Using SAT-Solvers to Compute Inference-Proof Database Instances

Cornelia Tadros and Lena Wiese

Technische Universität Dortmund, 44221 Dortmund, Germany
{tadros,wiese}@ls6.cs.uni-dortmund.de
http://ls6-www.cs.tu-dortmund.de/issi/

Abstract. An inference-proof database instance is a published, secure view of an input instance containing secret information with respect to a security policy and a user profile. In this paper, we show how the problem of generating an inference-proof database instance can be represented by the partial maximum satisfiability problem. We present a prototypical implementation that relies on highly efficient SAT-solving technology and study its performance in a number of test cases.

1 Introduction and System Settings

Controlled Query Evaluation (CQE) is a framework for inference control in logical database systems. In [3] a preprocessing procedure (which we call *pre*CQE here) is described that accepts propositional input (an input instance, a confidentiality policy, an availability policy and a user profile). It outputs an "inference-proof" solution instance; this output instance is secure in the sense that it can be published to provide answers to any user queries without enabling the user to deduce any confidential information – and without the need to maintain a history of previous user queries. As secondary and tertiary goals, the output instance is meant to preserve maximum availability (of entries in the availability policy) as well as minimize the amount of modifications ("distortions") with respect to the input instance. The aims of this article are twofold:

1. We show that precomputing an inference-proof, availability-preserving, and distortion-minimal database instance can be reduced to a weighted partial MAXSAT (W-PMSAT) problem with three weights.
2. We present and evaluate a prototypical implementation where highly efficient third-party SAT solving tools can be plugged in – instead of implementing the algorithm (as theoretically exposed in [3]) directly.

Our preprocessing approach stands orthogonal to history-based inference control mechanisms in logical databases (as for example in [2,11]) that compute the (possibly distorted) answers at runtime. Yet, it is akin to the use of cover stories (see [8,5]) in multilevel secure databases while adding the bonuses of availability preservation and distortion minimization.

We now describe components (visualized in Figure 1) and settings of the CQE system that are assumed in this article. The system is based on a propositional

J. Garcia-Alfaro et al. (Eds.): DPM 2009 and SETOP 2009, LNCS 5939, pp. 65–77, 2010.

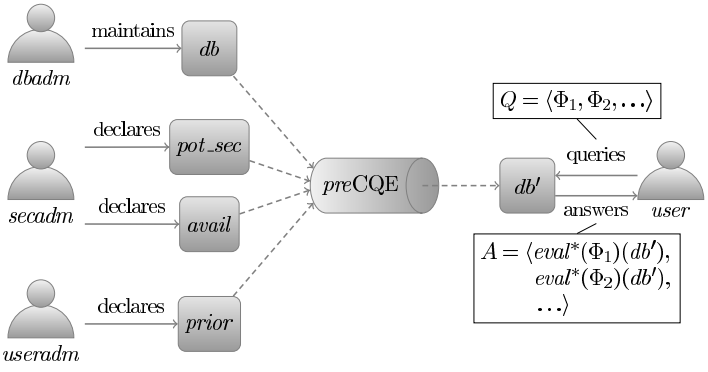

Fig. 1. Concept of the algorithm

language with an infinite number of propositional variables (the propositional "alphabet" $\mathcal{P}$). For our running example, $\mathcal{P}$ is the vocabulary for a medical record with diseases and medications:

$$\mathcal{P} = \{\texttt{cancer}, \texttt{aids}, \texttt{flu}, \texttt{cough}, \ldots, \texttt{medA}, \texttt{medB}, \texttt{medC}, \ldots\}$$

Propositional formulas are built from $\mathcal{P}$ with the connectives $\wedge$, $\vee$ and $\neg$. A propositional variable is also called a "positive literal"; a propositional variable preceded by a negation sign is called a "negative literal".

Data Model. The input database instance *db* is a finite set of propositional variables (where each variable represents a tuple in the database); hence $db \subset \mathcal{P}$. It represents a complete interpretation I^{db} for all variables in $\mathcal{P}$: a variable $A \in db$ is interpreted as *true*, otherwise it is interpreted as *false*. In our example, $db = \{\texttt{cancer}, \texttt{aids}, \texttt{medA}, \texttt{medB}\}$ comprises the set of all *true* propositions, while all other variables (from $\mathcal{P} \setminus db$) are *false*. The input instance is maintained by the database administrator *dbadm*.

Interaction Model. A user is assumed to interact with a database instance via an evaluation function $eval^*$; it takes a query formula and a database instance as inputs and returns the query formula or its negation depending on which of the two formulas is true in the instance:

$$eval^*(\Phi)(db) = \begin{cases} \Phi & \text{if } I^{db} \models \Phi \quad (\text{with } \models \text{ being the model operator}) \\ \neg\Phi & \text{else} \end{cases}$$

Eg., $eval^*(\texttt{flu})(db) = \neg\texttt{flu}$ and $eval^*(\texttt{medB} \wedge \texttt{medA})(db) = \texttt{medB} \wedge \texttt{medA}$.

Confidentiality Model. The confidentiality policy *pot_sec* is a finite set of formulas. An entry of *pot_sec* is a "potential secret": the user must not know that a potential secret is *true* in *db*, but he may assume that it is *false*. The confidentiality policy is declared by the security administrator *secadm*, for example as $pot\_sec = \{\texttt{cancer}, \texttt{aids}\}$: the user may not know the fact `cancer` (or `aids`),

but he may learn ¬cancer (and ¬aids). As a rational, sophisticated person, the user is assumed to know the policy specification *pot_sec*. The only protection mechanism analyzed in this article is modification of some *db*-entries; hence a solution instance may contain the entry flu. This is called "uniform lying" in the CQE context.

Availability Model. The availability policy *avail* is a finite set of formulas. It specifies important information, that should at best not be distorted by the lying mechanism. That is, whenever it is possible to distort information not contained in *avail* (while still protecting the secrets), we prefer this distortion to a distortion affecting *avail* entries. The availability policy is also declared by the security administrator *secadm*, for example as $avail = \{\texttt{medA} \wedge \texttt{medB}, \texttt{medB}\}$ stating that the information whether *both* medA and medB or medB *alone* are prescribed should not be distorted (due to side effects or mutual reactions with substances that must be considered). Beyond this explicit goal to preserve availability, there is a tertiary goal to distort as few database entries as possible.

User Model. The user profile *prior* is a finite set of formulas containing a specification of the knowledge the user had prior to interacting with the CQE system. The user profile is declared by the user administrator *useradm*; eg., the user knows that a patient taking medicine A is ill with Aids or Cancer, and a patient taking medicine B is ill with Cancer or Flu: $prior = \{\neg\texttt{medA} \vee \texttt{cancer} \vee \texttt{aids}, \neg\texttt{medB} \vee \texttt{cancer} \vee \texttt{flu}\}$. He is able to use full implication on his knowledge and the database answers to deduce other facts from them. Hence, the user knowledge and the database answer must never be inconsistent as from inconsistent knowledge the user can deduce any facts (including the secrets): from a contradiction, anything follows by logical implication. Beyond the mere representation of the user knowledge in the user profile, the user is assumed to be aware of the system settings (that is, complete database, known policy, lying).

Execution Model. The *pre*CQE procedure takes *db*, *pot_sec*, *avail* and *prior* as inputs and outputs a complete instance *db′*. The output instance *db′* has the property that it is consistent with the a priori knowledge *prior* and that no truthful answer to any user query enables the user to infer a potential secret from *pot_sec*. More formally, in the case of a complete *db′* and lying, we define "inference-proofness" of *db′* as follows:

Definition 1 (Inference-proofness). *A complete database instance db′ is called* inference-proof *(with respect to prior and pot_sec) iff*

1. $I^{db'} \models prior$
2. $I^{db'} \not\models \Psi$ *for every* $\Psi \in pot\_sec$

A user (modeled by *prior*) can pose any query sequence $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$ and retrieve truthful responses $A = \langle eval^*(\Phi_1)(db'), eval^*(\Phi_2)(db'), \ldots \rangle$ from an inference-proof instance without being able to deduce a secret. Eg., neither $db'_1 = \{\texttt{medB}, \texttt{flu}\}$ nor $db'_2 = \emptyset$ disclose any of the secrets aids and cancer but they obey *prior* – hence both are inference-proof.

In [3] it is shown that with these system settings, the problem of finding an inference-proof instance db' amounts to finding a model (a satisfying interpretation) $I^{db'}$ for a constraint set C. This set C consists of the user profile and the negations of the potential secrets (a condition for consistency of C – and hence existence of a db'– is identified in [3]):

Definition 2 (Constraint set). *For a set prior and a set pot_sec, the* constraint set *is*

$$C := prior \cup Neg(pot\_sec) \qquad where\ Neg(pot\_sec) := \{\neg\Psi \mid \Psi \in pot\_sec\}$$

Hence, in our example the constraint set C is:

$$C := \{\neg\texttt{medA} \vee \texttt{cancer} \vee \texttt{aids}, \neg\texttt{medB} \vee \texttt{cancer} \vee \texttt{flu}, \neg\texttt{cancer}, \neg\texttt{aids}\}$$

and it holds that $I^{db'_1} \models C$ as well as $I^{db'_2} \models C$.

To meet the availability requirements and thus retain as much correct information in db' as possible, we define two distance measures: the first one to measure how many *avail* entries are affected by distortion and the second one to measure how many *db* entries are affected by distortion:

Definition 3 (Availability preservation/distortion minimization). *The* availability distance *(for inference-proof db') is defined as*

$$avail\_dist(db') := ||\{\Theta \in avail \mid eval^*(\Theta)(db') \neq eval^*(\Theta)(db)\}||$$

An inference-proof db' is availability-preserving *iff there is no db'' such that $avail\_dist(db') > avail\_dist(db'')$.*

The distortion distance *(for inf.-proof and availability-preserving db') is*

$$db\_dist(db') := ||\{A \in \mathcal{P} \mid eval^*(A)(db') \neq eval^*(A)(db)\}||$$

An inference-proof and availability-preserving db' is distortion-minimal *iff there is no db'' such that $db\_dist(db') > db\_dist(db'')$.*

We first of all minimize *avail_dist* and among the *avail_dist*-minimal solutions search for one that minimizes *db_dist*. Yet, due to the model requirement, inference-proofness and hence confidentiality of the secrets is our main goal and the two distances are availability optimization functions. In our example, we see that db'_1 preserves availability better than db'_2: while in the input instance *db* both entries of *avail* are *true*, in db'_1 the first entry is *false* but the second is *true*, such that $avail\_dist(db'_1) = 1$; in db'_2 both entries are *false*, such that $avail\_dist(db'_2) = 2$. Hence, db'_1 is our unique optimal solution (and distortion minimality has no effect).

A crucial point for the efficiency of the *pre*CQE algorithm is that only a finite subset of the infinite $\mathcal{P}$ of "decision variables" that are contained in *prior*, *pot_sec* or *avail* have to be considered when searching for an inference-proof, availability-preserving and distortion-minimal solution:

Definition 4 (Decision variables). *The* decision variables *are*

$$\mathcal{P}_{decision} := \{A \in \mathcal{P} \mid A\ occurs\ in\ prior, pot\_sec\ or\ avail\}$$

In our example, $\mathcal{P}_{decision} = \{\texttt{cancer}, \texttt{aids}, \texttt{flu}, \texttt{medA}, \texttt{medB}\}$.

2 Encoding as SAT Problem

*pre*CQE for propositional logic can be represented (by a transformation of the input constraints) as a variant of an optimization problem for the satisfiability (SAT) problem; in this case (as opposed to the Branch and Bound approach in [3]) the availability and distortion distances need not be maintained explicitly but are encoded into "weights". However, SAT solving normally refers to input formulas in conjunctive normal form (CNF) such that all *pre*CQE input formulas have to be converted into an equivalent set of "clauses" (a clause is a disjunction of literals).

In the following we present the representation of the *pre*CQE problem as a weighted partial MAXSAT (W-PMSAT) optimization problem. Here it is crucial to see the input as a set of clauses. Each clause has an associated non-negative integer as a weight. We use three weights: the highest one to account for inference-proofness (and hence confidentiality-preservation) for the so called "hard constraints", an intermediate one to account for availability preservation, and the lowest weight 1 for distortion minimization. The W-PMSAT optimization function is to maximize the sum of weights of satisfied clauses in an interpretation (or, equivalently, minimize the sum of weights of unsatisfied clauses). Hard constraints necessarily have to be satisfied; that is why the optimization is partial: the W-PMSAT solver only has to maximize the summed weight of the remaining satisfied "soft constraints". Our three weights are computed such that if all clauses with a lower weight are satisfied at the cost of *not* satisfying a clause with a higher weight, the summed total weight is lower and hence the solution is worse: this nicely encodes the fact that inference-proofness is our main, availability preservation our secondary, and distortion minimization our tertiary goal.

2.1 Clauses and Weights

The *pre*CQE inputs db, $avail$ and the constraint set C (see Def. 2) are transformed into three sets of clauses: one set C_1 of soft constraints containing all clauses with lowest weight 1, a second set C_2 of ("auxiliary") soft constraints with an intermediate weight and a third set C_3 of hard constraints with highest weight. At first, all decision variables are transformed to soft constraints according to their evaluation in db. That is:

$$C_1 := eval^*(\mathcal{P}_{decision})(db) := \bigcup_{A \in \mathcal{P}_{decision}} eval^*(A)(db)$$

is the set of soft constraints that all have weight 1; in our example, $C_1 = \{\texttt{cancer}, \texttt{aids}, \neg\texttt{flu}, \texttt{medA}, \texttt{medB}\}$ (recall that $eval^*(\texttt{flu})(db) = \neg\texttt{flu}$).

Second, an intermediate weight has to be determined when considering the formulas in $avail$. Recall that the semantics of the availability policy is that only a maximal number but possibly not all of the formulas in $eval^*(avail)(db)$ can be satisfied in the solution instance db' (we use $eval^*(avail)(db)$ as an abbreviation for $\bigcup_{\Phi \in avail} eval^*(\Phi)(db)$). This optimization requirement leads to the problem

of loss of structural information when transforming formulas in $eval^*(avail)(db)$ into CNF: If we take a formula Θ from $eval^*(avail)(db)$ and determine its CNF representation $cnf(\Theta)$ (in order to be processable by a W-PMSAT solver), all the clauses of $cnf(\Theta)$ have to be treated as "belonging together" when counting their weight. We can achieve this with the help of auxiliary propositional variables denoted S_Θ. The second set C_2 of auxiliary soft constraints consists exactly of the auxiliary variables: for each S_Θ, we add a clause S_Θ with weight $card(C_1) + 1$ to C_2. In our example, the second set of auxiliary constraints with weight $card(C_1) + 1 = 6$ is $C_2 = \{S_{\texttt{medA}\wedge\texttt{medB}}, S_{\texttt{medB}}\}$.

Next, for a formula Θ in $eval^*(avail)(db)$, $cnf(\Theta)$ is transformed as:

1. To each clause c of $cnf(\Theta)$ conjoin $\neg S_\Theta$ which gives us $c \vee \neg S_\Theta$
2. Add these augmented clauses to the constraint set C_3

Finally, for each constraint formula $\Phi \in C$, add the clauses of $cnf(\Phi)$ to C_3. All the clauses in the constraint set C_3 have as weight the sum of the weights of all the constraints at lower levels plus 1: $card(C_2) \cdot (card(C_1) + 1) + card(C_1) + 1$.

In our example, the set of hard constraints with weight $card(C_2) \cdot (card(C_1) + 1) + card(C_1) + 1 = 18$ is:

$$C_3 := \{\neg\texttt{medA} \vee \texttt{cancer} \vee \texttt{aids}, \neg\texttt{medB} \vee \texttt{cancer} \vee \texttt{flu}, \neg\texttt{cancer}, \neg\texttt{aids},$$
$$\texttt{medA} \vee \neg S_{\texttt{medA}\wedge\texttt{medB}}, \texttt{medB} \vee \neg S_{\texttt{medA}\wedge\texttt{medB}}, \texttt{medB} \vee \neg S_{\texttt{medB}}\}$$

2.2 Solution Instance

We can show that a solution of this W-PMSAT input represents an inference-proof, availability-preserving and distortion-minimal propositional solution instance for the *pre*CQE input.

Proposition 1. *Let I^* be a solution of the W-PMSAT input, specified in Section 2.1, and db' the solution instance as obtained by*

$$db' := \{A \mid A \in db, A \notin \mathcal{P}_{decision}\} \cup \{A \mid A \in \mathcal{P}_{decision} \text{ with } I^* \models A\}.$$

Then db' is inference-proof, availability-preserving and distortion-minimal in the sense of Definition 1 and Definition 3.

We sketch the proof in the following: All hard constraints in C_3 must be satisfied in I^*, in particular the constraint set C from Definition 2 and thus db' is **inference-proof**. As for **availability preservation**, assume that $\tilde{db}$ is an inference-proof instance with better availability distance than db'. The interpretation $I^{\tilde{db}}$ over $\mathcal{P}$ can be extended to an interpretation $\tilde{I}$ over the variables $\mathcal{P} \cup \{S_\Theta \mid \Theta \in avail\}$ by setting S_Θ to true whenever $eval^*(\Theta)(\tilde{db}) = eval^*(\Theta)(db)$ and to false otherwise. (By the choice of the values of all S_Θ and the inference-proofness of $\tilde{db}$ all hard constraints C_3 are satisfied in $\tilde{I}$.) The total weight of all satisfied soft constraints $C_1 \cup C_2$ given $\tilde{I}$ is greater than the sum of weights of all satisfied clauses $S_\Theta \in C_2$, which amounts to

$$
\begin{aligned}
&(card(avail) - avail\_dist(\tilde{db})) \cdot (card(\mathcal{P}_{decision}) + 1) \\
\geq\ &(card(avail) - (avail\_dist(db') - 1)) \cdot (card(\mathcal{P}_{decision}) + 1) \\
>\ &(card(avail) - avail\_dist(db')) \cdot (card(\mathcal{P}_{decision}) + 1) + card(\mathcal{P}_{decision})
\end{aligned}
$$

As we can achieve at most that S_Θ is satisfied iff $cnf(\Theta)$ is satisfied, the value $(card(avail) - avail\_dist(db')) \cdot (card(\mathcal{P}_{decision}) + 1)$ is an upper bound to the sum of weights of all satisfied clauses in C_2 given I^*. Further, $card(\mathcal{P}_{decision})$ is an upper bound to the sum of weights of all satisfied clauses in C_1 given I^*. Hence, following the inequalities above, $\tilde{I}$ is more optimal then I^*, which is a contradiction to the optimality of I^*. Lastly, the sum of weights of unsatisfied clauses from C_1 is equal to db_dist, hence the instance db' is **distortion-minimal**. □

3 A *pre*CQE Implementation for Propositional Logic

In recent years, propositional SAT solving has seen a huge improvement in performance. Several highly efficient implementations take part in the yearly SAT competition (in conjunction with the SAT conference). As part of the SAT competition there also is a "MAXSAT evaluation" [6,1] that includes competition categories for W-PMSAT problems. Those SAT solvers often employ a Branch and Bound strategy for propositional input (similar to the one described in [3]) and beyond that implement highly efficient heuristics to speed up the search. While the SAT competition is already quite established, the MAXSAT evaluation has been organized just for the fourth time in 2009. This shows that the interest in efficient solving strategies for this optimization problem has come up very recently.

We wanted to apply this highly efficient W-PMSAT technology to our problem and benefit from up-to-date solver implementations instead of implementing our approach in [3] by hand; we developed a program that translates propositional *pre*CQE input formulas into a W-PMSAT instance. In particular, the program offers the following functionality:

1. It offers a graphical interface for the specification of the input (db, pot_sec, $avail$ and $prior$) and the presentation of the solution db'.
2. It transforms the specified input into a W-PMSAT instance by converting the input into CNF, creating the auxiliary constraints and computing the weights.
3. It transforms this input into the input format of the selected solver.
4. It calls the selected solver on this instance (in W-PMSAT encoding).
5. It measures the runtime of the whole computation as well as the runtime for the solver alone.
6. It transforms the solver output into the solution instance db'.

As the input format we chose the TPTP format for first-order formulas (see [10]) as we plan to extend our work to relational databases. It is a standard format for Automated Theorem Proving and is much more convenient to use than the propositional SAT solver input formats (e.g. DIMACS; see the rules of [1]): while

with DIMACS variables are encoded by numbers, TPTP variables can be any user-defined strings. This is a great advantage because our administrators specify their input in TPTP. The SAT solvers we chose are all able to process the DIMACS format such that the *pre*CQE input is converted into this format by calling the external TPTP conversion library; the mapping from TPTP variables to propositional DIMACS variables is recorded on this occasion. In a separate step, *pre*CQE creates the necessary auxiliary constraints. Afterward, *pre*CQE calculates the weights of the W-PMSAT clauses and sets the weight for each clause as described above. With this step, the CQE input has been fully transformed into a W-PMSAT instance. On this instance, an external W-PMSAT solver is run to find an optimal solution; the runtime of the solver is internally recorded. *pre*CQE uses the mapping information between TPTP formulas and DIMACS variables to translate the SAT solver solution into a *pre*CQE output instance db'.

Our program has been tested with three W-PMSAT solvers:

- MiniMaxSAT (see [7])
- MAX-DPLL (as part of the SAT solver Toolbar; see [9])
- SAT4J (`http://www.sat4j.org/`)

MiniMaxSAT was run on a Linux system while we executed Toolbar on a Solaris platform. SAT4J is written purely in Java. With our system settings, MiniMaxSAT showed the best runtime performance; hence the test runs described in the upcoming section were all done with MiniMaxSAT.

3.1 Test Cases

To test our prototype we made an effort to simulate problems specific to databases. Tests were run with differently sized inputs and for every input size we tested 10 random permutations to avoid a bias caused by the input order. The runtime graphs below show the average runtime taken from all 10 instances per size as well as the deviation of the individual running times (in seconds for better readability); the runtime tables detail the number of decision variables and clauses for each input size as well as the running times in milliseconds (msec). The number of decision variables and clauses are decisive values when comparing the performance.

The first tests are a generalization of our running example: We identified 24 combinations of medicines and diseases (the "patient types") that are consistent with the a priori knowledge *prior* and hence permitted in *db*. They are listed in Table 1. We used the abbreviations `N1` to `N24` to denote 24 different patient names. Then we (in the role of the *dbadm*) entered a propositional input instance *db* that contains each patient type exactly once; that is, if the *db* contains the entry[1] `'n1_aids'`, it means that patient `N1` suffers from Aids. Note that there are 66 propositional variables in the propositional *db*. Next, the potential secrets

[1] Actually, the exact TPTP syntax is `fof(r0,axiom,'n1_aids').`; we only state the relevant part here.

Table 1. Permissible patient types in *db*

n1_aids	n2_cancer	n3_flu	n4_aids, n4_cancer	n5_aids, n5_flu	n6_cancer, n6_flu
n7_aids, n7_cancer, n7_flu	n8_medA, n8_aids	n9_medA, n9_cancer	n10_medA, n10_aids, n10_cancer	n11_medA, n11_aids, n11_flu	n12_medA, n12_cancer, n12_flu
n13_medA, n13_aids, n13_cancer, n13_flu	n14_medB, n14_cancer	n15_medB, n15_flu	n16_medB, n16_aids, n16_cancer	n17_medB, n17_aids, n17_flu	n18_medB, n18_cancer, n18_flu
n19_medB, n19_aids, n19_cancer, n19_flu	n20_medA, n20_medB, n20_cancer	n21_medA, n21_medB, n21_aids, n21_cancer	n22_medA, n22_medB, n22_aids, n22_flu	n23_medA, n23_medB, n23_cancer, n23_flu	n24_medA, n24_medB, n24_aids, n24_cancer, n24_flu

and the a priori knowledge are entered (in the roles of *secadm* and *useradm*) in TPTP syntax for each of the 24 patient names as propositional formulas. For N1 the set *prior* contains[2] `'n1_medA'=>('n1_aids'|'n1_cancer')` and `'n1_medB'=>('n1_cancer'|'n1_flu')` and the set *pot_sec* contains `'n1_aids'` as well as `'n1_cancer'`. These entries are entered for all 24 patients; that is, we have 48 entries in *prior*, and 48 entries in *pot_sec*, too. In the first test, we did not use an explicit availability policy; that is, $avail = \emptyset$. As mentioned previously, all input is permuted at random to make tests independent of the order of input.

As for the weights, they are calculated for this example as follows: all the $24 \cdot 5 = 120$ decision variables are transformed into soft constraints receiving the weight 1. As there is no availability policy, there is no need for auxiliary constraints. All constraint formulas in C receive the weight 121. For this simplest input, a solution was found in milliseconds.

Obviously, we are interested in more meaningful results for databases with much more entries. The general idea for the expansion of our tests was to uniformly repeat the 24 patient types and test up to what number of repetitions a moderate runtime performance can be achieved. So, our first step was to repeat each patient type 10 times (each repetition with a new name) such that we have a *db* with 660 entries, *prior* with 480 entries and *pot_sec* with 480 entries; for 10 repetitions there are hence $24 \cdot 5 \cdot 10 = 1200$ decision variables. We ran tests up to 150 repetitions with 9900 *db* entries, 7200 *prior* and *pot_sec* entries each and 18000 decision variables. Figure 2 shows the results; what can be seen is that a huge amount of time is needed for the creation of the DIMACS input – this includes the creation of *Neg*(*pot_sec*) and the auxiliary constraints, the calculation and assignment of weights as well as the TPTP conversion, – whereas the MiniMaxSAT solver appears quite unimpressed by the increased size of the input. We

[2] Full TPTP syntax is `fof(r0,axiom,'n1_medA'=>('n1_aids'|'n1_cancer')).` and `fof(r1,axiom,'n1_medB'=>('n1_cancer'|'n1_flu')).`

	total runtime (msec)			solver runtime			dec.	clauses	
rep.	min	max	avg.	min	max	avg.	vars.	soft	hard
1	1832	2175	1930	178	208	184	120	120	96
25	10981	12246	11974	2214	3206	3092	3000	3000	2400
50	29333	32149	31304	4412	6360	6135	6000	6000	4800
75	58530	62026	60459	6503	9439	8991	9000	9000	7200
100	93275	101551	95792	8803	9001	8902	12000	12000	9600
125	139835	150095	142843	11000	11472	11171	15000	15000	12000
150	197389	206099	202067	13231	18253	16429	18000	18000	16800

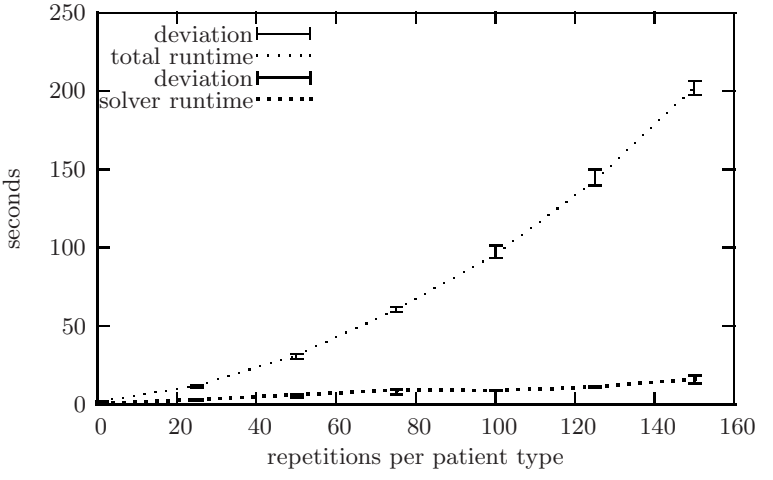

Fig. 2. Performance of *pre*CQE for 24 patient types

made two more test runs with different patient types without availability policy: a thorough analysis of the patient types reveals that there are four patient types with multiple optimal solutions. We separated them from the remaining 20 patient types with unique solution and tested the two sets separately. The existence of multiple optima slowed down the SAT solver only slightly.

After these promising results, we introduced an explicit availability policy; that is, we supplied a set *avail* with two entries for each patient type: $avail = \{\texttt{n1\_medA}, \texttt{n1\_medB}, \texttt{n2\_medA}, \texttt{n2\_medB}, \dots\}$. In the *pre*CQE implementation, they are first of all evaluated according to *db* (that is, $eval^*(avail)(db)$ is computed). As described in Section 2.1, the resulting formulas are transformed into hard constraints and auxiliary constraints with auxiliary propositional variables. In the simplest case with one repetition per patient type we thus have 120 decision variables with lowest weight 1. As there are 48 formulas in *avail*, we have 48 auxiliary constraints with weight 121. Finally there are $48 + 48 + 48 = 144$ hard constraints with weight $(48 \cdot 121) + 120 + 1 = 5929$. That is, when satisfying all hard constraints, the solution has a weight of at least 853776. We experienced problems with these high weight values, because after 55 repetitions of patient types, we faced an integer overflow: the computed solution had a negative weight. To avoid this, we then examined the performance of a reduced set of patient

types. We removed the patient types with `medA`-entries, such that the first *prior* constraint will never be violated. We kept 13 patient types: N1, N2, N3, N4, N5, N6, N7, N14, N15, N16, N17, N18, N19 and their corresponding entries in *db*, *prior*, *pot_sec* and *avail*. The results can be found in Figure 3. We were able to repeat these 13 patient types much more often (up to 10150 *db* entries) than the full 24 patient type set; that is, only the search with the full set led to the integer overflow, while for the reduced set this was not the case. In comparison to tests without availability policy, runtimes increased only little (comparing the results for similar amounts of decision variables).

Lastly, we made a test with the full set of 24 patient types but we changed the potential secrets into a conjunctive format:

$$pot\_sec = \{\texttt{n1\_aids} \wedge \texttt{n1\_cancer}, \texttt{n2\_aids} \wedge \texttt{n2\_cancer}, \dots\}$$

This means that for every patient it is allowed to know if the patient has either aids or cancer but it is not allowed to know that a patient has both aids and

	total runtime (msec)			solver runtime			dec.	clauses		
rep.	min	max	avg.	min	max	avg.	vars.	low	aux.	hard
1	1744	2102	1841	142	162	146	65	65	26	78
25	8068	8308	8166	2051	2118	2077	1625	1625	650	1950
50	19028	20650	19423	4009	4121	4076	3250	3250	1300	3900
75	35061	37300	35706	5981	6266	6132	4875	4875	1950	5850
100	54295	63201	57153	5712	8375	8002	6500	6500	2600	7800
150	107971	117968	113187	8695	12533	12017	9750	9750	3900	11700
200	187700	195847	190946	11601	16924	16131	13000	13000	5200	15600
250	277757	296878	289068	15119	21247	20257	16250	16250	6500	19500
300	397551	425732	407416	18031	26073	24910	19500	19500	7800	23400
350	537890	562223	548090	22343	31778	30152	22750	22750	9100	27300

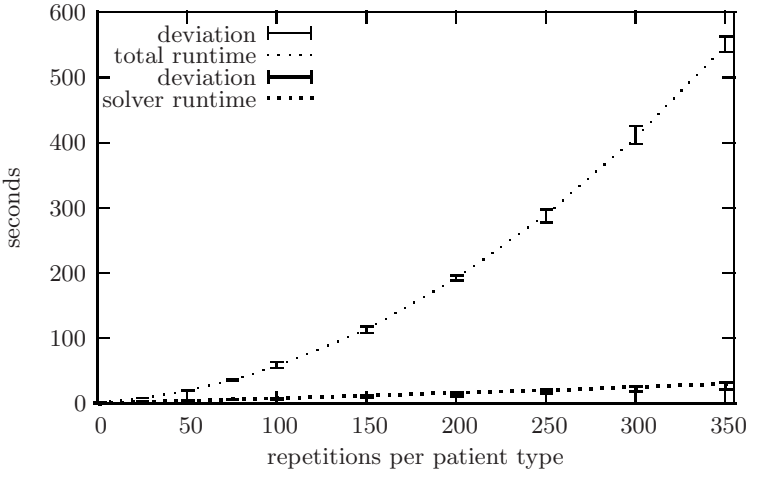

Fig. 3. Performance of *pre*CQE without `medA`-entries

	total runtime (msec)			solver runtime			dec.	clauses		
rep.	min	max	avg.	min	max	avg.	vars.	low	aux.	hard
1	1941	2934	2182	225	337	264	120	120	48	120
25	18283	23445	20439	4630	6530	5362	3000	3000	1200	3000
50	50486	58167	52449	9651	12052	9966	6000	6000	2400	6000
75	101453	105935	103266	15904	16270	16115	9000	9000	3600	9000
100	164611	175434	170920	18185	23374	22623	12000	12000	4800	12000
125	252737	276016	260537	28020	32737	31255	15000	15000	6000	15000
150	351488	380984	367471	32437	40160	39087	18000	18000	7200	18000

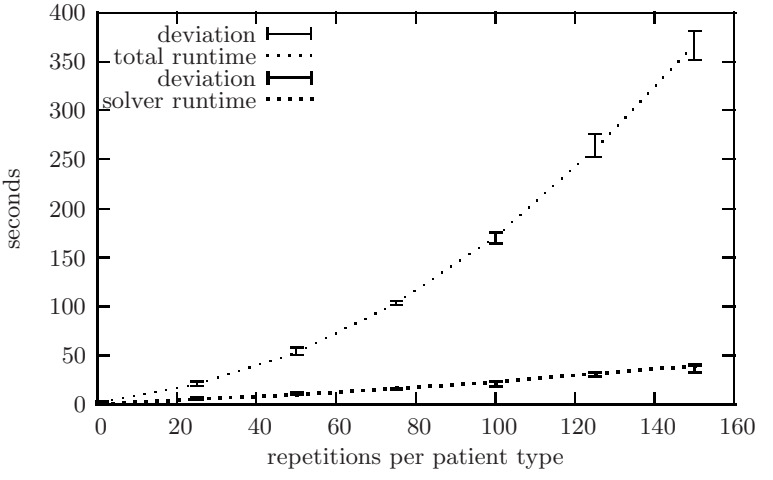

Fig. 4. Performance of *pre*CQE with conjunctive secrets

cancer at the same time. This offers a greater set of possible solutions and the SAT solver is forced to make more decision steps. Yet, as the amount of formulas in *pot_sec* is half of what it was before – only one entry per patient – the number of hard clauses is reduced: for one repetition we have 120 low level constraints, 48 auxiliary constraints and $48 + 48 + 24 = 120$ hard constraints. The results (for up to 9900 *db* entries) are detailed in Figure 4. Again, there is only a slight increase in runtime (compared with respect to the number of decision variables).

The above stated "medical record" tests contained independent subproblems in the sense that for each patient a satisfaction of the constraints could be reached without affecting the entries of other patients. Hence, as a second class of test cases we took a set of "cascading constraints" where the search for a solution requires several splitting and backtracking steps because the clauses share variables and thus are interconnected. Moreover these cascading constraints lead to test formulas with increasing length. We perceived this to be a lot more challenging task for the SAT solvers. The simplest test input with 9 decision variables was *db* = {c1, c2, c3}, *pot_sec* = {c3}, and lastly *prior* = {c3<=>((~v3_1|~v3_2|~v3_3)& c2), c2<=>((~v2_1|~v2_2|~v2_3)& c1)}. Without going into detail, the modification of v-variables is always suboptimal but the solver still searches on them. When comparing the runtime for 18000

decision variables, the runtime for the cascading constraints was only 2 minutes slower than for the medical records. Hence, the overall performance was still favorable.

4 Conclusion

We showed that (and how) in the CQE setting for a complete database with a confidentiality policy of potential secrets and lying as the protection mechanism, the problem of finding an inference-proof, availability-preserving and distortion-minimal database instance can be represented as a W-PMSAT problem. The presented prototype makes use of current SAT solver technology. Two classes of test cases showed that the preprocessing approach is feasible for a large number of database entries. Ongoing work at our department includes a prototypical implementation for relational database systems whose theoretical foundation is described in [4]. A major open question is whether after an update of the database instance or the policy parts of a previous solution can be reused for example with an incremental SAT solver.

References

1. Argelich, J., Li, C.M., Manyà, F., Planes, J.: MaxSAT evaluation, `http://www.maxsat.udl.cat/`
2. Biskup, J., Bonatti, P.A.: Controlled query evaluation with open queries for a decidable relational submodel. Annals of Mathematics and Artificial Intelligence 50(1-2), 39–77 (2007)
3. Biskup, J., Wiese, L.: Preprocessing for controlled query evaluation with availability policy. Journal of Computer Security 16(4), 477–494 (2008)
4. Biskup, J., Wiese, L.: Combining consistency and confidentiality requirements in first-order databases. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.): ISC 2009. LNCS, vol. 5735, pp. 121–134. Springer, Heidelberg (2009)
5. Cuppens, F., Gabillon, A.: Cover story management. Data & Knowledge Engineering 37(2), 177–201 (2001)
6. Heras, F., Larrosa, J., de Givry, S., Schiex, T.: 2006 and 2007 Max-SAT Evaluations: Contributed Instances. Journal on Satisfiability, Boolean Modeling and Computation 4(1), 239–250 (2008)
7. Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSAT: An efficient Weighted Max-SAT Solver. Journal of Artificial Intelligence Research 31, 1–32 (2008)
8. Jukic, N., Nestorov, S., Vrbsky, S.V., Parrish, A.S.: Enhancing database access control by facilitating non-key related cover stories. Journal of Database Management 16(3), 1–20 (2005)
9. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient Max-SAT solving. Artificial Intelligence 172(2-3), 204–233 (2008)
10. Sutcliffe, G.: TPTP, TSTP, CASC, etc. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 6–22. Springer, Heidelberg (2007)
11. Toland, T.S., Farkas, C., Eastman, C.M.: Dynamic disclosure monitor (D^2Mon): An improved query processing solution. In: Jonker, W., Petković, M. (eds.) SDM 2005. LNCS, vol. 3674, pp. 124–142. Springer, Heidelberg (2005)