Inference-Proof Updating of a Weakened View Under the Modification of Input Parameters

Joachim Biskup(✉) and Marcel Preuß(✉)

Technische Universität Dortmund, Dortmund, Germany
{joachim.biskup,marcel.preuss}@cs.tu-dortmund.de

Abstract. We treat a challenging problem of confidentiality-preserving data publishing: how to repeatedly update a released weakened view under a modification of the input parameter values, while continuously enforcing the confidentiality policy, i.e., without revealing a prohibited piece of information, neither for the updated view nor retrospectively for the previous versions of the view. In our semantically ambitious approach, a weakened view is determined by a two-stage procedure that takes three input parameters: (i) a confidentiality policy consisting of prohibitions in the form of pieces of information that the pertinent receiver of the view should not be able to learn, (ii) the assumed background knowledge of that receiver, and (iii) the actually stored relation instance, or the respective modification requests. Assuming that the receiver is aware of the specification of both the underlying view generation procedure and the proposed updating procedure and additionally of the declared confidentiality policy, the main challenge has been to block all meta-inferences that the receiver could make by relating subsequent views.

Keywords: Background knowledge · Inference-proofness · History-awareness · Meta-inference · Policy of prohibitions · Relational database · Semantic confidentiality · Update · View generation · Weakened information

1 Introduction

Within a framework of cooperating with partners and sharing resources with them, managing the fundamental asset of own information – whether personal or institutional – has evolved as a main challenge of IT-security, leading to diverse computational techniques to enforce all kinds of an owner's interests. This includes confidentiality-preserving data publishing [8] aiming at hiding specific pieces of information while still providing sufficient availability. One class of techniques for confidentiality-preserving data publishing distorts data by weakening the still true information content of released data, e.g., by explicitly erasing sensitive data or by substituting sensitive data items by suitably generalized ones, as for instance applied for k-anonymization with l-diversification [12,15,17].

Whereas the effectiveness of many such techniques relies on the appropriateness of more or less intuitive concepts, like, e.g., quasi-identifiers, our own approach has more ambitiously been based on a fully formalized notion of semantic confidentiality in terms of inference-proofness. This notion considers an authorized receiver that profits from some background knowledge and unlimited computational resources for rational reasoning. More specifically, in previous work [4] we conceptually designed a two-stage *view generation* procedure that weakens the information content of an actually stored relation instance, and we verified the requested confidentiality property and experimentally evaluated the runtime efficiency. This procedure takes three input parameters, (i) a *confidentiality policy* consisting of prohibitions in the form of pieces of information that the pertinent receiver of the view should not be able to learn, (ii) the assumed *background knowledge* of that receiver in the form of single-premise tuple-generating data dependencies, and (iii) the actually stored *relation instance*.

Example 1 (weakened view). Let R be a relation symbol (table name) with three attributes (columns) with (conceptually) countably infinite domains, having the current relation instance $r = \{(a, b, c), (a, c, c), (b, a, c)\}$ under closed-world assumption. Expressed in terms of first-order logic as a basis for formal semantics of relational databases [1], this means that the three ground atoms $R(a, b, c)$, $R(a, c, c)$ and $R(b, a, c)$ are evaluated to *true*, whereas all other syntactically possible ground atoms are considered to be evaluated to *false*. For the moment still neglecting background knowledge, let us further suppose that the data owner wants to prohibit that the anticipated receiver of the view to be generated could ever learn that $R(a, b, c)$ is *true*, and so for $R(a, b, d)$. Obviously, that view should not reveal that the tuple (a, b, c) is an element of the relation instance r.

The view generation procedure of [4] achieves this goal as follows. In the first stage, only treating the considered prohibitions, the procedure forms a disjunctive template $R(a, b, c) \lor R(a, b, d)$ (notably, the truth evaluation of which is *not* prohibited to be known). In the second stage, the procedure checks each ground atom that is *true* in the instance whether it entails a disjunctive template. If this is the case the procedure replaces the ground atom by all those templates, thus weakening the originally complete information about the ground atom into still *true* disjunctions. Thus, so far, the view consists of the (distorted) *disjunctive part* $R(a, b, c) \lor R(a, b, d)$ and the (untouched) *positive part* formed by $R(a, c, c)$ and $R(b, a, c)$. Moreover, these parts are complemented by an (adapted) *negative part* that replaces the original closed-world assumption by a first-order sentence intuitively expressing that any ground atom that does not entail any element of the disjunctive part and of the positive part should be evaluated to *false*:

$$(\forall X)(\forall Y)(\forall Z)\,[(X \equiv a \land Y \equiv b \land Z \equiv c) \lor (X \equiv a \land Y \equiv b \land Z \equiv d)\lor \\ (X \equiv a \land Y \equiv c \land Z \equiv c) \lor (X \equiv b \land Y \equiv a \land Z \equiv c) \lor \neg R(X, Y, Z)\,]\,.$$

The weakened view consisting of the three parts does not entail any of the prohibited sentences. Moreover, capturing the receiver's assumed awareness of the control mechanism, the view is even *inference-proof* in the sense that for each prohibited sentence Ψ there is a fictitious "alternative" relation instance r^{Ψ} that would generate the same view as r and make Ψ *false*. In fact, after seeing the view and in particular learning the truth of the disjunction $R(a,b,c) \vee R(a,b,d)$, the receiver can not distinguish whether only $R(a,b,c)$ is *true* or only $R(a,b,d)$ is *true* or both $R(a,b,c)$ and $R(a,b,d)$ are *true*.

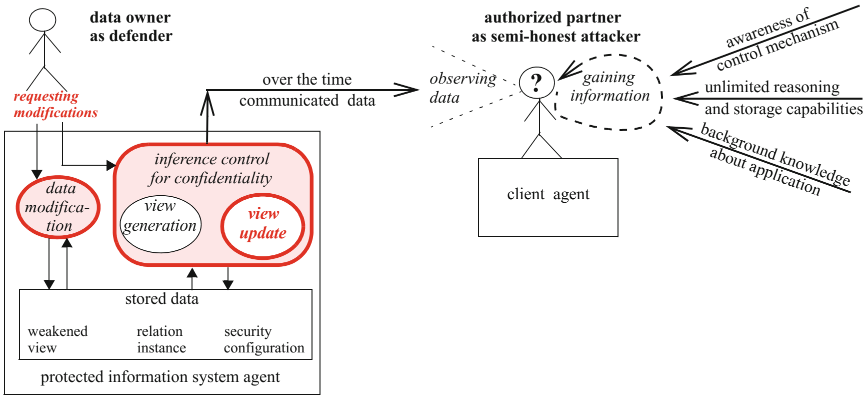

data owner as defender
requesting modifications
data modification
inference control for confidentiality
view generation
view update
over the time communicated data
stored data
weakened view
relation instance
security configuration
protected information system agent
authorized partner as semi-honest attacker
observing data
?
gaining information
client agent
awareness of control mechanism
unlimited reasoning and storage capabilities
background knowledge about application

Fig. 1. Visualization of the problem of confidentiality-preserving view updating

Considering the situation roughly visualized in Fig. 1, in the present work we address and solve the problem of efficiently *updating* a released weakened view under a modification of the input parameter values, while *continuously* enforcing the confidentiality policy, i.e., without revealing a prohibited piece of information, neither for the updated view nor retrospectively for the previous versions of the view. Conservatively assuming that the receiver is aware of the specification of both the view generation procedure and the updating procedure and, additionally, of the declared confidentiality policy – and thus of the whole security configuration consisting of the policy and the background knowledge – the main challenge has been to block all *meta-inferences* that the receiver could draw by relating subsequent views. The wanted blocking is achieved by establishing sufficient *indistinguishability* between the actual, possibly harmful situation and a fictitious harmless situation.

In Sect. 2, besides briefly discussing related work, we identify some basic conditions for achieving our goal in a still intuitive style. Then, in Sect. 3 we introduce our formal framework in order to prepare for proving precise assurances about our solution. This solution is presented and analyzed in Sect. 4. Finally, in Sect. 5 we report on the practical efficiency of a prototype implementation.

2 Conditions for Inference-Proof View Updating

Example 2 (instance modification). Continuing Example 1, let the owner now insert the tuple (a, b, d) into the relation instance r. The corresponding ground fact $R(a, b, d)$ entails the disjunctive template $R(a, b, c) \vee R(a, b, d)$, which however is already contained in the view, such that the view generation algorithm applied to the updated relation r' returns the same view as before, which per se appears to be harmless. However, if the receiver got informed about the mere fact of a successful insertion of a new tuple, by recognizing that nevertheless the view remained unchanged the receiver could figure out that originally only exactly one of the ground atoms $R(a, b, c)$ and $R(a, b, d)$ has been *true* and, thus, now both of them are *true*. But this would violate the requested confidentiality.

Now, suppose we start with the instance $r' = \{(a, b, c), (a, c, c), (b, a, c), (a, b, d)\}$ and then the owner deletes first (a, b, c) and then (a, b, d). At the beginning and also after the first deletion, the respective views are the same as above. But after the second deletion, the disjunction $R(a, b, c) \vee R(a, b, d)$ is removed from the view and the negative part is suitably adapted. Again, if the receiver got informed about the mere fact of two successful deletions, by recognizing that first the view remained unchanged and then the disjunction was dropped, the receiver could figure out that originally both ground atoms $R(a, b, c)$ and $R(a, b, d)$ have been *true*. But this would violate the requested confidentiality.

Such a kind of challenge has been identified earlier for diverse and only partially comparable settings, briefly and selectively classified in the following and further surveyed in Table 1; see also Sect. 6 of [8]. The *owner's data* might be either a relation instance focused on individuals [2,7,11,16,18–20] or, more generically, any logic-oriented knowledge or belief base which includes any relation instance under closed world assumption (this work, [3,5,6]). The *protection need* might refer to either the values of a sensitive attribute [2,7,11,16,18–20] or a suitable class of sentences in the underlying logic (this work, [3,5,6]), aiming at either a suitably strengthened version of l-diversity (with match-uncertainty [11]) or a general notion of continuous semantic indistinguishability, respectively. Similarly, the *background knowledge* might dedicatedly consist of either the population concerned [2,7,11,16,18–20] (under uniform publication procedures [11]) or, more generically, of a suitable class of sentences in the underlying logic (this work, [3,5,6]). Regarding *modifications*, there might be either none but only sequential releases of different views [18,20] or independently by other publishers [11], or insertion of tuples only [7,16] or both insertion and deletion of tuples [19] or, additionally, also value modification [2,3,5] or belief modification [6] or transactional modifications of not only the instance but also of the background knowledge and the confidentiality policy (this work). The *modification request* might be issued by either the information owner (this and most other work), or by the attacking receiver [3,5,6], as already earlier studied for mandatory multilevel databases with polyinstantiation. And the main *distortion* kind might be either lying [3] or refusals [5,6] or value generalization [2,7,11,16,18–20] or weakening by disjunctions (this work), the three latter ones possibly complemented by either restricted lying by fake tuples [2,19] or sampling and noise addition [11] or restricted refusals (this work).

Table 1. Properties of selected approaches to confidentiality-preserving updating

Approach/ Reference	Original Data	Background Knowledge	Confident. Policy	Kind of Modification	Notion of Confidentiality	Interaction with Receiver	Kind of Distortion	Communicated Data
Yao/Wang Jajodia 2005 [20]	relation instance (about individuals)	represented individuals with their identifiers; functional dependencies	values of a sensitive attribute for individuals	none (fixed set of tuples)	possibilistic k-anonymity for sensitive attribute	sequential release publishing of (select-project views)	none (only checking)	select-project views
Wang/ Fung 2006 [18]	relation instance (about individuals)	represented individuals with their identifiers	values of sensitive attributes for individuals	none (fixed set of tuples)	probabilistic l-diversity for sensitive attribute (as (X,Y)-privacy)	sequential release publishing of projections	generalization of attributes leading to a quasi-identifier	anonymized (generalized) projections
Xiao/ Tao 2007 [19]	relation instance about individuals	"lifespans" of represented individuals with their quasi-identifiers	values of a sensitive attribute for individuals	tuple insertion and deletion	probabilistic (lifespan-persistent) m-invariance for sensitive attribute	continuous (full) data publishing	generalization of quasi-identifiers and (lied) fake tuples	anonymized (generalized) instances with counts for fake tuples
Byun et al 2009 [7]	relation instance about individuals	"lifespans" of represented individuals with their quasi-identifiers	values of a sensitive attribute for individuals	tuple insertion	probabilistic "cross-version safe" k-anonymity with l-diversity for sensitive attribute	continuous (full) data publishing	generalization of quasi-identifiers with attack-checking of a "history table"	anonymized (generalized) instances
Shmueli/ Tassa 2015 [16]	relation instance about individuals	represented individuals with their quasi-identifiers	values of a sensitive attribute for individuals	tuple insertion	probabilistic l-diversity of "possible worlds" for sensitive attribute	sequential release and continuous data publishing (of projections)	generalization of quasi-identifiers using (lied) "fake worlds"	anonymized (generalized) projections
Li/ et al 2016 [11]	relation instance about individuals	represented individuals with their quasi-identifiers; uniform publishing	values of a sensitive attribute for individuals	none (fixed set of tuples)	probabilistic uncertainty about matches with independent releases	single (full) publishing, independently of overlapping further releases	generalization of quasi-identifiers and sensitive values; sampling; noise addition	anonymized (generalized, distorted) instance
Anjum et al 2017 [2]	relation instance about individuals	represented individuals with their quasi-identifiers and event-lists	values of a sensitive attribute for individuals	tuple insertion and deletion; modification in quasi-identifiers and sensitive att.	probabilistic τ-safety for sensitive attribute	continuous (full) data publishing	generalization of quasi-identifiers and (lied) fake tuples	anonymized (generalized) instances
Biskup et al 2011 [3]	complete propositional knowledge base	propositional sentences	any propositional sentences	view update transactions; view refreshments	continuous semantic possibilistic indistinguishability	query and update evaluations	lying	controlled (refreshed) query answers and notifications
Biskup/ Tadros 2012 [5]	complete propositional knowledge base	propositional sentences	any propositional sentences	view update transactions	continuous or temporary semantic possibilistic indistinguishability	(knowledge) query and update evaluations	refusal	controlled query answers and notifications
Biskup/ Tadros 2013 [6]	(incomplete) propositional belief base	initial belief approximation; class of nonmonotonic (belief) reasoning	any propositional sentences	belief revisions	temporary semantic skeptically possibilistic indistinguishability	(belief) query and revision evaluations	refusal	controlled query answers and notifications
published weakened views [this work]	relation instance with closed world assumption	single-premise tuple-generating dependencies	first-order sentences restricted to existential facts	transactions for modification of instance, background and policy	continuous semantic possibilistic indistinguishability	full view publishing	weakening by disjunctions (with some refusals)	positive, disjunctive, negative and refused data; notifications

Example 3 (policy modification). Again extending Example 1, let the owner now specify $R(a, c, c)$ as a new prohibition in the confidentiality policy. The first stage of the view generation procedure would aim at forming a disjunctive template covering the specified new prohibition and also being disjoint from all other templates. To achieve these goals, the procedure has to select an additional (artificial) prohibition, say $R(b, c, c)$, and might then add $R(a, c, c) \vee R(b, c, c)$ as a further disjunctive template. Since the tuple (a, c, c) is an element of the relation instance r, the ground fact $R(a, c, c)$ should no longer appear in the positive part of the modified view generated in the second stage. Instead, the weakening disjunction $R(a, c, c) \vee R(b, c, c)$ should become a further element of the disjunctive part, with the negative part being suitably adapted.

However, if the receiver could be sure that the relation instance r has not been modified, he would still know that $R(a, c, c)$ is *true*. This would violate the new prohibition and, thus, the weakening would be useless. In other words, if previous knowledge about the instance already indicates a violation of the modified policy, then inference-proofness of the updated view can not be achieved. This problem can be resolved by requiring that each modification of the confidentiality policy occurs as part of a *transaction* that might also comprise instance modifications, and thus previous knowledge about the instance could be no longer be valid.

More generally, also dealing with background knowledge as discussed below, we will show that always leaving the receiver uninformed about the kind of the requested modifications – in particular uncertain about additional instance modifications that are not reflected in the new weakened view – is sufficient to enforce the wanted notion of confidentiality. The examples considered so far together with the claimed generalized insights indicate that the underlying view generation procedure enjoys reasonable robustness regarding modifications of the instance and the policy. This behavior mainly results from two fundamental features of the overall approach: the *two-stage design* dealing with the policy and the instance separately, and the *strict isolation* of the three parts of a weakened view regarding entailments. However, achieving this isolation in the presence of *background knowledge*, so far neglected, requires quite subtle considerations presented in [4] and in more detail in [14].

In particular, and only briefly sketched, background knowledge affects the forming of disjunctive templates in the first stage of the view generation procedure in two ways. First, it might become necessary to introduce further prohibitions, which in particular strengthens the needs to clean the (extended) policy from redundancies. Second, the background knowledge has to be partitioned regarding unwanted joint entailment effects such that, roughly described, disjunctive templates have to be formed of suitably "independent" prohibitions that are not affected by sentences of the same partition block. In more general terms, the set of disjunctive templates might be modified. Moreover, in some cases the weakened view has to additionally comprise a *refused* part consisting of so-called *refusals*, i.e., sentences whose truth evaluations are explicitly denied whatever the stored relation might look like.

Example 4 (background modification). Again continuing Example 1, let now the database application have been changed such that *in future* all relation instances will satisfy the data dependency $R(a,b,d) \Rightarrow R(c,c,c)$. Moreover, let the owner assume that the receiver can henceforth exploit this dependency as his background knowledge. As known to him by the negative part of the view, $(a,b,d) \notin r$ and thus the premise of the dependency is not *true*. So, at first glance the dependency seems to be not helpful for the receiver. A second thought, however, easily indicates that the following inference would be enabled.

Since also $(c,c,c) \notin r$ and thus the conclusion $R(c,c,c)$ of the dependency is not *true* as well, applying the dependency in contraposition, i.e., $\neg R(c,c,c) \Rightarrow \neg R(a,b,d)$, the receiver can learn that $R(a,b,d)$ is *false*. Thus, given the truth of the disjunction $R(a,b,c) \vee R(a,b,d)$, the receiver can conclude that $R(a,b,c)$ is *true*. Hence, without suitable further precautions, the confidentiality policy would be violated. In fact, the underlying view generation procedure would already treat the conclusion $R(c,c,c)$ as a further prohibition.

The insights gained from the given examples and the lessons learnt from elaborating the above sketched solutions lead to the following list of conditions for inference-proof view updating:

- **C1:** (only) **conflict-free requests**:
  An initial input control checks whether a modification request of the owner consists of insertions and deletions that are not conflicting. In particular, an item should not be required to be both inserted and deleted within the same modification request, and the items to be modified should be consistent.
- **C2:** (only) **transactions**:
  The accepted inputs of the owner are processed as a transaction with semantics that lead to either a commit (all temporary modifications of the relation instance and the security configuration are made persistent) or an abort (the relation instance and the security configuration remain unchanged, i.e., all temporary actions are rollbacked).
- **C3:** (only) **possibly comprehensive transactions**:
  Extending condition C2, additionally, (from the point of view of the receiver) the inputs for each transaction might be comprehensive, i.e., they might always comprise *all kinds* of modifications, i.e., simultaneously instance modifications, background modifications and policy modifications.
- **C4:** (only) **state-related invariants**:
  Each invariant whose satisfaction is checked for the final decision on either committing or aborting the transaction only refers to the preliminarily generated internal situation, but not to the relationship between the previous one and the still preliminary one.
- **C5: notifications**:
  The receiver is always notified about a request of the owner for modifications by either returning the updated weakened view or sending a note about an input rejection or a transaction abort, respectively.

- **C5*:** only **notifications of effective and committed transactions**: More restrictively, the receiver is notified about a request of the owner for modifications only if (i) the inputs are not rejected, (ii) the transaction has been committed and (iii) the view update has been effective, i.e., the weakened view has actually been changed. Otherwise, an owner's input attempt is totally invisible to the receiver.
- **C6: observability of the security configuration**: The receiver can always learn the somehow "posted" current security configuration which includes the awareness of related requests for modification (but the receiver can never see the current relation instance nor requests for instance modifications).

For most applications, we see no need to inform the receiver about internal modification requests that do not actually change the published view. Accordingly, as expressed by condition C5*, it appears to be reasonable to completely hide the processing of requests that do change the external view. Technically, condition C5* would require to consider possibly differing local times of the owner and of the receiver, respectively. Then, to distinguish points in time local to the owner that are observable by the receiver and those points that are not, we would have to employ a rather complicated formal representation of our approach.

However, under the remaining conditions C1–C6 we can show that our view update procedure is inference-proof even if the receiver can observe the fact (but not the internal effects) of unsuccessful internal processing. Accordingly, to simplify the presentation by avoiding asynchronous local times, we will elaborate our approach based on conditions C1–C6, with a global discrete time for both agents, with points in time $1, 2, \dots$ used as synchronous timestamps.

3 Basic Concepts and Formal Definitions

We will formally define the *basic concepts* leading to the underlying *view generation* procedure, briefly summarize the assurances proved before in [4,14], and introduce a precise notion of *continuous inference-proofness* to be enforced by the new *view update* procedure. Figure 2 outlines the framework.

3.1 Database Management System

We consider a *relational database* management system, which is operated under the control of the data *owner*. The system is based on a single relational *schema*, which comprises a fixed relation *symbol* (table name) R, a fixed finite set of *attributes* (column names) $\mathcal{A} = \{A_1, \dots, A_n\}$, each of which has the same *infinite domain Dom* of constants, and some possibly time-varying set SC_t of *semantic constraints*. At each point in time t the system maintains a database *instance* r_t, which is a finite set of tuples over $\mathcal{A}$ with values in *Dom*, satisfying the current semantic constraints in SC_t. Such an instance is treated as being *complete*: each tuple in r_t represents a fact that is *true* in some fictitious "real world"; whereas,

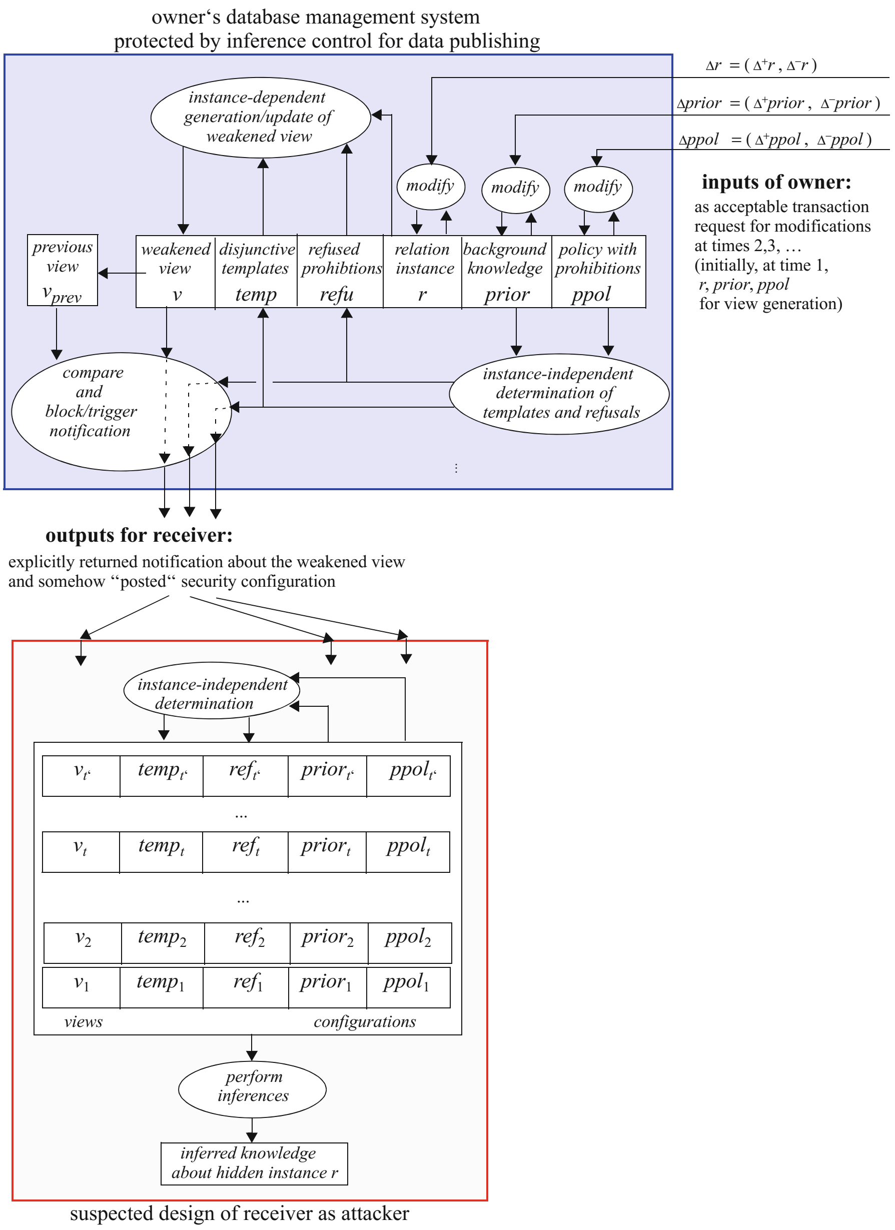

owner's database management system
protected by inference control for data publishing
$\Delta r = (\Delta^+ r, \Delta^- r)$
$\Delta prior = (\Delta^+ prior, \Delta^- prior)$
$\Delta ppol = (\Delta^+ ppol, \Delta^- ppol)$
inputs of owner:
as acceptable transaction request for modifications at times 2,3, …
(initially, at time 1, r, prior, ppol for view generation)
instance-dependent generation/update of weakened view
modify
modify
modify
previous view v_{prev}
weakened view v
disjunctive templates $temp$
refused prohibtions $refu$
relation instance r
background knowledge $prior$
policy with prohibitions $ppol$
compare and block/trigger notification
instance-independent determination of templates and refusals
outputs for receiver:
explicitly returned notification about the weakened view and somehow "posted" security configuration
instance-independent determination
$v_{t'}$ $temp_{t'}$ $ref_{t'}$ $prior_{t'}$ $ppol_{t'}$
v_t $temp_t$ ref_t $prior_t$ $ppol_t$
v_2 $temp_2$ ref_2 $prior_2$ $ppol_2$
v_1 $temp_1$ ref_1 $prior_1$ $ppol_1$
views
configurations
perform inferences
inferred knowledge about hidden instance r
suspected design of receiver as attacker

Fig. 2. Outline of the owner's protection (upper part) against the anticipated receiver seen as a rational, omnipotent and too-curious attacker (lower part)

by Closed World Assumption (CWA), each other tuple over $\mathcal{A}$ with values in Dom represents a possible fact which is *false* in that world.

We follow a foundation of the relational model of data in terms of *first-order logic* with equality, as also used, e.g., in [1,10]. Syntactically, the logic is specified by a language $\mathscr{L}$ over $\equiv$, R, $\mathcal{A}$, *Dom*, variables, propositional connectives and first-order quantifiers in the usual way. Semantically, for this logic we treat a database tuple $(a_1, \ldots, a_n)$ as a ground fact $R(a_1, \ldots, a_n) \in \mathscr{L}$ and a database instance as a finite Herbrand interpretation of $\mathscr{L}$ with the infinite universe *Dom* assuming *unique names*. Using an instance in this way, we can inductively assign a truth value to each sentence in $\mathscr{L}$. This foundation also provides us with the pertinent notions of *satisfaction* and *entailment*: an instance r, seen as an Herbrand interpretation of the kind described above, *satisfies* a sentence $\Phi \in \mathscr{L}$ (r is a model of Φ, $r \models \Phi$) iff the truth evaluation according to r returns the truth value *true*; a set $\mathcal{S} \subseteq \mathscr{L}$ of sentences *entails* a sentence $\Phi \in \mathscr{L}$ ($\mathcal{S} \models \Phi$) iff each instance r satisfying $\mathcal{S}$ also satisfies Φ.

For confidentiality policies we employ the sublanguage $\mathscr{L}_{exist}$ of *existential facts*, which are sentences in $\mathscr{L}$ of the form $(\exists X_{i_1}) \ldots (\exists X_{i_m}) R(t_1, \ldots, t_n)$ with pairwise different variables $X_{i_1}, \ldots, X_{i_m}$ and terms $t_{i_j} = X_{i_j}$ for $i_j \in \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$ and $t_i \in Dom$ otherwise. Such a sentence corresponds to a subtuple where the components for the attributes in $\{A_{i_1}, \ldots, A_{i_m}\}$ are dropped. We also see ground facts (without any existentially quantified variables) as elements of $\mathscr{L}_{exist}$. For weakening we employ the sublanguage $\mathscr{L}^{\vee}_{exist}$ of strict and non-redundant *disjunctions* over $\mathscr{L}_{exist}$, i.e., all sentences of the form $\Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_k$ such that $k \geq 2$, $\Psi_i \in \mathscr{L}_{exist}$ and $\Psi_i \not\models \Psi_j$ for $i \neq j$.

3.2 Inference Control for Data Publishing

The database management system is protected by an *inference control* system for *data publishing*, which for each point in time t internally determines a receiver-specific current weakened view v_t on the current relation instance r_t. The current view v_t also depends on the current *security configuration*, which consists of the currently assumed background knowledge $prior_t \supseteq SC_t$ of the receiver, and the currently declared confidentiality policy $ppol_t$ for the receiver.

The initial view v_1 is generated by applying the underlying *view generation* procedure *vgen* [4,14], i.e., $v_1 = vgen(r_1, prior_1, ppol_1)$. During its first still instance-independent stage, this procedure *vgen* also internally determines the initial set $temp_1$ of disjunctive templates and the initial set $refu_1$ of refusals.

All further weakened views v_t, for times $t > 1$, are determined by a *view update* procedure *vupd*, to be presented in the remainder of this article, i.e., $v_t = vupd(\,(\Delta^{+}r_t, \Delta^{-}r_t), (\Delta^{+}prior_t, \Delta^{-}prior_t), (\Delta^{+}ppol_t, \Delta^{-}ppol_t)\,)$. The following definition specifies the envisioned structure of such a procedure.

Definition 1 (specification of view update procedure). *At times* $t = 2, 3, \ldots$ *a* view update *procedure vupd determines a weakened view* $v_t = vupd(\,(\Delta^{+}r_t, \Delta^{-}r_t), (\Delta^{+}prior_t, \Delta^{-}prior_t), (\Delta^{+}ppol_t, \Delta^{-}ppol_t)\,)$*, based on the previous* internal owner state*, defined by*

$$own_{t-1} = (v_{t-1}, temp_{t-1}, refu_{t-1}, r_{t-1}, prior_{t-1}, ppol_{t-1})\,, \tag{1}$$

and satisfying the following conditions:

- *The previous internal state own_{t-1} is accessible for the view update procedure vupd, but more aged internal states are not memorized.*
- *The* explicit input parameter *values* $(\Delta^{+}r_t, \Delta^{-}r_t)$, $(\Delta^{+}prior_t, \Delta^{-}prior_t)$ *and* $(\Delta^{+}ppol_t, \Delta^{-}ppol_t)$ *for the requested modifications are internally specified by the owner, for each parameter indicating which elements are to be inserted in and which elements are to be deleted from the previous state.*
- *A requested modification is actually* accepted *and* committed only if *(i) the input parameter values are conflict-free and (ii) all pertinent invariants expressed in terms of a state are maintained; otherwise the request would be* rejected *or* aborted*, respectively.*
- *The components of the internal state are* updated *as follows:*
  - $ppol_t := [\,ppol_{t-1} \cup \Delta^{+}ppol_t\,] \setminus \Delta^{-}ppol_t$ *;*
  - $prior_t := [\,prior_{t-1} \cup \Delta^{+}prior_t\,] \setminus \Delta^{-}prior_t$ *;*
  - $r_t := [\,r_{t-1} \cup \Delta^{+}r_t\,] \setminus \Delta^{-}r_t$ *;*
  - $temp_t$ *and* $refu_t$ *are assigned the same results as the underlying view generation procedure vgen would do in its first stage;*
  - v_t *is assigned the* return value *of the view update procedure vupd.*
- *The receiver always gets* notified *about the fact of a modification request.*

Cautiously assuming condition C5 and suspecting the memorization of the full history, the observable effects of initial view generation and repeated view updates on the side of the receiver are represented by the sequence of current *attacker states*, each of them defined by $att_t = (v_t, temp_t, refu_t, prior_t, ppol_t)$.

3.3 The Underlying View Generation Procedure

The underlying *view generation* procedure $vgen(r, prior, ppol)$ [4,14] takes three inputs from the owner: a relation instance r, the assumed background knowledge *prior* of the receiver, and the confidentiality policy *ppol* for that receiver. During a first, still instance-independent stage, a subprocedure $vgen\_stage1(prior, ppol)$ internally determines a set *temp* of disjunctive templates and a set *refu* of elements stemming mainly from *ppol* and leading to refusals. In a second, instance-dependent stage, a subprocedure $vgen\_stage2(temp, refu, r)$ generates a weakened view v that consists of four parts: the refused knowledge $v^{?} := refu$, the positive knowledge v^{+}, the disjunctive knowledge $v^{\vee}$, and the negative knowledge v^{-}. In order to avoid inferences based on the editorial representation of these parts some final *normalization* based on standardized sorting is due. Accordingly, the view generation procedure has the following overall structure:

```
PROCEDURE $vgen(r, prior, ppol)$ {
$(temp, refu) := vgen\_stage1(prior, ppol)$;
$(v^{?}, v^{+}, v^{\vee}, v^{-}) := vgen\_stage2(temp, refu, r)$;
$v := norm(v^{?}, v^{+}, v^{\vee}, v^{-})$;
notify receiver by sending $v$ }
```

Employing first-order logic, all items are formalized by means of suitable subsets of $\mathscr{L}$. Capturing the intuitions and our goals on the one hand and facing the well-known difficulty of the computational unsolvability of the general entailment problem for the full first-order logic language $\mathscr{L}$ on the other hand, see, e.g., [13], we will apply the conventions summarized in the following.

Regarding the input parameters:

– The *relation instance* r is seen as a finite set of ground facts of the form $R(a_1, \ldots, a_n)$, complemented with a pertinent completeness sentence $Comp(r)$; i.e., for $r = \{(a_{1,1}, \ldots, a_{1,n}), \ldots, (a_{m,1}, \ldots, a_{m,n})\}$ we get

$$(\forall X_1) \ldots (\forall X_n)[\bigvee_{(a_{j,1},\ldots,a_{j,n}) \in r} (\bigwedge_{i \in \{1,\ldots,n\}} X_i \equiv a_{j,i}) \ \vee \neg R(X_1, \ldots, X_n)].$$

– Establishing knowledge about the relationship of one single fact with another single fact, the *background knowledge prior* is a finite set of single-premise tuple-generating dependencies [1] of the syntactic form[1]

$$(\forall X_1) \ldots (\forall X_k) [R(t_1, \ldots, t_n) \Rightarrow (\exists Y_1) \ldots (\exists Y_l) R(\bar{t}_1, \ldots, \bar{t}_n)], \qquad (2)$$

where $X_1, \ldots, X_k, Y_1, \ldots, Y_l$ are pairwise different variables, each universally quantified variable X_i occurring exactly once in $R(t_1, \ldots, t_n)$ and at most once in $R(\bar{t}_1, \ldots, \bar{t}_n)$, each existentially quantified variable Y_j occurring exactly once in $R(\bar{t}_1, \ldots, \bar{t}_n)$, and – preferably to avoid an overall refusal – in both $R(t_1, \ldots, t_n)$ and $R(\bar{t}_1, \ldots, \bar{t}_n)$ at least one constant of Dom occurs.

– The *confidentiality policy* $ppol \subset \mathscr{L}_{exist}$ is a finite set of existential facts.

Regarding the first stage, further outlined below:

– As far as possible, the finite set *temp* of disjunctive *templates* with $temp \subset \mathscr{L}^{\vee}_{exist}$ should only be formed by elements of the confidentiality policy, covering all of them. Moreover, all disjunctions seen together should be *mutually independent* in the following sense: for each two different disjunctions $\Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_k$ and $\bar{\Psi}_1 \vee \bar{\Psi}_2 \vee \ldots \vee \bar{\Psi}_{\bar{k}}$ we have $\Psi_i \not\models \bar{\Psi}_j$ and $\bar{\Psi}_j \not\models \Psi_i$.

– The finite set *refu* of refusals contains selected policy elements and possibly further prohibition sentences and, thus, $refu \subset \mathscr{L}_{exist}$.

Regarding the second stage:

– The *refused knowledge* is instance-independent and just comprises the refusals determined in the first stage, i.e., $v^? := refu$.

– The *positive knowledge* gathers all ground facts (tuples) of the relation instance r that entail neither a refusal nor a disjunctive template, i.e., $v^+ := \{ \Phi \mid \Phi \in r \text{ and for all } \Psi \in refu : \Phi \not\models \Psi, \text{for all } \tau \in temp : \Phi \not\models \tau \}$.

[1] To simplify our treatment, we do not consider definite background knowledge with an empty premise part. Thus, in particular, background knowledge can not per se entail any possible prohibition.

- The *disjunctive knowledge* $v^{\vee}$ is formed as follows. As far as needed for confidentiality, a ground fact (tuple) $R(a_1,\ldots,a_n)$ in the relation instance r is disjunctively *weakened* by replacing it in a *context-free* way by a disjunction $\Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_k$ taken from the previously, in the first stage determined set of disjunctive templates *temp* such that $R(a_1,\ldots,a_n) \models \Psi_1 \vee \Psi_2 \vee \ldots \vee \Psi_k$. In fact, in order to conveniently capture many simultaneous threats to confidentiality, the replacement is performed with *all* such disjunctions. Formally, $v^{\vee} := \{\, \tau \mid \tau \in \mathit{temp}$ and there exists $\Phi \in r : \Phi \models \tau \,\}$[2].
- The *negative knowledge* consists of the suitably adapted pertinent completeness sentence, i.e., $v^{-} := \mathit{Comp}(\, v^{+}, v^{\vee}, \mathit{temp}, \mathit{refu}\,)$.

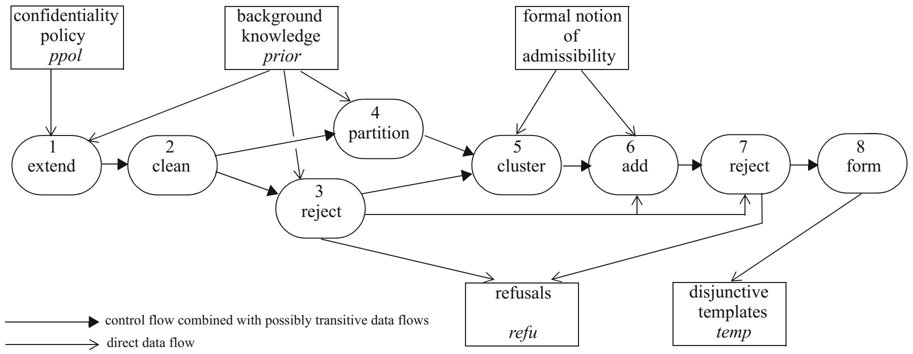

confidentiality policy *ppol*
background knowledge *prior*
formal notion of admissibility
1 extend
2 clean
3 reject
4 partition
5 cluster
6 add
7 reject
8 form
refusals *refu*
disjunctive templates *temp*
control flow combined with possibly transitive data flows
direct data flow

Fig. 3. Direct and transitive data flows and control flow in stage 1

The first stage, which is still independent of r, can be further outlined as follows, and as also visualized in Fig. 3:

1. *extend* the policy by *implicit prohibitions* caused by a single dependency;
2. *clean* the policy from semantically *redundant prohibitions*;
3. *reject* (delete) conflicting prohibitions and establish *refusals* in *refu* instead;
4. *partition* the set of dependencies according to interactions with prohibitions;
5. respecting the partitioning, *cluster* prohibitions into admissible[3] groups;
6. if possible, *add synthetic prohibitions* for completing a partial match;
7. *reject* prohibitions remained isolated and establish additional *refusals* in *refu*;
8. *form templates* of *temp* as disjunctions, one for each group of the clustering.

[2] In reference [4], we additionally required a void overlapping with the refused knowledge to ensure unique interactive control decisions; the current weaker requirement is in accordance with the detailed elaboration in reference [14].

[3] As elaborated in [14], the notion of admissibility is intended to formally capture application-oriented needs, in particular aiming at the plausibility of a disjunctive template and an approximate equal likelihood of its disjuncts.

3.4 Continuous Inference-Proofness

Intuitively, as inspired by [9] and closely following [3,5,6], a prohibition sentence $\Psi \in ppol_t$ is intended to express a strong *semantic confidentiality* requirement: from the point of view of the receiver, based on the explicitly returned pieces of data and the somehow "posted" security configurations, at all times $t' \geq t$ it should appear to be *possible* that the prohibition sentence Ψ has *not* been *true* at time t. In other words, even if Ψ has actually been *true* in the (hidden) relation instance r_t, the receiver should not be sure about this situation.

This intuition will be formalized as roughly outlined in the following. Based on (i) his (assumed) time-depending background knowledge $prior_1$, $prior_2, \ldots, prior_{t'}$, (ii) his awareness of the time-depending confidentiality policy $ppol_1, ppol_2, \ldots, ppol_{t'}$, and (iii) the observed notifications of either rejection, abortion or commitment, including the weakened views $v_1, v_2, \ldots, v_{t'}$ – originating from the actual (hidden) initial instance r_1 and the actual (hidden) instance modification parameters $\Delta r_2, \Delta r_3, \ldots, \Delta r_{t'}$ –, the receiver can imagine that the same observations could result from a (fictitious) alternative instance r_1^Ψ and (fictitious) instance modifications Δr_2^Ψ, $\Delta r_3^\Psi, \ldots,$ $\Delta r_{t'}^\Psi$ such that the then resulting (also fictitious) relation instance r_t^Ψ does *not* satisfy Ψ.

Definition 2 (continuous (possibilistic) inference-proofness). *Under conditions C1–C6, a* view update *procedure vupd according to Definition 1* continuously *complements the* view generation *procedure vgen in an* inference-proof *way iff, from the point of view of the receiver:*
for each (hidden) initial relation instance r_1,
for each (known) initial background knowledge $prior_1$,
for each (known) initial confidentiality policy $ppol_1$ and
for each (totally hidden) sequence of instance modifications $\Delta r_2, \Delta r_3, \ldots, \Delta r_{t'}$,
for each (known) sequences of background modifications $\Delta prior_2, \Delta prior_3, \ldots,$ $\Delta prior_{t'}$ and policy modifications $\Delta ppol_2, \Delta ppol_3, \ldots, \Delta ppol_{t'}$
under the procedures vgen and vupd leading to
the (known) attacker states $att_1, att_2, \ldots, att_{t'}$ and
the (hidden) relation instance r_t at a point in time $t \leq t'$,
for each prohibition sentence $\Psi \in ppol_t$
there exists an "alternative hidden situation", i.e., there exist a (fictitious) relation instance r_1^Ψ and a (fictitious) sequence of instance modifications Δr_2^Ψ, $\Delta r_3^\Psi, \ldots,$ $\Delta r_{t'}^\Psi$ such that

1. indistinguishability of the alternative hidden situation:
under the procedures vgen and vupd, the instance parameters r_1^Ψ and Δr_2^Ψ, $\Delta r_3^\Psi, \ldots,$ $\Delta r_{t'}^\Psi$ together with the background parameters $prior_1$ and $\Delta prior_2, \Delta prior_3, \ldots, \Delta prior_{t'}$ and with the policy parameters $ppol_1$ and $\Delta ppol_2, \Delta ppol_3, \ldots, \Delta ppol_{t'}$ generate the same notifications and the same attacker states $att_1^\Psi = att_1$, $att_2^\Psi = att_2, \ldots,$ $att_{t'}^\Psi = att_{t'}$, in particular the same weakened views, i.e., regarding Ψ, the hidden items are indistinguishable *from the fictitious items;*

2. credibility of the alternative situation: r_j^{Ψ} *satisfies* $prior_j$*, for* $j = 1, \ldots, t'$*;*
3. harmlessness of the alternative situation: r_t^{Ψ} *does* not *satisfy* Ψ*.*

If we restrict Definition 2 to the special case $t' = 1$, i.e., that only initially, at time 1, the view generation procedure *vgen* has been applied but subsequently no modifications have been requested, we just obtain the notion of (static) semantic confidentiality dealt with in our previous work [4,14] and, thus, according to Theorem 1 and Theorem 2 of [4], the following proposition holds.

Proposition 1. *The view generation procedure vgen (restricted to acceptable input parameter values) complies with* static *(possibilistic)* inference-proofness*, i.e., from the point of view of the receiver: for each (hidden) initial relation instance* r_1*, for each (known) initial background knowledge* $prior_1$*, for each (known) initial confidentiality policy* $ppol_1$*, for each prohibition sentence* $\Psi \in ppol_1$ *there exists an "alternative hidden situation", i.e., there exists a (fictitious) relation instance* r_1^{Ψ} *such that*

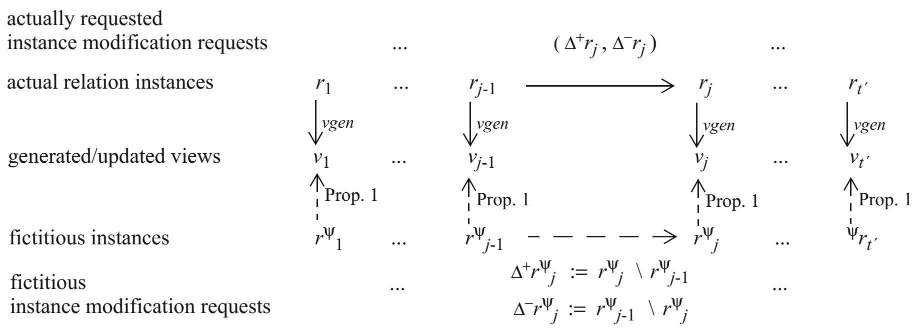
1. indistinguishability of the alternative situation: *under the procedure vgen,* r_1^{Ψ} *together with* $prior_1$ *and* $ppol_1$ *generates the same weakened view* v_1*;*
2. credibility of the alternative situation: r_1^{Ψ} *satisfies* $prior_1$*;*
3. harmlessness of the alternative situation: r_1^{Ψ} *does* not *satisfy* Ψ*.*

4 The View Update Procedure

Based on the informally stated conditions C1–C6 identified in Sect. 2 and the formal specifications outlined in Sect. 3, we are now ready to present our main contribution: a concrete view update procedure for weakened views and a verification of its compliance with continuous inference-proofness. As discussed in Sect. 2, to show inference-proofness under as weak conditions as reasonable, we define the procedure in accordance with condition C5 (notifications), such that the receiver gets notified about the fact of any owner request. However, for practical applications, we do not recommend to do so but, following condition C5*, to inform the receiver only about actually changed views. Moreover, condition C6 (observability of security configuration) is not explicitly expressed in the procedure but only employed in its verification assuming an utmost powerful attacking receiver.

```
PROCEDURE $vupd( (\Delta^{+}r, \Delta^{-}r), (\Delta^{+}prior, \Delta^{-}prior), (\Delta^{+}ppol, \Delta^{-}ppol) )$ {
IF modification requests are not conflict-free
  THEN notify owner in detail about the detected conflicts;
          notify receiver only about the mere fact of a conflict
  ELSE*****if there are no conflicts
    BEGIN_TRANSACTION
      $own_{prev} := own$;
      $prior := [prior \cup \Delta^{+}prior] \setminus \Delta^{-}prior$;
      $r \quad := [r \cup \Delta^{+}r] \setminus \Delta^{-}r$;
      IF invariants satisfied (here: $prior$ is satisfied by $r$)
```

```
        THEN $ppol := [\,ppol \cup \Delta^{+}ppol] \setminus \Delta^{-}ppol$;
             IF $(prior_{prev}, ppol_{prev}) \neq (prior, ppol)$
               THEN $(temp, refu) := vgen\_stage1(prior, ppol)$ FI;
             IF $(temp_{prev}, refu_{prev}, r_{prev}) \neq (temp, refu, r)$
               THEN $(v^{?}, v^{+}, v^{\vee}, v^{-}) := vgen\_stage2(temp, refu, r)$;
                    $v := norm(v^{?}, v^{+}, v^{\vee}, v^{-})$ FI;
             commit (make all modifications persistent);
             notify owner about commit;
             notify receiver about commit by sending $v$
        ELSE abort (restore previous values of $prior$ and $r$);
             notify owner in detail about violation of invariants;
             notify receiver only about the mere fact of a violation
      FI
    END_TRANSACTION
FI }
```



actually requested instance modification requests
...
$(\Delta^{+}r_j, \Delta^{-}r_j)$
...
actual relation instances
r_1
r_{j-1}
r_j
$r_{t'}$
vgen
generated/updated views
v_1
v_{j-1}
v_j
$v_{t'}$
Prop. 1
fictitious instances
r^{Ψ}_1
r^{Ψ}_{j-1}
r^{Ψ}_j
$r^{\Psi}_{t'}$
fictitious instance modification requests
$\Delta^{+}r^{\Psi}_j := r^{\Psi}_j \setminus r^{\Psi}_{j-1}$
$\Delta^{-}r^{\Psi}_j := r^{\Psi}_{j-1} \setminus r^{\Psi}_j$

Fig. 4. Construction of "alternative situations" regarding instance modifications

Theorem 1. *Procedure vupd complies with* continuous inference-proofness *in the sense of Definition 2.*

Proof. Let the procedures *vgen* and *vupd* inductively determine a sequence of internal owner states $own_1, own_2, \ldots, own_{t'}$, as defined by (1) within Definition 1. Basically, besides dedicated arguments for conflicting parameter values and transaction abortion, for the standard case of transaction commitment we will apply Proposition 1 for each point in time individually, as indicated in Fig. 4.

More specifically, for $j = 1$, Proposition 1 directly ensures the existence of an "alternative situation" with the same notification.

Inductively, for $j > 1$, according to the declaration of *vupd* we have to distinguish three mutually excluding cases.

Case 1, parameter values conflicting: A conflict can only occur for the following reasons: contradictory insert and delete requests or inconsistent modification requests. Both an actual contradiction and an actual inconsistency could always be imagined to result from alternative fictitious ones, respectively, leading to the same notification and leaving the internal owner state unchanged, and thus the induction hypothesis applies.

Case 2, transaction aborted: The transaction for the modification of the instance and the security configuration is only aborted if the tentatively modified (known) background knowledge $prior_j$ does not satisfy the tentatively modified (hidden) instance r_j. So, there exists a single-premise tuple-generating dependency in $prior_j$ that is violated by r_j and is of the form defined by (2) in Sect. 3.3, e.g.,

$$(\forall X_1)\dots(\forall X_k)\,[\,R(t_1,\dots,t_n) \Rightarrow (\exists Y_1)\dots(\exists Y_l)\,R(\bar{t}_1,\dots,\bar{t}_n)\,]\ .$$

Accordingly, there exists a substitution σ that replaces the universally quantified variables $X_1,\dots,X_k$ with the constants $c_1,\dots,c_k$ such that (i) $\sigma[(t_1,\dots,t_n)] \in r_j$ but (ii) for all constant substitutions τ of the existentially quantified variables $Y_1,\dots,Y_l$ we have $\tau[\sigma[(\bar{t}_1,\dots,\bar{t}_n)]] \notin r_j$.

This actual situation could also result from a fictitious instance modification regarding the fictitious instance r^{Ψ}_{j-1} that requests to insert $\sigma[(t_1,\dots,t_n)]$ and to delete all those (finitely many) $\tau[\sigma[(\bar{t}_1,\dots,\bar{t}_n)]]$ which have been in r^{Ψ}_{j-1}. Accordingly, the abort notification for the actual situation equals the abort notification for the fictitious situation. Moreover, in both situations the internal owner state and thus also the attacker state remains the same as at time $j-1$ such that the induction hypotheses about the situation at time $j-1$ immediately implies the assertion about time j.

Case 3, transaction committed: Consider the committed execution of

$$vupd(\,(\Delta^{+}r_j, \Delta^{-}r_j), (\Delta^{+}prior_j, \Delta^{-}prior_j), (\Delta^{+}ppol_j, \Delta^{-}ppol_j)\,)\,.$$

This execution first determines new components $prior_j$, $ppol_j$ and r_j for the internal state, and then determines the same updated view v_j as the view generation procedure *vgen* would have done applied to these components. Thus, according to Proposition 1, there exists an "alternative" fictitious instance r^{Ψ}_j leading to the same view v_j under the procedure *vgen* applied to $prior_j$, $ppol_j$ and r^{Ψ}_j. Now, we observe that this fictitious instance r^{Ψ}_j can also be obtained from the inductively assumed fictitious instance r^{Ψ}_{j-1} by an instance modification request with parameter values

$$\Delta^{+}r^{\Psi}_j := r^{\Psi}_j \setminus r^{\Psi}_{j-1} \text{ and } \Delta^{-}r^{\Psi}_j := r^{\Psi}_{j-1} \setminus r^{\Psi}_j\ .$$

Then, the fictitious execution of

$$vupd(\,(\Delta^{+}r^{\Psi}_j, \Delta^{-}r^{\Psi}_j), (\Delta^{+}prior_j, \Delta^{-}prior_j), (\Delta^{+}ppol_j, \Delta^{-}ppol_j)\,)$$

would also generate the same view v_j for the following reasons: by condition C3, these parameter values are possible, and by condition C4, the transaction would commit for these parameter values as well. Accordingly, the notification for the actual situation equals the notification for the fictitious situation. □

5 Experimental Runtime Evaluation

We presented the view *update* procedure *vupd* in a straightforward way in order to facilitate its verification. However, we might attempt to replace the employed *recomputation* of the new internal state by means of the two subprocedures *vgen_stage1* and *vgen_stage2* of the underlying view *generation* procedure *vgen* by a more efficient *incremental* determination of the new internal state.

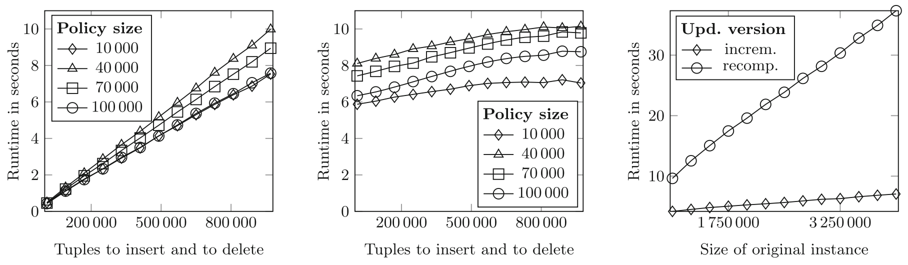
Regarding the subprocedure *vgen_stage1*, the outline given in Sect. 3.3 and visualized by Fig. 3 already roughly indicates that the final results *temp* and *refu* depend on the inputs *ppol* and *prior* in a transitively dependent way, along the whole chain of the eight steps. For example, the insertion of a new prohibition into *ppol* might raise further extensions in step 1, which in turn might introduce new redundancies that in step 2 can trigger rather involved non-monotonic cleaning effects: a previously kept prohibition is sometimes removed in favor of a new prohibition, but sometimes it remains untouched causing the removal of new prohibitions as being redundant. The alternatives decided in step 2, adjusted by identifying refusals in step 3, further effects all succeeding steps, both by using the result of step 3 as direct input and indirectly via the transitive data flows, in particular incorporated by the partition generated in step 4. A more detailed analysis and corresponding options for optimized, partly incremental computations are beyond the scope of the present work.

Regarding the subprocedure *vgen_stage2*, we can replace its simple call by essentially more refined operations if the results of the first stage, *temp* and *refu*, have remained unchanged. This is an outline of an incremental approach:

- The instance-independent *refused knowledge* remains $v^{?} := \mathit{refu}$.
- For updating the *positive knowledge*, basically only the elements of the input parameter value $(\Delta^{+}r, \Delta^{-}r)$ have to be processed, rather than the whole modified relation r, i.e., under the precondition $\Delta^{+}r \cap \Delta^{-}r = \emptyset$, $v^{+} := [v^{+}_{prev} \cup \{\, \Phi \mid \Phi \in \Delta^{+}r \text{ and for all } \tau \in \mathit{temp} \cup \mathit{refu} : \Phi \not\models \tau \}\,] \setminus \Delta^{-}r$.
- For updating the *disjunctive knowledge*, similarly only the elements of the input parameter value $(\Delta^{+}r, \Delta^{-}r)$ have to be processed, i.e., $v^{\vee} := [v^{\vee}_{prev} \setminus \{\, \tau \mid \tau \in v^{\vee}_{prev} \text{ and for all } \Phi \in r_{prev} \text{ with } \Phi \models \tau : \Phi \in \Delta^{-}r\}] \cup \{\, \tau \mid \tau \in \mathit{temp} \text{ and there exists } \Phi \in \Delta^{+}r : \Phi \models \tau \,\}$.
- The completeness sentence for *negative knowledge* is adapted accordingly.

We also extended the prototype implementation of [4,14] to instantiate the new view update procedure *vupd* in two versions, straightforward and incremental. All crucial subroutines of this implementation, which are employed for view generations and view updates, are developed in Java 8 and parallelized to benefit from modern hardware. The experiments were run under Ubuntu 14.04 on a machine with two "Intel Xeon E5-2690" CPUs, providing a total number of 16 physical and 32 logical cores (due to hyperthreading) running at 2.9 GHz.

Within Experiment 1 an original instance with 1 000 000 database tuples is modified by inserting and deleting the same number of randomly chosen database tuples, varying from 10 000 to 970 000. Comparing the Figs. 5(a) and (b),



(a) Incremental view updates under varying size of instance modifications

(b) Recomputed view updates under varying size of instance modifications

(c) View updates under increasing size of original instance

Fig. 5. Experimental runtime comparison of the incremental and the straightforward recomputation (of stage 2) version of the view update procedure

it becomes clear that in terms of runtime an incremental view update is nearly always better than a recomputation of a weakened view with *vgen_stage2*. Even if about the full database instance is to be replaced, there is usually little reason *not* to employ the incremental procedure.

Experiment 2 then applies a sequence of instance modifications to an original instance with initially 1 000 000 tuples. Each of these modifications inserts 500 000 random tuples and deletes only 250 000 random tuples, resulting in modified original instances enlarged up to 4 250 000 tuples. A quick look at Fig. 5(c) reveals that the incremental procedure clearly outperforms recomputations.

6 Conclusion

For a specific approach to confidentiality-preserving data publishing, we addressed the challenging problem of how to *update a published view* according to modifications of the underlying original data or of the security configuration *without revealing sensitive information.* Basically, as far as needed for complying with a declarative confidentiality policy, and whenever possible, that approach weakens the knowledge embodied in a tuple of a complete relation instance into a piece of disjunctive knowledge formed from elements of the policy. In a first still instance-independent stage disjunctive templates (and, if required, refusals) are suitably determined, and in a second instance-dependent stage each tuple is inspected individually whether it has to be disjunctively weakened according to one or more of the disjunctive templates (or even be refused). The first stage guarantees that all templates are sufficiently mutually isolated regarding logic entailments – even under background knowledge in the form of data dependencies – such that afterwards in the second stage for any actual relation instance a strong kind of (possibilistic) *semantic confidentiality* will always be achieved (leaving open the problem of probabilistic inference-proofness).

Exploiting the basic features of this approach, namely instance-independent *mutual isolation of templates* (and refusals) in the first stage and *individual* *treatment of tuples* in the second stage, we showed how confidentiality-preserving updating of views is possible while complying with an extended notion of *continuous inference-proofness*. Essentially, this goal can be achieved by conceptually rerunning the two stages of the underlying view generation procedure, provided some precautions are enforced: modification requests have to be formed as *transactions*, in general possibly dealing with modifications of both the relation instance and the security configuration, and *invariants* to be maintained by transaction processing should refer to committed *internal states* of the overall system of the underlying relational database. Due to simplification avoiding asynchronous time, we always made transactions explicit, though essentially the same confidentiality guarantees can be obtained by *completely hiding* rejected, non-committed and non-effective modification requests (see condition C5*).

The updating procedure preserves the *practical efficiency* of the underlying view generation procedure, again due to the basic features summarized above, and as confirmed by runtime experiments with a *prototype implementation*. Moreover, the updating procedure also preserves and extends the *availability* properties of the underlying procedure. As discussed in [4,14], the latter one minimally distorts data only if *locally necessary* under the given setting, and the introductory examples *necessitate some restrictions*, and motivate the concrete ones expressed by conditions C1–C4. However, *global optimization* is likely to be related to NP-hardness and thus would be in conflict with efficiency.

So far, we only deal with a *single* relation governed by single-premise tuple-generating dependencies rather than with a *multi-relational* database with any intrarelational and interrelational constraints. Though any attempt towards the latter goal would be highly worthwhile to enhance practicality, it will always face substantial limitations regarding efficiency or even computability.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Anjum, A., Raschia, G., Gelgon, M., Khan, A., Malik, S.U.R., Ahmad, N., Ahmed, M., Suhail, S., Alam, M.M.: τ-safety: a privacy model for sequential publication with arbitrary updates. Comput. Secur. **66**, 20–39 (2017)
3. Biskup, J., Gogolin, C., Seiler, J., Weibert, T.: Inference-proof view update transactions with forwarded refreshments. J. Comput. Secur. **19**, 487–529 (2011)
4. Biskup, J., Preuß, M.: Information control by policy-based relational weakening templates. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 361–381. Springer, Cham (2016). doi:10.1007/978-3-319-45741-3_19
5. Biskup, J., Tadros, C.: Inference-proof view update transactions with minimal refusals. In: Garcia-Alfaro, J., Navarro-Arribas, G., Cuppens-Boulahia, N., de Capitani di Vimercati, S. (eds.) DPM/SETOP -2011. LNCS, vol. 7122, pp. 104–121. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28879-1_8
6. Biskup, J., Tadros, C.: Preserving confidentiality while reacting on iterated queries and belief revisions. Ann. Math. Artif. Intell. **73**(1–2), 75–123 (2015)

7. Byun, J., Li, T., Bertino, E., Li, N., Sohn, Y.: Privacy-preserving incremental data dissemination. J. Comput. Secur. **17**(1), 43–68 (2009)
8. Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: a survey of recent developments. ACM Comput. Surv. **42**(4), 14:1–14:53 (2010)
9. Halpern, J.Y., O'Neill, K.R.: Secrecy in multiagent systems. ACM Trans. Inf. Syst. Secur. **12**(1), 5.1–5.47 (2008)
10. Levesque, H.J., Lakemeyer, G.: The Logic of Knowledge Bases. MIT Press, Cambridge (2000)
11. Li, J., Baig, M.M., Sattar, A.H.M.S., Ding, X., Liu, J., Vincent, M.W.: A hybrid approach to prevent composition attacks for independent data releases. Inf. Sci. **367–368**, 324–336 (2016)
12. A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. ℓ-diversity: privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data **1**(1) (2007). Article 3
13. Nerode, A., Shore, R.: Logic for Applications, 2nd edn. Springer, Heidelberg (1997)
14. Preuß, M.: Inference-proof materialized views. Ph.D. thesis, Dortmund University of Technology, Germany (2016)
15. Samarati, P.: Protecting respondents' identities in microdata release. IEEE Trans. Knowl. Data Eng. **13**(6), 1010–1027 (2001)
16. Shmueli, E., Tassa, T.: Privacy by diversity in sequential releases of databases. Inf. Sci. **298**, 344–372 (2015)
17. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. Uncertainty Fuzziness Knowl.-Based Syst. **10**(5), 557–570 (2002)
18. Wang, K., Fung, B.C.M.: Anonymizing sequential releases. In: Eliassi-Rad, T., Ungar, L.H., Craven, M., Gunopulos, D. (eds.) Knowledge Discovery and Data Mining, KDD 2006, pp. 414–423. ACM (2006)
19. Xiao, X., Tao, Y.: M-invariance: towards privacy preserving re-publication of dynamic datasets. In: Chan, C.Y., Ooi, B.C., Zhou, A. (eds.) Management of Data, SIGMOD 2007, pp. 689–700. ACM (2007)
20. Yao, C., Wang, X.S., Jajodia, S.: Checking for k-anonymity violation by views. In: Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.-Å., Ooi, B.C. (eds.) Very Large Data Bases, VLDB 2005, pp. 910–921. ACM (2005)