Signature-Based Inference-Usability Confinement for Relational Databases under Functional and Join Dependencies*

Joachim Biskup[1] and Sven Hartmann[2] and Sebastian Link[3] and Jan-Hendrik Lochner[1] and Torsten Schlotmann[1]

[1] Fakultät für Informatik, Technische Universität Dortmund, Germany
{joachim.biskup|jan-hendrik.lochner|torsten.schlotmann}@cs.tu-dortmund.de
[2] Institut für Informatik, Technische Universität Clausthal, Germany
sven.hartmann@tu-clausthal.de
[3] Department of Computer Science, The University of Auckland, New Zealand
s.link@auckland.ac.nz

Abstract. Inference control of queries for relational databases confines the information content and thus the usability of data returned to a client, aiming to keep some pieces of information confidential as specified in a policy, in particular for the sake of privacy. In general, there is a tradeoff between the following factors: on the one hand, the expressiveness offered to administrators to declare a schema, a confidentiality policy and assumptions about a client's a priori knowledge; on the other hand, the computational complexity of a provably confidentiality preserving enforcement mechanism. We propose and investigate a new balanced solution for a widely applicable situation: we admit relational schemas with functional and join dependencies, which are also treated as a priori knowledge, and select-project sentences for policies and queries; we design an efficient signature-based enforcement mechanism that we implement for an Oracle/SQL-system. At declaration time, the inference signatures are compiled from an analysis of all possible crucial inferences, and at run time they are employed like in the field of intrusion detection.

Keywords: a priori knowledge, confidentiality policy, functional dependency, inference control, inference-usability confinement, interaction history, join dependency, refusal, relational database, select-project query, inference signature, SQL, template dependency

1 Introduction

Inference control for information systems in general and relational databases in particular is a mechanism to confine the information content and thus the usability of data made accessible to a client to whom some piece(s) of information

* This work has been partially supported by the Deutsche Forschungsgemeinschaft under grant BI 311/12-2 and under grant SFB 876/A5 for the Collaborative Research Center "Providing Information by Resource-Constrained Data Analysis".

should be kept confidential. Thus inference control aims at protecting *information* rather than just the underlying *data*, as achieved by traditional access control or simple encryption. Though protection of information is a crucial requirement for many public and commercial applications, the actual enforcement is facing great challenges arising from conceptual and computational problems.

On the conceptual side, among others the following main factors have to be considered: a client-specific and declaratively expressed *confidentiality policy* which might be balanced with availability demands; an assumption about the client's *a priori knowledge* regarding the information managed by the information system, which will include schema information in many cases; the client's postulated *system awareness* regarding the semantics of both the underlying information system and the monitoring control mechanism.

On the computational side, the high runtime complexity is a major concern. In fact, the fundamental semantics of a well-designed information system can be defined in terms of an appropriate logic. In particular, a *relational database* comes along with the *relational calculus* for querying and some class of *dependencies* (semantic constraints) for declaring schemas [1]. Thus, data managed by such a system can be interpreted as sentences in the underlying first-order logic. Accordingly, confining the usability of data comprises the task of monitoring all options for inferring implied (entailed) sentences from the sentences available to a client, at any point in time while the client is interacting with the system. Unfortunately, as well-known from the discipline of *theorem proving*, the computational treatment of entailment problems might be inherently complex.

Consequently, a major research task regarding information protection is to identify practically relevant situations that still enable a reasonably efficient enforcement of suitably restricted conceptual requirements. In our previous work [8] we already introduced and theoretically analyzed the following situation as highly promising: Using the *refusal* approach, where harmful correct answers are replaced by a refusal notification denoted by `mum`, we protect *select-project sentences* under closed (yes/no-)*select-project queries* evaluated for relational database instances of a schema with *functional and join dependencies*. In the present article, we present a successful elaboration of the proposed approach:

- Based on the theoretical analysis, we have designed, implemented and investigated a practical, SQL-conforming *signature-based* enforcement method.
- The inference signatures are *compiled* from an analysis of all crucial inferences that are possible for the given situation, and later on *monitored* like in the field of intrusion detection.

To set up a larger perspective, we observe that the conceptual requirements of inference-usability confinement can be captured by an invariant that a control mechanism has to guarantee for all sequences of query-response interactions. Such an *invariant* might have several forms, which are equivalent under careful formalizations [13,10]. E.g., the invariant might require that for any sentence in the confidentiality policy, based on his current knowledge, which results from the a priori knowledge and previous interactions, and his system awareness,

- the client cannot exclude that this sentence is *not* valid in the instance;
- the client does not know that this sentence is valid in the instance.

To enforce such an invariant, a control mechanism has to inspect each query considered as an interaction request and the answer to be returned whether or not they satisfy an adequate *control condition*. Clearly, a *necessary* control condition is that the current knowledge updated with the answer will not entail any sentence in the confidentiality policy. Unfortunately, however, this condition is not sufficient in general, since it neglects the impact of a client's system awareness, which might enable so-called meta-inferences. Thus, in general, we have to strengthen this condition to become *sufficient*, while preferably remaining to be necessary for the sake of availability. Moreover, as far as achievable, checking a sufficient (and necessary) control condition should be *computationally feasible*.

Several sufficient and "reasonably necessary" control conditions for comprehensive and general situations have been proposed in the past [3], which, however, inevitably tend to be infeasible in the worst case. Moreover, dedicated narrower situations have been investigated to find effective control conditions that are also efficiently testable. The contribution of the present article is particularly related to the following situations, all of which consider the refusal approach for relational databases, assuming that the client knows the confidentiality policy.

- *Situation 1.* As long as decidability is achieved, any a priori knowledge, confidentiality policy and closed (yes/no-)queries (of the relational calculus) are admitted. For the confinement, while maintaining and employing a log file that represents the a priori knowledge and the answers to previous queries, we have to ensure that adding neither the correct answer to the current query nor the negation of the correct answer will be harmful; additional refusals for harmful negations of correct answers guarantee that an observed "refused" answer `mum` cannot be traced back to its actual cause by exploiting the system awareness [4].
- *Situation 2.* The a priori knowledge may only comprise a schema declaration with functional dependencies that lead to Boyce-Codd normal form with a unique minimal key. Confidentiality policies are restricted to select-project sentences of a special kind referring to "facts", and queries are restricted to arbitrary select-project sentences. For the confinement, it suffices to ensure that the query sentence does not "cover" any policy element [6].
- *Situation 3.* The a priori knowledge may only comprise a schema declaration with functional dependencies. Confidentiality policies are restricted to select-project sentences, whereas queries must be closed select-queries. For the confinement, it suffices to ensure that the query sentence does not "cover" any policy element [9].
- *Situation 4.* The a priori knowledge may only comprise a schema declaration with functional dependencies and full join dependencies (without any further restrictions). Confidentiality policies and queries are restricted to select-project sentences. For the confinement, we have to ensure two conditions: (1) The query sentence does not "cover" any policy element. (2) Previous positive answers together with a positive answer to the current query

do not “instantiate” any template dependency implied by the schema dependencies and “covering” an element of the confidentiality policy [8]. In the present article, we will show how this requirement can be converted into an efficient enforcement mechanism.

Notably, Situations 2 to 4 postulate that the client's a priori knowledge only comprises *schema declarations*. Accordingly, if additional a priori knowledge was assumed, further potential sources of inferences should be considered, and thus the respective confinement method would have to be appropriately enhanced.

The control conditions sketched so far are devised to be used *dynamically* at run time to detect current options for crucial entailments. To avoid the runtime overhead, one might prefer a *static* approach [11]: We then interpret the confidentiality policy and the a priori knowledge as constraints to be satisfied by an alternative instance that minimally distorts the actual instance, precompute a solution to such a constrained optimization problem, and let the solution instance be queried by the client without any further control.

We might also follow a *mixed* approach that suitably splits the workload among (1) some precomputations before the client is involved at all, (2) appropriate dynamic control operations after receiving a specific client request and before returning an answer, and (3) some follow-up adaptation actions between two requests [2]. The present article follows a mixed approach for the specific relational framework of [8] described as Situation 4 above; roughly outlined, our new signature-based enforcement mechanism consists of a two-phase protocol:

- At declaration time, we compile inference signatures as representatives of “forbidden structures” in the sense of [8]: “instantiations” of template dependencies implied by the schema dependencies and “covering” a policy element.
- At run time, we monitor these inference signatures for the actual queries.

2 Formal Framework

In this section we summarize our formal framework and restate the theorem that justifies our signature-based enforcement mechanism, referring the reader to [1,8] for more details. Examples can be found in the next section.

A *relation schema* $RS = \langle R, \mathcal{U}, \Sigma \rangle$ consists of a *relation symbol* R, a finite set $\mathcal{U}$ of *attributes*, and a finite set Σ of dependencies (semantic constraints). Σ comprises either functional dependencies, assumed to be a minimal cover, or full join dependencies, or both kinds of dependencies. An *instance* r is a finite dependency-satisfying Herbrand interpretation of the schema, considering the relation symbol as a predicate. A *tuple* is denoted by $\mu = R(a_1, \ldots, a_n)$ where $n = |\mathcal{U}|$ and $a_i \in Const$, an infinite set of constants. If μ is an element of r, we write $r \models_M \mu$. More generally, $\models_M$ denotes the *satisfaction* relation between an interpretation and a sentence. The corresponding notion of logical *implication* (entailment) between sentences is denoted by $\models$.

Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{U}$ be attribute sets. A relation r over $\mathcal{U}$ satisfies the *functional dependency* $\mathcal{A} \rightarrow \mathcal{B}$ if any two tuples that agree on the values of attributes in $\mathcal{A}$ also agree on the values of the attributes in $\mathcal{B}$.

Let $\mathcal{C}_1, \ldots, \mathcal{C}_l \subseteq \mathcal{U}$ be attribute sets such that $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_l = \mathcal{U}$. A relation r over $\mathcal{U}$ satisfies the (full) *join dependency* $\bowtie [\mathcal{C}_1, \ldots, \mathcal{C}_l]$ if whenever there are tuples $\mu_1, \ldots, \mu_l$ in r with $\mu_i[\mathcal{C}_i \cap \mathcal{C}_j] = \mu_j[\mathcal{C}_i \cap \mathcal{C}_j]$ for $1 \leq i, j \leq l$, there is also a tuple μ_{l+1} in r with $\mu_{l+1}[\mathcal{C}_i] = \mu_i[\mathcal{C}_i]$ for $1 \leq i \leq l$.

Join dependencies are a special case of *template dependencies*. A *template dependency* $TD[h_1, \ldots, h_l|c]$ over $\mathcal{U}$ has one or more *hypothesis rows* $h_1, \ldots, h_l$ and a *conclusion row* c. Each row consists of abstract symbols (best seen as variables), one symbol per attribute in $\mathcal{U}$. A symbol may appear more than once but only for one attribute, i.e., in a *typed* way. For t and t' denoting tuples or rows, respectively, over $\mathcal{U}$, $ag(t, t') := \{A \mid A \in \mathcal{U} \text{ and } t(A) = t'(A)\}$ is the *agree set* of these tuples or rows, respectively. The aggregated agree sets of the conclusion $\cup_{j=1}^{l} ag(c, h_j)$ form the *scheme* of the template dependency. A symbol occurring in the conclusion for an attribute A in the scheme is often called "distinguished" (free variable) and denoted by a_A. Any other symbol in the template dependency is often called "nondistinguished" (existentially quantified variable) and denoted by b_i, where each such symbol gets a different index i.

A relation r over $\mathcal{U}$ *satisfies* the template dependency $TD[h_1, \ldots, h_l|c]$ if whenever there are tuples $t_1, \ldots, t_l$ in r with $ag(h_i, h_j) \subseteq ag(t_i, t_j)$ for all $i, j \in \{1, \ldots, l\}$ there is also a tuple t in r with $ag(c, h_i) \subseteq ag(t, t_i)$ for $i = 1, \ldots, l$.

A template dependency $TD[h_1, \ldots, h_l|c]$ is called (hypothesis-)*minimal* with respect to Σ if dropping a full hypothesis row h_i or replacing any symbol in a hypothesis by a new symbol, different from all others – thus (potentially) deleting an equality condition – would result in a template dependency that is not implied by Σ. Moreover, a minimal template dependency is called (conclusion-)*maximal* with respect to Σ if the following additionally holds: if we replace a symbol in the conclusion row c that so far is not involved in any agree set with a hypothesis by another symbol that already occurs in some hypothesis, then we would obtain a template dependency that is not implied by Σ. Finally, a template dependency enjoying both optimization properties is called a *basic* implication of Σ.

Queries and elements of a confidentiality policy *psec*, called *potential secrets*, are expressed in a fragment of the relational calculus, using a set of variables *Var*. This fragment is given by the *language* $\mathscr{L}$ of *existential-R-sentences*, or *select-project sentences* which are *sentences* (closed formulas) of the form $(\exists X_1) \ldots (\exists X_l) R(v_1, \ldots, v_n)$ with $0 \leq l \leq n$, $X_i \in Var$, $v_i \in Const \cup Var$, $\{X_1, \ldots, X_l\} \subseteq \{v_1, \ldots, v_n\}$, and $v_i \neq v_j$ if $v_i, v_j \in Var$ and $i \neq j$; these properties and the closedness imply that $\{X_1, \ldots, X_l\} = \{v_1, \ldots, v_n\} \cap Var$. For $\Phi \in \mathscr{L}$, we define the *scheme* $\mathcal{P}$ of Φ as the set of attributes for which a constant appears. For a sentence in $\mathscr{L}$ let its corresponding "generalized tuple" denote the sentence without its prefix of existential quantifiers. In this case we think of the variables in the generalized tuple as the null value "exists but unknown".

A sentence (generalized tuple) Φ is defined to *cover* a sentence (generalized tuple) Ψ if every constant c that appears in Ψ appears in Φ at the same position. The following equivalence can be easily verified: Φ covers Ψ if and only if $\Phi \models \Psi$.

Restating Theorem 2 of [8] below, we specify a "forbidden structure" an instantiation of which is necessary for any violation of a policy element by exploiting the a priori knowledge about the dependencies. Additionally, Theorem 1 of [8] indicates that an occurrence of such a structure is also sufficient for exploiting the dependencies. Accordingly, we obtain a necessary and "reasonably sufficient" *control condition* by avoiding both an immediate violation by "covering" a potential secret and an instantiated "forbidden structure"; the latter consists of an implied template dependency whose scheme comprises the scheme of a potential secret, while the schemes of the query answers "uniformly cover" the hypotheses. Our mechanism will be based on that control condition.

Theorem 1 (forbidden structures, necessary for a violation by exploiting the dependencies). *Let $RS = \langle R, \mathcal{U}, \Sigma \rangle$ be a relation schema where the dependency set Σ consists of functional and full join dependencies, and r an instance of RS. Let $\Psi \in \mathscr{L}$ be a potential secret with scheme $\mathcal{P} \subseteq \mathcal{U}$, and $\Phi_1, \ldots, \Phi_l \in \mathscr{L}$ queries with schemes $\mathcal{F}_1, \ldots, \mathcal{F}_l$ such that:*

1. *$\Phi_i \not\models \Psi$, for $i = 1, \ldots, l$, i.e., all queries do not cover the potential secret;*
2. *$r \models_M \Phi_i$, for $i = 1, \ldots, l$, i.e., all queries are true in the instance r;*
3. *$\Sigma \cup \{\Phi_1, \ldots, \Phi_l\} \models \Psi$, i.e., the answers violate the confidentiality policy.*

Then there exists a nontrivial template dependency $TD[h_1, \ldots, h_l | c]$ implied by Σ such that $\mathcal{P} = \cup_{j=1}^{l} ag(c, h_j)$ and $\cup_{j \in \{1,\ldots,l\} \smallsetminus \{i\}} ag(h_i, h_j) \subseteq \mathcal{F}_i$, for $i = 1, \ldots, l$.

3 Examples

We will outline the fundamental features of the signature-based enforcement mechanism by means of two examples. Though only dealing with functional dependencies specifying a key, the first example is beyond the scope of the Situations 2 and 3 sketched before and thus cannot be treated by the mechanisms of [6,9]. The second example introduces join dependencies as a priori knowledge.

Example 1. At *declaration time*, we consider the following items: a relation schema $RS = \langle R, \mathcal{U}, \Sigma \rangle$ with attribute set $\mathcal{U} = \{K, A, B\}$ and dependencies $\Sigma = \{K \rightarrow A, K \rightarrow B\}$, i.e., attribute K is the unique minimal key; an instance $r = \{R(c_K, c_A, c_B), R(\tilde{c}_K, c_A, c_B)\}$, where c_K, $\tilde{c}_K$, c_A and c_B are constants in *Const*; and a single potential secret $\Psi = (\exists X_K) R(X_K, c_A, c_B)$. We will compile inference signatures in four steps.
In step 1, we see that Σ entails the template dependency
$td := TD[\ a_K\ a_A\ b_1\ ,\ a_K\ b_2\ a_B\ |\ a_K\ a_A\ a_B\ ]$
as a "forbidden structure" that must not be instantiated by the potential secret and query answers according to the instance.
In step 2, first treating the potential secret, we find that the scheme KAB of the template dependency td covers the scheme AB of the potential secret Ψ.
In step 3, we specialize the conclusion $(\ a_K\ a_A\ a_B\ )$ of td with the constants appearing in the potential secret Ψ, yielding $(\ a_K\ c_A\ c_B\ )$. Then we propagate this specialization to the hypotheses on common attributes, i.e., according to agree sets, getting $(\ a_K\ c_A\ b_1\ )$ for the first hypothesis, and $(\ a_K\ b_2\ c_B\ )$ for the second hypothesis. In this way we get the instantiated template dependency

$td[\mathit{\Psi}] := TD[\, a_K\ c_A\ b_1\,,\ a_K\ b_2\ c_B \,|\, a_K\ c_A\ c_B \,]$.

In step 4, finally considering the instance r, we further uniformly instantiate the hypotheses on the distinguished symbol a_K for the further agree set $ag(h_1, h_2) = \{K\}$ with $h_1 = (\, a_K\ c_A\ b_1 \,)$ and $h_2 = (\, a_K\ b_2\ c_B \,)$ according to tuples in the instance r. For the single tuple $R(c_K, c_A, c_B) \in r$ used twice, we get

$Sig_1 := TD[\, c_K\ c_A\ b_1\,,\ c_K\ b_2\ c_B \,|\, c_K\ c_A\ c_B \,]$

as an inference signature; similarly, for the tuple $R(\tilde{c}_K, c_A, c_B) \in r$ we get

$Sig_2 := TD[\, \tilde{c}_K\ c_A\ b_1\,,\ \tilde{c}_K\ b_2\ c_B \,|\, \tilde{c}_K\ c_A\ c_B \,]$

as another inference signature. Each of them indicates that the user must not learn all of its hypotheses, and thus later on we can ignore its conclusion.

Once the inference signatures have been compiled at declaration time, they have to be monitored at *run time* according to the queries requested by the pertinent client. Suppose the client issues

$\mathit{\Phi}_1 := R(c_K, c_A, c_B)$,
$\mathit{\Phi}_2 := (\exists X_B)R(c_K, c_A, X_B)$, and
$\mathit{\Phi}_3 := (\exists X_A)R(c_K, X_A, c_B)$.

Covering the potential secret $\mathit{\Psi}$, the first query $\mathit{\Phi}_1$ is immediately refused. Though the second query $\mathit{\Phi}_2$ does not cover $\mathit{\Psi}$, it nevertheless might contribute to a forbidden structure together with other queries. So we consider the inference signatures: $\mathit{\Phi}_2$ only covers the first hypothesis $(\, c_K\ c_A\ b_1 \,)$ of Sig_1. Observing that Sig_1 has another hypothesis still uncovered, we can determine the correct query evaluation, yielding a positive answer $(\exists X_B)R(c_K, c_A, X_B)$ to be returned to the client. Moreover, we have to mark the covered hypothesis as already hit.

The third query $\mathit{\Phi}_3$ again does not cover $\mathit{\Psi}$, but the second hypothesis $(\, c_K\ b_2\ c_B \,)$ of Sig_1: independently of the correct query evaluation we have to refuse the answer for the following reasons. If the correct answer is positive, the knowledge about all hypotheses of the inference signature would enable the client to *directly infer* the validity of the conclusion and thus of the potential secret $\mathit{\Psi}$. If the correct answer is negative, this additional knowledge does not directly lead to the crucial inference; however, *only* refusing a positive answer would enable a *meta-inference* of the following kind: "the only reason for the refusal is a positive answer, which thus is valid".

Example 2. To further exemplify the compiling phase in some more detail, we now consider the relation schema $RS = \langle R, \mathcal{U}, \mathit{\Sigma} \rangle$ with attribute set $\mathcal{U} = \{S(ymptom), D(iagnosis), P(atient)\}$ and two join dependencies in $\mathit{\Sigma} = \{\bowtie[SD, SP], \bowtie[DS, DP]\}$, the instance r comprising the four tuples $R(\mathit{Fever}, \mathit{Cancer}, \mathit{Smith})$, $R(\mathit{Fever}, \mathit{Fraction}, \mathit{Smith})$, $R(\mathit{Fever}, \mathit{Cancer}, \mathit{Miller})$, and $R(\mathit{Fever}, \mathit{Fraction}, \mathit{Miller})$, and the confidentiality policy $psec = \{\mathit{\Psi}\}$ containing the single potential secret $\mathit{\Psi} = (\exists X_S)R(X_S, \mathit{Cancer}, \mathit{Smith})$.

The upper part of Figure 1 shows the dependencies as template dependencies in graphical notation known as tableau. Intuitively, the first dependency expresses the following: a symptom a_S that both contributes to a diagnosis a_D for some patient b_1, whose identity does not matter, and applies for the patient a_P contributes to the diagnosis a_D for the patient a_P as well. The meaning of the second dependency has a similar flavor. As proved in [12], the two join dependencies

S	D	P
a_S	a_D	b_1
a_S	b_2	a_P
a_S	a_D	a_P

S	D	P
a_S	a_D	b'_1
b'_2	a_D	a_P
a_S	a_D	a_P

S	D	P
(a_S, a_S)	(a_D, a_D)	(b_1, b'_1)
$(\mathbf{a_S}, \mathbf{b'_2})$	(a_D, a_D)	(b_1, a_P)
(a_S, a_S)	$(\mathbf{b_2}, \mathbf{a_D})$	(a_P, b'_1)
$(\mathbf{a_S}, \mathbf{b'_2})$	$(\mathbf{b_2}, \mathbf{a_D})$	(a_P, a_P)
(a_S, a_S)	(a_D, a_D)	(a_P, a_P)

S	D	P
a_S	a_D	b_1
$\mathbf{b_2}$	a_D	b_3
a_S	$\mathbf{b_4}$	b_5
$\mathbf{b_2}$	$\mathbf{b_4}$	a_P
a_S	a_D	a_P

Fig. 1. $\bowtie[SD, SP]$, $\bowtie[DS, DP]$, and their direct product as tableaus

together are equivalent to their *direct product* exhibited in the lower part of Figure 1, both as constructed by definition and rewritten by substituting each pair of variables by a single variable.

By Theorem 1, we have to consider all template dependencies that are implied by Σ. However, it suffices to finally employ only the basic implications. Unfortunately, so far we do not know an efficient algorithm to compute the set Σ^{+basic} of all basic implications, which even might be infinite. But, we can somehow successively generate all candidates, in turn check each candidate whether it is an implication by applying the chase procedure, see [16,12,1], and finally minimize the hypotheses and maximize the conclusion of the implied candidates.

Figure 2 shows those elements of Σ^{+basic} that have at most three hypotheses: we get the two declared dependencies and two basic versions of their direct product, obtained by deleting the third or the second hypothesis, respectively. In step 2 of the compiling phase, we identify those dependencies in Σ^{+basic} such that the scheme $\{D(iagnosis), P(atient)\}$ of the potential secret Ψ is contained in the scheme of the dependency; in this example, so far the condition is always satisfied. Accordingly, for each element of Σ^{+basic} determined so far, attributes D and P in the conclusion are instantiated with the constants *Cancer* and *Smith* occurring in Ψ. These instantiations are then propagated to the hypotheses. In the next step 3, the hypotheses must be further instantiated according to the instance r. We have to determine minimal sets of tuples such that all equalities

S	D	P
a_S	a_D	b_1
a_S	b_2	a_P
a_S	a_D	a_P

S	D	P
a_S	a_D	b_1
b_2	a_D	a_P
a_S	a_D	a_P

S	D	P
a_S	a_D	b_1
$\mathbf{b_2}$	a_D	b_3
$\mathbf{b_2}$	b_4	a_P
a_S	a_D	a_P

S	D	P
a_S	a_D	b_1
a_S	$\mathbf{b_4}$	b_5
b_2	$\mathbf{b_4}$	a_P
a_S	a_D	a_P

Fig. 2. All basic implications of Σ having two or three hypotheses

S	D	P
Fever	*Cancer*	b_1
Fever	b_2	*Smith*
Fever	*Cancer*	*Smith*

S	D	P
Fever	*Cancer*	b_1
Fever	*Fraction*	b_5
b_2	*Fraction*	*Smith*
Fever	*Cancer*	*Smith*

Fig. 3. Instantiated signatures

expressed in the respective template dependency are satisfied and their values equal the values of already instantiated entries. Finally, all remaining agree sets not considered so far are instantiated with the values of those tuples.

The instantiated template dependencies obtained so far are candidates to become inference signatures. However, we do not have to retain all them. Firstly, if an instantiated hypothesis of a candidate covers a potential secret, we can discard the candidate, since in the monitoring phase a query whose answer would reveal such a hypothesis would be refused anyway. So, in the example the instantiation of the second basic implication is discarded. Secondly, if the hypotheses of a candidate constitute a superset of the hypotheses of another candidate, then the former candidate is redundant and can be discarded as well. So, in the example the instantiation of the third basic implication is discarded as well.

For the given simple instance r, we do not have to consider basic implications with more than three hypotheses, and thus we finally keep the inference signatures shown in Figure 3. However, due to the cyclic structure of the dependencies in the example, there are basic implications with arbitrarily many hypotheses. So we can extend the basic implications having three hypotheses by a suitable fourth hypothesis, as shown in Figure 4. In fact, e.g., we can generalize the structure of the fourth dependency shown in Figure 3 by extending the present "path" a_S,a_S,b_4,b_4 by b_2,b_2,b_7,b_7,b_8,b_8,b_{10},b_{10} , as exhibited by the rightmost dependency shown in Figure 4. However, the instance r lacks sufficient

S	D	P
a_S	a_D	b_1
$\mathbf{b_2}$	a_D	b_3
$\mathbf{b_2}$	$\mathbf{b_4}$	b_5
b_6	$\mathbf{b_4}$	a_P
a_S	a_D	a_P

S	D	P
a_S	a_D	b_1
a_S	$\mathbf{b_4}$	b_5
$\mathbf{b_2}$	$\mathbf{b_4}$	b_6
$\mathbf{b_2}$	b_7	a_P
a_S	a_D	a_P

S	D	P
a_S	a_D	b_1
a_S	$\mathbf{b_4}$	b_5
$\mathbf{b_2}$	$\mathbf{b_4}$	b_6
$\mathbf{b_2}$	$\mathbf{b_7}$	b_9
$\mathbf{b_8}$	$\mathbf{b_7}$	b_{11}
$\mathbf{b_8}$	$\mathbf{b_{10}}$	b_{13}
b_{12}	$\mathbf{b_{10}}$	a_P
a_S	a_D	a_P

Fig. 4. Basic implications of Σ having four hypotheses and an example of a basic implication of Σ having "many" hypotheses

diversity to instantiate such a long path with different constants. But we could employ a single element of the instance for instantiating several hypotheses and would then obtain instantiated signatures that we already got before.

4 Compiling and Monitoring Signatures

Generalizing the example, we now present the new signature-based enforcement mechanism as a two phase protocol.

The *compiling phase* takes the dependencies Σ declared in the schema, the confidentiality policy *psec*, and the instance r as inputs, and proceeds as follows to generate the set *psig* of all inference signatures:

1. It successively, with an increasing number of hypotheses, generates all basic template dependencies implied by Σ, i.e.,
   $\Sigma^{+basic} := \{td \mid \Sigma \models td, td$ is hypothesis-minimal and conclusion-maximal$\}$,
   until no further ones can exist or further ones would not lead to nonredundant instantiations for the given instance r.
2. It determines all pairs (Ψ, td) with $\Psi \in psec$ and $td \in \Sigma^{+basic}$ as generated so far, whose components match in the sense that the scheme of Ψ is a subset of the scheme of td, i.e., of the conclusion's aggregated agree set $\cup_{j=1}^{l} ag(c, h_j)$, where $td = TD[h_1, \ldots, h_l|c]$.
3. For each such pair, the (distinguished) symbols (seen as free variables) in the scheme of td are instantiated with the respective constants appearing in Ψ. Then the instantiation is propagated from the conclusion to the hypotheses. The result is denoted by $td[\Psi]$.
4. For each $td[\Psi]$ obtained so far, the instance r is searched for a minimal set of tuples $r_{td[\Psi]}$ that "uniformly covers" all hypotheses: (i) all equalities required by $td[\Psi]$ are satisfied and, (ii) the tuple values equal the respective already propagated instantiations. For each such set, the hypotheses are further instantiated on agree sets not captured before with the respective values found in the uniform covering. The remaining symbols are left unchanged. If none of the hypotheses covers any of the potential secrets in *psec*, then the resulting inference signature $Sig(\Psi, td, r_{td[\Psi]})$ is inserted into *psig*.
5. If the hypotheses of a result of step 3 or 4 constitute a superset of the hypotheses of another result of step 3 or 4, respectively, then the former element is discarded, since it is redundant.

Given the fullness of the join dependencies in Σ, step 1 can be based on the chase algorithm [16] together with bounded searching for minimization and subsequent maximization. Though being complex in general, the computation is expected to be feasible in practice, since we only deal with schema items. Moreover, in practice, a database administrator will only admit "minor" deviations from Boyce-Codd normal form having a unique key, for example, to ensure *faithful representation* of all dependencies by relaxing Boyce-Codd normal form to 3NF or to provide support of expected queries by a dedicated *denormalization*.

Similarly, seeing the elements of the confidentiality policy as a declaration of exceptions from the general default rule of permission, we expect that in many applications steps 2 and 3 will produce only a manageable number of templates for inference signatures. Moreover, as far as the constants occurring in these templates achieve a high selectivity regarding the instance considered, step 4 will not substantially increase the number of final inference signature.

The *monitoring phase* takes a query Φ, the confidentiality policy $psec$, the set $psig$ of all inference signatures determined in the compiling phase, and the instance r as inputs, and proceeds as follows:

1. It checks whether some $\Psi \in psec$ is covered by Φ (equivalently, Ψ is entailed by Φ, see Section 2), and if this is the case, the answer is immediately refused.
2. Otherwise, it determines all hypotheses Π occurring in $psig$ and not marked before such that Π is covered by Φ, and it tentatively marks them. If now for some signature in $psig$ all hypotheses are marked, then the answer is refused and the tentative marking is aborted. Otherwise, the correct answer is determined from r and then returned, and the tentative marking is committed if a positive answer Φ is returned; otherwise, if $\neg\Phi$ is returned, the tentative marking is aborted.

A straightforward implementation of the monitoring phase keeps the potential secrets and the suitably tagged hypotheses of inference signatures in two dedicated relations. Given a query, these relations are searched for covered tuples and inspected for an inference signature becoming fully marked. Approximating the computational costs of these actions for one instantiated hypothesis by a constant, the overall runtime complexity of an execution of the monitoring phase is at most linear in the size of the dedicated relations. In the next section we will present how this rough design has been converted into an SQL-based prototype.

Theorem 2. *Assume the Situation 4 sketched in Section 1 and analyzed in [8]: the a priori knowledge is restricted to comprise only a schema declaration with functional dependencies and join dependencies, and confidentiality policies and queries are restricted to select-project sentences. Then the signature-based enforcement mechanism preserves confidentiality (in the sense of Section 1).*

Sketch of Proof. “Negative” answers of the form $\neg\Phi_i$ do not contribute to a harmful inference of a potential secret Ψ: on the one hand, the confidentiality policy contains only positive sentences and, on the other hand, the dependencies only generate positive conclusions from positive assumptions.

So let us assume indirectly that there is a harmful inference based on some minimal set of positive answers $\{\Phi_1, \ldots, \Phi_m\}$ to derive Ψ. Then, by Theorem 1 (Theorem 2 of [8]), there exists a corresponding nontrivial template dependency that witnesses such an inference. In step 1 of the compiling phase, a hypothesis-minimal and conclusion-maximal version td of this dependency is added to Σ^{+basic}; this version then witnesses the inference considered as well. In the subsequent steps 2 and 3 of the compiling phase, td together with Ψ is further processed to set up a generic signature of the form $td[\Psi]$. Furthermore, in step 4 of the compiling phase, the tuples in the instance r leading to the harmful positive answers $\Phi_1, \dots, \Phi_m$ or these answers themselves, respectively, contribute to generate an (instantiated) inference signature of the form $Sig(\Psi, td, r_{td[\Psi]})$.

Finally, in the monitoring phase, when the last of these positive answers is controlled, the tentative marking of this inference signature results in a marking of all its hypotheses, and thus the answer is refused. This contradicts the assumption that the positive answers $\Phi_1, \dots, \Phi_m$ are all returned to the client. □

5 A Prototype for Oracle/SQL

We implemented[4] the signature-based enforcement mechanism as part of a larger project to realize a *general prototype* for inference-usability confinement of reactions generated by the server of a relational database management system (see Section 9 of [3]).

This prototype has been designed as a *frontend* to an Oracle/(SQL)-system: the *administration interface* enables officers to declare and manage client-specific data like the postulated a priori knowledge, a required confidentiality policy, a permitted interaction language, and the kind of distortion; the *interaction interface* enables registered clients to send requests like queries and receive reactions. The *interaction language* is uniformly based on the relational calculus as a specific version of first-order logic, which provides the foundation of the semantics of relational databases.

Accordingly, if a client should be permitted to issue queries under a schema $RS = \langle R, \mathcal{U}, \Sigma \rangle$ but be confined by the signature-based enforcement mechanism, then the following has to happen: the client is granted a permission to query the current instance r; the dependencies in Σ are added to that client's a priori knowledge; the language $\mathscr{L}$ is made available to the client to submit queries; a confidentiality policy is declared to confine the client's permission; and refusals are specified as the wanted kind of distortion. Additionally, a compatible enforcement mechanism is selected, either automatically by an optimizer or explicitly by an administrator. In the remainder of this section, we assume that inference signatures are both applicable and necessary as described in Section 1.

In a first attempt, considering the general prototype to mediate access to the underlying Oracle-system would suggest to let a wrapper translate a query $\Phi \in \mathscr{L}$ into an *SQL-query* during the *monitoring phase*. In our case, for instance, assuming that R denotes the Oracle-table for the instance r, a closed (yes/no)-query $(\exists X_1) \dots (\exists X_l) R(X_1, \dots, X_l, c_{l+1}, \dots, c_n)$ would be converted into

$$\textsc{Select}\ A_{l+1}, \dots, A_n\ \ \textsc{From}\ R\ \ \textsc{Where}\ A_{l+1} = c_{l+1}\ \ \textsc{And} \dots\ \textsc{And}\ A_n = c_n,$$

which returns either the empty set or a singleton with a tuple μ of the form $(A_{l+1} : c_{l+1}, \dots, A_n : c_n)$ over the attribute set $\{A_{l+1}, \dots, A_n\}$.

[4] The following exposition only outlines the implementation and slightly differs from the presently employed version of the code, which is under ongoing development for both improved usability and further optimization.

R	Sym	Dia	Pat
	Fever	*Cancer*	*Smith*
	Fever	*Fraction*	*Smith*
	Fever	*Cancer*	*Miller*
	Fever	*Fraction*	*Miller*

SEC	Sym	Dia	Pat
	$*$	*Cancer*	*Smith*

QUE	Sym	Dia	Pat	Rea
	Fever	*Cancer*	X	

SIG	Sym	Dia	Pat	Id	Imp	Old
	Fever	*Cancer*	b_1	1		
	Fever	b_2	*Smith*	1		
	Fever	*Cancer*	b_1	2		
	Fever	*Fraction*	b_5	2		
	b_2	*Fraction*	*Smith*	2		

Fig. 5. Oracle-tables for signature-based enforcement applied to hospital database

However, while forwarding this query to the Oracle-system, we want the server not only to evaluate the query but to perform the further actions on the inference signatures described in Section 4 as well. In principle, this goal can be accomplished by the features of Oracle for *active databases*, i.e., by *triggers*. Since Oracle does not provide means to define a trigger on a query directly, we instead employ a suitable *update command* to an auxiliary Oracle-table *QUE(RY)*, which together with two further tables (which will be described below) has already been created during the *compiling phase*.

- Basically, the table *QUE(RY)* has the attributes in $\mathcal{U}$ specified in the schema for the table R and a further attribute *Rea(ction)*, which has a three-valued type $\{ref(used), pos(itive), neg(ative)\}$: a tuple of *QUE(RY)* denotes a query as a generalized tuple combined with an indicator how to react.

Regarding Oracle-*privileges*, any access right the client might have before on the Oracle-table R must be revoked, and instead the client is only granted the Insert-right on the auxiliary table *QUE(RY)*. The needed trigger *CQE* is declared for insertions into the table *QUE(RY)*, and this trigger is executed with the access rights of the owner (administrator) of the table R. Accordingly, we employ some kind of "right amplification", as for example offered by the operating system UNIX by means of setting the suid-flag for an executable file: the client only receives a privilege to *initiate* the query-and-control activities, as predefined by the trigger, without being permitted to perform such activities at his own discretion.

The trigger *CQE* operates on the auxiliary Oracle-table *QUE(RY)* and the two further Oracle-tables *(POT)SEC* and *SIG(NATURE)* that already have been created separated from the Oracle-table R during the compiling phase.

- The table *(POT)SEC* has the same attribute set $\mathcal{U}$ as R such that a declared potential secret $\Psi \in \mathscr{L}$ can be represented as a generalized tuple; however, each (originally existentially quantified) variable is uniformly replaced by a special placeholder " * ".
- The table *SIG(NATURE)* has the attributes in $\mathcal{U}$ as well such that a hypothesis of an inference signature can again be represented as a generalized tuple, and three further attributes to be used as follows: the attribute *Id*

specifies an inference signature the represented hypothesis belongs to; the Boolean attribute *(Flag)Imp* refers to a tentative marking during a current monitoring phase; and the Boolean attribute *(Flag)Old* refers to a marking already committed while controlling a preceding query.

Unfortunately, it turned out that we need the two flags, since we could not employ standard transaction functionality to freshly mark hypotheses tentatively and finally either commit the fresh markings or abort them by a rollback instruction: Oracle does not offer to direct the needed transaction functionality within trigger executions.

Figure 5 shows the Oracle-tables for the small hospital database introduced as Example 2 in Section 3 with the instance inserted into table R, after filling the table *(POT)SEC* with the declared confidentiality policy, populating the table *SIG(NATURE)* with the compiled inference signatures, and forwarding the query $(\exists X)R(\mathit{Fever}, \mathit{Cancer}, X)$ to the table *QUE(RY)*.

Activated by the insertion of the query into the Oracle-table *QUE(RY)*, the trigger *CQE* basically proceeds as follows:

1. The trigger extracts the query submitted by the client from *QUE*, constructs an SQL-query to determine whether or not the extracted query covers an element in *SEC*, executes the constructed SQL-query, and then checks the result for emptiness: if the result is nonempty, i.e., a covering has been detected, the trigger modifies the attribute *Rea(ction)* of the single tuple in *QUE* into $ref(\mathit{used})$ (which the frontend retrieves subsequently), and the trigger exits. The following SQL-query is constructed for the example:

   Select *Sym*,*Dia*,*Pat* From *SEC* Where $(\mathit{Sym} = \mathit{Fever}$ Or $\mathit{Sym} = *)$
   And $(\mathit{Dia} = \mathit{Cancer}$ Or $\mathit{Sym} = *)$ And $\mathit{Pat} = *$

2. Otherwise, the trigger continues to perform the actions already described in Section 4 by suitably employing the Oracle-tables in a similar way as in the first step; see Figure 6 for the rough design.

6 Experimental Evaluation

To determine the runtime overhead inherently caused by the signature-based enforcement mechanism, we measured the query processing times of the implementation described in Section 5. We started with the following observation. Given the dependencies declared in the schema and the confidentiality policy, a generic inference signature signifies a typical "forbidden structure", and thus all of them together represent all possibilities for harmfulness. Thus, we aimed at constructing the instances to be used for runtime evaluations by varying the following parameters: (1) the included forbidden structures; (2) the instantiations of included forbidden structures; and (3) the fraction of confined "exceptions". Moreover, we expected an impact of the length of query sequences, since with increasing length more markings of hypotheses will be found.

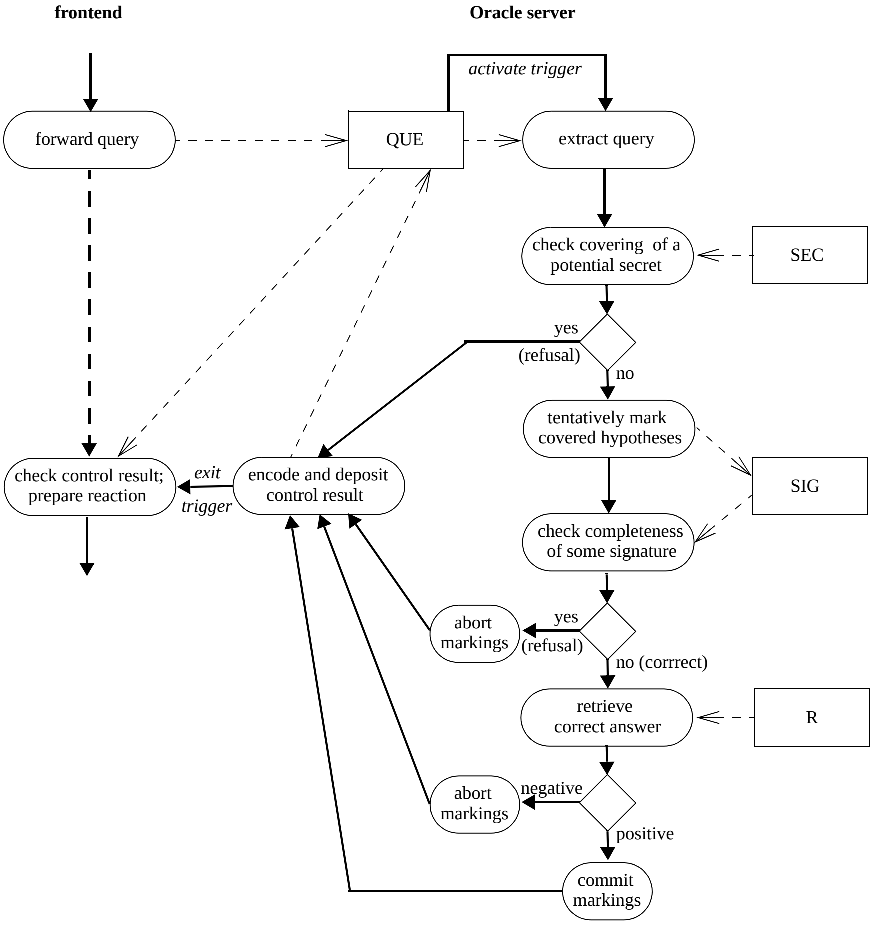

Fig. 6. Design of the trigger *CQE* to control a submitted query

Accordingly, we measured the following query processing times: the *maximum time* that occurred up to the last query of a sequence for the *whole frontend* and the *trigger alone*, respectively; and the *average time* over a sequence for the *whole frontend* and the *trigger alone*, respectively.

Discarding exceptional measurements caused by external factors and applying suitable roundings, we depict the results for Example 2 in Table 1: We restricted to the forbidden structures shown in Figure 2 and 4; varied the *number of instantiations* from 1 over 10 and 100 up to 1000, in this way getting instances (by applying the chase algorithm to satisfy the dependencies) of size from 79 tuples up to 79000 tuples; declared for each instantiated forbidden structure just one potential secret (as an “exception”); and formed query sequences of length from 100 up to 100000, suitably covering all relevant cases in a random way.

Table 1. Maximum and average query processing times experienced for Example 2

Instantiations	Instance size	Queries	Processing time pro query in msec			
			whole frontend		trigger alone	
			maximum	average	maximum	average
1	79	100	22	15	10	2
10	790	1000	54	17	10	3
100	7900	10000	421	23	150	7
1000	79000	100000	711	78	580	61

The results for the particular example suggest the practical feasibility of our approach, including scalability: a human user acting as a client will basically not realize a query processing time in the range of a few milliseconds up to around half a second. Of course, general practicality still has to be justified by statistically evaluating more advanced experiments for “real-world” applications using a more mature implementation with enhanced functionality and further optimizations.

7 Conclusions

Summarizing and concluding, we presented a signature-based enforcement mechanism that satisfies confidentiality requirements that are very general and have been considered in many contexts before, as summarized and further investigated by Halpern/O’Neill [13] and suitably extended to include policies by Biskup/Tadros [10]. The mechanism can be seen as a variation of a security automaton for the run time enforcement of security properties in the sense of Ligatti/Reddy [15] and others. We demonstrated the effectiveness of the mechanism for relational databases that are constrained by the large class of functional dependencies and join dependencies, which capture a wide range of applications, see, e.g., Abiteboul/Hull/Vianu [1].

Our mechanism differs from previously considered monitoring systems by taking advantage of the particular properties of functional dependencies and join dependencies, without imposing any further restrictions on these dependencies. We provide a proactive control functionality avoiding any confidentiality breach, in contrast to auditing approaches as described by, e.g., Kaushik/Ramamurthy [14], which can only detect violations after the fact.

There are several lines of further research and development, dealing with the following issues: tools for the compiling phase, the distribution of functionality between the two phases, the complete integration into a database management system like Oracle, more advanced interactions like open queries, updates and transactions, and experimental evaluations with “real-world” applications.

Regarding tools for the compiling phase, a major open problem is to design a generally applicable algorithm to effectively and efficiently determine all basic implications up to a suitably chosen number of hypotheses for any set of functional dependencies and join dependencies. We conjecture that properties regarding the occurrence of cyclic structures in the hypergraph of the dependencies has a major impact. The computational complexity of the compiling phase should also be investigated.

Regarding distribution of functionality, we already designed a more dynamic version of the signature-based enforcement mechanism. In this version, we initially keep the inference signatures generic, without instantiating them with specific values from the instance already at compile time. Rather, instantiations are dynamically generated at run time only employing instance tuples actually returned as responses to the client. This more dynamic version can be derived from the static version detailed in this article but some subtle optimization problems still have to be solved in a satisfactory way.

Regarding a complete integration, we first of all face problems of modifying proprietary software, but we would also be challenged to make the added functionality fully compatible with all the many services already offered. Of course, from the point of view of both administrators and clients, in general a full integration would be advantageous: conceptually for employing uniform interfaces, and algorithmically for avoiding the overhead raised by the communication of a separate frontend with a server and for including the security functionality into the scope of the server's optimizer.

Regarding advanced interactions, on the one hand we have to suitably adapt previous theoretical results [5,7] and, again, to exploit the features of the underlying database management system as far possible. On the other hand, in general an update of the database instance will require to update the (instantiated) inference signatures as well. Clearly, both aspects would have to be suitably combined, while also considering the optimization problems mentioned above.

Finally, regarding "real-world" applications, we would have to identify suitable classes of applications, clarify in detail how far the assumptions underlying the signature-based approach are actually satisfied by such applications, and then overcome essential mismatches by additional mechanisms. However, as pointed out in the introduction, there is an inevitable tradeoff between conceptual expressiveness and computational complexity: any extension of the work presented in this article will be challenged to maintain an appropriate balance between the conflicting goals.

Acknowledgments: We would like to sincerely thank Martin Bring and Jaouad Zarouali for improving the implementation and conducting the experiments.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, MA, 1995.
2. J. Biskup. History-dependent inference control of queries by dynamic policy adaption. In Y. Li, editor, *Data and Applications Security and Privacy, DBSec*, volume 6818 of *Lecture Notes in Computer Science*, pages 106–121. Springer, Berlin/Heidelberg, 2011.

3. J. Biskup. Inference-usability confinement by maintaining inference-proof views of an information system. *International Journal of Computational Science and Engineering*, 7(1):17–37, 2012.
4. J. Biskup and P. A. Bonatti. Lying versus refusal for known potential secrets. *Data Knowl. Eng.*, 38(2):199–222, 2001.
5. J. Biskup and P. A. Bonatti. Controlled query evaluation with open queries for a decidable relational submodel. *Ann. Math. Artif. Intell.*, 50(1-2):39–77, 2007.
6. J. Biskup, D. W. Embley, and J.-H. Lochner. Reducing inference control to access control for normalized database schemas. *Inf. Process. Lett.*, 106(1):8–12, 2008.
7. J. Biskup, C. Gogolin, J. Seiler, and T. Weibert. Inference-proof view update transactions with forwarded refreshments. *Journal of Computer Security*, 19:487–529, 2011.
8. J. Biskup, S. Hartmann, S. Link, and J.-H. Lochner. Chasing after secrets in relational databases. In A. H. F. Laender and L. V. S. Lakshmanan, editors, *Alberto Mendelzon International Workshop on Foundations of Data Management, AMW 2010*, volume 619 of *CEUR*, pages 13.1–12, 2010.
9. J. Biskup, J.-H. Lochner, and S. Sonntag. Optimization of the controlled evaluation of closed relational queries. In D. Gritzalis and J. Lopez, editors, *Advances in Information and Communication Technology – Emerging Challenges for Security, Privacy and Trust, IFIP 2009*, volume 297 of *IFIP Series*, pages 214–225. Springer, 2009.
10. J. Biskup and C. Tadros. Policy-based secrecy in the Runs & Systems Framework and controlled query evaluation. In I. Echizen, N. Kunihiro, and R. Sasaki, editors, *Advances in Information and Computer Security – International Workshop on Security, IWSEC 2010, Short Papers*, pages 60–77. Information Processing Society of Japan, 2010.
11. J. Biskup and L. Wiese. A sound and complete model-generation procedure for consistent and confidentiality-preserving databases. *Theoretical Computer Science*, 412:4044–4072, 2011.
12. R. Fagin, D. Maier, J. D. Ullman, and M. Yannakakis. Tools for template dependencies. *SIAM J. Comput.*, 12(1):36–59, 1983.
13. J. Y. Halpern and K. R. O'Neill. Secrecy in multiagent systems. *ACM Trans. Inf. Syst. Secur.*, 12(1):5.1–5.47, 2008.
14. R. Kaushik and R. Ramamurthy. Efficient auditing for complex SQL queries. In T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors, *ACM SIGMOD International Conference on Management of Data, SIGMOD 11*, pages 697–708. ACM, 2011.
15. K. Ligatti and S. Reddy. A theory of runtime enforcement, with results. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *European Symposium on Research in Computer Security, ESORICS 2010*, volume 6345 of *Lecture Notes in Computer Science*, pages 87–100. Springer, Berlin/Heidelberg, 2010.
16. F. Sadri and J. D. Ullman. Template dependencies: A large class of dependencies in relational databases and its complete axiomatization. *J. ACM*, 29(2):363–372, 1982.