History-Dependent Inference Control of Queries by Dynamic Policy Adaption*

Joachim Biskup

Technische Universität Dortmund, Dortmund, Germany
joachim.biskup@cs.tu-dortmund.de

Abstract. Policy-based inference control of queries submitted to a logic-oriented information system requires us to consider the history of queries and answers to a particular user. In most previous approaches, the control system captures the history by maintaining a fictitious view the user is supposed to generate by exploiting rational reasoning. In this paper, we propose and explore an alternative option to represent the history, namely by suitably adapting the confidentiality policy after returning an answer to a query. Basically, such a policy adaption precomputes all relevant steps of formal proofs that the fictitious view logically implies some policy element. We focus on propositional information systems.

Keywords: a priori knowledge, closed query, confidentiality policy, Controlled Query Evaluation, inference control, information system, interaction history, policy adaption, propositional logic, refusal, view.

1 Introduction

Inference control is a crucial though costly mechanism to protect *information* rather than just the underlying *data*, as achieved by traditional access control or simple encryption [4]. In general, dynamic *inference control* of *queries* submitted to an *information system* necessarily requires us to consider the *history* of queries and answers related to a particular user. In most of the previous work, including those on Controlled Query Evaluation (CQE) [5], the control employs the user's history in two ways: First, the control generates an (assumption about the) *view* that the user (supposingly) infers to represent his knowledge about the instance of the information system. This instance itself, however, remains hidden to the user, except that he has seen the *previous answers* and might have access to some *a priori knowledge*. Second, the control investigates whether that view combined with the correct answer to the next submitted query (or some closely related information) would be harmful w.r.t. a *confidentiality policy* specifically declared for the user.

In this context the intuitive meaning of *harmful* is the following: the user will be able to infer that some sentence contained in the policy actually holds in the

* This work has been performed within the framework of the Collaborative Research Center "Providing Information by Resource-Constrained Data Analysis", supported by the Deutsche Forschungsgemeinschaft under grant SFB 876/A5.

Y. Li (Ed.): Data and Applications Security and Privacy XXV, LNCS 6818, pp. 106–121, 2011.

instance. If this will be the case, the control reacts with a suitable *distortion* of the correct answer to avoid a security violation. In any case, after returning a reaction to the user, the control has to appropriately adjust the view generated for the user. Thus, over the time, the control enforces a suitable *invariant* to ensure that the view will be never harmful.

Notably, the view is *dynamically* updated after each reaction to a query, whereas the policy is kept unchanged once it has been *statically* declared by a security officer. We can rephrase this approach to dealing with the history as follows: at any point in time, the control has to confine the entailment relationship between the increasingly powerful (knowledgeable) view and the static policy.

We will illustrate this *view-based approach* to inference control by the following simple and straightforward example. Suppose that the policy requests to keep the propositional sentence $\varphi_1 \wedge \varphi_2$ secret. Furthermore, the user is assumed to have no a priori knowledge about the instance, for which both φ_1 and φ_2 are supposed to hold, and thus $\varphi_1 \wedge \varphi_2$ as well. Initially, the control generates an empty view. Then, as a first query, the user submits the sentence φ_1 in order to ask whether this sentence holds. The correct answer, φ_1, i.e., that this sentence holds, together with the empty view does not entail the single policy element, and thus the control returns the correct answer to the user in undistorted form and, accordingly, updates the view, which now comprises just the returned answer φ_1. Finally, as a second query, the user submits φ_2. Now, the correct answer, φ_2, together with the content of the updated view, φ_1, obviously entails the policy element, $\varphi_1 \wedge \varphi_2$, and thus the control must suitably distort the answer. Note the dynamic "last-minute behavior" of the control: if the queries were submitted in reverse order, first φ_2 and then φ_1, then φ_2 would have been correctly answered and the answer to φ_1 would have been distorted.

In this work, we will explore an alternative approach to employ the user's history. The alternative approach aims to represent the user's history by *dynamically adapting the policy*, thereby getting rid of the need to generate and maintain a view for the user. Intuitively, over the time, we will increasingly strengthen the policy, making it more and more restrictive as a countermeasure to the knowledge accumulated by previous answers.

To illustrate this alternative *policy-adaption based approach*, we reconsider the example presented above. Initially, the policy contains the sentence $\varphi_1 \wedge \varphi_2$. Since the first query, φ_1, is harmless, the correct answer is returned to the user. Now, once one of the conjuncts occurring in the original policy element is known to the user, he must not learn the other conjunct as well. Accordingly, the control replaces the previous policy element $\varphi_1 \wedge \varphi_2$ by φ_2 to be kept secret in future. If afterwards the second query, φ_2, is submitted, the control will immediately detect that the correct answer would violate the adapted policy and thus will distort the answer, as in the view-based approach.

We can also describe the policy-adaption based approach in terms of *theorem-proving*, as sketched in the following and elaborated in more detail in the remainder of this paper. In the starting step, for each sentence contained in the declared

policy and thus explicitly wanted to be kept secret to the user while returning answers to him, the user is supposed to aim at proving (the validity of) that sentence from the answers received. Acccordingly, for simplicity here assuming no a priori knowledge, the user initially considers every sentence contained in the declared policy to be a *current proof obligation*. Having received a new answer Φ_i in step i, the user can analyze all possible formal proofs for any of the current proof obligations whether and how Φ_i will be helpful to prove it. If the user detects such a situation, he can determine the resulting remaining proof obligations and, potentially, try to satisfy them by issuing further queries. Correspondingly, inference control can track the user's abilities, and thus control can dynamically adapt the policy by always setting it to the current set of proof obligations. In the example given above, the sole initial proof obligation is $\varphi_1 \wedge \varphi_2$, which can be replaced by the new proof obligation φ_2, once φ_1 is known.

Dynamic inference control is costly, at least in general, due to the inevitable need to suitably keeping track of the history and performing some kind of theorem-proving. The basic features of policy adaption suggest the possibility of substantial improvements in computational costs *at query time* in comparison with the view-based approach, at least in special situations: (1) the control no longer has to maintain a separate data structure for reflecting the user's view, and (2) analyzing and remembering remaining proof obligations can bee seen as a kind of stored *precomputation* for the task of checking whether subsequent queries are harmful or not. Moreover, we might be able to find appropriate *data structures* to actually benefit from the potentials.

In the following we roughly outline such an improvement for a restricted *propositional* situation, where *queries* are just propositional *atoms* of the form a_i and elements of the *confidentiality policy* are *conjunctions* of such atoms, thus of the form $a_{i_1} \wedge \ldots \wedge a_{i_k}$ with $1 \leq k$. Moreover, we will make policies *redundancy-free* in the sense that no policy element is a subconjunction of another policy element, just by discarding the larger one. As an example, let the policy be $\{a_1 \wedge a_2 \wedge a_3, a_3 \wedge a_4, a_4 \wedge a_5, a_6\}$, and consider the query sequence $\langle a_1, a_2 \rangle$.

The current policy will be represented by a data structure that is composed of two linked parts. The "look-up part" contains all *atoms* still occurring in the policy, and the "reduced part" comprises the *nontrivial conjunctions* (having at least 2 different atoms) still to be checked. Moreover, each atom in the former part is linked to each of the conjunctions in which it occurs in the latter part. Fig. 1 shows the initial state of the data structure for the example.

If an atom a_i is submitted as a query, the control first searches for that atom in the look-up part. If the atom is not found there, the query is censored to be *harmless* and correctly answered. Otherwise, there are two cases: If the atom is not linked to any nontrivial conjunction, then the atom is *harmful* by itself and the answer must be distorted. Otherwise, if there are links, the query is censored to be *harmless* and correctly answered, but the policy must be adapted by manipulating the current state of the data structure appropriately: (1) the query atom a_i is removed from the look-up part; (2) the query atom a_i is deleted from all the conjunctions in which it occurs; (3) if after the deletion a remaining

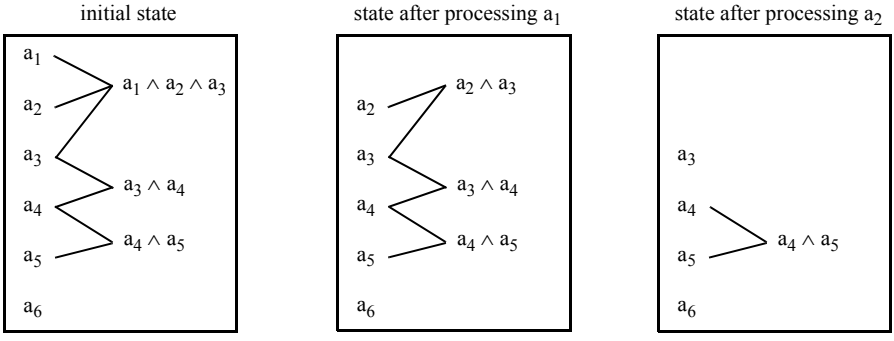

Fig. 1. Initial and subsequent states of a data structure for dynamic policy adaption

conjunction is reduced to a single atom a_j, then the conjunction is dropped at all and the corresponding link from a_j is deleted as well; moreover, all other conjunctions in which a_j occurs are deleted with all their links, too. Finally, if – by these deletions – another atom in the look-up part has lost all its links, then that atom is deleted from the look-up part.

Fig. 1 visualizes how the control operates for the parameters specified above. Querying the atom a_1 is harmless and leads to its removal from the look-up part by (1) and its deletion from the first conjunction, which is thus reduced to $a_2 \wedge a_3$ by (2). Then querying the atom a_2 is harmless again and leads to its removal from the look-up part by (1) and its deletion from the reduced conjunction, which thus becomes the single atom a_3 by (2); but this trivial conjunction is then totally dropped by (3), and the conjunction $a_3 \wedge a_4$ is deleted as well.

Since only some searching and elementary link manipulations are used, the *efficiency* of the procedure should be evident. A full justification of the *correctness* is elaborated in Sect. 3 for a more general situation. Roughly summarizing, in this article we will provide the following main contributions:

- We propose the policy-adaption based approach to keeping track of the history as a promising alternative to the view-based approach (this Sect. 1).
- After introducing our basic notations, briefly reviewing the view-based approach and commenting on complexity issues (Sect. 2), we fully elaborate the new approach for a special but reasonably expressive situation of Controlled Query Evaluation. This situation employs refusal as the sole distortion option and deals with a propositional information system (Sect. 3).
- We relate our approach to previous work, briefly discuss first-order information systems and evaluate the expected potentials and limitations (Sect. 4).

2 Basic Notations and View-Based Approach

Restricting to propositional information systems, we first introduce our basic notations. Then we briefly describe the view-based approach and state some observations on the complexity of deciding the pertinent logical implications.

2.1 Basic Notations

We employ a logic-oriented approach to information systems (see, e.g., [1]), which establishes formal semantics for both query answering and updating (not considered in this paper). For simplicity, we only consider *complete* information systems, and we focus on *propositional logic*. We assume a vocabulary of propositional *atoms*, from which we can construct propositional *sentences* in the standard way, using the propositional connectives of *negation* and *disjunction* and further derived connectives. A *literal* is either an atom or a negated atom.

The *schema* (of the information system) is given by the vocabulary and the *integrity constraints*, which are expressed as a finite set *con* of sentences over this vocabulary. We consider the integrity constraints as part of any user's a priori knowledge, which in each case is given as a set of sentences over the vocabulary.

An *instance db* (of the information system) is a set of literals formed as follows: For each atom α of the vocabulary, either the atom α itself or the negated atom $\neg\alpha$ is an element. Given the vocabulary, it suffices to explicitly specify only those atoms that are contained in an instance (implicitly assuming for the remaining atoms that their negations are elements by default, as a kind of closed world assumption). An instance *db* defines a *truth-value assignment* to propositional atoms by making each atom $\alpha \in db$ *true* and all the remaining atoms *false*. Such a truth-value assignment (interpretation) is inductively extended to arbitrary sentences Φ by giving the connectives the standard meaning; $eval(\Phi)(db)$ denotes the truth value assigned to Φ by *db*. The standard notion of logical *implication*, or *entailment*, between (sets of) sentences is designated by $\models$.

As a (closed, yes/no-)*query*, we allow any sentence Φ of the underlying propositional logic. The *correct answer* to the query Φ under an instance *db* is given by the pertinent truth value $eval(\Phi)(db)$; however, for convenience, we alternatively express the correct answer by $eval^*(\Phi)(db)$ that denotes either Φ or $\neg\Phi$ in a straightforward way. We aim at controlling any sequence of queries $Q := \langle\, \Phi_1, \Phi_2, \ldots, \Phi_i, \ldots, \Phi_k \,\rangle$ where the query Φ_i is submitted by some user at the point in time i; for simplicity of the presentation, we focus on only one user.

While the user is granted a general access right for reading (querying), a security officer declares a *confidentiality policy* as a finite set *psec* of propositional sentences, called *potential secrets*, in order to confine the actual information gain that can be achieved by the user. Here the qualification "potential" indicates that these sentences are not necessarily true in the actual instance. Following the principle of open design, the user is supposed to be *aware* of this declaration, as well as of all other features of the control mechanism. In order to prevent the user from ever inferring that any sentence $\Psi \in psec$ actually holds, we follow the *refusal* approach to inference control [11,6,5], i.e., if an informative answer to a query would be harmful, then the control reacts by returning a special symbol `mum`. In general, the refusal approach has to examine not only whether the correct answer to a query is harmful but also whether its negation would be harmful, in order to prevent so-called meta-inferences.

Besides the policy, in general the control mechanism also has to consider the (postulated) *a priori knowledge* of the user and the *answers* to previously issued

queries. To do so, the control might maintain a *user log*. Basically, such a user log then just contains a set *log* of propositional sentences. In principle, both the policy and the user log might be updated while processing queries; the current versions of them taken together form the current *state* $s_i := (psec_i, log_i)$ of the control mechanism. The *initial state* is obtained by setting $psec_0 := psec$ and $log_0 := prior$, where *prior* can be any suitable superset of the constraints *con* not being in conflict with *psec*, i.e., $prior \not\models \Psi$ for all $\Psi \in psec$. In this work, for simplicity, we will not elaborate the treatment of the a priori knowledge *prior* in depth: we just leave it empty in our examples, and we simply process it like a sequence of queries within our initialization subprotocol.

Definition 1 (controlled query evaluation). *Let be given an instance db, a finite set* log_{i-1} *of sentences (for explicitly reflecting the assumed user's current knowledge about the instance), and a finite set* $psec_{i-1}$ *of sentences (for representing the current version of the confidentiality policy). Then a function* $cqe(db, psec_{i-1}, log_{i-1}, \Phi_i)$ *defines a* controlled query evaluation *of a query* Φ_i *by generating a triple* $(ans_i, psec_i, log_i)$*, where* ans_i *is the answer returned to the user, and* $psec_i$ *and* log_i *together form the updated state.*

Furthermore, for the initializations specified above, this function is inductively extended to any query sequence $Q := \langle\, \Phi_1, \ldots, \Phi_i, \ldots, \Phi_k \,\rangle$ *by applying it stepwise in a straightforward way:*

$$\begin{aligned} &cqe(db, psec_0, log_0, Q) := \\ &\big\langle (ans_1, psec_1, log_1), \ldots, (ans_i, psec_i, log_i), \ldots, (ans_k, psec_k, , log_k) \big\rangle \end{aligned} \tag{1}$$

We are now ready to present our formal definition of the confidentiality requirement we want to achieve by a controlled query evaluation. Roughly summarized, given a potential secret Ψ declared in the (original) policy *psec*, this requirement is expressed in terms of the *indistinguishability* – from the point of view of the user – of the actual instance *db* from an alternative instance db^s that does not satisfy the potential secret considered.

Definition 2 (confidentiality). *A controlled query evaluation cqe* preserves confidentiality *iff*
for all instances db,
for all finite sets of sentences psec (original confidentiality policy),
for all finite sets of sentences prior (a priori knowledge)
satisfied by db and such that prior $\not\models \Psi$ *for all* $\Psi \in psec$*,*
for all query sequences Q, and
for all potential secrets $\Psi \in psec$
there exists an alternative instance db^s *satisfying prior such that:*

1. [indistinguishability]*:*

$$cqe(db, psec, prior, Q) = cqe(db^s, psec, prior, Q) \tag{2}$$

2. [possibility of false potential secrets]*:*

$$eval^*(\Psi)(db^s) = \neg\Psi \tag{3}$$

2.2 View-Based Approach

The view-based approach to Controlled Query Evaluation, as surveyed in [5], keeps track of the history by only updating the user log, while leaving the original policy unchanged. For the specific setting described above, i.e., refusal under known potential secrets for a propositional information system dealing with closed (yes/no-)queries, the function cqe^{view} is defined by its outputs as follows:

$$\begin{aligned} ans_i \quad &:= \texttt{if } log_{i-1} \models eval^*(\Phi_i)(db) \texttt{ then } eval^*(\Phi_i)(db) \texttt{ else} \\ &\quad \texttt{if } (\text{exists } \Psi)\,(\Psi \in psec \text{ and} \\ &\qquad\qquad (log_{i-1} \cup \{\Phi_i\} \models \Psi \text{ or } log_{i-1} \cup \{\neg\Phi_i\} \models \Psi)) \\ &\quad \texttt{then mum else } eval^*(\Phi_i)(db) \end{aligned} \tag{4}$$

$$psec_i \quad := psec \tag{5}$$

$$log_i \quad := \texttt{if } ans_i = \texttt{mum then } log_{i-1} \texttt{ else } log_{i-1} \cup \{ans_i\} \tag{6}$$

Proposition 1 ([6]). *The function cqe^{view} preserves confidentiality in the sense of Def. 2.*

Definition (4) of the controlled answer indicates that the task of inference control is closely related to the problem of deciding on logical implications of the form $\chi \models \Psi$, where the finite set of sentences χ – equivalently identified with the corresponding sentence formed as the conjunction over this set – denotes some potential knowledge of the user and Ψ is a policy element. This decision problem is well-known to be of high computational complexity in general, and thus we can expect to control answers efficiently only under some restrictions of the expressiveness of the languages for the sentences χ and Ψ, respectively.

As a starting point, we first observe the following: If both χ and Ψ are already specified in *disjunctive normal form* for a finite vocabulary, i.e., as a disjunction of so-called *minterms* that are built as a conjunction of literals (atoms or negated atoms) ranging over all atoms in the vocabulary, then $\chi \models \Psi$ holds if and only if each minterm of χ is also a minterm of Ψ.

For a slightly relaxed situation where both χ and Ψ are specified as a *disjunctive form*, i.e., a disjunction of conjunctions of literals ranging over different atoms in the vocabulary, the sufficiency part of this observation can be generalized along the following lines of reasoning, often referred to as subsumption. First, if some disjunctive sentences η_1 and η_2 are (syntactically) related such that each disjunct of η_1 is also a disjunct of η_2 – or at least (semantically) implies some disjunct of η_2 –, then the (semantic) implication $\eta_1 \models \eta_2$ holds, since η_2 is an obvious weakening of η_1. Dually, if some conjunctive sentences θ_1 and θ_2 are (syntactically) related such that each conjunct of θ_1 is also a conjunct of θ_2 – or is at least (semantically) implied by some conjunct of θ_2 –, then the (semantic) implication $\theta_2 \models \theta_1$ holds, since θ_2 is an obvious strengthening of θ_1.

Unfortunately, the necessity part of the observation stated above cannot be generalized for arbitrary disjunctive forms. However, the necessity part holds indeed, if the sentence Ψ consists of *all* the *prime implicants* of Ψ, i.e., (1) each disjunct of Ψ is *minimal* in the sense that discarding any of the literals in the

conjunction that constitutes this disjunct would result in a non-equivalent sentence, and (2) Ψ contains all minimal disjuncts (conjunctions of literals ranging over different atoms in the vocabulary) that imply Ψ.

Proposition 2. *Let χ be a disjunctive form and Ψ a disjunctive form that consists of all its prime implicants. Then $\chi \models \Psi$ holds if and only if for each disjunct of χ there is a disjunct of Ψ such that each literal occurring as a conjunct of the latter disjunct also appears as a conjunct of the former disjunct.*

3 Policy Adaption for Propositional Information Systems

We now present our new concept of the policy-adaption based approach in detail, exhibit an appropriate data structure for representing the current policy, and then demonstrate the correctness and comment on the efficiency.

3.1 Outline and Examples

To elaborate the policy-adaption based approach, we aim at defining the corresponding function cqe^{pol} for controlled query evaluation such that the following properties (further explained below) hold:

1. The parameter *log* could be dropped.
2. The history is reflected in the current version $psec_i$ of the policy.
3. The generated outputs ans_i are the same as for cqe^{view}.
4. The current version $psec_i$ is converted to be redundancy-free (see below).
5. The current version $psec_i$ is converted to be fully vulnerable (see below).

We first outline the basic techniques to achieve these properties, then exemplify these techniques, and finally present and verify a comprehensive algorithm for cqe^{pol} leading to a controlled query evaluation based on these techniques.

By property 3 and as a corollary to the result for cqe^{view} stated in Prop. 1, the function cqe^{pol} will preserve confidentiality in the sense of Def. 2 as well.

Regarding property 4, demanding the policy to be *redundancy-free*, we can observe the following by inspecting the guarding condition in the second line and the third line of (4): If a policy *psec* contains two different potential secrets Ψ_1 and Ψ_2 such that $\Psi_1 \models \Psi_2$, then we can remove Ψ_1 from the policy without affecting the answer. For, if a user knowledge $log \cup \{\Phi\}$ or $log \cup \{\neg\Phi\}$, respectively, implies Ψ_1, then that knowledge also implies Ψ_2; thus the outcome of the guarding condition remains the same after removing Ψ_1. Accordingly, we will keep the set *psec* redundancy-free in the sense that none of its elements implies another one.

Regarding property 5, demanding the policy to be *fully vulnerable*, we further observe the following: If a policy *psec* contains a potential secret Ψ such that $log \models \neg\Psi$ holds for the current user knowledge *log*, then we can remove Ψ from the policy. For, by monotonicity, this property will always be preserved later on and thus the confidentiality requirement expressed by Ψ will never be hurt.

Regarding the properties 2 and 3, which demand an *appropriate reflection of the history* in $psec_i$ such that the same outputs are generated as in the view-based approach, again by inspecting the guarding condition in the second line and the third line of (4), we have to inductively achieve an equivalence of the following kind (to be made more precise later on), where Δ_i denotes the query Φ_i or its negation $\neg\Phi_i$, respectively:

$$(\text{exists } \Psi)(\Psi \in psec \text{ and } log_{i-1} \cup \{\Delta_i\} \models \Psi) \text{ iff} \tag{7}$$

$$(\text{exists } \Psi)(\Psi \in psec_{i-1} \text{ and } \{\Delta_i\} \models \Psi) \tag{8}$$

To attain such a goal, we first impose all queries Φ_i and all policy elements Ψ to be given as a disjunctive form. Moreover, we additionally extend each policy element such that it contains all its prime implicants in order to profit from the efficiently verifiable characteristic property of $\chi \models \Psi$ given in Prop. 2.

Next, again for easily exploiting that property, in general we aim at representing a policy element of the form $\Psi = \Psi_1 \vee \ldots \vee \Psi_m$ that constitutes a nontrivial disjunction with $2 \leq m$ as the *set* of its disjuncts $\{\Psi_1, \ldots, \Psi_m\}$. To achieve a homogeneous treatment with a policy element of the form $\Psi = \Psi_1$ having only one disjunct, we then have to represent such an element as the singleton set $\{\Psi_1\}$. The set representations introduced will not affect the wanted equivalence, since they are functional equivalent with the original forms. If there are no semantic ambiguities, i.e., from a special context under consideration it is clear whether two disjuncts (implicants) belong to the same policy element or not, as in the examples below, we will omit the set notation for the sake of readibility.

Finally, to deal with disjunctive answers of the form $\Phi = \Phi_1 \vee \ldots \vee \Phi_n$ with $2 \leq n$, we will introduce *policy branches*: for each disjunct Φ_l, a copy of the current policy is generated and then inspected regarding implications that result from Φ_l alone. Subsequently, each of these branches has to be maintained with reference to the pertinent Φ_l until a definite answer that $\neg\Phi_l$ holds is given; then the branch is obviously contradictory and thus must be removed.

Example 1. Consider the following situation:
$db := \{a_1, \neg a_2, \neg a_3, a_4\}$ is the instance,
$Q := \langle\ a_1, a_2, a_3, a_4\ \rangle$ is the query sequence,
$psec := \{\neg a_1 \wedge \neg a_2 \wedge a_3 \wedge \neg a_4, a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge a_4\}$ is the policy, and
$log_0 := \emptyset$ is the void a priori knowledge.

Then $\langle\ a_1, \neg a_2, \neg a_3, a_4\ \rangle$ is the correct answer sequence, and the instance defines the first potential secret to be *false* and the second one to be *true*.

Controlling the first query a_1, we see that neither a_1 nor $\neg a_1$ implies any of the potential secrets, and thus the correct answer a_1 can be returned, and it would be inserted into the user log by the view-based approach such that we would have $log_1 := \{a_1\}$. Since the first potential secret is no longer vulnerable, we can remove it from the policy. Furthermore, once the user knows a_1, we now have to protect the remainder of the second potential secret, i.e., we can drop a_1 from $a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge a_4$. Thus we get
$psec_1 := \{\neg a_2 \wedge \neg a_3 \wedge a_4\}$.

Description and Branch	**Disjuncts (given prime implicants)**	**Disjuncts (additional prime implicants)**
sole element (original)	$a_1 \wedge \neg a_3 \wedge \neg a_4$ $a_1 \wedge \neg a_2 \wedge a_4$ $a_2 \wedge a_3 \wedge a_4$ $\neg a_1 \wedge a_2 \wedge \neg a_3$	$a_2 \wedge \neg a_3 \wedge \neg a_4$ $a_1 \wedge \neg a_2 \wedge \neg a_3$ $a_1 \wedge a_3 \wedge a_4$ $\neg a_1 \wedge a_2 \wedge a_4$
sole element (after answer a_1)	$\neg a_3 \wedge \neg a_4$ $\neg a_2 \wedge a_4$ $a_2 \wedge a_3 \wedge a_4$ (complementary)	$a_2 \wedge \neg a_3 \wedge \neg a_4$ $\neg a_2 \wedge \neg a_3$ $a_3 \wedge a_4$ (complementary)
sole element (after answer a_1 and subsumption)	$\neg a_3 \wedge \neg a_4$ $\neg a_2 \wedge a_4$ (subsumed) (complementary)	(subsumed) $\neg a_2 \wedge \neg a_3$ $a_3 \wedge a_4$ (complementary)
sole element (after answers a_1, $\neg a_2 \vee a_3$) for branch $\{\neg a_2\}$	$\neg a_3 \wedge \neg a_4$ a_4 (subsumed) (complementary)	(subsumed) $\neg a_3$ $a_3 \wedge a_4$ (complementary)
sole element (after answers a_1, $\neg a_2 \vee a_3$) for branch $\{a_3\}$	(complementary) $\neg a_2 \wedge a_4$ (subsumed) (complementary)	(subsumed) (complementary) a_4 (complementary)
sole element (after answers a_1, $\neg a_2 \vee a_3$ and subsumption) for branch $\{\neg a_2\}$	(subsumed) a_4 (subsumed) (complementary)	(subsumed) $\neg a_3$ (subsumed) (complementary)
sole element (after answers a_1, $\neg a_2 \vee a_3$ and subsumption) for branch $\{a_3\}$	(complementary) (subsumed) (subsumed) (complementary)	(subsumed) (complementary) a_4 (complementary)

Fig. 2. A converted and then stepwise adapted confidentiality policy

Similarly, stepwise controlling the second query a_2 and the third query a_3, we (would) get the following:

$log_2 := \{a_1, \neg a_2\}$, $psec_2 := \{\neg a_3 \wedge a_4\}$,
$log_3 := \{a_1, \neg a_2, \neg a_3\}$, $psec_3 := \{a_4\}$.

Finally, controlling the fourth query a_4, we immediately see that the correct answer violates the policy, and thus the answer must be refused. Notably, if the correct answer was $\neg a_4$, then that answer would have to be refused as well.

Example 2. Consider the following situation, the processing of which is further illustrated in Fig. 2:

$db := \{a_1, \neg a_2, \neg a_3, a_4\}$ is the instance, the same as before,
$Q := \langle\ a_1, \neg a_2 \vee a_3, a_3, a_4\ \rangle$ is the query sequence,
$psec := \{\ a_1 \wedge \neg a_3 \wedge \neg a_4 \quad \vee \quad a_1 \wedge \neg a_2 \wedge a_4 \quad \vee \quad a_1 \wedge a_2 \wedge a_3 \wedge a_4$
$\vee \quad \neg a_1 \wedge a_2 \wedge a_3 \wedge a_4 \quad \vee \quad \neg a_1 \wedge a_2 \wedge \neg a_3\ \}$ is the policy, and
$log_0 := \emptyset$ is the void a priori knowledge.

Obviously, then $\langle\ a_1, \neg a_2 \vee a_3, \neg a_3, a_4\ \rangle$ is the correct answer sequence, and the instance defines the sole potential secret to be *true*.

At initialization time, we observe that the sole policy element can be equivalently simplified by combining the two disjuncts $a_1 \wedge a_2 \wedge a_3 \wedge a_4$ and $\neg a_1 \wedge a_2 \wedge a_3 \wedge a_4$ into the prime implicant $a_2 \wedge a_3 \wedge a_4$. Furthermore, even afterwards the policy element does not contain all its prime implicants; in fact, we have to add four further prime implicants, namely

$a_2 \wedge \neg a_3 \wedge \neg a_4$, $a_1 \wedge \neg a_2 \wedge \neg a_3$, $a_1 \wedge a_3 \wedge a_4$, and $\neg a_1 \wedge a_2 \wedge a_4$.

Subsequently, we replace the sole policy element by the set of its disjuncts (prime implicants) and get the following representation of the policy:

$$psec_0 := \{a_1 \wedge \neg a_3 \wedge \neg a_4,\ a_1 \wedge \neg a_2 \wedge a_4,\ a_2 \wedge a_3 \wedge a_4,\ \neg a_1 \wedge a_2 \wedge \neg a_3,\ a_2 \wedge \neg a_3 \wedge \neg a_4,\ a_1 \wedge \neg a_2 \wedge \neg a_3,\ a_1 \wedge a_3 \wedge a_4,\ \neg a_1 \wedge a_2 \wedge a_4\}.$$

Controlling the first query a_1, we see that neither a_1 nor $\neg a_1$ implies any of the potential secrets, and thus the correct answer a_1 can be returned. Since the policy elements containing the complementary literal $\neg a_1$ are no longer vulnerable, we can remove them from the policy. Furthermore, once the user knows a_1, we can drop a_1 from the remaining elements. Additionally, we can remove elements that have become redundant, which is equivalent to being subsumed by a shorter disjunct. Altogether we get

$psec_1 := \{\neg a_3 \wedge \neg a_4,\ \neg a_2 \wedge a_4,\ \neg a_2 \wedge \neg a_3,\ a_3 \wedge a_4\}$.

Controlling the second query $\neg a_2 \vee a_3$, we see again that neither the positive answer $\neg a_2 \vee a_3$ nor the negative answer $a_2 \wedge \neg a_3$ implies any of the potential secrets, and thus the correct answer can be returned. However, since the correct answer is a disjunction, we split the policy into branches, one for the case that $\neg a_2$ is actually *true* and another one for the case that a_3 is actually *true*.

In the branch for $\neg a_2$, we can drop the occurrences of $\neg a_2$ from two of the elements, yielding the reduced elements a_4 and $\neg a_3$. As there are no occurrences of the complementary literal a_2, all elements are still vulnerable. Additionally, however, we can remove the then subsumed elements $\neg a_3 \wedge \neg a_4$ and $a_3 \wedge a_4$. Thus we get

$psec_2[\neg a_2] := \{a_4, \neg a_3\}$.

In the branch for a_3, we can drop the occurrence of a_3 from one of the elements, yielding the reduced element a_4, and we can remove the elements $\neg a_3 \wedge \neg a_4$ and $\neg a_2 \wedge \neg a_3$, in which the complementary literal $\neg a_3$ occurs. Additionally, the element $\neg a_2 \wedge a_4$ is now subsumed and thus can be removed. Thus we get

$psec_2[a_3] := \{a_4\}$.

Controlling the third query a_3, we see that the correct answer $\neg a_3$ makes the policy branch for a_3 contradictory and implies an element in the remaining branch for $\neg a_2$. Accordingly, the answer must be refused and both policy branches remain unchanged. Finally, controlling the fourth query a_4, we see that the correct answer a_4 implies a policy element in both branches, and thus the answer must be refused as well.

3.2 Protocol for Policy Adaption and Correctness

Having introduced the basic techniques, we are now ready to specify the types, inputs and methods of our new approach of policy adaption more formally.

Protocol for Policy Adaption.
types.
$\mathscr{L}$ propositional sentences;
$\mathscr{L}_{\mathrm{df}} \subseteq \mathscr{L}$ propositional sentences in disjunctive form;
$\mathscr{L}_{\mathrm{pi}} \subseteq \mathscr{L}_{\mathrm{df}}$ propositional sentences that consist of all their prime implicants;
$\mathscr{L}_{\mathrm{li}} \subseteq \mathscr{L}_{\mathrm{pi}}$ literals;
$\mathscr{L}_{\mathrm{im}} \subseteq \mathscr{L}_{\mathrm{df}}$ implicants (conjunctions of literals over distinct atoms);
$\mathscr{C} \subseteq_{finite} \wp\mathscr{L}$ declared confidentiality policies;
$\mathscr{M} \subseteq_{finite} \wp\wp\mathscr{L}_{\mathrm{im}}$ converted confidentiality policies
as multisets of "identified policy elements";
$\mathscr{B} \subseteq_{finite} \mathscr{M} \times \wp\mathscr{L}_{\mathrm{li}}$ policy branches; //written as $imsets[liset]$;
$\mathscr{Q} \subseteq \mathscr{L}$ queries.

subprotocol: initialization.
input: $psec : \mathscr{C}$;
$prior : \wp\mathscr{L}$;
method:
1. $sec_\emptyset := psec$;
2. modify $sec_\emptyset$ as follows:
   `foreach` $\Psi \in sec_\emptyset$ `do`
   convert Ψ such that it becomes the disjunction of all its prime implicants;
   `foreach` $\Psi \in sec_\emptyset$ `do`
   replace Ψ having form $\Psi_1 \vee \ldots \vee \Psi_m$ by the representing set $\{\Psi_1, \ldots, \Psi_m\}_\Psi$;
3. $psecb_0 := \{sec_\emptyset[\emptyset]\}$; // only one policy branch of form $\{\{\ldots\}, \ldots, \{\ldots\}\}[\emptyset]$
4. process $prior$ like a sequence of queries. //not elaborated for lack of space

subprotocol: generation (of answer and policy).
input: $\Phi_i : \mathscr{L}$;
$psecb_{i-1} : \mathscr{B}$;
method:
1. convert Φ_i into disjunctive form $\Phi_{i,1} \vee \ldots \vee \Phi_{i,n}$;
2. $ans_i :=$ `if` Φ_i violates $psecb_{i-1}$ `or` $\neg\Phi_i$ violates $psecb_{i-1}$
   `then mum`
   `else` $eval^*(\Phi_i)(db)$;
3. `if` $ans_i = \Phi_i$ (let $\Phi_i = \Phi_{i,1} \vee \ldots \vee \Phi_{i,n}$)
   `then` $psecb_i := \emptyset$;
   `foreach` disjunct $\Phi_{i,j}$ of Φ_i `do`
   $lit_j := \{\varphi \mid \varphi$ occurs in $\Phi_{i,j}\}$;
   $copy_j := \{sec[D \cup lit_j] \mid sec[D] \in psecb_{i-1}\}$;
   `foreach` literal φ of $\Phi_{i,j}$ `do` perform policy adaption for φ and $copy_j$;
   $psecb_i := psecb_i \cup copy_j$

   `elseif` $ans_i = \neg\Phi_i$ (let $\neg\Phi_i = \neg\Phi_{i,1} \wedge \ldots \wedge \neg\Phi_{i,n}$)
   `then` $psecb_i := psecb_{i-1}$;
   `foreach` conjunct $\neg\Phi_{i,j}$ of $\neg\Phi_i$ (let $\neg\Phi_{i,j} = \varphi_1 \vee \ldots \vee \varphi_k$) `do`
   $copy := \emptyset$;
   `foreach` literal φ_l of $\neg\Phi_{i,j}$ `do`
   $copy_l := \{sec[D \cup \{\varphi_l\}] \mid sec[D] \in psecb_i\}$;
   perform policy adaption for φ_l and $copy_l$;
   $copy := copy \cup copy_l$;
   $psecb_i := copy$.

subprotocol: violation (test).
input: $\Phi : \mathscr{L}$;
$psecb : \mathscr{B}$;
method:
convert Φ into disjunctive form;
//nothing to do if $\Phi = \Phi_i$, i.e., violation test is performed for current query
`if` there exists a branch $sec[D]$ of policy $psecb$ and
there exists a disjunct Φ_j of (negated) query Φ
such that $\Phi_j \wedge \bigwedge_{\varphi \in D} \varphi$ is not contradictory
//guaranteed if $\Phi = eval^*(\Phi_i)(db)$
`and`
there exists $\{\ldots\}_{\tilde{\Psi}} \in sec_\emptyset$ such that // $\tilde{\Psi}$ "uniformly identifies" a policy element
for all branches $sec[D]$ of policy $psecb$ and
for all disjuncts Φ_j of (negated) query Φ
such that $\Phi_j \wedge \bigwedge_{\varphi \in D} \varphi$ is not contradictory
there exists a disjunct $\tilde{\Psi}_r \in \{\ldots\}_{\tilde{\Psi}} \in sec$ such that $\Phi_j \models \tilde{\Psi}_r$ (by subsumption)
`then` return *true* (violation)
`else` return *false* (no violation).

subprotocol: adaption (for literal and policy copy).
input: $\varphi : \mathscr{L}_{\mathrm{li}}$;
`var` $copy : \mathscr{B}$; // $copy$ is used as input-and-output parameter
method: // modify $copy$ as follows
`foreach` policy branch $sec_j[D_j] \in copy$ `do`
1. `if` $\neg\varphi \in D_j$
`then` delete branch $sec_j[D_j]$
`else foreach` $\{\chi_1, \ldots, \chi_r\}_\Psi \in sec_j$ `do`
`foreach` $\chi \in \{\chi_1, \ldots, \chi_r\}_\Psi$ `do`
`if` φ occurs in χ `then` drop φ from χ;
`if` $\neg\varphi$ occurs in χ `then` remove χ from $\{\chi_1, \ldots, \chi_r\}_\Psi$;
`foreach` distinct $\chi_1, \chi_2 \in \{\chi_1, \ldots, \chi_r\}_\Psi$ `do`
`if` $\chi_1 \models \chi_2$ (by subsumption) `then` remove χ_1 from $\{\chi_1, \ldots, \chi_r\}_\Psi$;
2. `foreach` $\{\chi_1, \ldots, \chi_r\}_\Psi, \{\bar{\chi}_1, \ldots, \bar{\chi}_{\bar{r}}\}_{\bar{\Psi}} \in sec_j$ with $\Psi \neq \bar{\Psi}$ `do`
`if` $\chi_1 \vee \ldots \vee \chi_r \models \bar{\chi}_1 \vee \ldots \vee \bar{\chi}_{\bar{r}}$
`then` replace $\{\chi_1, \ldots, \chi_r\}_\Psi$ by $\emptyset_\Psi$ // consider $\emptyset_\Psi$ as removed.

As explained in Sect. 3.1, the protocol for policy adaption has been designed to achieve the same effects as the view-based approach. Thus the protocol is claimed to be *correct* with respect to the view-based approach and, accordingly by Prop. 1, to *preserve confidentiality*. The latter claim is stated in the following theorem, the proof of which justifies the former claim.

Theorem 1. *The function cqe^{pol} as defined by the Protocol for Policy Adaption preserves confidentiality in the sense of Def. 2.*

Proof. For lack of space, we only outline the inductive proof, which follows the informal arguments presented in Sect. 3.1. Basically, the induction will deal with the following items and notations:

- $hist_{i-1} := \bigvee_k \bigwedge_l \beta_{k,l}$ equivalently represents the user log log_{i-1} under the view-based approach as a single sentence converted into disjunctive form.
- $\Delta_i := \bigvee_{k''} \chi_{k''}$ in disjunctive form denotes the query Φ_i or its negation $\neg\Phi_i$.
- $tent_i := hist_{i-1} \wedge \Delta_i = \bigvee_{k,k''}(\bigwedge_l \beta_{k,l} \wedge \chi_{k''})$ then represents a left-hand side in a violation test according to (4), but so far ignoring that contradictory disjuncts might occur.
- $tent_i^{red} := \bigvee_{\bar{k},\bar{k}''}(\bigwedge_l \beta_{\bar{k},l} \wedge \chi_{\bar{k}''})$ in disjunctive form results from $tent_i$ by discarding all contradictory disjuncts (containing both an atom α and the negated literal $\neg\alpha$). The special case that $tent_i^{red}$ becomes the empty disjunction only happens if $log_{i-1} \models eval^*(\Phi_i)(db)$ and $\Delta_i = \neg eval^*(\Phi_i)(db)$.
- $\mathscr{D}_{i-1}$ is the set of tags D occurring in the current policy $psecb_{i-1}$.
- $psecb_{i-1} := \{sec_D[D] \mid D \in \mathscr{D}_{i-1}\}$ then describes the elements of that policy.

One can verify that the generation subprotocol establishes a one-to-one correspondance between the set of non-contradictory disjuncts $\bigwedge_l \beta_{k,l}$ of $hist_{i-1}$, ranging over all pertinent k, and $\mathscr{D}_{i-1}$, such that for each k the corresponding tag D satisfies $D = \{\beta \mid \beta = \beta_{k,l} \text{ for some } l\}$. Note that if the generation subprotocol tentatively forms a branch corresponding to a contradictory disjunct, then this fact is detected by performing the adaption subprotocol, which leads to an immediate deletion of that branch.

Then we assert and comment the equivalence of the following assertions:

1. (exists Ψ)($\Psi \in psec$ and $log_{i-1} \cup \{\Delta_i\} \models \Psi$).
   Such a kind of assertion is checked by the view-based approach according to (4), to be shown to satisfy the equivalence given by "(7) iff (8)".
2. (exists Ψ)($\Psi \in psec$ and $tent_i^{red} \models \Psi$).
   The set on the left-hand side of $\models$ is represented as a single sentence, which is formed as the conjunction over all elements of that set and then converted into disjunctive form (with discarding of contradictory disjuncts).
3. (exists $\tilde{\Psi}_s$)($\tilde{\Psi}_s = \{\tilde{\Psi}_{s,1}, \ldots, \tilde{\Psi}_{s,m}\} \in sec_\emptyset$ and $\bigvee_{\bar{k},\bar{k}''}(\bigwedge_l \beta_{\bar{k},l} \wedge \chi_{\bar{k}''}) \models \bigvee_r \tilde{\Psi}_{s,r}$).
   Here $\{\tilde{\Psi}_{s,1}, \ldots, \tilde{\Psi}_{s,m}\}$ are the initially determined prime implicants of $\tilde{\Psi}_s$.
4. (exists $\tilde{\Psi}_s$)($\tilde{\Psi}_s = \{\tilde{\Psi}_{s,1}, \ldots, \tilde{\Psi}_{s,m}\} \in sec_\emptyset$ and (for all $\bar{k}''$)(for all $\bar{k}$)
   (exists $\tilde{\Psi}_{s,r}$)($\tilde{\Psi}_{s,r} \in \tilde{\Psi}_s$ and $\bigwedge_l \beta_{\bar{k},l} \wedge \chi_{\bar{k}''} \models \tilde{\Psi}_{s,r}$)).
   We have exploited Prop. 2 for treating the implication problems.
5. (exists $\tilde{\Psi}_s$)($\tilde{\Psi}_s = \{\tilde{\Psi}_{s,1}, \ldots, \tilde{\Psi}_{s,m}\} \in sec_\emptyset$ and
   (for all $\bar{k}''$)(for all "non-contradictory"$\bar{D} \in \mathscr{D}_{i-1}$)
   (exists $\tilde{\Psi}_{s,r}$)($\tilde{\Psi}_{s,r} \in \tilde{\Psi}_s$ and $\bigwedge_l \beta_{k(\bar{D}),l} \wedge \chi_{\bar{k}''} \models \tilde{\Psi}_{s,r}$)).
   We have employed the correspondance between disjuncts of $hist_{i-1}$ and branches, where $k(\bar{D})$ corresponds to $\bar{D}$.
6. (exists $\tilde{\Psi}_s$)($\tilde{\Psi}_s \in sec_\emptyset$ and (for all $\bar{k}''$)(for all "non-contradictory"$\bar{D} \in \mathscr{D}_{i-1}$)
   (exists $\tilde{\Psi}_{s,r}^{\bar{D}}$)($\tilde{\Psi}_{s,r} \in \tilde{\Psi}_s^{\bar{D}}$ and $\chi_{\bar{k}''} \models \tilde{\Psi}_{s,r}^{\bar{D}}$)).
   Here $\tilde{\Psi}_s^{\bar{D}}$ is the version of $\tilde{\Psi}_s$ in the branch $sec[\bar{D}]$. The simplifications of the adaption subprotocol preserve the applicability of the efficient implication check, as stated in Prop. 2. Bascially, this kind of assertion is checked by the violation subprotocol of the policy-adaption approach. □

3.3 Efficiency of Policy Adaption

Without restrictions the worst-case complexity of policy adaption is inevitably determined by the complexity of the decision problems for propositional logic and thus expected to be exponential. Exponential efforts might also be hidden in transforming sentences into disjunctive forms or even determining all prime implicants. However, queries or negated queries that consist of strict disjunctions or generate strict disjunctions, respectively, are the *sole cause of branching* and thus of an exponential explosion of the size of an adapted policy. Besides these general remarks, analytical complexity results on "average"-case complexity appear to be hardly obtainable and are beyond the scope of this paper. It is left open to future work to implement a prototype and to set up practical experiments. If we then aim at empirically comparing policy adaption and view generation for special cases, we will be challenged to identify the best available optimization techniques for each of the two approaches.

If we restrict on queries that are single literals and then inspect such a literal, we have to determine whether and how the atom involved occurs in one of the implicants in the current policy data. To generalize the data structure exemplified in Fig. 1, we could maintain an efficiently searchable structure of all relevant atoms, together with the set structure comprising all current implicants (then including single literals), linking an atom with all pertinent implicants.

4 Related Work, Extensions and Conclusions

Though the policy-adaption based approach is innovative for inference control by means of Controlled Query Evaluation, some of the underlying ideas are already implicitly present in various previous work. First of all, we observe that a mechanism for enforcing inference control can be seen as an *automaton* that is basically specified by its set of *internal states*, its *state transition function* and its output or *reaction function*. In principle, for Controlled Query Evaluation a state has to reflect both a user's history and the confidentiality policy suitably. Accordingly, in a straightforward approach, a state can just be formed by a combination of two components: a current log of the user's history and a current version of the policy. In fact, the view-based approach explicitly maintains these two components. In contrast, the policy-adaption based approach aims at representing both of the needed features within one component.

All work on state-dependent control is somehow related to our contribution, as can be seen from the following examples. The works on "enforceable security properties" [10,9] treat states as abstract objects, without indicating implementations. Advanced discretionary access control based on logic programming, like the Flexible Authorization Framework [8] maintains a special "done-predicate", which can be seen as a kind of a user log or as a kind of a dynamic component of the access control policy, depending on the point of view. The Dynamic Authorization Framework [3] additionally selects a current model as a dynamic policy component to determine the current semantics. Dynamic mandatory access control [2] offers to adapt security labels assigned to objects as a classification like

"high-water marks", where classifications can be seen as a part of the access control policy. Many further examples stem from the dynamic control of workflows. Control of probabilistic inferences [7] uses a Bayesian network, which is updated after returning some piece of information to a user; the current network reflects the confidentiality requirements still to be enforced.

We demonstrated in detail that the proposed policy-adaption approach can be employed *effectively* for a specific situation of Controlled Query Evaluation, and we also indicated how to implement this approach such that inference control can be performed *efficiently* for special cases. It would be worthwhile to also consider more expressive situations, including incomplete instances and open queries. Such extensions will challenge us to transfer the current considerations to the more complex modal first-order logic. Seen from a even more general perspective, the ultimate goal of further efforts should be the following: We should aim at finding suitable combinations of the view-based approach and the policy-adaption based appraoch, in order to achieve the best possible efficiency for specific situations; and maybe we could further aim at constructing an optimizer that automatically recognizes the best combination for a current situation.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Bell, D.E., LaPadula, L.J.: Secure computer systems: A mathematical model, volume II. Journal of Computer Security 4(2/3), 229–263 (1996); reprint of MITRE Corporation (1974)
3. Bertino, E., Buccafurri, F., Ferrari, E., Rullo, P.: A logic-based approach for enforcing access control. Journal of Computer Security 8(2/3) (2000)
4. Biskup, J.: Security in Computing Systems – Challenges, Approaches and Solutions. Springer, Heidelberg (2009)
5. Biskup, J.: Usability confinement of server reactions: Maintaining inference-proof client views by controlled interaction execution. In: Kikuchi, S., Sachdeva, S., Bhalla, S. (eds.) DNIS 2010. LNCS, vol. 5999, pp. 80–106. Springer, Heidelberg (2010)
6. Biskup, J., Bonatti, P.A.: Controlled query evaluation for enforcing confidentiality in complete information systems. Int. J. Inf. Sec. 3(1), 14–27 (2004)
7. Chen, Y., Chu, W.W.: Protection of database security via collaborative inference detection. IEEE Trans. Knowl. Data Eng. 20(8), 1013–1027 (2008)
8. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible support for multiple access control policies. ACM Trans. Database Syst. 26(2), 214–260 (2001)
9. Ligatti, J., Reddy, S.: A theory of runtime enforcement, with results. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 87–100. Springer, Heidelberg (2010)
10. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (2000)
11. Sicherman, G.L., de Jonge, W., van de Riet, R.P.: Answering queries without revealing secrets. ACM Trans. Database Syst. 8(1), 41–59 (1983)