a), (b) and (c) of Definition 4.27 hold.

In case $eval_{ps}(\Psi)(db) = true$, we have by Lemma 7.6

$$log_n \not\models_{S5} \Psi. \tag{7.40}$$

Thus, there must be a structure $M = (S, \mathcal{K}, \pi)$ and a state $s \in S$ such that

$$(M, s) \models log_n \text{ and} \tag{7.41}$$
$$(M, s) \not\models \Psi. \tag{7.42}$$

We generate a second database instance $db' := db_{M,s}$ as specified in Definition 5.3. According to Lemma 5.4, db' is consistent and deductively closed wrt. $\models_{PL}$. Furthermore, by Corollary 5.6, we have

$$eval_{ps}(\Psi)(db') = false, \tag{7.43}$$

which satisfies condition (b) of Definition 4.27. We construct a confidentiality policy $policy_{ps}'$ by

$$policy_{ps}' := \{\ eval^*(\Phi_j)(db') \mid ans_j \text{ was refused under } (db, policy_{ps})\ \}. \tag{7.44}$$

By Corollary 4.19, we have for each propositional sentence Φ_j

$$eval_{ps}(eval^*(\Phi_j)(db'))(db') = true, \tag{7.45}$$

and thus

$$eval_{ps}(\Psi')(db') = true \quad \text{for each } \Psi' \in policy'_{ps}. \tag{7.46}$$

First, we show that $precondition^{u,R}_{ps}(db', prior_{ps}, policy'_{ps})$ holds. The second part, $(\forall \phi \in prior_{ps})[\ eval_{ps}(\phi)(db') = true\ ]$, follows from Lemma 5.5. For the first part,

$$\neg violates^{truesingle}(db', prior_{ps}, policy'_{ps}), \tag{7.47}$$

or equivalently

$$prior_{ps} \not\models_{S5} \Psi' \text{ for all } \Psi' \in policy'_{ps} \tag{7.48}$$

(as $eval_{ps}(\Psi)(db') = true$ for all $\Psi \in policy'_{ps}$, as shown above), let

$$\Psi' = eval^*(\Phi_j)(db') \in policy'_{ps} \tag{7.49}$$

be one of these potential secrets. Indirectly assume that

$$prior_{ps} \models_{S5} eval^*(\Phi_j)(db'). \tag{7.50}$$

As $prior_{ps} \subseteq log_n$, this means that also

$$log_n \models_{S5} eval^*(\Phi_j)(db'). \tag{7.51}$$

As (M, s) is a model of log_n, we then have

$$(M, s) \models eval^*(\Phi_j)(db'), \tag{7.52}$$

and by Lemma 5.5 also

$$eval_{ps}(eval^*(\Phi_j)(db'))(db) = true. \tag{7.53}$$

By Corollary 4.20, we then get

$$eval(\Phi_j)(db) = eval(\Phi_j)(db'). \tag{7.54}$$

We can thus rewrite (7.51) as

$$log_n \models_{S5} eval^*(\Phi_j)(db). \tag{7.55}$$

Now remember that the answer to Φ_j was refused under $(db, policy_{ps})$, which means by definition of the *censor* function that

$$log_{j-1} \cup \{eval^*(\Phi_j)(db)\} \models_{S5} \Psi^* \tag{7.56}$$

for some $\Psi^* \in policy_{ps}$ with $eval_{ps}(\Psi^*)(db) = true$, and by $log_{j-1} \subseteq log_n$ also

$$log_n \cup \{eval^*(\Phi_j)(db)\} \models_{S5} \Psi^*. \tag{7.57}$$

From (7.55) and (7.57), we then get

$$log_n \models_{S5} \Psi^*, \tag{7.58}$$

which is a contradiction to Lemma 7.6.

Finally, we show by induction on the query number i that the answers ans_i (ans'_i) are identical under $(db, policy_{ps})$ and $(db', policy'_{ps})$. Note that, if the answers are the same, the log files are identical as well. For $i = 0$, this condition is trivial. We now assume $i > 0$ and consider the answer ans_i given under $(db, policy_{ps})$. As lies are not allowed, the answer is either *refuse* or $eval(\Phi_i)(db)$.

Case 1 ($ans_i = refuse$)**.** As the i-th query was refused under $(db, policy_{ps})$, and by the construction of $policy'_{ps}$ (7.44), there is a potential secret

$$\Psi' = eval^*(\Phi_i)(db) \in policy'_{ps}. \tag{7.59}$$

Trivially, we have

$$log_{i-1} \cup \{eval^*(\Phi_i)(db')\} \models_{S5} eval^*(\Phi_i)(db'), \tag{7.60}$$

and thus also

$$log_{i-1} \cup \{eval^*(\Phi_i)(db')\} \models_{S5} \Psi'. \tag{7.61}$$

As shown above, all potential secrets $\Psi' \in policy'_{ps}$ are *true* in db'. By the definition of the censor (7.30), we thus have

$$ans'_i = refuse. \tag{7.62}$$

Case 2 ($ans_i = eval(\Phi_i)(db)$)**.** Then the sentence

$$\Delta^*(\Phi_i, \{eval(\Phi_i)(db)\}) \tag{7.63}$$
$$= \; eval^*(\Phi_i)(db) \tag{7.64}$$

is added to the log file log_{i-1} in order to establish log_i, and that sentence is also contained in log_n at the end of the query sequence: By Corollary 5.7, we then have

$$eval(\Phi_i)(db') = eval(\Phi_i)(db). \tag{7.65}$$

We thus know that Φ_i has the same value in db and db', and as lies are not allowed, we have

$$ans'_i \in \{eval(\Phi_i)(db), refuse\}. \tag{7.66}$$

We will now indirectly prove that $ans'_i = eval(\Phi_i)(db)$, and assume that

$$ans'_i = refuse. \tag{7.67}$$

By the definition of the censor function (7.30), there must be a potential secret $\Psi' \in policy'_{ps}$ with $eval_{ps}(\Psi')(db') = true$ (which is the case for all potential secrets from $policy'_{ps}$, as shown above) so that

$$log_{i-1} \cup \{eval^*(\Phi_i)(db')\} \models_{S5} \Psi'. \tag{7.68}$$

Due to (7.65), we can substitute $eval^*(\Phi_i)(db')$ by $eval^*(\Phi_i)(db)$:

$$log_{i-1} \cup \{eval^*(\Phi_i)(db)\} \models_{S5} \Psi'. \tag{7.69}$$

As $eval^*(\Phi_i)(db)$ is the sentence added to the log file log_{i-1} under $(db, policy_{ps})$, we have

$$log_i \models_{S5} \Psi'. \tag{7.70}$$

By the construction of $policy'_{ps}$, Ψ' has the form $eval^*(\Phi_j)(db')$, where Φ_j is a query which was refused under $(db, policy_{ps})$, so we have

$$log_i \models_{S5} eval^*(\Phi_j)(db'). \tag{7.71}$$

Note that log_i does only contain information which is *true* in db: The precondition (7.31) guarantees that any sentence from $log_0 = prior$ is *true* in db. Furthermore, as no lies are given, any sentence added to the log file is also *true* in db. We thus have

$$eval_{ps}(\psi)(db) = true \quad \text{for all } \psi \in log_i, \tag{7.72}$$

and by (7.71) also

$$eval_{ps}(eval^*(\Phi_j)(db'))(db) = true. \tag{7.73}$$

From Corollary 4.20, we then conclude that

$$eval(\Phi_j)(db) = eval(\Phi_j)(db'), \tag{7.74}$$

and thus also

$$eval^*(\Phi_j)(db) = eval^*(\Phi_j)(db'). \tag{7.75}$$

By (7.71), we then have

$$log_i \models_{S5} eval^*(\Phi_j)(db), \tag{7.76}$$

and as $log_i \subseteq log_n$ also

$$log_n \models_{S5} eval^*(\Phi_j)(db). \tag{7.77}$$

Remember that $ans_j = \mathit{refuse}$, so there must be a potential secret $\Psi^* \in \mathit{policy}_{ps}$ so that $\mathit{eval}_{ps}(\Psi^*)(db) = \mathit{true}$ and

$$\mathit{log}_{j-1} \cup \{\mathit{eval}^*(\Phi_j)(db)\} \models_{S5} \Psi^*. \tag{7.78}$$

As $\mathit{log}_{j-1} \subseteq \mathit{log}_n$, and by (7.77), we then have

$$\mathit{log}_n \models_{S5} \Psi^*, \tag{7.79}$$

which is a contradiction to Lemma 7.6. Thus, $ans'_i \neq \mathit{refuse}$. As lies aren't allowed either, we have $ans'_i = \mathit{eval}(\Phi_i)(db) = ans_i$.

□

We will now show that $cqe^{u,R}_{ps}$ preserves ps-normality, which means that it is suitable for the reduction from confidentiality targets.

Theorem 7.9 (Normality-preservation of the uniform refusal method for unknown policies)**.** $cqe^{u,R}_{ps}$ is ps-normality-preserving in the sense of Definition 4.35.

Proof. Reconsider the way the alternative database instance (7.44) in the proof of Theorem 7.8. Each potential secret $\Psi' \in \mathit{policy}'$ is a sentence

$$\mathit{eval}^*(\Phi_j)(db') = \Delta(\Phi_j, \mathit{eval}(\Phi_j)(db')),$$

where Φ_j is a query which was refused under $(db, \mathit{policy}_{ps})$. According to Definition 4.30, these sentences are ps-normal. □

Corollary 7.10. The function

$$cqe^{u,R}_{ct}(Q, db, \mathit{prior}_{ct}, \mathit{policy}_{ct}) := cqe^{u,R}_{ps}(Q, db, \mathit{conv}_{ps}(\mathit{prior}_{ct}), \mathit{conv}_{ps}(\mathit{policy}_{ct}))$$

with the precondition

$$\mathit{precondition}^{u,R}_{ct}(db, \mathit{prior}_{ct}, \mathit{policy}_{ct}) := \mathit{precondition}^{u,R}_{ps}(db, \mathit{conv}_{ps}(\mathit{prior}_{ct}), \mathit{conv}_{ps}(\mathit{policy}_{ct}))$$

preserves confidentiality wrt. confidentiality targets in the sense of of Definition 3.10.

The question remains whether $cqe^{u,R}_{ps}$ is also ps/ct-normality-preserving, i. e., suitable for the reduction from propositional potential secrets to epistemic potential secrets. The corresponding Definition 4.42 demands the potential secrets from policy'_{ps} to be ps/ct-normal, i. e., that they have the form $K\alpha$, where α is propositional. When we construct policy'_{ps} in the above mentioned manner, this is not necessarily true: it might happen that $\mathit{eval}(\Phi_j)(db') = \mathit{undef}$. Then $\mathit{eval}^*(\Phi_j)(db)$ is not ps/ct-normal. In a proof for ps/ct-normality, we would have to show that this cannot happen if the input policy policy_{ct}, only consists of ps/ct-normal potential secrets. However, it might also be possible to construct a situation in which policy_{ps} is in fact ps/ct-normal, but there is no database instance db' and ps/ct-normal policy'_{ps} so that the properties stated in Definition 4.27 are satisfied. In that case, we would be able to show that $cqe^{u,R}_{ps}$ does *not* preserve ps/ct-normality.

7.3 Combined Lying and Refusal

We will now try to apply the basic heuristics – "only protect the true potential secrets" – to the algorithm for combined lying and refusal from Section 6.4.

Let $cqe_{ps}^{u,C}$ be an enforcement method for combined lying and refusal under unknown potential secrets, i. e., built up of $inference^{ans}$, $violates^{truesingle}$, and an arbitrary safe lying and refusal censor (cf. Definition 6.25). (Note that there are eight safe censors, so there are eight different $cqe_{ps}^{u,C}$ methods as well.)

Unfortunately, the resulting enforcement method does *not* preserve confidentiality.

Theorem 7.11 (Non-confidentiality of the combined lying and refusal method for unknown policies)**.** $cqe_{ps}^{u,C}$ does *not* preserve confidentiality in the sense of Definition 4.27.

Proof. We give a counter-example:

$$\begin{aligned} DS &:= \{\ a, b, c\ \} \\ db &:= \{\ a, \neg b\ \} \\ policy_{ps} &:= \{\ K(a \vee b \vee c),\ K\neg b,\ \neg Ka \vee \neg Kb \vee \neg Kc\ \} \\ prior_{ps} &:= \emptyset \\ Q &:= \langle\ a \vee b \vee c\ \rangle \end{aligned}$$

Then we have

$$eval_{ps}(\Psi)(db) = true \text{ for each } \Psi \in policy_{ps}, \tag{7.80}$$

and for each $v \in \{true, false, undef\}$ there exists a $\Psi \in policy_{ps}$ so that

$$\{\Delta^*(a \vee b \vee c, \{v\})\} \models_{S5} \Psi. \tag{7.81}$$

Thus, the security configuration for the query $\Phi_1 = a \vee b \vee c$ is

$$C_1 = \{\{true\}, \{false\}, \{undef\}\}, \tag{7.82}$$

and the answer will be refused. Indirectly assume that $cqe_{ps}^{c,U}$ preserves confidentiality. We pick the potential secret $\Psi = K(a \vee b \vee c)$, and Definition 4.27 guarantees that there is an argument $(db', policy'_{ps})$ so that $precondition_{ps}(db', prior_{ps}, policy'_{ps})$ is satisfied, for which

$$eval_{ps}(K(a \vee b \vee c))(db') = false, \tag{7.83}$$

and so that the same answers are returned. $ans_1 = refuse$, so the security configuration must be $C_1' = \{\{true\}, \{false\}, \{undef\}\}$. In particular, by definition of $violates^{truesingle}$ (5.6), there must be a potential secret $\Psi' \in policy'_{ps}$ with

$$eval_{ps}(\Psi')(db') = true \tag{7.84}$$

and, as $prior_{ps}$ is the empty set,

$$\{K(a \vee b \vee c)\} \models_{S5} \Psi'. \tag{7.85}$$

Since the vocabulary is $DS = \{a, b, c\}$, we either have $\Psi' \equiv K(a \vee b \vee c)$, or Ψ' must be a tautology. In the former case, (7.83) and (7.84) are contradictory. In the latter case, $precondition_{ps}(db', prior_{ps}, policy'_{ps})$ is not satisfied, as a tautology is also implied by the empty set $prior_{ps}$. Thus, there is no suitable argument $(db', policy'_{ps})$. □

Obviously, the refusal causes a meta inference: The user does not only learn that there is a certain potential secret, but also that this secret must be true in the actual database instance.

This negative result does not come as a surprise. The corresponding attempts to find a combined lying and refusal method for unknown potential secrets in the context of complete databases failed as well, and a similar counter-example can be constructed [5].

Of course, one can still use the algorithm for known policies, even if the policy is assumed to be unknown. However, that algorithm protects *all* potential secrets, and has a potentially lower availability. All attempts to design a special, confidentiality-preserving algorithm for combined lying and refusal under unknown policies have failed so far. The question remains how to take advantage of the "unknown policy" assumption when lying and refusal is combined.

8 Additional Properties

In the previous chapters, we have presented various enforcement methods, and we have shown that they all preserve confidentiality in the sense of Definition 4.27. The purpose of this chapter is to point out further properties of our enforcement methods for special cases. In particular, we consider the situation in which the actual database instance *db* is complete, and we compare our methods to those specifically designed for complete databases. Additionally, we consider users who are not as powerful or have less knowledge about the system as assumed by Definition 4.27.

8.1 Dealing with Complete Databases

Enforcement methods for uniform lying, uniform refusal and combined lying and refusal, and for known and unknown potential secrets, have been presented in the Chapters 6 and 7, respectively. Similar enforcement methods can be found in literature in the context of *complete databases*, namely in the work by Biskup and Bonatti [5].

In this section, we investigate how our methods for incomplete databases relate to those for complete databases. In particular, we show how complete databases can be modelled within our framework for incomplete databases. We also discuss how to account for the fact that the user might be aware of the completeness of the database. As you will see, this is easily possible by extending the initial log file by a special sentence. Finally, we compare the algorithms from [5] to those from this thesis when operating on a complete database. However, we remain on a fairly informal level. Future work might introduce a formal notion of equivalence and analyze the respective enforcement methods more precisely.

Also note that the methods from [5] sometimes use different notations. For example, in the framework for complete databases, the answer *ans* to a query Φ is denoted by either Φ, $\neg\Phi$ or `mum` instead of *true*, *false* or *refuse*. Throughout this section, we will implicitly adapt the notations from Chapter 5 in order to ensure an easier comparison.

We start our investigations with a general comparison of the enforcement methods for complete databases and those for incomplete databases; we then take a closer look at the underlying database models, the log update mechanisms and the confidentiality definitions.

Model-theoretic vs. proof-theoretic approach

Previous work on complete databases relies on a *model-theoretic* approach: A (complete) logic database *db* is defined as a structure of some logic, for example, a propositional interpretation over some database schema (set of propositions) DS. A query Φ within *db* is evaluated as follows: If *db* is a model of Φ, the value is *true*, otherwise the value is *false*.

In the *proof-theoretic* approach used in this thesis, we can define a complete logic database as follows:

Definition 8.1 (Complete logic database)**.** A complete logic database db over a schema DS is a consistent set of propositional sentences, using only propositions from DS, so that for each proposition $\Pi \in DS$ either $db \models_{PL} \Pi$ or $db \models_{PL} \neg\Pi$ holds.

Lemma 8.2. Let db be a complete logic database according to Definition 8.1. Then we have for each query Φ: $eval(\Phi)(db) \in \{true, false\}$.

Propositional vs. epistemic potential secrets

In the context of complete databases, the confidentiality policy is specified as a set of propositional potential secrets. In contrast, our methods for incomplete databases expect the potential secrets to be epistemic sentences of the restricted language $\mathcal{L}_{PS}$ (cf. Definition 4.5). However, in Chapters 2 and 3, we have shown that propositional sentences can be protected as potential secrets in the context of incomplete databases as well: We can convert them into confidentiality targets (using the function $conv_{ct}$) and then into epistemic potential secrets (using the function $conv_{ps}$). This reduction is possible if the underlying enforcement method preserves ps/ct-normality (cf. Corollary 4.45).

Completeness

It might be useful to assume that the user is *aware* of the fact that the database is complete. In other words, the user knows that each proposition $\Pi \in DS$ evaluates to either *true* or *false*. This knowledge can be incorporated into the user's initial assumptions by adding the following epistemic sentence to the initial log file log_0:

$$complete(DS) := \bigwedge_{\Pi \in DS} K\Pi \vee K\neg\Pi \tag{8.1}$$

The awareness of a complete database does also have an impact on our notion of confidentiality as stated in Definition 4.27, which requires an alternative database instance db' in which a given potential secret Ψ is not *true*, and so that the same answers are returned. It is reasonable to demand this alternative database instance to be a complete one as well.

Fortunately, the inclusion of $complete(DS)$ into log_0 guarantees this property. The alternative database db' is constructed as stated in Definition 5.3: We choose an $\mathcal{M}_{DS}$-structure $M = (S, \mathcal{K}, \pi)$ and a state $s \in S$ such that (M, s) is a model of log_n but not a model of Ψ, and we derive $db' := db_{M,s}$ from (M, s). As $complete(DS)$ is included in log_n, (M, s) is a model of $complete(DS)$. Hence, each proposition Π from DS will occur as either Π or $\neg\Pi$ in db'. Thus, db' is also complete according to Definition 8.1.

Log update

In our framework for incomplete databases, the log file is a set of epistemic sentences, and we use two different approaches when adding the information gained from an answer to

the log file: answer inferences and meta inferences. In contrast, the methods for complete databases only allow propositional logic within the log file, and all methods use the same log file update mechanism:

$$log_i := \begin{cases} log_{i-1} \cup \{\Phi_i\} & \text{if } ans_i = true, \\ log_{i-1} \cup \{\neg\Phi_i\} & \text{if } ans_i = false, \\ log_{i-1} & \text{otherwise} \end{cases}$$

This mechanism is similar to answer inferences approach (5.1). In particular, note that answer log update uses the inference $\{true, false, undef\}$ in order to represent refusal, which results in a tautology being added to the log file. This has the same effect on the logical consequences as keeping the previous log file.

How about meta inferences? Any non-empty subset of $\{true, false, undef\}$ can occur as a meta inference V. However, as $complete(DS)$ is already included in the log file, the user can implicitly rule out the value *undef*. Formally, we say that the user is able to draw a *secondary meta inference* $V' := V \cap \{true, false\}$, which, in general, contains more information than the original meta inference. This secondary meta inference is either $\{true, false\}$ (which means no gain of information, as this disjunction is already implied by $complete(DS)$), $\{true\}$ or $\{false\}$, where the latter two inference sets represent definitive information about the query value. Thus, in principle, we have the same three cases as with answer log update.

Despite these similarities, we must be careful when claiming two methods to be equivalent (as we do in the sections below) just because the censors make the same decision, and the sentences added to the log file are similar to a certain extend. For example, for a true functional equivalence we would have to prove that adding $K\Phi$ to an epistemic log file has the same effect (wrt. logical implication) as adding Φ to a propositional log file.

The censor

In the following, we assume that we have a complete database *db* and that $complete(DS)$ is included in log_0, and we compare the censors of our methods to those from [5]. In general, the completeness of *db* has a twofold effect on our algorithm when issuing a query Φ:

1. The original query value $eval(\Phi)(db)$ can never be *undef*, so we can eliminate the *undef*-column from the censor table.

2. It is always harmful to "learn" that the query value is *undef*, as this would contradict to $complete(DS)$ and thus lead to the inconsistency of the log file. Thus, $\{undef\}$ is always a harmful inference, and we only need to consider the security configurations (and rows in the censor table) which contain $\{undef\}$.

We can therefore specify a *restricted censor table* for each method and then easily compare the remaining cases to the algorithm of the respective method for complete databases.

There is one exception: The improved uniform refusal method for known policies from Section 6.3 explicitly considers the user's a priori knowledge about a certain query value

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

Figure 8.1: The uniform lying censor from Section 6.1, restricted to the relevant cases for complete databases

and picks a particular censor function. Thus, only the censor functions for the pre-inferences $\{true, false\}$, $\{true\}$ and $\{false\}$ will be relevant.

We are now ready to investigate the particular enforcement methods for similarities and equivalence.

Known Potential Secrets

Uniform Lying (Section 6.1)

The algorithm of the uniform lying censor for known potential secrets from [5] can be summarized as follows:

> If adding the correct answer to the log file implies the disjunction of all potential secrets, then return the negation of the correct answer (i. e., *false* instead of *true*, and *true* instead of *false*).
> Otherwise, return the correct answer.

Figure 8.1 shows the uniform lying censor table restricted by the constraints mentioned above. Only three possible security configurations remain. Apparently, this censor pursues the same strategy as the one for complete databases: If an answer (and *undef*) is harmful, choose the respective harmless answer. If no answer (only *undef*) is harmful, return the correct answer. This corresponds to the above sketched algorithm of the uniform lying censor for complete databases.

Uniform Refusal (Section 6.2)

The algorithm of the uniform refusal censor for known potential secrets from [5] can be summarized as follows:

> If adding the correct answer or adding the negated answer to the log file implies any potential secret, then return *refuse*.
> Otherwise, return the correct answer.

Implicitly, this censor uses additional *refuse*-conditions as introduced in Section 6.2: A refusal is issued even if only the negated answer would reveal a secret, in order to protect the "real" refusal under the correct answer.

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

Figure 8.2: The uniform refusal censor from Section 6.2, restricted to the relevant cases for complete databases

Figure 8.2 shows the restricted censor table of the uniform refusal censor from Section 6.2. The first line represents all supersets of $\{\{true\}, \{false\}, \{undef\}\}$, all of which are handled in the same way. In the second line, $\{true\}$ is a harmful inference, and of course also $\{undef\}$. As the user knows that the actual query value cannot be *undef*, the disjunctive inference $\{true, undef\}$ must be harmful as well, as its secondary meta inference $\{true, undef\} \cap \{true, false\} = \{true\}$ is harmful. The same argument applies to the third line of the censor table: When $\{false\}$ is harmful, $\{false, undef\}$ must be harmful as well.

The first three lines of the restricted censor table exactly correspond to the algorithm for complete databases. Interestingly, under the last security configuration $\{undef\}$ (where neither answer *true* or *false* is harmful), our new censor behaves differently: If the actual query value is *false*, the answer is refused. But why? This refusal originates from an additional *refuse*-condition introduced in order to protect a harmful meta inference from the original refusal for $eval(\Phi)(db) = undef$. Although the original refusal will never occur, the additional *refuse*-condition is still active. This has a negative impact on the availability of the censor, as more answers will be refused.

What is the impact of the secondary meta inference V'? For the first three lines of the censor table, the secondary meta inference is $\{true, false\}$. This results in the log file entry $K\Phi \vee K\neg\Phi$, which is a consequence of $complete(DS)$ anyway and thus adds no new information to the log file. In the last line of the censor table, we have the following situation:

- For $ans = true$, the secondary meta inference is $\{true\}$, and thus coincides with the primary meta inference.
- For $ans = refuse$, the primary meta inference is $\{false, undef\}$ (cf. Figure 6.2, line 10), whereas the secondary meta inference is $\{false, undef\} \cap \{true, false\} = \{false\}$. In this case, the awareness of the completeness of *db* does lead to a gain of information: The user learns that the actual query value must be *false*. However, this information is not harmful in the case of this security configuration (as we have $\{false\} \notin \{\{undef\}\}$).

Thus, the secondary meta inference has no serious impact in any case, and it does not need to be considered formally.

Uniform Refusal with Improved Availability (Section 6.3)

Unlike the other enforcement methods, the improved uniform refusal method $cqe_{ps}^{k,R*}$ explicitly considers the user's a priori knowledge about a certain query value and then employs a specific censor for this pre-inference. As $complete(DS)$ is included in log_0, learning that a query value is *undef* will always make the log file inconsistent. Thus, the pre-inference will always be a non-empty subset of $\{true, false\}$. As a result, only the censor functions for the pre-inferences $\{true, false\}$, $\{true\}$ and $\{false\}$ are relevant when db is complete. The censor tables are given in Figure 6.3.

In principle, $cqe_{ps}^{k,R*}$ uses meta log update. However, note that under the above mentioned assumptions there are only three possible meta inferences: $\{true, false\}$, $\{true\}$, and $\{false\}$. In particular, any refusal will result in the meta inference $\{true, false\}$, which does not provide any useful information to the user as it coincides with the least pre-inference. On the other hand, each non-refused answer will result in either $\{true\}$ or $\{false\}$. We thus have the same behavior as the log file update for complete databases: If the answer is not refused, the definite information about the answer is added to the log file. If the answer is refused, no information is added to the log file.

For complete databases, the algorithm of the uniform refusal censor [5] can be summarized as follows:

> If adding the correct answer or adding the negated answer to the log file implies any potential secret, then return *refuse*.
> Otherwise, return the correct answer.

Obviously, this algorithm corresponds to $censor_{\{true,false\}}$ from Figure 6.3. Hence, in case the user does not have any useful pre-inference about the query value (apart from the knowledge that db is complete), the method for complete databases and $cqe_{ps}^{k,R*}$ work in the same way and are equivalent. But what if the user does already know the value of the query?

- $cqe_{ps}^{k,R*}$ will detect the unary pre-inference and choose one of the censors $censor_{\{true\}}$ or $censor_{\{false\}}$. These censors always return the correct answer (cf. Figure 6.3).
- The method for complete databases will consider that adding the false answer to the log file will lead to an inconsistency, and thus to the implication of all potential secrets. Hence, the answer is refused.

With regard to these observations, $cqe_{ps}^{k,R*}$ provides better availability for the special case in which a query value is already known to the user. However, [6] proposes an "improved" refusal method, with a pre-processing step filtering out this type of queries, and returning the correct answer before employing the censor in the first place. This is similar to the idea of our improved refusal method, which considers the pre-inference of the query. This improved refusal method is then fully equivalent to $cqe_{ps}^{k,R*}$.

Combined Lying and Refusal (Section 6.4)

The combined lying and refusal censor for complete information systems works as follows:

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

Figure 8.3: The combined lying and refusal censor from Section 6.4, restricted to the relevant cases for complete databases

> If both, adding the correct answer, and adding the negated answer to the log file implies any potential secret, then return *refuse*.
> Else, if adding the correct answer to the log file implies any potential secret, return the negated answer.
> Otherwise, return the correct answer.

With regard to the censor table, the same restrictions apply as in the case of uniform lying (see above): We only have to consider the query values *true* and *undef*, and only the security configurations C with $\{undef\} \in C$. The resulting censor is given in Figure 8.3. Again, the censor coincides with the one for complete databases from [5].

Unknown Potential Secrets

Uniform Lying (Section 7.1)

For complete databases, the uniform lying method for unknown potential secrets has been established by modifying the respective method for known potential secrets in a way that only those secrets are protected which are actually true in the given database instance. In particular, the disjunction of all *true* potential secrets is now protected, instead of the disjunction of *all* potential secrets.

For incomplete databases, we designed the respective enforcement method in the exact same manner, introducing the function $violates^{truedisj}$. Given these considerations, our enforcement method is equivalent to the one for complete databases.

Uniform Refusal (Section 7.2)

The same arguments apply for the uniform refusal method for unknown potential secrets $cqe_{ps}^{u,R}$: Only the true potential secrets are protected, by using the function $violates^{truesingle}$. This corresponds to the way the algorithm for complete databases from [5] is constructed. We also use simple answer log update, just like the enforcement method for complete databases.

Note that we have not been able to show whether $cqe_{ps}^{u,R}$ preserves ps/ct-normality or not. The protection of propositional potential secrets – as used in the framework for complete databases – is only possible if preservation of ps/ct-normality is in fact given. This remains to be an open question.

8.2 Dealing with Plain Users

Our notion of confidentiality, as formalized in Definition 4.27, is very careful when it comes to the assumptions about the user and his knowledge and abilities: The user is expected to be aware of the presence of an inference control mechanism and to know its algorithm. Additionally, the user is assumed to have unlimited computational power. In Chapters 6 and 7, we have shown that even then confidentiality can be preserved under the various parameters. However, our notion of confidentiality does not consider the opposite case: What happens if the user is *not* as powerful as assumed? The following example demonstrates how we might run into problems then.

Example 8.3. Imagine an enforcement method which returns the answer *true* on any query:

$$cqe_{ps}^{true}(Q, db, prior_{ps}, policy_{ps}) := \langle true, true, true, \ldots \rangle$$

Clearly, this enforcement method preserves confidentiality in the sense of Definition 4.27, as any other database instance db' would produce the same answer sequence.

Furthermore, suppose a system run with the following parameters:

$$\begin{aligned} Q &:= \langle s \rangle \\ db &:= \{s\} \\ prior_{ps} &:= \emptyset \\ policy_{ps} &:= \{s\} \end{aligned}$$

The answer sequence is $\langle true \rangle$. Did the system disclose the potential secret s?

The answer to this question depends on the user and his assumptions and capabilities. A highly sophisticated user who knows the algorithm of cqe_{ps}^{true} will surely not believe in the answer returned by the system – as he knows that any database instance would have produced the answer *true*, and it is thus not reliable at all. However, there might be "plain" users who do not know the algorithm of the enforcement method, do not know about or neglect the existence of an inference control mechanism at all, or lack the computational power needed to reason about the possible existence of db'. These users might simply believe in the answers returned by the system. This is certainly undesirable.

In essence, the problem is that Definition 4.27 allows the user to know the algorithm of the enforcement methods (and possibly the confidentiality policy $policy_{ps}$, and what the system regards as his a priori assumptions $prior_{ps}$). However, the confidentiality definition does not inhibit the construction of an enforcement method which explicitly *depends* on this initial knowledge and capabilities. The task of establishing a "well-behaving" enforcement method is left to the designer.

While we believe that the enforcement methods presented in Chapter 6 and 7 are "well-behaving" in this respect, we cannot provide a proof for this proposition unless we establish a formal notion for this property. The purpose of this section is to provide this kind of formal "definition" of confidentiality in the presence of "plain" users.

The idea can be summarized as follows: While the original confidentiality definition demands that there is an alternative database instance db' so that *Controlled* Query Evaluation would have returned the same answers, we now demand that there is a db' so that the same answers are returned under *ordinary* query evaluation, i. e., without any inference control mechanism. We thus implicitly ensure that there is an indistinguishable database instance in which a given potential secret is *false*; the user does not need to know anything about Controlled Query Evaluation or its functioning.

Definition 8.4 (Confidentiality for Plain Users)**.** Let cqe_{ps} be a method for Controlled Query Evaluation with $precondition_{ps}$ as its associated precondition for admissible arguments. cqe_{ps} is defined to preserve confidentiality for plain users iff

for all finite query sequences $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$,
for all instances db,
for all sets of potential secrets $policy_{ps}$,
for all potential secrets $\Psi \in policy_{ps}$,
for all a priori assumptions $prior_{ps}$
such that $(db, prior_{ps}, policy_{ps})$ satisfies $precondition_{ps}$,
there exists an instance db'
such that $eval_{ps}(\phi)(db') = true$ for each $\phi \in prior_{ps}$
and the following two conditions hold:

(a) [same answers]
$cqe_{ps}(Q, db, prior_{ps}, policy_{ps}) = \langle ans_1, \ldots, ans_n \rangle$
with $ans_i = eval(\Phi_i)(db')$

(b) [Ψ is *false* in db']
$eval_{ps}(\Psi)(db') = false$

Obviously, this notion of confidentiality is only suitable for a uniform lying method – as soon as an answer is refused, there will be no database instance db' that returns the same answer under ordinary query evaluation, as the latter only produces the answers *true*, *false*, or *undef*. How can we handle refusals? The easiest way is to assume that a "really plain" user will simply ignore refused answers, and does not reflect on the reason for the refusal. We can therefore modify our definition and restrict the "same answers"-condition to the non-refused answers:

Definition 8.5 (Confidentiality for Really Plain Users)**.** Let cqe_{ps} be a method for Controlled Query Evaluation with $precondition_{ps}$ as its associated precondition for admissible arguments. cqe_{ps} is defined to preserve confidentiality for really plain users iff

for all finite query sequences $Q = \langle \Phi_1, \ldots, \Phi_n \rangle$,
for all instances db,
for all sets of potential secrets $policy_{ps}$,
for all potential secrets $\Psi \in policy_{ps}$,
for all a priori assumptions $prior_{ps}$
such that $(db, prior_{ps}, policy_{ps})$ satisfies $precondition_{ps}$,

there exists an instance db'
such that $eval_{ps}(\phi)(db') = true$ for each $\phi \in prior_{ps}$
and the following two conditions hold:

(a) [same answers if no refusal]
$cqe_{ps}(Q, db, prior_{ps}, policy_{ps}) = \langle ans_1, \ldots, ans_n \rangle$
with $ans_i \in \{eval(\Phi_i)(db'), refuse\}$

(b) [Ψ is *false* in db']
$eval_{ps}(\Psi)(db') = false$

It is fairly obvious to see that the former definition is stronger than the latter, and that both definitions are equivalent in case no refusals are involved.

Lemma 8.6. Let cqe_{ps} be an enforcement method, preserving confidentiality for plain users in the sense of Definition 8.4. Then cqe_{ps} preserves confidentiality for really plain users in the sense of Definition 8.5 as well.

Proof. Follows from the fact that condition (b) of Definition 8.4 is stronger that the corresponding condition (b) of Definition 8.5. □

Lemma 8.7. Let cqe_{ps} be a uniform lying enforcement method, i. e., it never returns *refuse* as an answer. Then Definition 8.4 and Definition 8.5 are equivalent.

Comparing Definitions 8.4 and 8.5 to the original Definition 4.27, we observe two differences:

- Instead of demanding satisfaction of some specific precondition for db', the new definitions require that the initial user assumptions $prior_{ps}$ hold in the alternative database instance db'. This is a reasonable requirement, as it guarantees that the (ordinary) answers returned by the system under db' do not contradict to $prior_{ps}$. However, the original confidentiality definition does *not* require this condition, neither for the original database instance db nor for the alternative database instance db'. This was discussed in Section 5.5.

- Both definitions do not distinguish between known and unknown policies. There are two arguments why this kind of differentiation does not make sense here, a formal and an intuitive one.

  For the formal argument, take a closer look at the original Definition 4.27: A potential secret Ψ is considered protected if there is an alternative database instance db' and an alternative set of potential secrets $policy'_{ps}$ so that Ψ is *false* in db' and the same answers are returned under under $(db, policy_{ps})$ and $(db', policy'_{ps})$. In case the user is assumed to know the confidentiality policy, $policy'_{ps}$ must also correspond to $policy_{ps}$. On the other hand, our confidentiality definitions for plain users demand that the same answers are returned under *ordinary* query evaluation. However, ordinary query evaluation does not incorporate any confidentiality policy, so we don't consider an alternative policy $policy'_{ps}$ anyway, and it is irrelevant if it was identical to $policy_{ps}$ or not.

Intuitively, remember that the user does not assume an inference control mechanism to be present, and thus also no confidentiality policy. In this respect, the confidentiality policy is always "unknown" to the user.

In the following, we will investigate whether the enforcement methods from Chapters 6 and 7 satisfy these definitions. For the uniform lying methods, we will investigate confidentiality for plain users, and for the uniform refusal methods and the combined lying and refusal methods, we will consider confidentiality for really plain users.

Known Potential Secrets

Uniform Lying (Section 6.1)

Theorem 8.8 (Confidentiality for plain users of the uniform lying method for known policies)**.** $cqe_{ps}^{k,L}$ preserves confidentiality for plain users in the sense of Definition 8.4.

Proof. Consider the alternative database instance db' constructed in the proof of Theorem 6.4 by choosing an $\mathcal{M}_{DS}$-structure $M = (s, \mathcal{K}, \pi)$ and a state $s \in S$ so that $(M, s) \models log_n$ and $(M, s) \not\models \Psi$.

As $prior_{ps} \subseteq log_n$, $(M, s) \models \phi$ holds for each $\phi \in prior_{ps}$. By Lemma 5.5, we then have

$$eval_{ps}(\phi)(db') = true \text{ for each } \phi \in prior_{ps}. \tag{8.2}$$

Condition (b) of Definition 8.4 is satisfied by (6.16). Furthermore, it is shown in that proof that no query value is distorted under db'. We thus have

$$ans_i = eval(\Phi_i)(db') \text{ for each } 1 \leq i \leq n, \tag{8.3}$$

which satisfies condition (a) of Definition 8.4. □

Uniform Refusal (Section 6.2)

Theorem 8.9 (Confidentiality for really plain users of the uniform refusal method for known policies)**.** $cqe_{ps}^{k,R}$ preserves confidentiality for really plain users in the sense of Definition 8.5.

Proof. Consider the alternative database instance db' constructed in the proof of Theorem 6.12 by choosing an $\mathcal{M}_{DS}$-structure $M = (s, \mathcal{K}, \pi)$ and a state $s \in S$ so that $(M, s) \models log_n$ and $(M, s) \not\models \Psi$.

As $prior_{ps} \subseteq log_n$, $(M, s) \models \phi$ holds for each $\phi \in prior_{ps}$. By Lemma 5.5, we then have

$$eval_{ps}(\phi)(db') = true \text{ for each } \phi \in prior_{ps}. \tag{8.4}$$

Condition (b) of Definition 8.5 is satisfied by (6.33). Furthermore, $cqe_{ps}^{k,R}$ does never lie, so we have

$$ans_i \in \{eval(\Phi_i)(db'), refuse\} \text{ for each } 1 \leq i \leq n, \tag{8.5}$$

which satisfies condition (a) of Definition 8.5. □

Uniform Refusal with Improved Availability (Section 6.3)

Theorem 8.10 (Confidentiality for really plain users of the improved uniform refusal method for known policies)**.** $cqe_{ps}^{k,R*}$ preserves confidentiality for really plain users in the sense of Definition 8.5.

Proof. Consider the alternative database instance db' constructed in the proof of Theorem 6.20 by choosing an $\mathcal{M}_{DS}$-structure $M = (s, \mathcal{K}, \pi)$ and a state $s \in S$ so that $(M, s) \models log_n$ and $(M, s) \not\models \Psi$.

As $prior_{ps} \subseteq log_n$, $(M, s) \models \phi$ holds for each $\phi \in prior_{ps}$. By Lemma 5.5, we then have

$$eval_{ps}(\phi)(db') = true \text{ for each } \phi \in prior_{ps}. \tag{8.6}$$

Condition (b) of Definition 8.5 is satisfied by (6.60). Furthermore, $cqe_{ps}^{k,R*}$ does never lie, so we have

$$ans_i \in \{eval(\Phi_i)(db'), refuse\} \text{ for each } 1 \leq i \leq n, \tag{8.7}$$

which satisfies condition (a) of Definition 8.5. □

Combined Lying and Refusal (Section 6.4)

Theorem 8.11 (Confidentiality for really plain users of the combined lying and refusal method for known policies)**.** $cqe_{ps}^{k,C}$ preserves confidentiality for really plain users in the sense of Definition 8.5.

Proof. Consider the alternative database instance db' constructed in the proof of Theorem 6.27 by choosing an $\mathcal{M}_{DS}$-structure $M = (s, \mathcal{K}, \pi)$ and a state $s \in S$ such that $(M, s) \models log_n$ and $(M, s) \not\models \Psi$.

As $prior_{ps} \subseteq log_n$, $(M, s) \models \phi$ holds for each $\phi \in prior_{ps}$. By Lemma 5.5, we then have

$$eval_{ps}(\phi)(db') = true \text{ for each } \phi \in prior_{ps}. \tag{8.8}$$

Condition (b) of Definition 8.5 is satisfied by (6.87). Furthermore, it is shown in that proof that no lies are returned under db'. We thus have

$$ans_i = \{eval(\Phi_i)(db'), refuse\} \text{ for each } 1 \leq i \leq n, \tag{8.9}$$

which satisfies condition (a) of Definition 8.5. □

Unknown Potential Secrets

Uniform Lying (Section 7.1)

Theorem 8.12 (Non-confidentiality for plain users of the uniform lying method for unknown policies)**.** $cqe_{ps}^{u,L}$ does **not** preserve confidentiality for plain users in the sense of Definition 8.4.

Proof. We specify a counter-example:

$$\begin{aligned} DS &:= \{\ s_1,\ s_2\ \} \\ Q &:= \langle\ s_2\ \rangle \\ db &:= \{\ s_1\ \} \\ prior_{ps} &:= \{\ Ks_1 \vee Ks_2\ \} \\ policy_{ps} &:= \{\ Ks_1,\ Ks_2\ \} \end{aligned}$$

This is a valid input to $cqe_{ps}^{u,L}$ according to $precondition_{ps}^{u,L}$ (7.5), as $prior_{ps}$ does not imply the disjunction of all true potential secrets $true\_pot\_sec\_disj = Ks_1$. Regarding the query $\Phi_1 = s_2$, the security configuration is $C_1 = \{\{false\}, \{undef\}, \{false, undef\}\}$, because learning that Ks_2 does not hold would imply that Ks_1 must hold, which is a true potential secret. The resulting answer is $ans_1 = true$.

It is easy to see that there is no instance db' with $eval(s_2)(db') = true$ (to return the same answer under ordinary query evaluation) and $eval_{ps}(Ks_2)(db') = false$ (to make the potential secret false). □

We encounter a paradox behavior: Assuming that the user knows the confidentiality policy and is also aware of the presence and the algorithm of CQE, $cqe_{ps}^{u,L}$ is unsafe (this can be verified by the same counter-example). When we assume that the user does know the CQE algorithm but not the confidentiality policy, $cqe_{ps}^{u,L}$ is safe (cf. Theorem 7.2). When we assume that the user does neither know the CQE algorithm nor the confidentiality policy, $cqe_{ps}^{u,L}$ is unsafe again.

What is the reason for this oddity? Obviously, $cqe_{ps}^{u,L}$ explicitly relies on the fact that the user knows the CQE algorithm, and that he can tentatively test other database instances and confidentiality policies for the resulting answer sequence. In the example given above, the system did claim s_2 to be *true*; however, the user might still assume that Ks_2 is not a potential secret, in which case an indistinguishable input $(db', policy'_{ps})$ can be found.

This result suggests that we should be careful when employing the methods for unknown potential secrets – implicitly, they assume that the user does know about CQE and its algorithm. If he doesn't, confidentiality might be threatened.

Uniform Refusal (Section 7.2)

Theorem 8.13 (Confidentiality for really plain users of the uniform refusal method for unknown policies)**.** $cqe_{ps}^{u,R}$ preserves confidentiality for really plain users in the sense of Definition 8.5.

Proof. Let Ψ be the potential secret under consideration, as given in Definition 8.5.

If $eval_{ps}(\Psi)(db) = false$, choose $db' := db$. Condition (b) of Definition 8.5 is then trivially satisfied. By the precondition (7.31) for the uniform refusal method, and by $db' = db$, we have

$$(\forall \phi \in prior_{ps})[\ eval_{ps}(\phi)(db') = true\ ]. \tag{8.10}$$

Furthermore, the uniform refusal method does never lie, so we have for all $1 \leq i \leq n$,

$$ans_i \in \{eval(\Phi_i)(db'), refuse\}, \tag{8.11}$$

which satisfies condition (a).

Otherwise, if $eval_{ps}(\Psi)(db) = true$, consider the alternative database instance db' constructed in the proof of Theorem 7.8 which satisfied the conditions from the ordinary confidentiality Definition 4.27. The required properties can then be extracted from that proof:

- As $precondition_{ps}^{u,R}$ (7.31) is satisfied for $(db', prior_{ps}, policy'_{ps})$, we have $eval_{ps}(\phi)(db') = true$ for all $\phi \in prior_{ps}$.
- By condition (a) of Definition 4.27, the same answers are returned under both $(db, policy_{ps})$ and $(db', policy'_{ps})$. Furthermore, the system will never lie, so we have $ans_i \in \{eval_{ps}(\Phi_i)(db), refuse\}$ for all $1 \leq i \leq n$.
- By condition (b) of Definition 4.27, we have $eval_{ps}(\Psi)(db') = false$.

□

Discussion

The results from this section indicate that the user model implicitly incorporated in our fundamental notion of confidentiality stated in Definition 4.27 does not necessarily cover all aspects of a "reasonable" user. If we only rely on that definition when constructing an enforcement method, we might end up in a situation in which confidentiality is only protected against a sophisticated user, but not against other "reasonable" users.

The plain user model established in this section reveals two properties not found in the original "sophisticated" user model:

- *Consistency:* The plain user only accepts answers that are consistent with each other and with his a priori assumptions.
- *No secret told:* As the plain user believes in any information returned by the system, it is not acceptable to "tell" a secret and implicitly "hope" that the user does not believe in what he was told. This is different from the sophisticated user, who is assumed to challenge all answers.

It would be favorable to establish a user model which fits all kinds of "reasonable" users. This user model could be incorporated into a common confidentiality definition for all kinds of users, which combines the requirements stated in the current Definitions 4.27, 8.4 and 8.5. This is goes beyond the scope of this thesis and could be the topic of future work.

9 Conclusion

In this thesis, we have presented a framework for policy-driven inference control in incomplete logic databases. We have studied three different kinds of confidentiality policies with varying expressiveness: propositional potential secrets, confidentiality targets, and epistemic potential secrets. For each kind of policy, we have introduced an abstract definition of an enforcement method, and established a declarative notion of confidentiality, based on the idea that the actual database instance must always be indistinguishable from an alternative one in which certain secret information is not true. Furthermore, we have shown that policies of a less expressive kind can be converted into a different, more expressive kind, and that confidentiality is still preserved under this reduction, given that the underlying enforcement method satisfies certain properties of normality-preservation.

For the most expressive kind of confidentiality policies, epistemic potential secrets, we have then constructed an operational framework and presented multiple instantiations thereof: for known policies, we have studied uniform lying, uniform refusal, uniform refusal with improved availability, and combined lying and refusal; for unknown policies, we have considered uniform lying and uniform refusal, and we have also shown that there is no (naively constructed) combined lying and refusal method which takes advantage of the extra freedom provided by unknown policies. For each of the resulting enforcement methods, we have shown that it satisfies the declarative confidentiality definition for epistemic potential secrets. We have also shown that most of the methods satisfy the normality-preservation requirements which make them suitable for the reduction from the other kinds of confidentiality policies.

Furthermore, we have analyzed all enforcement methods wrt. certain additional properties they might have or not have. We have informally shown that they are, to a certain extend, equivalent to the respective existing methods for complete databases from [5]. We have also considered an alternative user model, "plain users", which revealed additional insights on "useful" assumptions about a real-world user, his abilities and his behavior.

A complete overview of all enforcement methods studied in this thesis, along with their respective properties, is given in Figure 9.1.

The starting point for this thesis was the work by Biskup and Bonatti [5], who investigated Controlled Query Evaluation for complete databases and thereby considered three parameters: user awareness (known or unknown policy), the kind of confidentiality policy (potential secrets or secrecies) and the allowed distortion methods (uniform lying, uniform refusal, combined lying and refusal). Two of these parameters – user awareness and allowed kinds of distortion – can also explicitly be found in the framework presented in this thesis. The remaining parameter – the kind of confidentiality policy – is implicitly encoded into our definition of epistemic potential secrets. In the work by Biskup and Bonatti, a secrecy $(\Psi, \neg\Psi)$ declares that the user may not learn the exact value of some sentence Ψ.

	Known Policies	Unknown Policies
Uniform Lying		
Inferences	$inference^{ans}$	$inference^{ans}$
Security violations	$violates^{disj}$	$violates^{truedisj}$
Confidentiality	yes	yes
Confidentiality for plain users	plain	no
Normality-preservation	ps/ct-normality	ps/ct-normality
Uniform Refusal		
Inferences	$inference^{meta}$	$inference^{ans}$
Security violations	$violates^{single}$	$violates^{truesingle}$
Confidentiality	yes	yes
Confidentiality for plain users	really plain	really plain
Normality-preservation	ps/ct-normality	ps-normality[a]
Improved Uniform Refusal		
Inferences	$inference^{meta}$	
Security violations	$violates^{single}$	
Confidentiality	yes	(no algorithm needed)
Confidentiality for plain users	really plain	
Normality	ps/ct-normality	
Combined Lying and Refusal		
Inferences	$inference^{ans}$	
Security violations	$violates^{single}$	
Confidentiality	yes	(no algorithm found)
Plain users	really plain	
Normality-preservation	ps/ct-normality	

[a] might even preserve ps/ct-normality, no proof found

Figure 9.1: Overview of the algorithms and their properties

For our framework, we have shown that this can be achieved by introducing three distinct potential secrets $\Delta^*(\Psi, \{true, false\})$, $\Delta^*(\Psi, \{true, undef\})$ and $\Delta^*(\Psi, \{false, undef\})$. In total, these secrets prevent the user from learning any (even disjunctive) information about the actual value of Ψ. (The same technique – emulating secrecies by means of a set of potential secrets – has been introduced as a "naive reduction" for complete information systems, see e. g. [3]). This is why we don't need to consider special enforcement methods for "secrecy-like" confidentiality policies, and why we only had to consider six cases instead of twelve. Generally speaking, our results coincide with those for complete databases:

- All algorithms are designed in a similar way and share the main ideas.
- We have shown that there is no way to "naively" design a specific algorithm for combined lying and refusal under unknown potential secrets. The same result can be found for complete databases in [3].
- For complete information systems, it was shown that it is not possible to perform uniform lying under known secrecies [5]: Given a secrecy $(\Psi, \neg\Psi)$, the user could simply issue the query $\Phi = \Psi$, and he would know that the answer is exactly the negation of the actual query value.

  Although we haven't explicitly introduced a concept like secrecies for incomplete information systems, we can show that the same property also holds for our "emulated" secrecies. Given the three potential secrets $\Delta^*(\Psi, \{true, false\})$, $\Delta^*(\Psi, \{true, undef\})$ and $\Delta^*(\Psi, \{false, undef\})$, the disjunction of these three secrets

  $$K\Psi \vee K\neg\Psi \quad \vee \quad K\Psi \vee \neg K\Psi \wedge \neg K\neg\Psi \quad \vee \quad K\neg\Psi \vee \neg K\Psi \wedge \neg K\neg\Psi$$

  is a tautology, so *pot_sec_disj* will always be implied by $prior_{ps}$, and accordingly the precondition will never be satisfied.

Despite the comprehensive results, this thesis still leaves open some questions. In the following, we will reconsider these issues and provide hints for future work.

Higher logics

The framework established in this thesis is restricted to propositional logic. However, propositional logic is of low interest for real-world applications. It is therefore favorable to adapt this framework to higher logics, for example to first-order logic. In particular, one will have to take special care of properties of the resulting modal logic when introducing the K operator on top of the underlying logic. For first-order logic, the textbook by Fitting and Mendelsohn [24] would be a good starting point.

For complete databases, Biskup and Bonatti [5] have even gone one step further: They did not choose a particular, fixed logic, but considered an abstract, "suitable" logic in the first place. For incomplete databases and the framework from this thesis, it should be possible to identify the exact requirements for the underlying logic, as well. The main task would be to review the definitions and lemmas of Chapter 4, and the construction and properties of the alternative database instance $db_{M,s}$ in Chapter 5.

Complexity issues

Even with propositional logic, we run into a very high complexity. The main reason is the implication problems which need to be solved for both the *eval* function and the various versions of the *violates* function. A first analysis of the computational complexity can be found in [17]. In a nutshell, our methods have a time-complexity of $\mathcal{O}(2^{N_v})$, where N_v is the number of propositions in the "needed" part of the database schema (i. e., occurring in at least one query, the a priori assumptions or the confidentiality policy).

If we extended our framework to first-order logic, things get even worse. In unrestricted first-order logic, logical implication is not decidable, and so are our algorithms.

In order to make CQE a useful approach for real-world applications, it is necessary to maintain decidability, or even lower the complexity. One promising approach is to restrict the query and/or policy language to certain syntactical sentences, for example horn clauses or sentences of the Bernays-Schönfinkel fragment of first-order logic. This has been proposed for open queries to complete databases in [8]. The questions remains open whether similar restrictions may be a useful approach for incomplete databases as well, and, in particular, whether these restrictions are affected by our use of modal logic in the log file.

Explicit availability

This thesis addresses the conflict between confidentiality and availability: We want to give as many "useful" answers while still protecting confidential information. Confidentiality is addressed by our central Definition 4.27, on which most of our considerations are based.

However, we have no formal definition for availability. For example, take a look at the enforcement method sketched in Example 8.3, which always returns the answer *true* to each query. According to Definition 4.27, this enforcement method does in fact preserve confidentiality, but it is obvious that it provides very little availability (in terms of non-distorted answers).

It is also easy to construct an enforcement method which preserves confidentiality both in the ordinary way (Definition 4.27) and for plain users (Definition 8.4): Simply choose an *arbitrary* "inference-proof" database instance db' in which all potential secrets are false. As the precondition, ensure that the disjunction of all potential secrets is no tautology (otherwise, no suitable db' can be found which makes all secrets false at the same time). Finally, answer each query within db' using ordinary query evaluation. Depending on how much the (arbitrarily chosen) db' differs from db, this method does also provide fairly low availability, as there might be a large number of unnecessary lies. In [14, 15], Biskup and Wiese present a similar technique as an alternative to (dynamic) CQE. However, as an explicit availability constraint, they demand that the alternative database instance should have a minimal distance to the original database instance (where both instances are represented by propositional interpretations), i. e., the number of propositions with a different value should be as low as possible.

The enforcement methods presented in this thesis were implicitly designed in a way that they distort the least number of answers as possible. For example, property (b) of the

safeness definition for uniform lying censors (cf. Definition 6.2) demands that an answer must not be distorted if it is safe to return the true answer. For uniform refusal, we have added only the required number of additional *refuse*-conditions so that no harmful meta inferences are possible and the safeness according to Definition 6.10 is given. If we took away any *refuse* from the censor table, this property would be violated.

However, there is still no formal definition for "maximum" availability. It would be nice to have this kind of formal notion. We could then try to prove that our mechanisms do in fact provide the best possible availability.

Combined lying and refusal for unknown policies

In Section 7.3, we have shown that the naive design heuristics for unknown policies — "protect only the true potential secrets instead of all potential secrets" — does not work for the combined lying and refusal method. The questions remains open whether the combined lying and refusal approach can take advantage of unknown policies in any way.

The counter-example used in the proof of Theorem 7.11 indicates that the problem must be related to meta inferences caused by refusals: The user can infer that a certain potential secret must exist, and that this secret must in fact be true in the actual database instance. Against this background, one idea is to modify the algorithm so that refusals are made instance dependent (i. e., the answer is refused if *any* potential secret could be inferred otherwise, just like the algorithm for known policies does), while the lying condition only checks for the *true* potential secrets. In the context of this thesis, we haven't been able to determine whether this approach is successful or not. This remains to be an open question.

Bibliography

[1] Joachim Biskup. For unknown secrecies refusal is better than lying. *Data & Knowledge Engineering*, 33:1–23, 2000.

[2] Joachim Biskup and Piero A. Bonatti. Lying versus refusal for known potential secrets. *Data & Knowledge Engineering*, 38:199–222, 2001.

[3] Joachim Biskup and Piero A. Bonatti. Confidentiality policies and their enforcement for controlled query evaluation. In *ESORICS*, volume 2502 of *Lecture Notes in Computer Science*, pages 39–54. Springer, 2002.

[4] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation for known policies by combining lying and refusal. In Thomas Eiter and Klaus-Dieter Schewe, editors, *FoIKS 2002*, volume 2284 of *Lecture Notes in Computer Science*, pages 49–66. Springer, 2002.

[5] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation for enforcing confidentiality in complete information systems. *International Journal of Information Security*, 3:14–27, 2004.

[6] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation for known policies by combining lying and refusal. *Annals of Mathematics and Artificial Intelligence*, 40:37–62, 2004.

[7] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation with open queries for a decidable relational submodel. In Jürgen Dix and Stephen J. Hegner, editors, *FoIKS 2006*, volume 3861 of *Lecture Notes in Computer Science*, pages 43–62. Springer, 2006.

[8] Joachim Biskup and Piero A. Bonatti. Controlled query evaluation with open queries for a decidable relational submodel. *Annals of Mathematics and Artificial Intelligence*, 50(1-2):39–77, 2007.

[9] Joachim Biskup, Dominique M. Burgard, Torben Weibert, and Lena Wiese. Controlled Query Evaluation as a constraint satisfaction problem. In *Proceedings of the Third International Conference on Information Systems Security (ICISS)*, 2007.

[10] Joachim Biskup, Dadvid W. Embley, and Jan-Hendrik Lochner. Reducing inference control to access control for normalized database schemas. *Accepted for publication in Information Processing Letters*, 2007.

[11] Joachim Biskup and Jan-Hendrik Lochner. Enforcing confidentiality in relational databases by reducing inference control to access control. In J. Garay, editor, *Proceedings of Information Security, 10th International Conference, Valparaiso, Chile, October 9-12, 2007*, volume 4779 of *LNCS*, pages 407–422. Springer, 2007.

[12] Joachim Biskup and Torben Weibert. Confidentiality policies for Controlled Query Evaluation. In Steve Barker and Gail-Joon Ahn, editors, *21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 4602 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007.

[13] Joachim Biskup and Torben Weibert. Keeping secrets in incomplete databases. International Journal of Information Security, 2007.

[14] Joachim Biskup and Lena Wiese. On finding an inference-proof complete database for controlled query evaluation. In Ernesto Damiani and Peng Liu, editors, *20th Annual IFIP WG 11.3 Conference on Data and Applications Security, Proceedings*, volume 4127 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2006.

[15] Joachim Biskup and Lena Wiese. Preprocessing for controlled query evaluation with availability policy. *Journal of Computer Security (submitted)*, 2007.

[16] Piero A. Bonatti, Sarit Kraus, and V.S. Subrahmanian. Foundations of secure deductive databases. *IEEE Trans. on Knowledge and Data Engineering*, 7(3):406–422, 1995.

[17] Boris Brodski. Untersuchungen zur algorithmischen Komplexität der kontrollierten Anfrageauswertung. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, Lehrstuhl VI, 2007.

[18] Alexander Brodsky, Csilla Farkas, and Sushil Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):900–919, 2000.

[19] Silvano Castano, Mariagrazia Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security*. ACM Press, 1995.

[20] X. Chen and R. Wei. A dynamic method for handling the inference problem in multilevel secure databases. In *International Conference on Information Technology: Coding and Computing (ITCC'05)*, volume 1, pages 751–756. IEEE, 2005.

[21] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[22] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[23] Csilla Farkas and Sushil Jajodia. The inference problem: A survey. *SIGKDD Explorations*, 4(2):6–11, 2002.

[24] Melvin Fitting and Richard L. Mendelsohn. *First-Order Modal Logic*, volume 277 of *Synthese Library*. Kluwer Academic Publishers, 1998.

[25] Institute of Computer Science and Applied Mathematics of the University of Bern. The Logics Workbench (LWB). http://www.lwb.unibe.ch.

[26] Sushil Jajodia and Catherine Meadows. Inference problems in multilevel secure database management systems. In Marshall D. Abrams, Sushil Jajodia, and Harold J. Podell, editors, *Information Security: An Integrated Collection of Essays*, pages 570–584. IEEE Computer Society Press, 1995.

[27] Ernst L. Leiss. *Principles of Data Security*. Plenum Press, 1982.

[28] Clarence Irving Lewis. *Symbolic Logic*. Dover Publ., 1959.

[29] Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, 1990.

[30] Xiaolei Qian and Teresa F. Lunt. A semantic framework of the multilevel secure relational model. *IEEE Transactions on Knowledge and Data Engineering*, 9(2):292–301, 1997.

[31] George L. Sicherman, Wiebren de Jonge, and Reind P. van de Riet. Answering queries without revealing secrets. *ACM Trans. on Database Systems*, 8(1):41–59, 1983.

[32] Jessica Staddon. Dynamic inference control. *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 94–100, 2003.

[33] University of Dortmund, Information Systems and Security. Controlled Query Evaluation for incomplete databases, prototype implementation. http://ls6-www.cs.uni-dortmund.de/issi/projects/cqe/.

[34] Lingyu Wang, Sushil Jajodia, and Duminda Wijesekera. Securing OLAP data cubes against privacy breaches. In *IEEE Symposium on Security and Privacy*, pages 161–178. IEEE Computer Society, 2004.

[35] Lingyu Wang, Yingjiu Li, Duminda Wijesekera, and Sushil Jajodia. Precisely answering multi-dimensional range queries without privacy breaches. In *ESORICS 2003*, volume 2808 of *Lecture Notes in Computer Science*. Springer, 2003.

[36] Marianne Winslett, Kenneth Smith, and Xiaolei Qian. Formal query languages for secure relational databases. *ACM Trans. on Database Systems*, 19(4):626–662, 1994.

[37] Yanjiang Yang, Yingjiu Li, and Robert H. Deng. New paradigm of inference control with trusted computing. In Steve Barker and Gail-Joon Ahn, editors, *21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 4602 of *Lecture Notes in Computer Science*. Springer, 2007.

Index

Danksagung

Eine Arbeit wie diese kann nicht entstehen ohne die richtige Unterstützung, das richtige Umfeld und die richtige Motivation. Deshalb bin ich einer Reihe von Personen zu Dank verpflichtet, die ich an dieser Stelle erwähnen möchte.

An erster Stelle sei Joachim Biskup genannt, der es mir durch die Aufnahme in seine Arbeitsgruppe ermöglicht hat, wissenschaftlich tätig zu werden und schließlich diese Dissertation zu schreiben. Ich habe in dieser Zeit viel über das wissenschaftliche Arbeiten gelernt – wie man komplexe Ideen präzise zu Papier bringt, aber auch wie man Ansporn und Freiräume geschickt kombiniert, um diese Ideen schließlich zu einer fertigen Arbeit reifen zu lassen. Unsere unzähligen Diskussionen haben dieser Dissertation an vielen Stellen den letzten Schliff gegeben. Ebenso danke ich Gabriele Kern-Isberner für ihr Interesse an meiner Arbeit, für viele nützliche Anmerkungen und Diskussionen und für die Bereitschaft, das Zweitgutachten zu dieser Dissertation zu erstellen.

Die Diplomanden, deren Arbeiten ich betreut habe, haben mir oftmals wichtige Impulse für meine eigenen Forschungen gegeben. Besonders danke ich hier Boris Brodski, der mich in unseren Diskussionen auf die Idee gebracht hat, das Verweigerungs-Verfahren mit verbesserter Verfügbarkeit zu konstruieren.

Von allen Konferenzen, die ich besuchen durfte, war die wichtigste sicherlich die ER 2006 in Tucson, Arizona, im November 2006. In der Abgeschiedenheit der Wüste habe ich die nötige Ruhe gefunden, um wichtige Ideen und Entwürfe endlich zusammenzutragen und grundlegende Teile dieser Dissertation zu schreiben. In diesem Zusammenhang danke ich Joachim Biskup und Ralf Menzel, die mir diese Reise ermöglicht haben.

Für die moralische Unterstützung und die perfekte Arbeitsatmosphäre bedanke ich mich bei allen aktuellen und ehemaligen Kollegen des LS VI, vor allem der ISSI-Kaffeerunde. Zu nennen wären hier insbesondere: Claudia Graute, Thomas Leineweber, Jan-Hendrik Lochner, Daniel Maliga, Michael Meier, Ralf Menzel, Frank Müller, Jörg Parthe, Manuela Ritterskamp, Barbara Sprick, Matthias Thimm, Lena Wiese und Sandra Wortmann.

Bedanken möchte ich mich auch bei meinen Freunden und Ruderkameraden Gesche Born, Erik Hauptmeier und Stefan Michalski. Alle drei haben erst vor kurzem ihre Promotion erfolgreich abgeschlossen und waren mir besonders zum Ende hin eine große Motivationshilfe. Als Vierter im Bunde freue ich mich jetzt auf die Fahrt im "gemischten Doktoren-Vierer" auf der nächsten Osterwanderfahrt.

Meiner ganzen Familie danke ich für all die Unterstützung, die ich im Laufe der Jahre erfahren habe. Mein größter Dank aber gilt meiner Frau Anne – für ihre Liebe, ihr Verständnis, ihre Geduld, ihre redaktionellen Fähigkeiten, ihre Englischkenntnisse und vor allem für die Erkenntnis, dass es manchmal ganz schön gut sein kann, einen Schuhkarton voller verrückter Ideen im Kopf zu haben.

Dortmund, im Oktober 2007 Torben Weibert