Privacy policy preferences enforced by SPARQL Query Rewriting

Said Oulmakhzoune*‡, Nora Cuppens-Boulahia*, Frederic Cuppens* and Stephane Morucci‡
*Institut Mines-Telecom/Telecom-Bretagne, 2 Rue de la Chataigneraie, 35576 Cesson Sevigne, France
{said.oulmakhzoune, nora.cuppens, frederic.cuppens}@telecom-bretagne.eu
‡Swid, 5 square du Chene Germain, Immeuble le Thulium, 35510 Cesson Sevigne - France
{said.oulmakhzoune, stephane.morucci}@swid.fr

***Abstract*—When specifying privacy preferences, the data owner can control who may access its personal data, for which purpose and under which accuracy. In this paper we present an approach that enforces the privacy policy preferences by query transformation. We present also how to instrument this rewriting query algorithm using a privacy-aware model like *PrivOrBAC*. We take into account various dimensions of privacy preferences through the concepts of consent, accuracy, purpose and recipient.**

***Index Terms*—Semantic Data, SPARQL, Query Rewriting, privacy-aware, PrivOrBAC**

I. Introduction

RDF [11] ($\mathcal{R}$esource $\mathcal{D}$efinition $\mathcal{F}$ramework) is an increasingly used framework for describing Web resources, including private, sensitive and confidential resources. It is a data model based on graphs of {subject, predicate, object} triples. SPARQL [18] has been defined to easily locate and extract data from an RDF graph or resources viewed as RDF graph. It was standardized by the RDF Data Access Working Group of the World Wide Web Consortium, and is considered a key semantic web technology. When private data are accessed, SPARQL queries must be filtered so that only authorized data are returned with respect to some confidentiality policy and privacy preferences.

The control of confidentiality policy is not investigated in this paper. It was already investigated in [16] for SPARQL queries. Their approach is also based on query rewriting by applying filters to SPARQL queries. However, they have not taken into account privacy policy specificities.

The goal of our approach is to enforce the privacy requirements by SPARQL query rewriting. That rewriting algorithm is instrumented by a privacy-aware model. Without any loss of generality, we propose to use the privacy-aware OrBAC model PrivOrBAC [6] to express privacy policies. [6] specifies the most relevant privacy requirements which are compliant with the current legislations in the field of privacy protection [15], [4]. These requirements are the consent, accuracy, provisional obligations and purposes. The *consent* is the user agreement for accessing and/or processing his/her data. It is the requirement before delivering the personal data to third parties. The *accuracy* is the level of anonymity and/or the level of the accuracy of the data. The *provisional obligations*, refer to the actions to be taken by the requestors after the access. The *purpose* is the goal that motivates of the access request.

Privacy is the right of individuals to determine for themselves when, how and to what extent information about them is communicated to others[1]. The data subject (*data owner*) is the one who specifies the privacy preferences, i.e. it decides how its data should be used (*purposes*), to whom it may be disclosed (*recipients*) and under which *accuracy* (anonymization, obfuscation, etc.).

Privacy policy can refer to "data item" at the level of a cell in the case of relational databases or the level of an individual property (RDF predicate) in the case of RDF databases.

In the literature there are two categories of approaches that handle the privacy dimension. The first one aims to define a model or language to express privacy requirements. For instance, models such as P-RBAC [14], Purpose-BAC [24] and Pu-RBAC [13] define new languages to express access contexts, and they focus on purpose entity and on other privacy requirements [15]. PrivOrBAC [6] extends the OrBAC model [5]. It reuses most of existing mechanisms implemented in OrBAC to express privacy requirements.

The second category aims to apply security requirements on data and/or using their own new security model for a specific kind of database, for instance, LeFevre et al. [12] and Huey [10]. [12] presents an approach that enforces limited disclosure expressed by privacy policy in the case of Hippocratic databases. They store privacy requirements in relational tables in the same database where the data to be protected is stored. Then they apply those requirements by a query rewriting approach. [10] is based on the Oracle Virtual Private Database (VPD) and supports fine-grained access control (FGAC) through policy functions. A function is associated with the table (or view) that needs protection. When it is invoked, it returns various pieces of SQL, called predicates, depending on the system context (e.g current time, current user, etc.) to enforce FGAC. Policy functions are expressed in PL/SQL. Their approach is based on query modification. When a query is issued, it is dynamically modified by appending predicates, returned by the policy function, to the where clause of the query.

Regarding the enforcement of privacy requirements, our

[1]Alan Westin, Professor Emeritus of Public Law and Government, Columbia University

approach belongs to the second category. But, in our case, requirements specification is based on an existing privacy-aware model independently of the data. Thus, we avoid empiricity, manage conflicts and can move to another privacy model when needed. We choose Priv-OrBAC as it takes into account most of the privacy recommendations specified by well known standards [1], [3], [2], [15].

Our contribution aims to enforce privacy policy preferences with the following features:

- Enforcing privacy policy does not require any modification to existing databases viewed as RDF graphs.
- Use of an existing privacy-aware model to store and manage privacy policy preferences.
- The following privacy requirements are taken into account: consent, purpose, recipient and accuracy.
- In the case of distributed systems, we only have to manage our privacy policy preferences in a given system as SPARQL service (endpoint).

In our approach, the answer to the rewritten query may differ from the result of the user's initial query. In that case and as suggested in [19], we can check the query validity of the rewritten query with respect to the initial query and notify the user when the answer to the query is not complete.

The rest of this paper is organized as follows. Section II presents an overview of our approach. Section III presents an example of privacy preferences ontology used to illustrate our approach. Section IV introduces some criteria that should be satisfied by our rewriting algorithm. Section V presents the principle of our rewriting algorithm. Section VI presents some related works. Finally, section VII concludes this paper.

II. Approach principle

A SPARQL query consists of triple patterns, conjunctions, disjunctions, and optional patterns. SPARQL allows users to write globally unambiguous queries. For example, the following query returns the name of all patients and their drug name.

```
PREFIX dt:<http://hospital.fr/patients/>
SELECT ?name ?drugName WHERE{
   ?p    rdf:type    dt:Patient.
   ?p    dt:name     ?name.
   ?p    dt:takes    ?drug.
   ?drug dt:drugName ?drugName. }
```

Basically, the SPARQL syntax looks like SQL, but the advantage of SPARQL is that it enables queries spanning multiple disparate (local or remote) data sources containing heterogeneous semi-structured data.

The principle of our approach is to rewrite the initial SPARQL query in order to protect personal data from unauthorized access. The privacy policy preferences is expressed and stored using the PrivOrBAC model. We take into account the concepts of *consent*, *accuracy*, *purpose* and *recipient*.

Figure 1 shows an overview of the approach. The *SPARQL Rewriting Engine* is the component that implements the rewriting algorithm. It takes as input (i) the initial SPARQL query, (ii) the preferences ontology $OWL_{privacy}$ and (iii) a table mapping $\mathcal{M}$. (ii) and (iii) are explained in detail in section III. *SPARQL Rewriting Engine* adds security constraints (SPARQL conditions, filters and services) to the initial query such that the returned result is compliant with the privacy policy defined by each data-owner. The output is the rewritten SPARQL query denoted as Q_{rw}.

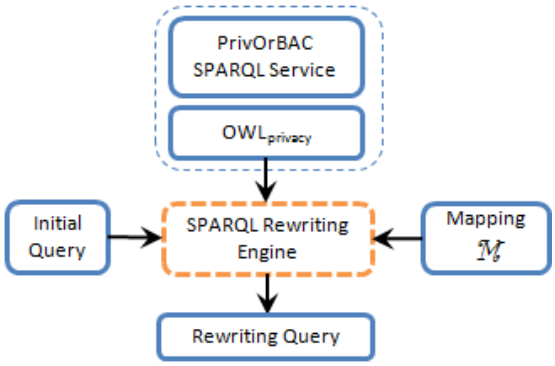

Fig. 1: Our approach principle

Let us first show a privacy unaware execution of a SPARQL query. We assume that doctor Alice tries to get the name of all patients, and patient Charlie does not want to disclose his information, including her name, to doctor Alice. Alice issues the following query Q_i:

```
PREFIX dt:<http://hospital.fr/patients/>
SELECT ?name WHERE{
  ?p rdf:type dt:Patient.
  ?p dt:name  ?name. }
```

The initial query Q_i is executed by the SPARQL engine which evaluates it (with no modifications and no filters) and gets the corresponding result from the queried RDF data sources. Of course the returned result contains Charlie's name.

PrivOrBAC proposes a set of web services used to access the privacy policy preferences. Using these web services we build our *PrivOrBAC SPARQL service* by implementing the approach proposed in [7]. In [7] the received SPARQL query is decomposed into a set of web services that covers the initial query requirements. *PrivOrBAC SPARQL service* is a server that receives a SPARQL query expressed in $OWL_{privacy}$. It decomposes this SPARQL query into a set of PrivOrBAC web services that will be invoked later. The collected result is correctly merged, filtered and formated, then transfered to the requestor (SPARQL Engine in our case). In the rest of this paper we assume that the privacy policy is accessed through SPARQL service based on preferences ontology $OWL_{privacy}$ presented in section III. That is, the privacy policy is viewed as RDF data.

In the case of privacy-aware model that does not provide web services, there exist other approaches that allow building SPARQL services. For instance [21] and [8] could be used when policies are stored in XML format.

Figure 2 illustrates the execution of the rewritten query Q_{rw}. It can be summarized as follows:

- SPARQL rewriting engine rewrites the initial query by inserting (i) a call to the service of policy preferences and (ii) inserting corresponding security constraints.

- SPARQL engine executes the rewritten query Q_{rw}. So, that execution will get the data from 'RDF databases' and the corresponding policy preferences from the service of preferences that is injected by the rewriting engine. Then it applies corresponding security filters and conditions.

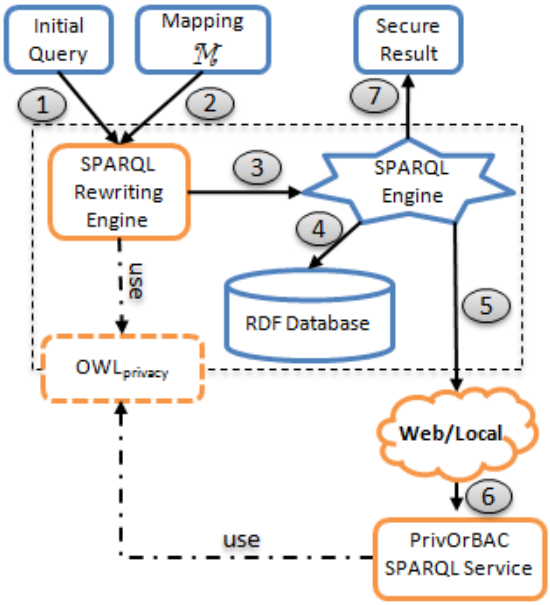

Fig. 2: Our approach using SPARQL Service

III. Privacy-aware Ontology

In this section we define an ontology of privacy preferences. This ontology allows us to query the privacy policy preferences of PrivOrBAC via SPARQL as explained in section II. This ontology is independent from the data structure. It is denoted $OWL_{private}$. Obligations [6] are not taken into account by this ontology. We handle the concepts of consent, accuracy, purpose and recipient as suggested in [6]. In the rest of the paper, we use the prefix *P* (resp. *dt*) for preference ontology (resp. data ontology). Figure 3 presents this ontology.

It is composed of three classes:

- *P:Dataowner*: represents the data-owner, he/she is identified by the property *P:hasId*. Each data-owner has a set of preferences *P:Preference* via the object property *P:hasPreference*.
- *P:Preference*: represents preferences associated with triples (data-owner, recipient, purpose). Each preference has a set of targets *P:Target* via the object property *P:hasTarget*. *P:hasPurpose* corresponds to the purpose on which this preference is defined. *P:hasRecipient* represents the *Requestor* targeted by this preference.
- *P:Target*: represents a particular preference item. It is composed of three properties. (1) *P:hasName* represents the name of a data item, e.g. age, name, address, etc. (2) *P:hasDecision* represents the data-owner choice (consent) associated with that target, e.g. Yes to indicate that the data-owner agrees to disclose the value of that target to (resp. for) the corresponding recipient (resp. purpose) of the preference, No otherwise. (3) *P:hasAccuracy* represents the accuracy that will be applied to that target in the case of positive consent (decision=Yes), e.g. Anonymization, Nullification, etc.

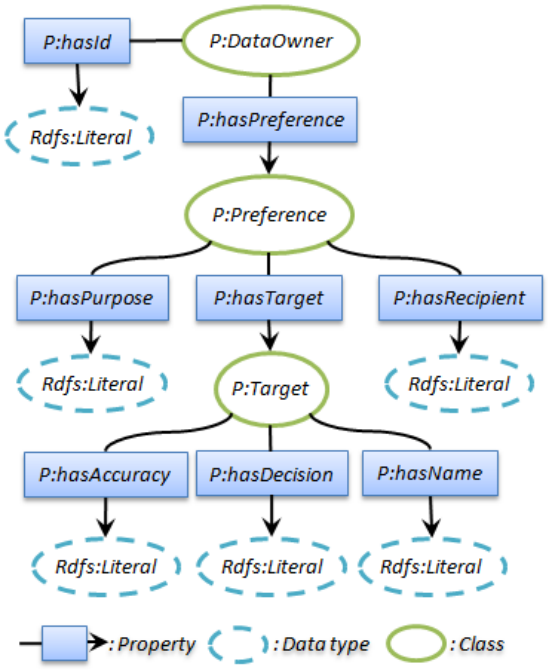

Fig. 3: Privacy Preferences Ontology

Figure 4 shows an example of privacy preferences defined by the data-owner Safaa for the purpose *purpose_1* and the recipient *Bob*. She decided to disclose her name in clear (without accuracy) and her age after k-anonymization [22] with $k = 15$.

Let OWL_{data} be the data ontology or the ontology on which the initial query is expressed. Properties of OWL_{data} are associated with the PrivOrBAC ontology $OWL_{privOrBAC}$ defined above, via a mapping table $\mathcal{M}$. It is defined as follows: each property of OWL_{data} (i.e. *dt:name*, *dt:age*) is mapped to a string value ('*name*', '*age*' resp.) which is the value of the property *P:hasName* of an instance of the class *P:Target*. We denote this mapping table as $\mathcal{M}$. For instance, in figure 4, '*_:t1*' is an instance of the class *P:Target*. It corresponds to the preference defined by the data-owner for the property *dt:age*.

The mapping table $\mathcal{M}$ represents also a mapping between values of the *Target* attribute defined in the *Consent_preference* view, and OWL_{data} properties.

IV. The correctness criteria

Before presenting our rewriting algorithm, we will first introduce some criteria that should be satisfied by any rewriting algorithms.

Wang et al. [23] proposed a formal notion of correctness for fine-grained access control in relational databases. They presented three correctness criteria (sound, secure and maximum)

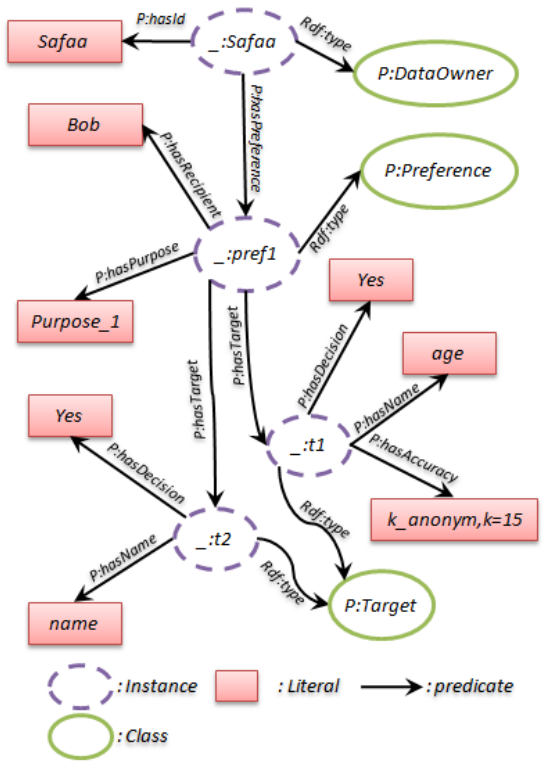

Fig. 4: Privacy Preferences Example

that should be satisfied by any query processing algorithm in order to be "correct".

Soundness : An algorithm is sound if and only if its rewritten query Q_{rw} returns only correct answers i.e. answers to the initial query.

Maximality : An algorithm is maximum if and only if Q_{rw} returns as much information as possible.

Security : An algorithm is secure if and only if the result of Q_{rw} respects the security and privacy policy of the queried system.

In the rest of this section, we present a condition that should be satisfied by our algorithm in order to satisfy the soundness, maximality and security criteria [23].

Let A be a query processing algorithm that enforces privacy policies. Let D be a database, P a disclosure policy and Q a query. Let $Q' = A(P, Q)$ be the rewriting query of Q using the policy P. We denote $R = A(D, P, Q)$ the output result of Q' on D.

Definition 1 : Given two tuples $t_1 = (x_1, x_2, ..., x_n)$ *and* $t_2 = (y_1, y_2, ..., y_n)$*, we say that* t_1 *is subsumed by* t_2*, denoted* $t_1 \sqsubseteq t_2$*, if and only if* $\forall i \in [1..n] : (x_i = y_i) \vee (x_i = unauthorized)$.

Definition 2 [23] : Given two relations R_1 *and* R_2*, we say that* R_1 *is subsumed by* R_2*, denoted* $R_1 \sqsubseteq R_2$*, if and only if:*

$$(\forall t_1 \in R_1)(\exists t_2 \in R_2)|t_1 \sqsubseteq t_2$$

Definition 3 [23] : Two databases states D *and* D' *are "equivalent" with respect to policy* P *(denoted as* $(D \equiv {}_P D')$*) if the information allowed by* P *in* D *is the same as that allowed by* P *in* D'.

Let S denote the standard query answering procedure and $S(D, Q)$ the result of the query Q in the database state D without any privacy restriction. [23] formalizes the three criteria as follows: a query processing algorithm A is :

- sound if and only if :$\forall P \forall Q \forall D$

$$A(D, P, Q) \sqsubseteq S(D, Q)$$

- secure if and only if:$\forall P \forall Q \forall D \forall D'$

$$[(D \equiv {}_P D') \longrightarrow (A(D, P, Q) = A(D', P, Q))]$$

- maximum if and only if: $\forall D' \forall$ relation R if $[(D \equiv {}_P D') \wedge (R \sqsubseteq S(D', Q))]$ then we have $R \sqsubseteq A(P, D, Q)$

V. Rewriting Algorithm Principle

For a given SPARQL query, we assume that we have the associated requestor (recipient) and purpose. We assume that there is only one data-owner for each data item [20]. We do not handle the case where several data-owners share one data item. This issue is not handled by PrivOrBAC.

The principle of our approach is summarized in the following items. For each property of the where clause of the initial query:

1) we normalize the triple pattern corresponding to that property (Algorithm 1)
2) we get its associated choice (*P:hasDecision*) and accuracy (*P:hasAccuracy*) for each data-owner.
3) we apply the corresponding choice and accuracy using algorithm 2.

The normalization is applied to extract implicit filters in SPARQL queries. It transforms the implicit filter to explicit one with respect to the semantic of the initial query.

The algorithm 1 aims to normalize a triple pattern tp, i.e. transforms tp to a new triple pattern tp_{norm} where its object is a SPARQL variable. For example the normalization of the triple pattern *{?p dt:age 25}* is the pattern *{?p dt:age ?age. Filter(?age=25)}*.

Algorithm 1 Normalize a triple pattern

Require: Triple pattern $tp \leftarrow (s, p, o)$
1: Let tp_{norm} be a triple pattern such that $tp_{norm} \leftarrow tp$
2: **if** the object o of tp is not a variable **then**
3: Let var be a SPARQL variable
4: $tp_{norm} \leftarrow (s, p, ?var)$
5: $tp_{norm} \leftarrow tp_{norm}.Filter(?var = o)$
6: **end if**
7: **return** tp_{norm}

For a given property $prop$ of a data-owner do, algorithm 2 returns the new value of $prop$ that do wants to return to the requester (recipient), by applying its corresponding choice and

accuracy. It checks if the choice (decision) is negative then it returns a null value. Otherwise, if the accuracy is defined then it returns the value of $prop$ by applying the corresponding accuracy, using the function *apply(value, accuracy)*. If the accuracy is not defined, it returns the original value of $prop$.

Algorithm 2 Apply the choice and its accuracy on a property value

Require: choice, accuracy, value
1: **if** choice = 'No' **then**
2: **return** null
3: **end if**
4: **if** accuracy is bound to a value **then**
5: **return** apply(value,accuracy)
6: **end if**
7: **return** value

A. Algorithm

Before defining the general algorithm we will first introduce some useful methods that are used by the general rewriting algorithm.

The second step of our algorithm, after normalization step, is to insert a call to the privacy service in order to get preferences of each data-owner. The service block corresponding to that call, denoted as $ServiceBlock(\$purpose, \$recipient)$, depends on the purpose *$purpose* and recipient *$recipient* of the initial query. This service block is defined as follow.

```
ServiceBlock($purpose, $recipient) =
SERVICE ps:preferences {
  ?dp   rdf:type P:DataOwner;
        P:hasId  ?id;
        P:hasPreference ?pref.
  ?pref P:hasPurpose    $purpose;
        P:hasRecipient  $recipient; }
```

This block of service SB returns all data-owner's preferences defined for the given purpose *$purpose* and recipient *$recipient*. To filter preferences for specific targets, we add corresponding triples to that service block. For that, we define a method, denoted *addTarget(SB:ServiceBlock, tn:Value)*, that takes as parameter a service block SB and the name of the target tn. The set of triples added to the block SB are to get the decision and accuracy of the given target name tn. The algorithm 3 presents the definition of the $addTarget$ method. For example, In order to get preferences of the target 'name' defined for the purpose *$purpose* and the recipient *$recipient*, we call the method *addTarget(SB, 'name')*. The new value of SB is as follows:

Algorithm 3 addTarget method *addTarget(SB:ServiceBlock, x:Value)*

Require: Service block SB and value x
1: add the triple $\{?pref \quad hastTarget \quad ?tx\}$ to SB
2: add the triple $\{?tx \quad hasName \quad x\}$ to SB
3: add the triple $\{?tx \quad hastDecision \quad ?xDecision\}$ to SB
4: Let Opt be the optional SPARQL element defined as follows
5: $Opt = Optional\{?tx \quad hasAccuracy \quad ?xAccuracy\}$
6: add Opt to SB

```
SERVICE ps:preferences {
  ?dp   rdf:type P:DataOwner; P:hasId  ?id;
        P:hasPreference ?pref.
  ?pref P:hasPurpose    $purpose;
        P:hasRecipient  $recipient;
        P:hasTarget     ?tname.
  ?tname  P:hasName     'name';
          P:hasDecision ?nameDecision.
  OPTIONAL{?tname P:hasAccuracy ?nameAcc}
}
```

The third step is to apply the data-owner preferences (decision and accuracy), for each target specified in the query, by using algorithm 2. This step aims to nullify unauthorized data and to apply the corresponding accuracy, if defined, for authorized ones. In our algorithm we handle a *Basic Graph Pattern*[2] Bgp [18] of the initial SPARQL query.

Let Bgp be a basic group of pattern, tn be a target name and $tp = (s, p, o)$ be the triple pattern of Bgp that corresponds to the target tn. The method *addBind* (algorithm 4) aims to assign the authorized value to the variable o of $tp = (s, o, p)$ after applying the corresponding privacy preferences. The *IF* keyword is one of the SPARQL 1.1 operators [9]. It is defined as follows:

Algorithm 4 addBind method: *addBind(Bgp:BGP, x:Value, tp:TriplePattern)*

Require: BGP Bgp, target name x and normalized triple pattern tp
1: Let o be the object variable of the triple $tp = (s, p, o)$
2: Let o' be a SPARQL variable such that $o \neq o'$
3: We rename the variable o of tp by o'
4: Let C_1 and C_2 be the two conditions defined as follow:
5: $C_1 = (?xDecision = \text{'}No')$
6: $C_2 = bound(?xAccuracy)$
7: Let V_1, V_2, $V_{accuracy}$ and V_{bind} be the values defined as follow:
8: $V_1 = null$ */*null to design unauthorized value*/*
9: $V_{accuracy} = eval(o', ?xAccuracy)$
10: $V_2 = IF(C_2, V_{accuracy}, o')$
11: $V_{bind} = IF(C_1, V_1, V_2)$
12: Let $B = Bind(V_{bind} \quad AS \quad o)$
13: add the bind B to the BGP Bgp

```
rdfTerm IF(expr1, expr2, expr3)
```

It evaluates the first argument *expr1*, interprets it as a boolean value, then returns the value of *expr2* if the boolean value is true, otherwise it returns the value of *expr3*. Only one of *expr2* and *expr3* is evaluated. For instance, the value of V_{bind} (line 11) is V_1 if the condition C_1 is satisfied, otherwise $V_{bind} = V_2$.

The *Bind* keyword is an explicit assignment of variables. It is included in SPARQL1.1 with the syntax:

[2]A simple set of SPARQL triples

```
BIND(expr AS ?var)
```

The value of the expression *'expr'* will be assigned to the SPARQL variable *'?var'* [9].

The SPARQL function *eval(value, accuracy)* is an implementation of the function *apply* used in algorithm 2.

The rewriting algorithm 5 aims to build the service block SB of the initial query, based on algorithm 3. Then, it transforms each basic group of pattern, of the initial query, based on algorithm 4.

Algorithm 5 Rewriting Query Algorithm

Require: purpose Pur, recipient Rec, query Q
1: Let Q' be the normalized query of Q
2: Let $SB = ServiceBlock(Pur, Rec)$
3: **for** each basic group of pattern Bgp **do**
4: Let Bgp' be its equivalent in Q'
5: **for** each triple pattern $tp = (s, p, o)$ of Bgp **do**
6: Let $tp' = (s, p, o')$ be the equivalent triple of tp in Bgp'
7: **if** the property p has mapping in M **then**
8: Let x be the mapping of p in M
9: *addTarget(SB, x)*
10: *addBind(Bgp', x, tp')*
11: **end if**
12: **end for**
13: **end for**
14: add the service bloc SB to Q'
15: **return** Q'

In the rest of this section we will start by analyzing the case of SPARQL query without filters after normalization. Then we will see the case of SPARQL query with filters.

B. SPARQL query without filters

Let us take a simple example to illustrate our approach. Bob tries to select the name and the age of all patients for the purpose *purpose_1*. He issues the following query:

```
SELECT ?name ?age FROM dt:infos WHERE {
  ?p  rdf:type  dt:Patient;
      dt:name ?name; dt:age ?age. }
```

We assume that in the mapping table $\mathcal{M}$, the property *dt:name* (resp. *dt:age*) corresponds to the string value 'name' (resp. 'age'). So, The transformed query is as follows:

```
01.SELECT ?name ?age FROM dt:infos WHERE {
02. ?p  rdf:type  dt:Patient;
03.     dt:id  ?id;
04.     dt:name ?n; dt:age  ?a.
05. SERVICE ps:preferences {
06.  ?dp   rdf:type P:DataOwner; P:hasId ?id;
07.        P:hasPreference ?pref.
08.  ?pref P:hasPurpose   'purpose_1';
09.        P:hasRecipient  'Bob';
10.        P:hasTarget    ?tp1, ?tp2.
11.  ?tp1  P:hasName 'name';
12.        P:hasDecision ?nameDecision.
13.  OPTIONAL{?tp1 P:hasAccuracy ?nameAccu}
14.  ?tp2  P:hasName 'age';
15.        P:hasDecision ?ageDecision.
16.  OPTIONAL{?tp2 P:hasAccuracy ?ageAccu}
17. }
18. BIND(IF(?nameDecision='No',null,
19.  IF(bound(?nameAccu),
20.    udf:eval(?n,?nameAccu),?n)) AS ?name).
21. BIND(IF(?ageDecision='No',null,
22.  IF(bound(?ageAccu),
23.    udf:eval(?a,?ageAccu),?a)) AS ?age).
24.}
```

ID	Target	Choice	Accuracy
1	age	Yes	anonym
2	age	No	-
3	age	Yes	k_anonym, k=15
4	age	Yes	-
1	name	Yes	anonym
2	name	No	-
3	name	Yes	-
4	name	Yes	-

TABLE I: Preferences of name and age properties for (*purpose_1*,*Bob*)

ID	age	Algo. 2
1	33	axfd14
2	42	*null*
3	30	[28,43]
4	27	27

ID	name	Algo. 2
1	Alice	bob15s
2	Charlie	*null*
3	Safaa	Safaa
4	Said	Said

TABLE II: Values of name and age properties after and before applying algorithm 2

Before Transformation		After Transformation	
name	**age**	**name**	**age**
Alice	33	bob15s	axfd14
Charlie	42	*null*	*null*
Safaa	30	Safaa	[28,43]
Said	27	Said	27

TABLE III: Result of the query before and after transformation

Lines 5 to 17 correspond to a call to the service of privacy preferences. In this clause we get the choice and its accuracy of properties *dt:name* and *dt:age* corresponding respectively to *'name'* and *'age'* (lines 11 to 16) for the triple (data-owner, purpose_1, Bob) (lines 6, 8, 9).

Lines 18 to 20 (resp. 21 to 23) correspond to algorithm 4 for the property *dt:name* (resp. *dt:age*).

Table I shows an example of privacy policy preferences of the properties *name* and *age* for the purpose *purpose_1* and the recipient *Bob*. The value 'anonym' (resp. 'k_anonym, k=15') of the accuracy means that the value of the corresponding target must be anonymized (resp. k-anonymized where k=15). Table II shows an example of values of the properties *name* and *age* before and after applying the algorithm 2 based on the preferences given in table I. For instance, the value 'axfd14' (resp. [28,43]) is the anonymization (resp. k-anonymization with k=15) of the value 33 (resp. 30). Finally, table III shows the result of the user query, presented in the example above, before and after query transformation.

In the rest of this section we will study the case of a query with one triple, then the case of a query with many triples. Let D be a database state, P be a disclosure policy, $tp = (s, pred, o)$ be a triple pattern where o is a variable, and $var(tp)$ be all variables defined in tp. We denote Q^D_{tp} the

query defined as follow:

```
SELECT var(tp) FROM D WHERE{s  pred  o}
```

The result of Q_{tp}^D represents all triples that match with tp.

If the predicate $pred$ does not have a mapping in M then the rewritten query of Q_{tp}^D is the same as Q_{tp}^D (line 7 of algorithm 5). Otherwise, the rewritten query is as follow:

```
SELECT var(tp)FROM D
WHERE{ SB.
       s  pred  o'. BIND(f_{P,pred}(o') AS o)}
```

SB corresponds to the service block and $f_{P,pred}(o')$ corresponds to V_{bind} which was defined in algorithm 4. V_{bind} can take three possible values: (i) $null$ for negative decision, (ii) $accuracy(o')$ for positive decision with the accuracy $accuracy$ and (ii) o' otherwise. The result $A(D, P, Q_{tp}^D)$ of the rewritten query corresponds to $S(D, Q_{tp}^D)$ by replacing values of o by $f_{P,pred}(o)$. So, for all $t_1 = (x_1, ..., x_n)$ of $A(D, P, Q_{tp}^D)$ there exists an element $t_2 = (y_1, ..., y_n)$ of $S(D, Q_{tp}^D)$ such that $\forall i \in [1..n]$ $x_i = y_i$ or $x_i = unauthorized$ where $unauthorized$ denotes $null$ or $accuracy(o')$. We deduce that in that case the soundness property is satisfied.

Let Q_2^D be a SPARQL query with two triples $tp_1 = (s_1, pred_1, o_1)$ and $tp_2 = (s_2, pred_2, o_2)$. If $pred_1$ and $pred_2$ do not have a mapping in M then the rewritten query of Q_2^D is the same as Q_2^D. Otherwise, we can write Q_2^D as a join of two SPARQL queries with one triple, as follow:

$Q_2^D = Q_{tp_1}^D \bowtie Q_{tp_2}^D$

If the join is based on the subject part of triples tp_1 and tp_2 i.e. $s_1 = s_2 = s$ then there is no impact for the join of the result of the rewriting queries of $Q_{tp_1}^D$ and $Q_{tp_2}^D$. If the join is based on the object part of tp_1 and subject part of tp_2 i.e. $o_1 = s_2 = o$ then the join of the result of the rewritten queries is based on $f_{P,pred_1}(o)$ and o. Since $f_{P,pred_1}(o_1)$ is not generally equal to s_2 then $A(D, P, Q_2^D)$ may contain unsound result. The problem of join occurs, in the case of SPARQL query Q, when the where clause of Q contains an object property that has a mapping in M.

By induction we can generalize the result bellow for SPARQL query composed of a set of triples.

C. SPARQL query with filter

SPARQL query may contain filters. Filters are restrictions on solutions over the whole group in which they appear. For instance, a query that gets name and age of patients who are more than 25 years old, is expressed as follows :

```
SELECT ?name ?age FROM dt:infos WHERE {
  ?p rdf:type dt:Patient; dt:name ?name;
     dt:age ?age. FILTER(?age>=25)}
```

As explained in section V, the result of the query could be heterogeneous (nullified, anonymized, etc.) after applying the policy preferences. There are two different strategies for query rewriting. The first one is to execute filters before applying the privacy policy preferences. It is obvious that this case is not secure. For instance, a malicious user could issue a query that returns the name and age of patients who are 25 years old.

name	age
Safaa	[28,43]
Said	27

TABLE IV: The expected result

name	age
Said	27

TABLE V: The obtained Result

The returned result corresponds exactly to patients who are 25 years old even if the value of the age is hidden.

The second strategy is to execute filters after applying the privacy policy preferences. In this case, filters are executed on heterogeneous blinded data. Thus the returned result could not be sound. For instance, Bob tries to select patients who live in Rennes. We assume that Alice lives in Paris and she chooses to disclose her city, to Bob, after anonymization to 'Rennes'. The result of the query issued by Bob will contain Alice who is living in Paris. The returned result could also be not maximum. For example, based on the data given in tables I and II, the result of the query above after transformation is obtained by applying the SPARQL filter *FILTER(?age>=25)* on the data of table III after transformation. Table V shows the result of the query after transformation[3] and table IV shows the expected result. As we can see the result is not maximum. The value of the age of Safaa is in the interval $[28, 43]$ (a k_anonymization of 30 with precision $k = 15$) which is always greater than 25. One way to correct this problem, is to extend the semantic of SPARQL filters. For example, in the case of comparison operators, we have take into account the comparison between numbers and intervals. It depends also on the types of anonymization. Another way is to categorize those types and see which ones are applicable for preserving and implementing the privacy requirements.

As explained above, in the case of normalized query with filters, the result of the rewritten query may not be sound.

VI. Related Works

LeFevre et al. [12] presented an approach that enforces limited disclosure expressed by privacy policy in the case of Hippocratic databases. It is implemented by rewriting queries. When a query Q is issued, it is transformed to Q' so that the result of Q' respects the cell-level disclosure policy P. Their approach is based on replacing all the cells that are not allowed to be seen by P with $NULL$. After that, Q' is evaluated as a normal query with evaluation rules "$NULL \neq NULL$" and "$NULL \neq cst$" for any constant value cst. [12] does not take into account the privacy requirement *accuracy*. They only replace unauthorized data with "NULL". Moreover, [12] does not satisfy the sound property and maximum property [23].

Another issue is that they only mask variables (fields) used in the header of the SQL query. They do not handle the qualification portion. For instance, for the following query *SELECT name FROM Patients WHERE age=25*, they only mask unauthorized *name*. However, if a data-owner Alice, who is

[3] $age \geq 25$ is interpreted as false for each value of age that is not a number

25 years old, choses to disclose her name but not her age, anyone looking at the results concludes that Alice is 25 years old. One way to correct this problem is to normalize the SQL query by adding all variables used in the where clause to the corresponding header of the query, e.g. the query above is normalized as follows: *SELECT name, age FROM Patients WHERE age=25*. Then we rewrite the normalized query and applies filter after transformation so that Alice will not appear in the result.

Oulmakhzoune et al. [16], [17] presented an approach that enforces data confidentiality based on SPARQL query rewriting for select [16] and update [17] queries. The confidentiality policy is modeled as a set of positive and negative filters (corresponding respectively to permissions and prohibitions) that apply to SPARQL queries. They transform the queries so that the results returned by transformed queries are compliant with the confidentiality policy. [16], [17] do not take into account the privacy policy preferences. Our approach could be considered an extension of [16], [17].

The Virtual Private Database (VPD) in Oracle [10] supports the fine-grained access control (FGAC) through policy functions. When a query is issued, it is dynamically modified by appending predicates, returned by the policy function, to the where clause of the query. The approach suggested by Oracle requires to know PL/SQL in order to implement the access control policy. This may lead to security policy complex to define and maintain. Moreover, the policy functions are stored and managed locally.

VII. Conclusion

In this paper we introduced an approach that preserves the privacy preferences by query modification in the case of SPARQL query. We take into account various dimensions of privacy preferences through the concepts of consent, accuracy, purpose and recipient. However, obligations are not taken into account. A possible extension will be to handle provisional obligations.

Our approach satisfies the security criteria. However, the maximality and soundness criteria may not be satisfied in the case of query with filters as illustrated in section V-C. They depend on the semantic of SPARQL filters and how they interpret obfuscated and anonymized data. An interesting future work is to extend the semantic of SPARQL filters, in the case of obfuscated data, in order to preserve the soundness and maximality criteria.

Acknowledgments: This research work is supported by the French National Research Agency project PAIRSE under grant number ANR-09-SEGI-008.

References

[1] Caslon analytics, caslon analytics privacy guide. http://www.caslon.com.au/privacyguide.htm.

[2] European commission, directive 95/46, "the processing of personal data and on the free movement of such data". http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML, October 1995.

[3] European commission, directive 02/58,"privacy and electronic communications". http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2002:201:0037:0047:EN:PDF, July 2002.

[4] Standards for privacy of individually identifiable health information: Final rule. http://www.hhs.gov/ocr/privacy/hipaa/administrative/privacyrule/privrulepd.pdf, August 2002.

[5] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Como, Italy, June 2003.

[6] N. Ajam, N. Cuppens-Boulahia, and F. Cuppens. Contextual privacy management in extended role based access control model. *Data Privacy Management and Autonomous Spontaneous Security*, pages 121–135, 2010.

[7] M. Barhamgi, P.-A. Champin, D. Benslimane, and A. Ouksel. Composing data-providing web services in p2p-based collaboration environments. *19th International Conference on Advanced Information Systems Engineering*, 2007.

[8] N. Bikakis, N. Gioldasis, C. Tsinaraki, and S. Christodoulakis. Semantic based access over xml data. *Visioning and Engineering the Knowledge Society. A Web Science Perspective*, pages 259–267, 2009.

[9] S. Harris, L. Garlik, and A. Seaborne. Sparql 1.1 query language. http://www.w3.org/TR/sparql11-query/, May 2011.

[10] P. Huey. Oracle database security guide : Chapter 7, using oracle virtual private database to control data access. http://download.oracle.com/docs/cd/E14072_01/network.112/e10574.pdf.

[11] G. Klyne and J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[12] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting disclosure in hippocratic databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 108–119. VLDB Endowment, 2004.

[13] A. Masoumzadeh and J. Joshi. Purbac: Purpose-aware role-based access control. *On the Move to Meaningful Internet Systems: OTM 2008*, pages 1104–1121, 2008.

[14] Q. Ni, A. Trombetta, E. Bertino, and J. Lobo. Privacy-aware role based access control. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 41–50. ACM, 2007.

[15] OECD. Organisation for economic co-operation and development. 'protection of privacy and transborder flows of personal data'. September 1980.

[16] S. Oulmakhzoune, N. Cuppens-Boulahia, F. Cuppens, and S. Morucci. fQuery: SPARQL Query Rewriting to Enforce Data Confidentiality. *Proc. of the 24th IFIP WG11.3 Working Conference on Data and Applications Security and Privacy. Rome, Italy*, 21-23 June 2010.

[17] S. Oulmakhzoune, N. Cuppens-Boulahia, F. Cuppens, and S. Morucci. Rewriting of sparql/update queries for securing data access. *International Conference on Information and Communications Security*, pages 4–15, 2010.

[18] E. Prud'Hommeaux and A. Seaborne. Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/, January 2010.

[19] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. *Proc. ACM Sigmod Conf.*, June 2004.

[20] A. Squicciarini, M. Shehab, and F. Paci. Collective privacy management in social networks. In *Proceedings of the 18th international conference on World wide web*, pages 521–530. ACM, 2009.

[21] I. Stavrakantonakis, C. Tsinaraki, N. Bikakis, N. Gioldasis, and S. Christodoulakis. Sparql2xquery 2.0: Supporting semantic-based queries over xml data. In *Semantic Media Adaptation and Personalization (SMAP), 2010 5th International Workshop on*, pages 76–84. IEEE, 2010.

[22] L. Sweeney et al. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty Fuzziness and Knowledge Based Systems*, 10(5):557–570, 2002.

[23] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin, and J. Byun. On the correctness criteria of fine-grained access control in relational databases. In *Proceedings of the 33rd international conference on Very large data bases*, pages 555–566. VLDB Endowment, 2007.

[24] N. Yang, H. Barringer, and N. Zhang. A purpose-based access control model. In *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*, pages 143–148. IEEE, 2007.