Security Mechanisms Planning to Enforce Security Policies

Anis Bkakria, Frédéric Cuppens, Nora Cuppens-Boulahia, David Gross-Amblard

Security mechanisms planning to enforce security policies

Anis Bkakria [1], Frédéric Cuppens [1], Nora Cuppens-Boulahia [1], and David Gross-Amblard [2]

Tél 'ecom Bretagne [1]
{anis.bkakria,frederic.cuppens,nora.cuppens}@telecom-bretagne.eu
IRISA,Université de Rennes 1 [2]
david.gross_amblard@irisa.fr

Abstract. This paper presents an approach allowing for a given security and utility requirements, the selection of a combination of mechanisms and the way it will be applied to enforce them. To achieve this goal, we firstly use an expressive formal language to specify the security and utility properties required by data owners and the security mechanisms that can be used to enforce them. Second, we extend and use a Graphplan-based approach to build a planning graph representing all possible transformations of the system resulting from the application of security mechanisms. Finally, we define a method to search the best security mechanisms execution plan to transform the used system from its initial state to a state in which the security requirements are enforced.

1 Introduction

In recent years, the concept of data outsourcing has become quite popular since it offers many features, including reduced costs from saving in storage, increasing availability as well as minimizing management effort. Many security-related research issues associated with data outsourcing have been studied focusing on data confidentiality [2, 12], data authentication and integrity [15, 16], Copyright protection [11], privacy and anonymity [21], because outsourced data often contains highly sensitive information which will be stored and managed by third parties. To tackle those traditional security issues, data protection mechanisms have recently been the focus of huge interest, especially cryptographic and information hiding techniques such as encryption, anonymization, watermarking, fragmentation, etc. These mechanisms are known to be efficient when used independently. However, in many situations they have to be combined to ensure security requirements.

To illustrate, let us take an example in which a company wants to outsource its stored and collected data. Let us suppose that the company considers that a first part of the data (Data I) to be outsourced are sensitive and must be protected, a second part of the data (Data II) is not sensitive but it could be stolen and used by competitors, so the company wants to be able to prove its ownership for the outsourced data. A third part of the data (Data III) could

disclose the identities of the customers of the company, and that it should be able to perform mathematical and statistical operations such as data mining over this part of data. In order to preserve confidentiality of Data I, the company may use encryption. It can also use a watermarking mechanism to embed some robust proof of ownership over Data II, and to anonymize Data III in order to preserve its privacy. As you know, many encryption, watermarking and anonymization mechanisms have been proposed recently. The problem then is to define a reasoning method allowing the company to choose the best mechanisms to enforce its security requirements. Moreover, if we suppose that Data I, Data II, and Data III are intersecting, then there will be a piece of data over which encryption, watermarking, and anonymization will be applied. Therefore, these mechanisms must be combined in an appropriate way to provide the security functionalities without one harming the other.

In this paper, we strive to design an approach allowing for given security requirements, selection of the combination of mechanisms and the way it will be applied (e.g, the order of application of the mechanisms) to enforce security requirements. To meet this goal, we present in section 3 an expressive language [3] to formally express the security requirements to be enforced and the security mechanisms that can be used to enforce the corresponding policy. Section 4 presents the problem we want to address in this paper. In Section 5,we extend and use the Graphplan approach [5] to build a planning graph representing all possible states of the system resulting from the application of security mechanisms. Section 6 presents our reasoning method to search the best security mechanisms execution plan that can transform the target system from its initial state to a state in which the security requirements are enforced. Section 7 presents the implementation and experimental testing results of our approach. Finally, Section 8 reports our conclusions.

2 Related Work

Few research efforts have investigated how to combine security mechanisms to enforce security policies over outsourced data. One of the firsts attempt is proposed in [8], it consists of combining data fragmentation together with encryption to protect outsourced data confidentiality and can only be applied over one-relation databases [1]. Recently, authors of [1, 2] have improved the approach presented in [8] in such a way that it can deal with multi-relation databases. They have also proposed a secure and effective technique for querying data distributed in several service providers and improve the security of their querying technique in order to protect data confidentiality under a collaborative Cloud storage service providers model. Popa et al. [19] and Bkakria et al. [4] have proposed approaches based on adjustable encryption, they combine different encryption schemes to get the best trade-off between data confidentiality and data utility for outsourced relational databases. Boho et al. [6] have proposed interesting approach combining watermarking and encryption to protect both the confidentiality and traceability

[1] Databases composed of only one table

of outsourced multimedia files. All previously cited approaches have three main limitations: First, they are defined in such a way that only two pre-selected security mechanisms can be combined together. Second, they cannot deal with all security properties that can be required by data owners as each approach can provide at most two security properties. Finally, they cannot deal with all data structures that can be used to store outsourced data. In our first attempt to overcome these limitations, we define in [3] an expressive formal language based on epistemic linear temporal logic (Epistemic LTL). This formal language allows as to: (1) formally model the system (e.g., data owner, cloud server, etc.) and the data structure on which the security policy should be enforced, (2) formally express the security policy, and (3) formally specify existing security mechanisms that can be used to protect outsourced data. Then, we have defined a reasoning method for our formal model allowing as to pick up the set of security mechanisms that can enforce each security property requirement by the data owner. However, the reasoning method we proposed in [3], does not take into consideration conflicts that may occur between security mechanisms which makes finding a combination of security mechanisms that satisfy many security requirements hard to fulfill.

3 System specification[3]

We define and use the language $\mathcal{L}$ to formalize our system. In particular, we define formulas describing the basic knowledge of each agent , the formalization of the security and utility requirements to be enforced, and the formalization of the security mechanisms that can be used to enforce the policy. The first-order temporal epistemic language $\mathcal{L}$ is made up of a set of predicates $\mathcal{P}$, propositional connectives $\vee$, $\wedge$, $\neg$, $\rightarrow$ and $\leftrightarrow$, the quantifiers $\forall$, $\exists$. We take the future connectives $\bigcirc$ (next), $\Diamond$ (eventually), $\Box$ (always) [10]. For knowledge we assume a set of agents $A_g = \{1, \cdots, m\}$ and use a set of unary modal connectives K_j, for $j \in A_g$, in which a formula $K_j\psi$ is to be read as "agent j knows ψ".

Definition 1. *Let φ and ψ be propositions and P_i be a predicate of arity n in $\mathcal{P}$. The set of well-formed formulas of $\mathcal{L}$ is defined as follows:*
$\phi ::= P_i(t_1, \cdots, t_n) | \; K_i\psi | \; \neg\varphi | \; \varphi \vee \psi | \; \varphi \wedge \psi | \; \bigcirc\varphi | \; \Diamond\varphi | \; \Box\varphi | \varphi \rightarrow \psi | \; \varphi \leftrightarrow \psi \mid \exists x\psi \mid \forall x\psi$

Definition 2. *An interpretation of the language $\mathcal{L}$ is the triple $\mathcal{K} = (\mathcal{W}, \mathcal{I}, \Phi)$ consisting of a sequence of states $\mathcal{W} = \{w_0, w_1, \cdots\}$, a set of classical first-order structures $\mathcal{I}$ that assigns for each states $w_i \in \mathcal{W}$ a predicate $I_{w_i}(P) : |I_{w_i}|^n \rightarrow \{True, False\}$ for each n-places predicate $P \in \mathcal{P}$ and Φ a transition function which defines transitions between states due to the application of mechanisms (actions). $\Phi(w_i, m_k) = w_j$ if the mechanism m_k transits our system from states w_i to state w_j.*

Definition 3. *Let $\mathcal{W}$ be a sequence of states, w_i $(i \geq 0)$ denote a state of $\mathcal{W}$, and let v be an assignment. The satisfaction relation $\models$ for a formula ψ of $\mathcal{L}$ is defined as follows:*

- $(w_i, \mathcal{W}) \models P(t_1, \cdots, t_n) \iff I_{w_i}(P)(v(t_1), \cdots, v(t_n)) = True$

- $(w_i, \mathcal{W}) \models \neg\psi \iff (w_i, \mathcal{W}) \not\models \psi$
- $(w_i, \mathcal{W}) \models \psi \rightarrow \varphi \iff (w_i, \mathcal{W}) \not\models \psi$ *or* $(w_i, \mathcal{W}) \models \varphi$
- $(w_i, \mathcal{W}) \models \psi \leftrightarrow \varphi \iff (w_i, \mathcal{W}) \models (\psi \rightarrow \varphi) \wedge (\varphi \rightarrow \psi)$
- $(w_i, \mathcal{W}) \models \forall x\psi \iff (w_i, \mathcal{W}) \models \psi[x/c]$ *for all* $c \in |I_{w_i}|$
- $(w_i, \mathcal{W}) \models \psi \wedge \varphi \iff (w_i, \mathcal{W}) \models \psi$ *and* $(w_i, \mathcal{W}) \models \varphi$
- $(w_i, \mathcal{W}) \models \psi \vee \varphi \iff (w_i, \mathcal{W}) \models \psi$ *or* $(w_i, \mathcal{W}) \models \varphi$
- $(w_i, \mathcal{W}) \models \bigcirc\psi \iff (w_{i+1}, \mathcal{W}) \models \psi$
- $(w_i, \mathcal{W}) \models \Diamond\psi \iff (w_k, \mathcal{W}) \models \psi$ *for some* $k \geq i$
- $(w_i, \mathcal{W}) \models \Box\psi \iff (w_k, \mathcal{W}) \models \psi$ *for all* $k \geq i$

3.1 Security policy specification

The security policy to be enforced over the target system are specified through a set of security constraints and a set of utility goals.

Security constraints: Using security constraints, the data owner specifies the security requirements that should be met by the target system. We define five types of security constraints.

• **Confidentiality constraint**: It requires the protection of the confidentiality of an object in the target system.

$$\Box\ [\forall o, \forall e.\ object(o) \wedge sensitive(o) \wedge untrusted(e) \rightarrow \neg K_e\ o]. \tag{1}$$

Formula 1 specifies that in any state of the target system, an untrusted entity e should not knows any sensitive object o.

• **Privacy constraint**: The data owner requires the prevention of identity disclosure.

$$\Box\ [\forall o, \forall e.\ object(o) \wedge identifier(o) \wedge untrusted(e) \rightarrow \neg K_e\ o]. \tag{2}$$

Formula 2 specifies that an object o that can be exploited to identify an identity should not be known by any untrusted entity e in any state of the used system.

• **Traceability constraint (Traitor detection)**: In the applications where a database contents are publicly available over a network, the content owner would like to discourage unauthorized duplication and distribution. For this the owner wants to give to a set of entities E' the ability to check that the content has been released to an authorized user.

$$\begin{aligned} &\Box\ [\forall o, \forall e.\ object(o) \wedge sensitive(o) \wedge untrusted(e) \wedge K_e o \\ &\rightarrow \bigwedge_{e_1 \in E'} K_{e_1}(\exists E_r.\ \bigwedge_{e_r \in E_r} (trusted(e_r) \wedge responsible(e_r, o)))]. \end{aligned} \tag{3}$$

Formula 3 means that in any state of the system, if an untrusted entity knows a sensitive object o, the set of entities E' should be able to know the set of entities E_r responsible of the disclosure of the sensitive object o.

• **Ownership constraint**: The data owner wants to give a set of entities E the ability to verify the ownership of the object o_2.

$$\begin{aligned} \Box\ [\forall o_1. \forall o_2, e\ &object(o_1) \wedge object(o_2) \wedge copy\_of(o_1, o_2) \wedge \\ &\bigwedge_{e_r \in E} K_{e_r} owner(e, o_1) \rightarrow \bigwedge_{e_r \in E} K_{e_r} owner(e, o_2)]. \end{aligned} \tag{4}$$

Formula 4 specifies that in any state of the system, if there are two objects o_1 and o_2 such that o_2 is a copy of o_1 and a set of entities E which know that the

owner of o_1 is e, therefore E should be able to know that o_2 belongs to e.

• **Integrity assessment constraint**: verifying the accuracy and consistency of an object o over its entire life-cycle. This means that data cannot be modified in an undetected manner. In the target system, we should be able to check if o has been modified or not. A data owner may want to give a set of entities E' the ability to check the integrity of an object o.

$$\Box\ [\ object(o) \rightarrow \bigwedge_{e_1 \in E'} K_{e_1}(is\_modified(o) \vee is\_unmodified(o))]. \tag{5}$$

Utility goals: Using utility goals, the data owner can require that some functionalities should be provided. An utility goals is specified using the formula 6 meaning that the utility requirement req is to be provided over the o.

$$utility\_requirement(req) \wedge provides(req, o) \tag{6}$$

3.2 Mechanisms specification

The security mechanisms to be used in the system are specified using preconditions formulas and effects formulas.

Preconditions. For each mechanism, preconditions are represented by a set of formulas which are necessary conditions required for the application of the security mechanism. We define the two-places predicate *is_applicable*. The formula $is\_applicable(m, o)$ is to be read "the mechanism m can be applied over the object o". Preconditions of a security mechanism m are specified using a formula of the following form:

$$\Box\ (is\_applicable(m, o)\ \rightarrow\ \Delta_m) \tag{7}$$

Where Δ_m represents necessary conditions for the applicability of the mechanism m. A formula of the form (7) is to be read "At any state of the system, m can be applied if the preconditions Δ_m hold". **Effects.** Modifications resulting from the application of a mechanism m that transits the system from a state w_i to a state w_j. We use the two-places predicate $apply(m, o)$ to say that the mechanism m is applied over the object o. For a mechanism m, effects are represented by a set of formulas Σ_m such that:

$$\Phi(w_i,\ apply(m, o)) = w_j\ \rightarrow\ (w_j\ \models\ \Sigma_m) \tag{8}$$

Axiom 8 states that if the application of the mechanism m over the object o transits the system from a state w_i to a state w_j, therefore the set of effects Σ_m of the application of the mechanism m is satisfied on the state w_j.

4 Problem statement

We strive to plan a sequence of mechanisms allowing to transform the system from its initial state to a state in which the goals are reached while respecting a set of defined constraints. Planning efficiency can be improved by allowing parallel application of mechanisms, which leads to minimize the number of parallel plan steps. In order to be able to apply mechanisms parallelly, we should

figure out which mechanisms are compatible by finding different kind of conflicts between mechanisms.

Definition 4. ***Conflicting mechanisms:*** *Two mechanisms M_1 and M_2 represented respectively by $(\Delta_{M_1}, \Sigma_{M_1})$ and $(\Delta_{M_2}, \Sigma_{M_2})$ where Δ_{M_i} and Σ_{M_i} represents respectively the specifications of preconditions and the effects of M_i, are effectively incompatible if and only if one of the following deductions hold:*

(i) $\Sigma_{M_1} \cup \Sigma_{M_2} \vdash \bot$
(ii) $\Sigma_{M_i} \cup \Delta_{M_j} \vdash \bot$ *with* $1 \leq i, j \leq 2$ *and* $i \neq j$
(iii) $\Delta_{M_1} \cup \Delta_{M_2} \vdash \bot$

Item (i) means that the effects of M_1 and M_2 are inconsistent. Item (ii) means that the effects of the application of M_i dissatisfy the preconditions of M_j. Item (iii) states that M_1 and M_2 have a competing preconditions such that they cannot be true in the same state.

Definition 5. ***Parallel plan:*** *Consider a set of available mechanisms $\mathcal{M}$. A parallel plan is a finite sequence of sets of mechanisms $\mathcal{P} = \{p_1, \cdots, p_n\}$ such that any $p_i \in \mathcal{P}$, $p_i \subseteq \mathcal{M}$.*

Definition 6. ***Correctness:*** *Given a system $\mathcal{S}$, its current state w_1, a finite set of mechanisms $\mathcal{M}$, a parallel plan $\mathcal{P} = \{p_1, \cdots, p_n\}$ is correct regarding $\mathcal{S}$ and w_1 if and only if the following conditions hold:*

(i) $\exists w_2, \cdots, w_n$ *such that :* $\forall M \in p_i, w_i \models \Delta_M$, $1 \leq i \leq n$.
(ii) $\forall p_i \in \mathcal{P}, \forall M_1, M_2 \in p_i :$ M_1 *and* M_2 *are not conflicting.*

Parallel Planning Problem: Consider a system $\mathcal{S}$, its current state w_1, a set of mechanisms $\mathcal{M}$ that can be applied over the system, a set of goals $\mathcal{G}$ that should be achieved, and a set of constraints $\mathcal{C}$ that should be respected. The Parallel Planning Problem consists on finding a sequence of sets of mechanisms $\mathcal{P} = \{p_1, \cdots, p_n\}$ such that the following conditions hold:

(i) $\mathcal{P}$ is correct regarding $\mathcal{S}$ and w_1.
(ii) $\forall w_i = \Phi(w_{i-1}, p_{i-1}),\ \forall c \in \mathcal{C} :\ w_i \models c$, $2 \leq i \leq n$.

In next part, we briefly introduce the Graphplan's basic operations as defined in [5]. Graphplan uses action schemata in the STRIPS format in witch each action is represented as preconditions and effects which is suitable with the representation of our mechanisms parallel planning problem.

4.1 Graphplan Description

Graphplan is a directed, leveled graph composed of two kinds of nodes and tree kinds of edges. Graphplan levels alternate between *fact levels* containing *fact nodes* (each node is labeled with a predicate), and *action levels* composed of *action nodes* (each labeled with some security mechanism). Relations between actions and predicates in a Graphplan are explicitly represented through edges.

Preconditions-edges are used to connect action nodes of an action level i to their preconditions in the fact level i. *Effects-edges* connect action nodes belonging to the action level i to their effects in the fact level $i+1$. *Mutual-exclusion edges* are relations connecting action nodes belonging to the same Graphplan level. They represent conflicts between action nodes identified according to Definition 4.

Graphplan is based on two main phases: The first is called Graphplan construction phase consisting of growing a planning graph. The second phase allows to extract possible solutions (plans) from the graphplan by performing a backward searching phase starting with the goals. In the graph construction phase, we start with a planning graph having only a single fact level which contains the initial specification of the used system. GraphPlan construction method runs in stages, in each stage i, it extends the planning graph resulting from the stage $i-1$ by adding one time step which contains the next action level and the following fact level. After each stage, Graphplan check if all predicates representing the goals are presented in the last fact level in the planning graph, if it is the case, search a valid plan that transform the system from its initial state to a state in which all the goals are achieved.

4.2 Graphplan modeling of the problem

The STRIPS system [9] used by Graphplan is represented by four lists, a finite set $\mathcal{C}_s$ of ground atomic formulas called conditions, a finite set of operators $\mathcal{O}_s$ where each operator is composed of two formulas (satisfiable conjunction of conditions) representing its preconditions and effects, a finite set of predicates $\mathcal{I}_s$ that denotes the initial state, and a finite set of predicates $\mathcal{G}_s$ that denotes goal state. As we have seen in the previous section, our planning problem is composed of a system $\mathcal{S}$, a set of security mechanisms $\mathcal{M}$, a set of constraints $\mathcal{C}$, and a set of goals $\mathcal{G}$. Obviously, $\mathcal{S}$, $\mathcal{M}$, and $\mathcal{G}$ can be easily modeled as a STRIPS planning problem by expressing $\mathcal{S}$ as $\mathcal{C}_s$ and $\mathcal{I}_s$, $\mathcal{M}$ as $\mathcal{O}_s$, and $\mathcal{G}$ as $\mathcal{G}_s$. According to the section 3.1, $\mathcal{C}$ will be composed of security constraints and utility goals. Utility goals specify the functionalities that should be provided for $\mathcal{S}$ (e.g. the ability to compare the equality of objects). A plan P satisfies a set of utility goals C_u if at its end none of the utility goals in C_u is violated. Consequently, utility constraints will be expressed as goals in the STRIPS planning problem. Security constraints specify the requirements that should be respected during the transformation of $\mathcal{S}$, they can be considered as safety constraint meaning that those requirements are to be satisfied in all states of $\mathcal{S}$. However, the STRIPS language as it is defined in [9] cannot express this kind of constraints. To overcome this limitation, we extend the STRIPS system language by adding the operator *Constraint* allowing to express the set of security constraints. For instance, a confidentiality constraint (rule 1) is to be expressed as follows:

Constraint ($\textbf{confidentiality\_constraint}_1(o, e)$:
Formula: $object(o) \wedge sensitive(o) \wedge untrusted(e) \wedge K_e o)$

In the previous expression, *confidentiality_constraint*$_1$ *(o,e)* is used to denote the name of the constraint and the variables (bounded variables of the rule 1) to be instantiated. The *Formula* field represents the conjunction of predicates

indicating the condition under which the constraint is violated (the negation of CNF representation of the constraint).

5 Extending Graphplan

5.1 Graph construction phase extension

We extend Graphplan's construction method of the planning graph in two ways. The first extension allows to build a planning graph of a planning problem which contains *Domain Axioms* (axioms that formally specify relations between different objects of the system). Second, we improve the Graphplan's construction method of the planning graph to avoid the violation of security constraints while building the planning graph.

The need of axioms. In Graphplan approach, the lack of axioms disrupts the ability to represent real-word domains containing normally quite complex conditions and rules. Without the use of axioms, mechanisms preconditions and effects can become quickly too complex and unreadable. In our approach, we believe that the use of axiom will provide a natural way of deriving supervenient properties, representing logical consequences of the effects of applied mechanisms.

Updating knowledge using an inference graph

In this part, we present how we define axioms in our approach and the way they will be used in the planning process. We define an axiom as an expression in the following form:

$$\bigwedge_{i=1}^{n} p_i \rightarrow \bigwedge_{j=1}^{m} q_j \tag{9}$$

Where each p_i and q_j are predicates of our defined language.

According to a state w of the used system, we want to be able to infer all possible new facts based on a set of axioms that represents relationships between different predicates in our language. To meet this goal, we utilize the same construction method used in Graphplan in order to build an inference graph. In this construction method, we consider each axiom in our system as an action, then the left part of the representation of the axiom (9) will be the preconditions of the action and the right part is its effects. The idea consists on applying in each layer of the graph the set of applicable actions (axioms) until we infer all possible new facts. Algorithm 1 describes how the inference graph is constructed. Once it is built, it allows to extract the set of facts that are derived using the set of defined axioms. In fact, the set of inferred facts is $\mathcal{IG}_l \backslash \mathcal{IG}_0$ were $\mathcal{IG}_0$ and $\mathcal{IG}_l$ represent respectively the set of predicate-nodes in the first fact level and the set of predicate-nodes in the last fact level of the inference graph.

```
   input :
           G    /* planning graph (Graphplan) */
           last_fl_G = {f_1, ··· , f_n} /* set of facts in the last fact level of G */
           A_x = {Ax_1, ··· , Ax_m}   /* the set of domain-axioms */
   output:
           inferred_facts    /* the set of derived new facts */
 1 Main
 2 IG = ∅    /* inference graph initialization */
 3 add_fact_level(IG, last_fl)    /* add the last fact level of G to the inference graph IG */
 4 for i = 0 to m do
 5 |   new_fact_level = ∅    /* new empty fact level */
 6 |   new_fact_level = last_level(IG)    /* copy the last fact level of IG to new_fact_level */
 7 |   foreach axiom in A_x do
 8 |   |   instances = instantiate(axiom)    /* get all instances of the axiom */
 9 |   |   foreach inst in instances do
10 |   |   |   /* axioms can be divided into left and right parts (rule 9) */
11 |   |   |   if (last_level(IG) ⊨ left_part(inst)) then
12 |   |   |   |   new_fact_level = new_fact_level ∪ right_part(inst)
13 |   |   |   end
14 |   |   endfch
15 |   endfch
16 |   if (new_fact_level == last_level(IG)) then
17 |   |   inferred_facts = new_fact_level \ last_fl_G
18 |   |   break
19 |   else
20 |   |   add_fact_level(IG, new_fact_level)
21 |   end
22 end
```

Algorithm 1: Building inference graph and getting new derived facts

Theorem 1. *Given the set of formulas Σ_w representing the system $\mathcal{S}$ in the state w, and a set of n consistent axioms $\mathcal{A} = \{ax_1, \cdots, ax_n\}$, the height of the graph representing the graph inference of $\mathcal{S}$ using $\mathcal{A}$ will be at most n.*

Theorem 2. *Consider a system $\mathcal{S}$ composed of n objects and represented by p predicates in a state w_i, and m axioms each having a constant number of bounded variables. Let q be the largest number of predicates in the right-side of each axiom (formula 9). Then, the size of a k-level inference graph and the time required to build it, are polynomial n, m, q, p and k.*

Building planning graph under security constraints

The specification of security constraints requires that some properties should be respected during all the states of the target system. Since each fact level of the planning graph is built using the construction method of Graphplan, it can be considered as a possible state of the system, our idea consists of verifying the satisfiability of security constraints on each new created fact level of the planning graph during its construction.

Definition 7. ***Violated security constraint:*** *Consider a planning graph G composed of n fact levels $fl_1, \cdots, fl_n$, each fact level fl_i is composed of a set of facts w_i. A security constraint C specified in our formal language using the set of formulas Σ_C and specified in the STRIPS system language by $\overline{\Sigma_C}$ (the negation of CNF of Σ_C) is violated in a fact level fl_i if and only if $w_i \models \overline{\Sigma_C}$.*

Graphplan uses directed edges to connect each action instance node belonging to the ith action level of the graph to the set of fact nodes belonging to the ith fact level representing its preconditions, and to the set of fact nodes belonging to the $(i+1)$th fact level representing its effects. Thanks to this property, we are able to find the combinations of actions belonging to the ith action level of the graph and leading to security constraints violation in the $(i+1)$th fact level.

Algorithm 2 describes the used method to get the combinations of actions leading to violate a security constraint. The correctness and the complexity of the Algorithm 2 are proved by the following theorems.

Theorem 3. ***(Correctness):*** *Given a violated security constraint C and a set of fact nodes cause_nodes that causes the violation of C, the Algorithm 2 terminates and computes all the combinations of actions that lead to violate C.*

Theorem 4. ***(Complexity):*** *Given a violated security constraint C, a set of cause nodes $CN = \{n_1, \cdots, n_n\}$ representing the set of fact nodes that causes the violation of C, the complexity of the algorithm 2 is $O(\prod_{i=1}^{n} l_i)$ in time, where l_i is the number of different actions providing the fact node n_i.*

```
   input :
           C   /* the violated security constraint */
           cause_nodes   /* the set of fact_nodes that causes the violation of C */
   output:
           action_combinations    /* the set of combinations of actions that violate the
   constraint C */
 1 Main
 2 combination = ∅
 3 all_combinations(causes_nodes, action_combination)
 4 End Main
 5
 6 Recursive Procedure all_combination(nodes, combination)
 7 if (Card(nodes) == 0) then
 8   |  add(combination, action_combination) /* add combination to action_combination */
 9 end
10 first_node = nodes.first    /* get the first node in the set nodes */
11 remove(nodes, first_nodes)    /* remove the first_node from the set nodes */
12 foreach action_node in first_nodes.in_edges do
13   |  copy_combination = combination
14   |  if (action_node ∉ copy_combination) then
15   |    |  add(action_node, copy_combination)
16   |  end
17   |  all_combinations(causes_nodes, copy_combination)
18 endfch
```

Algorithm 2: Getting all combinations of actions that violate a constraint

Once we know the combination of actions $\mathbb{C}_c$ that leads to the violation of the security constraint C. The trivial way to solve this violation problem would be to remove $\mathbb{C}_c$ and its corresponding effects from the planning graph. However, this solution can be useless in many cases as it can prevent some actions in $\mathbb{C}_c$ (a subset of $\mathbb{C}_c$) that do not violate C to be used.

Avoiding security constraints violation. In Graphplan, mutual exclusions are basically used to specify that no valid plan could possibly contain conflictual actions in the same plan step. Since, a security constraint C is violated if all actions in a combination $\mathbb{C}_c$ that violates C are applied in the same action level

of the planning graph, our solution to prevent this violation is to use mutual exclusion relations as following:

(i) If $|\mathbb{C}_c| \geq 2$: $\forall$ $node_a \in \mathbb{C}_c$, create a mutual-exclusion between $node_a$ and $\mathbb{C}_c \setminus \{node_a\}$.
(ii) If $|\mathbb{C}_c| = 1$, remove the action-node in $\mathbb{C}_c$ and its corresponding effects from the planning graph.

where $|\mathbb{C}_c|$ represents the number of action-nodes in $\mathbb{C}_c$. (i) ensures that if the number of action-nodes in $\mathbb{C}_c$ is more that one, therefore we will create a mutual-exclusion between each action-node $node_a$ in $\mathbb{C}_c$ and the set of other action-nodes in $\mathbb{C}_c$. This allows in one side to ensure that no correct plan could possibly contain $node_a$ and $\mathbb{C}_c \setminus \{node_a\}$ together which allows to avoid the violation of the security constraint C, and on the other side allows the largest subsets of action-nodes ($\mathbb{C}_c \setminus \{node_a\}$) in $\mathbb{C}_c$ that do not violate C to be used together in the same plan. (ii) states that if $\mathbb{C}_c$ is composed of only one action-node, therefore, the unique solution to avoid the violation of C is to remove the action-node in $\mathbb{C}_c$ as well as its corresponding effects from the planning graph.

6 Searching the best plan

Given a planning graph $\mathcal{G}$ constructed using our previously explained extension of GraphPlan, our goal is to find the best mechanisms execution plan (parallel plan) that enforces the chosen security and utility requirements. For this end, and to be able to compare different mechanisms execution plans, as a first step, we assign a weight for each action-node in $\mathcal{G}$ representing a security mechanism using the metric described in Definition 8. As a second step, we define a second metric to measure a score for each mechanisms execution plan that can satisfy the defined policy as described in Definition 9.

Definition 8. *Consider an action-node an_M in $\mathcal{G}$ representing the application of the security mechanism M over the object ob. Suppose that M provides n security properties $sp_1, \cdots, sp_n$ and m utility properties $up_1, \cdots, up_m$. The weight ω which will be assigned to an_M is measured as following:*

$$\omega = \alpha_{ob} \sum_{i=1}^{n} \tau_i + \beta_{ob} \sum_{i=1}^{m} \nu_i - \delta_{ob}\ \varepsilon_M$$

where $\tau_i \in [0,1]$ represents the robustness level of the provided security property sp_i, $\nu_i \in [0,1]$ represents the satisfiability level of the provided utility property up_i, $\varepsilon_M \in [0,1]$ is the deployment efficiency level of the mechanism M, and $\alpha_{ob} \in [0,1]$, $\beta_{ob} \in [0,1]$, and $\delta_{ob} \in [0,1]$ represents respectively the security, utility, and deployment efficiency factors of ob such that α_{ob}, β_{ob}, and δ_{ob} are complementary.

The intuitions behind the use of the robustness level τ (1), the satisfiability level ν (2), the deployment efficiency level ε (3), and the security, utility

and deployment efficiency factors (4) to measure the weight of an action-node is that: (1) Some security mechanisms are not as robust as they should be to fully ensure their provided security properties under well known attacks. For example, encryption-based mechanisms are supposed to ensure the confidentiality of the objects over which they are applied. However an Order-preserving encryption based mechanisms such as Boldyreva [7] preserves the order of the plaintexts, which may enable many attacks. It was concluded that order-preserving encryption leaks at least half of the plaintexts bits [22]. Hence, the confidentiality robustness level $\tau_{confidentiality}$ will be less that 0.5 for Boldyreva. (2) Some security mechanisms cannot fully provide some utility requirements. In these cases, the satisfiability level factor ν is used to specify the level of providability of an utility requirement. For illustrative purpose, let us take the example of homomorphic-based encryption mechanisms which are supposed to provide computation (addition + multiplication) over encrypted objects. However, Paillier cryptosystem [17] is homomorphic-based encryption mechanisms allowing to perform only addition over encrypted data. Therefore, satisfiability level factor of computation for Paillier cryptosystem will be $\nu_{computation} = 0.5$. (3) Some security mechanisms are expensive in terms of deployment time compared to other security mechanisms, we take this fact into consideration by using the deployment efficiency level ε_M, as much as the mechanism M can be efficiently deployed, ε_M will be closer to 1. (4) The weight of an_M representing the application of M over *ob* should also take into account the security, utility and deployment efficiency factors represented respectively by ϵ_{ob}, ρ_{ob}, and δ_{ob}, which are specified by the data owner for *ob*. For illustrative purpose, let us take a file f_1 storing information about the payment parameters used by the costumers of a company. The company attributes the value 0.8 to ϵ_{f_1}, 0.1 to ρ_{f_1}, and 0.1 to δ_{ob} as it considers that the utility of f_1 as well as deployment efficiency of the policy over f_1 are not important compared to its security. That is why the action-node in $\mathcal{G}$ representing a security mechanism applied over f_1 which ensures the highest level of robustness for security properties will have the highest weight compared to others having high providability of utility requirements, high deployment efficiency and weakly robustness for security properties.

Definition 9. *Consider a parallel plan $\mathcal{P} = \{p_1, \cdots, p_n\}$. Suppose that each $p_i \in \mathcal{P}$ is composed of m_i action-nodes $an_1^i, \cdots, an_{m_i}^i$. The score Sc of $\mathcal{P}$ is:*

$$Sc = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \omega_j^i$$

where ω_j^i is the weight of the action-node an_j^i measured according to the Definition 8.

Definition 10. *Consider a security policy $\mathcal{SP}$ and a set of parallel plans $\mathcal{P}_1, \cdots, \mathcal{P}_n$ in $\mathcal{G}$ each satisfying $\mathcal{SP}$ and all having respectively the scores $Sc_1, \cdots, Sc_n$. A parallel plan $\mathcal{P}$ having the score Sc is the best parallel plan in $\{\mathcal{P}_1, \cdots, \mathcal{P}_n\}$ if the following condition holds:*

$$\forall i \in 1 \cdots n,\ Sc \ \geq \ Sc_i$$

Obliviously, finding the best parallel plan in a planning graph $\mathcal{G}$ that enforces a security policy $\mathcal{SP}$ requires finding all parallel plans in $\mathcal{G}$ that satisfy $\mathcal{SP}$.

Theorem 5. *Computing all parallel plans in a planning graph that enforce a security policy $\mathcal{SP}$ is NP-hard.*

Heuristic search based planning

Our goal is to find the parallel plan having both the maximum score regarding our metric (defined in Definitions 8, 9, and 10), and the minimum number of steps. To this end, we use the cost-optimal planner CO-PLAN [20] which proceeds in four stages:

- Planning graph conversion to CNF wff: Convert the planning graph into a CNF notation by constructing proposition formula as described in [13].
- Wff solving: CO-PLAN uses a modified version of RSAT [18] called CORSAT to process the CNF formulae which allows to figure out: (1) If a solution exists for the given decision problem, and (2) if a solution exists, it is identified with minimal plan costs.
- Bounded forward-search: CO-PLAN uses the speed and efficiency of SAT-based planners allowing to obtain a good admissible initial bound on the cost of an optimal plan. In the second phase, CO-PLAN performs then a bounded forward-search in the problem state space.
- Plan extraction: If a model of the wff is found, then the model is converted to the corresponding plan; otherwise, the length of planing graph is incremented and the process repeats.

In fact, CO-PLAN identify the solution having the minimal parallel plan costs. To be able to use it, we transform our parallel plan score maximization problem to a minimization plan cost problem by considering that $Cost_{\mathcal{P}} = -Sc_{\mathcal{P}}$, where $Cost_{\mathcal{P}}$ and $Sc_{\mathcal{P}}$ represent respectively the cost of the parallel plan $\mathcal{P}$ and the score of $\mathcal{P}$ measured according to Definition 9.

7 Implementation and Evaluations

In the experimental part of this work, we measure the computational performance of our approach.

7.1 Implementation

We develop a prototype implementing our approach to find a near-optimal security mechanisms plan allowing to enforce security policies for outsourced data using available open source C++ libraries. For GraphPlan construction, we used the SATPLAN'06 library [14] allowing to create a planning graph up to some length k. We extend SATPLAN'06 library (as described in section 5) to support:

(1) the use of domain axioms allowing to deduce new facts about objects of the system to be used, and (2) we improve the Graphplans construction method of the planning graph to avoid the violation of security constraints while building the planning graph. For analyzing the planning graph and searching the best mechanisms plan, we used CO-PLAN library [20].

7.2 Experimental setup

The domain that we have used in evaluating our prototype is composed of:

- A data owner;
- A finite set of users:
  - Trusted users: which can access and use the outsourced data
  - Untrusted users: which are not supposed to be able to violate the policy. In all experiments, we suppose that we have two untrusted users, a cloud server and an external adversary.
- A finite set of objects that represents the data to be outsourced, we consider that the data owner wants to outsource a file system. So the objects are the set of files and directories in the file system to be outsourced.
- A finite set of security and utility requirements representing the policy to be enforced. We suppose that the data owner will specify some security constraints and utility goals over some objects in the file system to be outsourced. Only the objects over which the data owner has specified the policy will be considered in the planning problem.
- A finite set of security mechanisms that can be used to enforce the security policy. We specified 20 security mechanisms, including 8 encryption-based mechanisms, 4 anonymization-based mechanisms, 6 watermarking-based mechanisms, and 2 information transfer protocols HTTPS and SSH that can be used to send the objects to be outsourced to the cloud server.

We ran the all experiments on a server with Intel core i7 2.50 GHz, 16 GB of RAM, and running Debian 7.

7.3 Experimental results

We conducted a set of experiments to evaluate the performance of our prototype. Table 7.3 shows the parameters used in each experiment, the number of nodes in the planning graph built to resolve the problem, and the time needed to find a near-optimal solution using the method we presented in Section 6. Due to lack of space, we will not be able to include the specifications of the constraints and the security mechanisms that are used in each experiment.

8 Conclusion

In this paper, we have presented a new framework allowing: first, a data owner to formally specify different security and utility requirements that should be enforced over the data to be outsource. Second, to specify existing security mechanisms that can be used to enforce the policy defined by the data owner. Finally,

Parameters				number of nodes	time(s)
objects	constraints	users	mecanisms		
5	5	5	15	75952	1.9
10	10	5	15	121385	9.7
20	15	5	15	721385	97.65
100	50	5	20	1951423	721.5

Table 1. Our prototype performance with respect to: the number of objects that will be outsourced (objects), the number of constraints defined over the object to be outsourced (constraints), the number of users involved in the used system (users), the number of security mechanisms that can be used to enforce the policy (mechanisms). Column "number of nodes" indicates the number of nodes in the planning graph.

to choose the near-optimal security mechanisms execution plan that enforces the policy while offering the best tradeoff between security, utility and complexity.

References

1. Anis Bkakria, Frédéric Cuppens, Nora Cuppens-Boulahia, and José M. Fernandez. Confidentiality-preserving query execution of fragmented outsourced data. In *Information and Communicatiaon Technology - International Conference, ICT-EurAsia 2013, Yogyakarta, Indonesia, March 25-29, 2013. Proceedings*, pages 426–440, 2013.
2. Anis Bkakria, Frédéric Cuppens, Nora Cuppens-Boulahia, José M. Fernandez, and David Gross-Amblard. Preserving multi-relational outsourced databases confidentiality using fragmentation and encryption. *JoWUA*, 4(2):39–62, 2013.
3. Anis Bkakria, Frédéric Cuppens, Nora Cuppens-Boulahia, and David Gross-Amblard. Specification and deployment of integrated security policies for outsourced data. In *Data and Applications Security and Privacy XXVIII - 28th Annual IFIP WG 11.3 Working Conference, DBSec 2014, Vienna, Austria, July 14-16, 2014. Proceedings*, pages 17–32, 2014.
4. Anis Bkakria, Andreas Schaad, Florian Kerschbaum, Frédéric Cuppens, Nora Cuppens-Boulahia, and David Gross-Amblard. Optimized and controlled provisioning of encrypted outsourced data. In *19th ACM Symposium on Access Control Models and Technologies, SACMAT '14, London, ON, Canada - June 25 - 27, 2014*, pages 141–152, 2014.
5. Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1636–1642, 1995.
6. Andras Boho, Glenn Van Wallendael, Ann Dooms, Jan De Cock, Geert Braeckman, Peter Schelkens, Bart Preneel, and Rik Van de Walle. End-to-end security for video distribution: The combination of encryption, watermarking, and video adaptation. *IEEE Signal Process. Mag.*, 30(2):97–107, 2013.
7. Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 224–241, 2009.

8. Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Fragmentation and encryption to enforce privacy in data storage. In *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007, Proceedings*, pages 171–186, 2007.
9. Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.
10. Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '80, pages 163–173, New York, NY, USA, 1980. ACM.
11. David Gross-Amblard. Query-preserving watermarking of relational databases and xml documents. *ACM Trans. Database Syst.*, 36(1):3, 2011.
12. Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*, pages 216–227, 2002.
13. Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 2.*, pages 1194–1201, 1996.
14. Henry A. Kautz, Bart Selman, and Jörg Hoffmann. SatPlan: Planning as satisfiability. In *Abstracts of the 5th International Planning Competition*, 2006.
15. Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2004, San Diego, California, USA*, 2004.
16. Maithili Narasimha and Gene Tsudik. DSAC: an approach to ensure integrity of outsourced databases using signature aggregation and chaining. *IACR Cryptology ePrint Archive*, 2005:297, 2005.
17. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.
18. Knot Pipatsrisawat and Adnan Darwiche. Rsat 2.0: Sat solver description. Technical report, 2007.
19. Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 85–100, 2011.
20. Nathan Robinson, Charles Gretton, and Duc-Nghia Pham. Co-plan: Combining sat-based planning with forward-search. *Proc. IPC-6*, 2008.
21. Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
22. Liangliang Xiao and I-Ling Yen. Security analysis for order preserving encryption schemes. In *46th Annual Conference on Information Sciences and Systems, CISS 2012, Princeton, NJ, USA, March 21-23, 2012*, pages 1–6, 2012.