BOUNCER: Privacy-aware Query Processing Over Federations of RDF Datasets

Kemele M. Endris[1], Zuhair Almhithawi[2], Ioanna Lytra[2,4], Maria-Esther Vidal[1,3], Sören Auer[1,3]

[1] L3S Research Center, Germany {endris,auer}@L3S.de
[2] University of Bonn, Germany s6zualmh@uni-bonn.de,lytra@cs.uni-bonn.de
[3] TIB Leibniz Information Centre for Science and Technology, Hannover
[4] Fraunhofer IAIS, Germany Maria.Vidal@tib.eu

Abstract. Data provides the basis for emerging scientific and interdisciplinary data-centric applications with the potential of improving the quality of life for the citizens. However, effective data-centric applications demand data management techniques able to process a large volume of data which may include sensitive data, e.g., financial transactions, medical procedures, or personal data. Managing sensitive data requires the enforcement of privacy and access control regulations, particularly, during the execution of queries against datasets that include sensitive and non-sensitive data. In this paper, we tackle the problem of enforcing privacy regulations during query processing, and propose BOUNCER, a privacy-aware query engine over federations of RDF datasets. BOUNCER allows for the description of RDF datasets in terms of RDF molecule templates, i.e., abstract descriptions of the properties of the entities in an RDF dataset and their privacy regulations. Furthermore, BOUNCER implements query decomposition and optimization techniques able to identify query plans over RDF datasets that not only contain the relevant entities to answer a query, but that are also regulated by policies that allow for accessing these relevant entities. We empirically evaluate the effectiveness of the BOUNCER privacy-aware techniques over state-of-the-art benchmarks of RDF datasets. The observed results suggest that BOUNCER can effectively enforce access control regulations at different granularity without impacting the performance of query processing.

1 Introduction

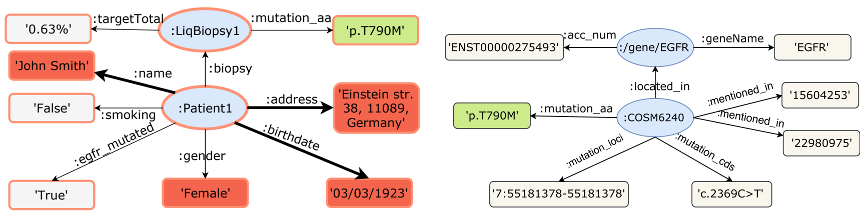
In recent years, the amount of both open data available on the Web and private data exchanged across companies and organizations, expressed as Linked Data, has been constantly increasing. To address this new challenge of effective and efficient data-centric applications built on top of this data, data management techniques targeting sensitive data such as financial transactions, medical procedures, or various other personal data must consider various privacy and access control regulations and enforce privacy constraints once data is being accessed by data consumers. Existing works suggest the specification of Access Control ontologies for RDF data [5,12] and their enforcement on centralized or distributed RDF stores (e.g., [2]) or federated RDF sources (e.g., [8]). Albeit expressive, these approaches are not able to consider privacy-aware regulations during the whole pipeline of a federated query engine, i.e., during source selection, query decomposition, planning, and execution. As a consequence, efficient query plans cannot be devised in a way that privacy-aware policies are enforced.

In this paper, we introduce a privacy-aware federated query engine, called BOUNCER, which is able to enforce privacy regulations during query processing over RDF datasets. In particular, BOUNCER exploits RDF molecule templates, i.e., abstract descriptions of the properties of the entities in an RDF dataset in order to express privacy regulations as well as their automatic enforcement during query decomposition and planning. The novelty of the introduced approach is (1) the granularity of access control regulations that can be imposed; (2) the different levels at which access control statements can be enforced (at source level and at mediator level) and (3) the query plans which include physical operators that enforce the privacy and data access regulations imposed by the sources where the query is executed. The experimental evaluation of the effectiveness and efficiency of BOUNCER is conducted over the state-of-the-art benchmark BSBM for a medium size RDF dataset and 14 queries with different characteristics. The observed results suggest the effective and efficient enforcement of access control regulations during query execution, leading to minimal overhead in time incurred by the introduced access policies.

The remainder of the article is structured as follows. We motivate the privacy-aware federated query engine BOUNCER using a real case scenario from the medical domain in Section 2. In Section 4, we introduce the BOUNCER access policy model and in Section 5 we formally define the query decomposition and query planning techniques applied inside BOUNCER and present the architecture of our federated engine. We perform an empirical evaluation of our approach and report on the evaluation results in Section 6. Finally, we discuss the related work in Section 7 and conclude with an outlook on future work in Section 8.

2 Motivating Example

We motivate our work using a real-world use case from the biomedical domain where data sources from clinical records and genomics data have been integrated into an RDF graph. For instance, Fig.1 depicts two RDF subgraphs or RDF molecules [7]. One RDF molecule represents a patient and his/her clinical information provided by source (S1), while the other RDF molecule models the results of liquid biopsy available in a research institute (S2). The privacy policy enforced at the hospital data source states that *projection (view)* of values is not permitted. Properties name, date of birth, and address of a patient (thicker arrows in Fig.1) are controlled, i.e., query operations are not permitted. Furthermore, it permits a *local join operation* (on premises of the hospital data server) of properties, such as `ex:mutation_aa` - peptide sequence changes that are studied for a patient, `ex:targetTotal` - percentage of circulating tumor DNA in the blood sample of liquid biopsy, `ex:egfr_mutated` - whether the patient has mutations that lead to EGFR over-expression, and `ex:smoking` - whether the patient is a smoker or not. Suppose a user requires to collect the Pubmed ID, mutation name, the genomic coordinates of the mutation and accession numbers of

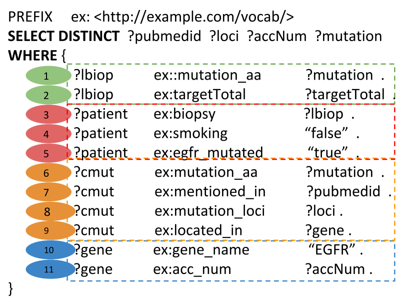


(a) Hospital Data (S1) (b) Research Institute Data (S2)

Fig. 1: **Motivating Example**. Federation of RDF data sources S1 and S2. (a) An RDF molecule representing a lung cancer patient; thicker arrows correspond to controlled properties. (b) An RDF molecule representing the results of a liquid biopsy of a patient. Servers at the hospital can perform join operations.

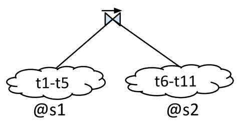
the genes associated with non-smoking lung cancer patients whose liquid biopsy has been studied for somatic mutations that involve EGFR gene amplification (over-expression). Fig.2a depicts a SPARQL query that represents this request; it is composed of 11 triple patterns. The first five triple patterns are executed against S1 while the last six triple patterns are evaluated over S2.

Existing federated query engines are able to generate query plans over these data sources. Fig.2b shows a query execution plan generated by FedX [11] federated query engine for the given query. FedX decomposes the query into two subqueries that are sent to each data source. FedX uses a nested loop join operator to join results from both sources. This operator pushes down the join operation to the data sources by binding the join variables of the right operand of the operator with values extracted from the left operand. First, triple patterns from $t1-t5$ are executed on S1, extracting values for the variables `?mutation_aa`, `?lbiop`, `?targetTotal`, and `?patient`. Then, the shared variable, `?mutation_aa`, is bound and the triple patterns $t6 - t11$ are executed over S2. However, executing this plan yields no answer since the privacy-policy of the hospital does not allow projection of values from the first subquery. Fig.2c shows the query execution plan generated by ANAPSID [1] federated query engine. ANAPSID creates a bushy plan where join operation is performed using GJoin operator (special type of symmetric hash join operator). This operator executes the left and right operands and makes join on the federated engine. In order to check whether the results returned from the subqueries on the left and right operand can be joined, the values of shared variables from both operands have to be checked by ANAPSID, which requires extracting all values for all variables in both sources. This ignores the privacy policy enforced which yields no answer for the given query. The MULDER [7] federated query engine generates a bushy plan and decomposes the query by identifying matching RDF Molecule Templates (PRDF-MTs) as a subquery, as shown in Fig.2d. PRDF-MT is a template that represents a set of RDF molecules that share the same RDF type (rdf:type). MULDER assigns nested hash join operator to join triple patterns $t3 - t5$ associated with `Patient` PRDF-MT and triple patterns $t1 - t2$ that are associated with `Liquid_Biopsy` PRDF-MT. Like in FedX, this operator extracts values for join and projection variables from the left operand, and then binds them to the

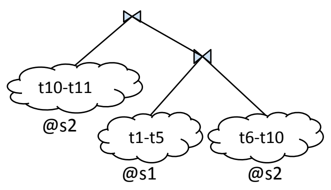


```
PREFIX    ex: <http://example.com/vocab/>
SELECT DISTINCT  ?pubmedid  ?loci  ?accNum  ?mutation
WHERE {
   1  ?lbiop      ex::mutation_aa      ?mutation  .
   2  ?lbiop      ex:targetTotal       ?targetTotal .
   3  ?patient    ex:biopsy            ?lbiop  .
   4  ?patient    ex:smoking           "false"  .
   5  ?patient    ex:egfr_mutated      "true" .
   6  ?cmut       ex:mutation_aa       ?mutation  .
   7  ?cmut       ex:mentioned_in      ?pubmedid  .
   8  ?cmut       ex:mutation_loci     ?loci .
   9  ?cmut       ex:located_in        ?gene .
  10  ?gene       ex:gene_name         "EGFR" .
  11  ?gene       ex:acc_num           ?accNum .
}
```

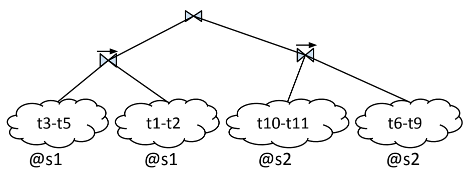
(a) SPARQL Query



(b) FedX Query Plan



(c) ANAPSID Query Plan



(d) MULDER Query Plan

Fig. 2: **Motivating Example**. (a) A SPARQL query composed of four star-shaped subqueries accessing controlled and public data from S1 and S2. (b) FedX generates a plan with two subqueries. (c) ANAPSID decomposed the query into three subqueries. (d) MULDER identifies a plan with four star-shape subqueries. None of the query plan respects privacy policies of S1 and S2.

same variables of the right operand. Like FedX and ANAPSID plans, the MULDER plan also ignores the privacy policy enforced at the hospital data source, which would yield an empty query answer. All of these federated engines fail to answer the query, because they ignore the privacy policy of the data sources during query decomposition as well as query execution plan generation (e.g., wrong join ordering). Also, MULDER ignores the privacy policy of the hospital during query decomposition and splits the triple patterns from this source. This leads to trying to extract results on the federation system which is not possible because of the restrictions enforced by the hospital. In addition to the join order problem, ANAPSID selects a wrong join operator which requires data from S1 to be projected for the restricted properties, i.e., $t1 - t5$. In this paper, we present BOUNCER a privacy-aware federated query engine able to identify plans that respect the above-mentioned privacy and access control policies.

3 Problem Statement and Proposed Solution

In this section, we formalize the problem of privacy-aware query decomposition over a federation of RDF data sources. First we define a set of privacy-aware predicates that represent the type of operations that can be performed over an RDF dataset according to the access regulations of the federation.

Definition 1 (Privacy-Aware Operations). *Given a federated query engine $\mathcal{M}$, a federation $\mathcal{F}$ of RDF datasets D, and a dataset D_i in D. Let p_{ij} be an RDF* *property with domain the RDF class C_{ij}. The set of operations to be executed by $\mathcal{M}$ against $\mathcal{F}$ is defined as follows:*

- *join_local(D_i, p_{ij}, C_{ij}) - this predicate indicates that the join operation on property p_{ij} can be performed on the dataset D_i.*
- *join_fed(D_i, p_{ij}, C_{ij}) - this predicate indicates that the join operation on property p_{ij} can be performed by $\mathcal{M}$. The truth value of join_fed(D_i, p_{ij}, C_{ij}) implies to the truth value of join_local(D_i, p_{ij}, C_{ij}).*
- *project(D_i, p_{ij}, C_{ij}) - this predicate indicates that the values of the property p_{ij} can be projected from dataset D_i. The truth value of project(D_i, p_{ij}, C_{ij}) implies to the truth value of join_fed(D_i, p_{ij}, C_{ij}).*

Definition 2 (Access Control Theory). *Given a federated query engine $\mathcal{M}$, a set of RDF datasets $D = \{D_1, .., D_n\}$ of a federation $\mathcal{F}$. An Access Control Theory is defined as the set of privacy-aware operations that can be performed on property p_{ij} of RDF class C_{ij} over dataset D_i in D.*

The access control theory for the federation described in our running example of Fig.2a can be defined as a conjunction of the following operations:

- *join_local(s1, ex:mutation_aa, Liquid_Biopsy)*,
- *join_local(s1, ex:biopsy, Patient)*, *project(s2, ex:located_in, Mutation)*,
- *join_local(s1, ex:targetTotal, Liquid_Biopsy)*, *project(s2, ex:acc_num, Gene)*,
- *join_local(s1, ex:smoking, Patient)*, *join_local(s1, ex:egfr_mutated, Patient)*,
- *project(s2, ex:mutation_aa, Mutation)*,*project(s2, ex:gene_name, Gene)*,
- *project(s2,ex:mutation_loci,Mutation)*,*project(s2,ex:mentioned_in,Mutation)*.

Note that the RDF properties `:name`, `:gender`, `:address`, and `:birthdate` of the `Patient` RDF class do not have operations defined in the access control theory. In our approach this fact indicates that these properties are controlled and any operation on these properties performed by the federated engine is forbidden.

Property 1. Given a property p_{ij} of an RDF class C_i from a dataset D_i in a federation $\mathcal{F}$ and an access control theory T. If there is no privacy-aware predicate in T that includes p_{ij}, then p_{ij} is a *controlled property* and no federation engine can perform operations over p_{ij} against D_i.

A basic graph pattern (BGP) in a SPARQL query is defined as a set of triple patterns $\{t_1, \ldots, t_n\}$. A BGP contains one or more triple patterns that involve a variable being projected from the original SELECT query. We call these triple patterns *projected triple patterns*, denoted as $PTP = \{t_1, \ldots, t_m\}$ such that $PTP \subseteq BGP$. A BGP includes at least one star-shaped subquery (SSQ), i.e., $BGP = \{SSQ_1, \ldots, SSQ_n\}$. A star-shaped subquery is a set of triple patterns that share the same subject variable or object [13]. Furthermore, an SSQ may contain zero or more triple patterns that involve a variable which is being projected from the original SELECT query. We call these triple patterns *projected triple patterns of an SSQ*, denoted as $PTS = \{t_1, \ldots, t_k\}$ where $PTS_i \subseteq SSQ_i$. Let PRJ be a set of triple patterns that involve a variable being projected from the original SELECT query, then projected triple patterns of a BGP, is a subset of PRJ, i.e., $PTP \subseteq PRJ$ and a projected triple pattern of SSQ_i is a subset of PTP, i.e., $PTS_i \subseteq PTP$. For example, in our running example, there is only one BGP, $BGP_1 = \{t_1, \ldots, t_{11}\}$, for which projected variables belong to triple patterns, $PRJ = \{t_6, t_7, t_8, t_{11}\}$. Projected triple patterns of BGP_1 are the same as PRJ, $PTP_{BGP_1} = \{t_6, t_7, t_8, t_{11}\}$, since there is only one BGP. Furthermore, BGP_1 can be clustered into four start-shaped subqueries, $SSQs_{BGP_1} = \{SSQ_{1=\{t_1-t_2\}}$, $SSQ_{2=\{t_3-t_5\}}, SSQ_{3=\{t_6-t_9\}}$, $SSQ_{4=\{t_{10}-t_{11}\}}\}$. Out of four $SSQs$ of BGP_1, only the last two $SSQs$ have triple patterns that are also in the projected triple patterns, i.e., $PTS_{SSQ_1} = \emptyset$, $PTS_{SSQ_2} = \emptyset$,$PTS_{SSQ_3} = \{t_6, t_7, t_8\}$, $PTS_{SSQ_4} = \{t_{11}\}$.

Property 2. Given a SPARQL query Q such that a variable $?v$ is associated with a property p of a triple pattern t in a BGP and $?v$ is projected in Q. Suppose an access control theory T regulates the access of the datasets in D of the federation $\mathcal{F}$. A federation engine $\mathcal{M}$ accepts Q iff there is a privacy-aware operation $project(D_i, p, C)$ in T for at least an RDF dataset D_i in D.

A privacy-aware query decomposition on a federation is defined. This formalization states the conditions to be met by a decomposition in order to be evaluated over a federation by enforcing their access regulations.

Definition 3 (Privacy-Aware Query Decomposition). *Let BGP be a basic graph pattern, PTP a set of projected triple patterns of a BGP, T an access control theory, and $D = \{D_1, \ldots, D_n\}$ a set of RDF datasets of a federation $\mathcal{F}$. A privacy-aware decomposition P of BGP in D, $\gamma(P|BGP, D, T, PTP)$, is a set of decomposition elements, $\Phi = \{\phi_1, .., \phi_k\}$, such that ϕ_i is a four-tuple, $\phi_i = (SQ_i, SD_i, PS_i, PTS_i)$, where:*

- *SQ_i is a subset of triple patterns in BGP, i.e., $SQ_i \subseteq BGP$, and $SQ_i \neq \emptyset$, such that there is no repetition of triple patterns, i.e., If $t_a \in SQ_i$, then $!\exists t_a \in SQ_j : SQ_j \subset BGP \wedge i \neq j$,*
- *SD_i is a subset of datasets in D, i.e, $SD_i \subseteq D$, and $SD_i \neq \emptyset$,*
- *PS_i is a set of privacy-aware operations that are permitted on triple patterns in SQ_i to be performed on datasets in SD_i and $PS_i \subseteq T$, and $PS_i \neq \emptyset$,*
- *PTS_i is a set of triple patterns in SQ_i that contains variables being projected from the original SELECT query, i.e., $PTS_i \subseteq SQ_i \wedge PTS_i \subseteq PTP$,*
- *The set composed of SQ_i in the decompositions $\phi_i \in \Phi$ corresponds to a partition of BGP and*
- *The selected RDF datasets are able to project out the attributes in the project clause of the query, i.e., $\forall t_a \in SQ_i : t_a \in PTP$, then $project(D_a, p_{aj}, C_{aj}) \in PS_i$ where $t_a = (s, p_{aj}, o)$, $D_a \in SD_i$, and $SQ_i \in \phi_i$.*

After defining what is a decomposition of a query, we state the problem of finding a suitable decomposition for a query and a given set of data sources.

Privacy-Aware Query Decomposition Problem. Given a SPARQL query Q, RDF datasets D=$\{\mathrm{D}_1, \ldots, \mathrm{D}_m\}$ of a federation $\mathcal{F}$, and access control theory T. The *problem of decomposing* Q in D restricted by T is defined as follows. For all BGPs, BGP=$\{\mathrm{t}_1, \ldots, \mathrm{t}_n\}$ in Q, find a query decomposition $\gamma(P|BGP, D, T, PTP)$ that satisfies the following conditions:

- The evaluation of $\gamma(P|BGP, D, T, PTP)$ in D is *complete* according to the privacy-aware policies of the federation in T. Suppose D^* represents the maximal subset of D where the privacy policies of each RDF dataset $D_i \in D^*$ allow for projecting and joining the properties from D_i that appear in Q[5]. Then the evaluation of BGP in D^* is equivalent to the evaluation of $\gamma(P|BGP, D, T, PTP)$ and the following expression holds:

$$[[BGP]]_{D*} = [[\gamma(P|BGP, D, T, PTP)]]_D$$

- The cost of executing the query decomposition $\gamma(P|BGP, D, T, PTP)$ is *minimal*. Suppose the execution time of a decomposition P' of BGP in D is represented as $cost(\gamma(P'|BGP, D, T, PTP))$, then

$$\gamma(P|BGP, D, T, PTP) = \underset{\gamma(P'|BGP,D,T,PTP)}{\operatorname{argmin}} cost(\gamma(P'|BGP, D, T, PTP))$$

To solve this problem, we present BOUNCER, a federated query engine able to identify query decompositions for SPARQL queries and query plans that efficiently evaluate SPARQL queries over a federation. Two definitions are presented for a query plan over a decomposition. The next two functions are presented in order to facilitate the understanding of the definition of a query plan.

Definition 4 (The property function prop(*)). *Given a set of triple patterns, TPS, the function $prop(TPS)$ is defined as follows:*

$$prop(TPS) = \{p \mid (s, p, o) \in TPS \wedge p \text{ is constant } \}$$

Definition 5 (The variable function var(*)). *Given a privacy-aware decomposition, Φ, the function $var(\Phi)$ is defined inductively as follows:*

1. ***Base case:*** *$\Phi = \{\phi_1\}$, then $var(\Phi) = \{?x \mid (s, p, o) \in SQ_1$, where $\phi_1 = (SQ_1, SD_1, PS_1, PTS_1)$, $?x = s \wedge s$ is a variable $\vee$ $?x = o \wedge o$ is a variable$\}$*
2. ***Inductive case:*** *Let Φ_1 and Φ_2 be disjoint decompositions such that Φ=$\Phi_1 \cup \Phi_2$ then, $var(\Phi) = var(\Phi_1) \cup var(\Phi_2)$.*

Definition 6 (A Valid Plan over a Privacy-Aware Decomposition). *Given a privacy-aware decomposition $\gamma(P|BGP, D, T, PTP)$: $\Phi = \{\phi_1, \ldots, \phi_n\}$, a valid query plan, $\alpha(\Phi)$, is defined inductively as follows:*

1. ***Base Case:*** *If only one decomposition ϕ_1 belongs to Φ, i.e., $\Phi = \{\phi_1\}$, the plan unions all the service graph patterns over the selected RDF sources. Thus, $\alpha(\Phi) = UNION_{d_i \in SD_1}(SERVICE\ d_i\ SQ_1)$ is a valid plan*[6][7]*, where:*
   - *$\phi_1 = (SQ_1, SD_1, PS_1, PTS_1)$ is a valid privacy-aware decomposition;*
   - *All the variables projected in the query have the permission to be projected, i.e., $\forall p_{i1} \in prop(PTS_1)$, $project(Di, pi1, Ci1) \in PS_1$.*

[5] Predicates $project(Di, p_{ij}, C_{ij})$, $join\_fed(Di, p_{ij}, C_{ij})$ and $join\_local(Di, p_{ij}, C_{ij})$ are part of T for all properties in triple patterns in Q that can be answered by Di.
[6] For readability, UNION$_{di \in SD+i}$ represents SPARQL UNION operator
[7] $SERVICE$ corresponds to the SPARQL SERVICE clause

2. ***Inductive Case:*** *Let Φ_1 and Φ_2 be disjoint decompositions such that $\Phi=\Phi_1 \cup \Phi_2$. Then, $\alpha(\Phi) = (\alpha(\Phi_1) * \alpha(\Phi_2))$ is a valid plan, where:*
   (a) *$\alpha(\Phi_1)$ and $\alpha(\Phi_2)$ are valid plans.*
   (b) *The join variables appear jointly in the triple patterns of Φ_1 and Φ_2, i.e., $joinVars = var(\Phi_1) \cap var(\Phi_2)$.*
   (c) *$\mathcal{J}$ is a set of joint triple patterns involving join variables in BGP:*
      - $\mathcal{J} = \{t | variable(t) \subseteq joinVars, (t \in \Phi_{1(SQ)} \vee t \in \Phi_{2(SQ)})\}$
      - $\Phi_{1(SQ)} = \{SQ_i | \forall \phi_i \in \Phi_1, \phi_i = (SQ_i, SD_i, PS_i, PTS_i)\}$, *and*
      - $\Phi_{2(SQ)} = \{SQ_j | \forall \phi_j \in \Phi_2, \phi_j = (SQ_j, SD_j, PS_j, PTS_j)\}$.
   (d) *The operator * is a JOIN operator, i.e., $\alpha(\Phi) = (\alpha(\Phi_1)$ JOIN $\alpha(\Phi_2))$ is a valid plan, iff $\forall p_{ij} \in prop(\mathcal{J})$, $join\_fed(D_i, p_{ij}, C_{ij}) \in (\Phi_{1(PS)} \cap \Phi_{2(PS)})$, $\Phi_{1(PS)} = \{PS_i | \forall \phi_i \in \Phi_1, \phi_i = (SQ_i, SD_i, PS_i, PTS_i)\}$, and $\Phi_{2(PS)} = \{PS_j | \forall \phi_j \in \Phi_2, \phi_j = (SQ_j, SD_j, PS_j, PTS_j)\}$.*
   (e) *The operator * is a DJOIN operator, i.e., $\alpha(\Phi) = (\alpha(\Phi_1)$ DJOIN $\alpha(\Phi_2))$ is a valid plan iff $\forall p_{ij} \in prop(\mathcal{J})$, $join\_fed(D_i, p_{ij}, C_{ij}) \in \Phi_{1(PS)}$ and $join\_local(D_i, p_{ij}, C_{ij}) \in \Phi_{2(PS)}$[8].*

Next, we define the BOUNCER architecture and the main characteristics of the query decomposition and execution tasks implemented by BOUNCER.

4 BOUNCER: A Privacy-Aware Engine

Web interfaces provide access to RDF datasets, and can be described in terms of resources and properties in the datasets. BOUNCER employs privacy-aware RDF Molecule Templates for describing and enforcing privacy policies.

Definition 7 (Privacy-Aware RDF Molecule Template(PRDF-MT)). *A privacy-aware RDF molecule template (PRDF-MT) is a 5-tuple=<WebI, C, DTP, IntraL, InterL>, where:*
- *WebI – is a Web service API that provides access to an RDF dataset G via SPARQL protocol;*
- *C – is an RDF class such that the triple pattern (?s rdf:type C) is true in G;*
- *DTP – is a set of triples (p, T, op) such that p is a property with domain C and range T, the triple patterns (?s p ?o) and (?o rdf:type T) and (?s rdf:type C) are true in G, and op is an access control operator that is allowed to be performed on property p;*
- *IntraL – is a set of pairs (p,C_j) such that p is an object property with domain C and range C_j, and the triple patterns (?s p ?o) and (?o rdf:type C_j) and (?s rdf:type C) are true in G;*
- *InterL – is a set of triples (p,C_k,SW) such that p is an object property with domain C and range C_k; SW is a Web service API that provides access to an RDF dataset K, and the triple patterns (?s p ?o) and (?s rdf:type C) are true in G, and the triple pattern (?o rdf:type C_k) is true in K.*

[8] DJOIN- is a dependent JOIN [14].

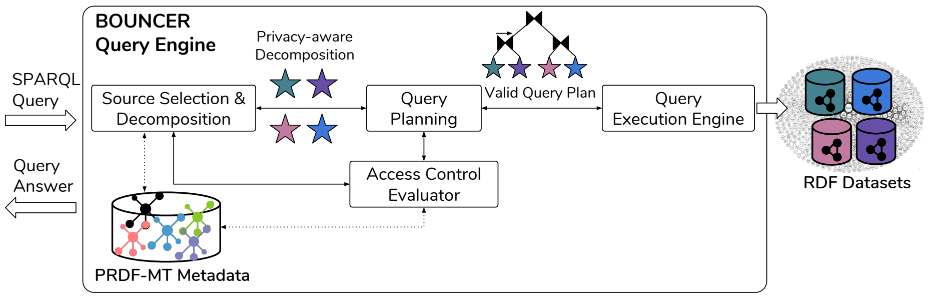


Fig. 3: **BOUNCER Architecture**. BOUNCER receives a SPARQL query and outputs the results of executing the SPARQL query over a federation of SPARQL endpoints. It relies on PRDF-MT descriptions and privacy-aware policies to select relevant sources, and perform query decomposition and planning. The query engine executes a valid plan against the selected sources.

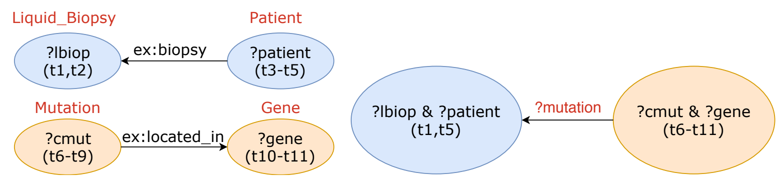
Fig.3 depicts BOUNCER architecture. Given a SPARQL query, the source selection and query decomposition component solves the problem of identifying a privacy-aware query decomposition; they select PRDF-MTs for subqueries (SSQs) by consulting PRDF-MT metadata store and the access control evaluator component. The source selection and decomposition component is privacy-aware decomposition; it is given to the query planning component for creating a valid plan, i.e., access policies of the selected data sources should be respected. The valid plan is executed in a bushy-tree fashion by the query execution.

5 Privacy-Aware Decomposition and Execution

This section presents the privacy-aware techniques implemented by BOUNCER. They rely on the description of the RDF datasets of a federation in terms of privacy-aware RDF molecule templates (PRDF-MTs) to identify query plans that enforce data access control regulations. More importantly, these techniques are able to generate query execution plans whose operators force the execution of queries at the dataset sites in case data cannot be transferred or accessed.

5.1 Privacy-Aware Source Selection and Decomposition

The BOUNCER privacy-aware source selection and query decomposition is sketched in Algorithm 1. Given a *BGP* in a SPARQL query Q, BOUNCER first decomposes the query into star-shaped subqueries (SSQs), (Line 2). For instance, our running example query, in Fig.2a, is decomposed into four SSQs, as shown in Fig.4, i.e., SSQs around the variables `?lbiop`, `?patient`, `?cmut`, and `?gene`, respectively. The first SSQ (denoted `?lbiop`-SSQ) has two triple patterns, t1-t2, the second SSQ (`?patient`-SSQ) is composed of three triple patterns, t3-t5, the third SSQ (`?cmut`-SSQ) includes four triple patterns, and the fourth SSQ (`?gene`-SSQ) is composed of two triple patterns, t10-t11.



(a) Initial query decomposition (b) Privacy-aware Query Decomposition

Fig. 4: **Example of Privacy-Aware Decompositions**. Decompositions for SPARQL query in the motivating example. Nodes represent SSQs and colors indicate datasets where they are executed; edges correspond to join variables. a) Initial query decomposed into four SSQs. b) Decomposition result where the subqueries `?lbiop`-SSQ and `?patient`-SSQ are composed into a single subquery to comply with the privacy policy of data source S1, while `?cmut`-SSQ and `?gene`-SSQ are also composed to push down the join operation to the data source S2.

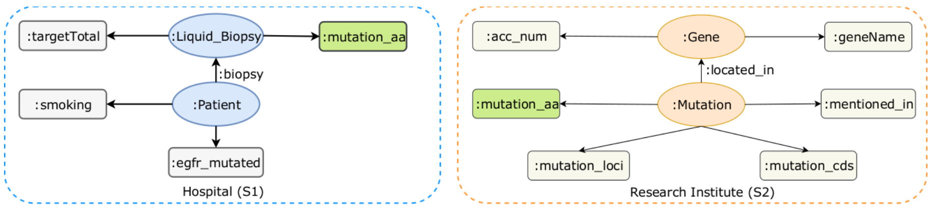


Fig. 5: **Example of Privacy-aware RDF Molecule Templates (PRDF-MTs)**. Two PRDF-MTs for the SPARQL query in the motivating example. According to the privacy regulations the properties **:name**, **:birthdate**, and **:addresss** are controlled; they do not appear in the PRDF-MTs.

Fig.4a presents an initial decomposition with the selected PRDF-MTs for each SSQs. The subquery `?patient`-SSQ is joined to the subquery `?lbiop`-SSQ via `ex:biopsy` property. Similarly, `?cmut`-SSQ is joined to `?gene`-SSQ via the `ex:located_in` property. Given the set of properties in each SSQ and the joins between them, BOUNCER finds a matching PRDF-MT for each SSQs (Line 3), i.e., it matches the subqueries `?patient`-SSQ, `?lbiop`-SSQ, `?cmut`-SSQ, and `?gene`-SSQ to the PRDF-MTs `Patient`, `Liquid_Biopsy`, `Mutation`, and `Gene`, respectively. Once the PRDF-MTs are identified for the SSQs, BOUNCER verifies the access control policies associated with them (Line 4). A subquery SSQ associated with an PRDF-MT(s) that grants the `project()` permission to all of its properties is called *Independent SSQ*; otherwise, it is called *Dependent SSQ*. An SSQ in a SPARQL query Q is called *dependent* iff a property of at least one triple pattern in SSQ is associated with the privacy-aware operation `join_local()`. On the other hand, an SSQ is *independent* iff the privacy-aware operation `project()` is true for the properties of the triple patterns in SSQ.

If the value of the controlled property is in the projection list, i.e., if the property of a triple pattern in an SSQ have `join_local()` or `join_fed()` predicate, then the decomposition process exits with empty result (Line 6). Once the SSQs are associated with PRDF-MTs, the next step is to merge the SSQs with

Algorithm 1 Privacy-Aware Query Decomposition: BG - Basic Graph Pattern, Q - Query, $PRMT$ - Access-aware RDF Molecule Templates

```
1: procedure DECOMPOSE($BGP$, $Q$, $PRMT$)
2:     $SSQs \leftarrow getSSQs(BGP)$                          ▷ Partition the BGP to SSQs
3:     $RES \leftarrow selectSource(PRMT, PRMT)$   ▷ RES=[(SSQ, PRMT, DataSource)]
4:     $A \leftarrow getAccessPolicies(RES)$; $\Phi \leftarrow [\,]$; $DR \leftarrow \{\,\}$ ▷ access control statements
5:     for $(SSQ, RMT, p, ds, pred) \in A$ do
6:         if $p \in Query.PRJ \wedge pred\,!= project(ds, p, RMT.type)$ then return [ ]
7:         $DR[SSQ][PTS].append(t) \mid t = (s, p, o) \wedge t \in SSQ \mid p \in Query.PRJ$
8:         $DR[SSQ][SD].append(ds) \wedge DR[SSQ][PS].append(pred)$
9:     end for
10:    for $(SSQ_i, SD_i, PS_i, PTS_i) \in DR$ do
11:        $\phi_i = (SQ_i, SD_i, PS_i, PTS_i) \mid SQ_i \leftarrow SSQ_i$
12:        if $join\_local() \in PS_i$ then            ▷ If $SSQ_i$ contains restricted property
13:            for $(SSQ_j, SD_j, PS_j, PTS_j) \in DR$ do
14:                if $SD_i \cap SD_j \not\equiv \emptyset$ then
15:                    $\phi_i.extend(SSQ_j, SD_j, PS_j, PTS_j)$
16:                    $DR.remove((SSQ_j, SD_j, PS_j, PTS_j)) \wedge done \leftarrow True$
17:            end for
18:            if $NOT\ done$ then return [ ]
19:        end if
20:        $\Phi.append(\phi_i)$
21:    end for
22:    return $\Phi$                                            ▷ decomposed query
23: end procedure
```

the same source and push down the join operation to the data source. To comply with access control policies of a dataset, i.e., when the properties of an SSQ have only the `join_local()` permission, the join operation with this SSQ should be done at the data source. Hence, if two SSQs can be executed at the same source, then BOUNCER decomposes them as a single subquery (SQ) (Line 10-21). This technique may also improve query execution time by performing join operation at the source site. Fig.4b shows a final decomposition for our running example. `?lbiop`-SSQ and `?patient`-SSQ are merged because they are dependent and the join operation can be executed at the source.

5.2 BOUNCER Privacy-Aware Query Planning Technique

Algorithm 2 sketches the BOUNCER privacy-aware query planing technique. Given a privacy-aware decomposition Φ of a query Q, BOUNCER finds a valid plan that respects the privacy-policy of the data sources. For each subquery in ϕ_i a service-graph pattern is created (Line 4 & 6) and the SPARQL UNION operator is used whenever the subquery can be executed over more than one data source. Then, BOUNCER selects another subquery, ϕ_j that is joinable with ϕ_i (Line 5). If ϕ_i is composed of dependent SSQ(s) (resp., independent SSQ(s)) and ϕ_j is composed of an independent SSQ(s) (resp., dependent SSQ(s)), then a dependent join operator (DJOIN) is selected (Line 9-12). If both ϕ_i and ϕ_j are merged of an independent SSQ(s), then any JOIN operator can be chosen (Line

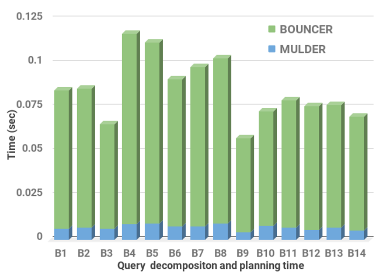
Algorithm 2 Query Planning over Privacy-Aware Decomposition: Φ - Privacy-Aware query decomposition, Q - SELECT query

```
 1: procedure makePlan($\Phi$, $Q$)
 2:     $\alpha \leftarrow []$
 3:     for $\phi_i \in \Phi$ do
 4:         $\sigma_1 \leftarrow UNION_{d_i \in SD_i \wedge SD_i \in \phi_i}(SERVICE\ d_i\ \ SQ_i)$
 5:         for $\phi_j \in \Phi \mid \phi_i \neq \phi_j \wedge var(SQ_i)\ \cap\ var(SQ_j)\ \not\equiv\ \emptyset$ do        ▻ If joinable
 6:             $\sigma_2 \leftarrow UNION_{d_j \in SD_j}(SERVICE\ d_j\ \ SQ_j)$
 7:             $\mathcal{J} \leftarrow \{\ t \mid vari(t) \subseteq [var(SQ_i) \cap\ var(SQ_j)] \wedge t \in [SQ_i \cup SQ_j]\}$
 8:             $\rho \leftarrow prop(\mathcal{J})$        ▻ Properties of join variables
 9:             if $\exists join\_local() \in PS_i \wedge \forall pred_{p\in\rho} \in PS_j \mid pred_{p\in\rho} \Rightarrow join\_fed()$ then
10:               $\alpha.append((\sigma_2\ \ DJOIN\ \ \sigma_1))$; $joined \leftarrow True$        ▻ Dependent JOIN
11:             if $\exists join\_local() \in PS_j \wedge \forall pred_{p\in\rho} \in PS_i \mid pred_{p\in\rho} \Rightarrow join\_fed()$ then
12:               $\alpha.append((\sigma_1\ \ DJOIN\ \ \sigma_2))$; $joined \leftarrow True$        ▻ Dependent JOIN
13:             if $\forall pred_{p\in\rho} \in [PS_i \cup PS_j] \mid pred_{p\in\rho} \Rightarrow join\_fed()$ then
14:               $\alpha.append((\sigma_1\ \ JOIN\ \ \sigma_2))$; $joined \leftarrow True$        ▻ Independent JOIN
15:         end for
16:         if $\exists join\_local() \in PS_i \wedge\ NOT\ joined$ then return [ ]        ▻ No valid plan
17:     end for
18:     return $\alpha$
19: end procedure
```

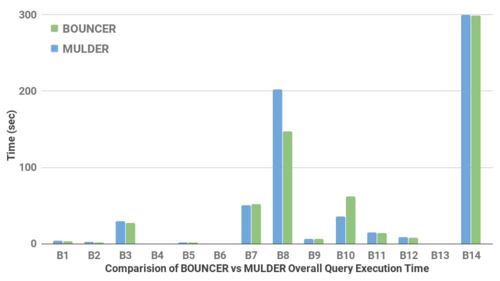
13-14). Finally, otherwise, an empty plan is returned indicating that there is no valid plan for the input query (Line 16).

6 Empirical Evaluation

We study the efficiency and effectiveness of BOUNCER. First, we assess the impact of access-control policies enforcement and BOUNCER is compared to ANAPSID, FedX, and MULDER. Then, the performance of BOUNCER is



(a) Decompositon and Planning



(b) Overall Execution

Fig. 6: **Decomposition and Execution Time**. BOUNCER decomposition and planning are more expensive than baseline (MULDER), but BOUNCER generates more efficient plans and overall execution time is reduced.

evaluated. We study the following research questions: **RQ1)** Does privacy-aware enforcement employed during source selection, query decomposition, and planning impact query execution time? **RQ2)** Can privacy-aware policies be used to identify query plans that enhance execution time and answer completeness?
Benchmarks: The Berlin SPARQL Benchmark (*BSBM*) generates a dataset of 200M triples and 14 queries; answer size is limited to 10,000 per query.
Metrics: *i*) `Execution Time`: Elapsed time between the submission of a query to an engine and the delivery of the answers. Timeout is set to 300 seconds. *ii*) `Throughput`: Number of answers produced per second; this is computed as the ratio of the number of answers to execution time in seconds.
Implementation: BOUNCER privacy-aware techniques are implemented in Python 3.5 and integrated into the ANAPSID query engine. The BSBM dataset is partitioned into 8 parts (one part per RDF type) and deployed on one machine as SPARQL endpoints using Virtuoso 6.01.3127, where each dataset resides in a dedicated Virtuoso docker container. Experiments are executed on a Dell PowerEdge R805 server, AMD Opteron 2.4GHz CPU, 64 cores, 256GB RAM.
Experiment 1: Impact of Access Control Enforcement. The impact of privacy-aware processing techniques is studied, as well as the overhead on source selection, decomposition, and execution. In this experiment, the privacy-aware theory enables all the operations over the properties of the federation, i.e., all the operations are defined for each property and dataset. MULDER and BOUNCER are compared; Fig.6 reports on decomposition, planning, and execution time per query. Both engines generate the same results and BOUNCER consumes more time in query decomposition and planning. However, the overall execution time is lower in almost all queries. These results suggest that even there is an impact on query processing, BOUNCER is able to exploit privacy-aware polices, and generates query plans that speed up query execution.
Experiment 2: Impact of Privacy-Aware Query Plans. The privacy-aware query plans produced by BOUNCER are compared to the ones generated by state-of-the-art query engines. In this experiment, the privacy-aware theory enables local joins for `Person`, `Producer`, `Product`, and `ProductFeature`, and projections of the properties of `Offer`, `Review`, `ProductType`, and `Vendor`. Fig.7 reports on the throughput of each query engine. As observed, the query engines produced different query plans which allow for high performance. However, many of these plans are not valid, i.e., they do not respect the privacy-aware policies in the theory. For instance, ANAPSID produces bushy tree plans around `gjoins`; albeit efficient, these plans violate the privacy policies. FedX and MULDER are able to generate some valid plans–*by chance*– but fail in producing efficient executions. On the contrary, BOUNCER generates valid plans that in many cases increase the performance of the query engine. Results observed in two experiments suggest that efficient query plans can be identified by exploiting the privacy policies; thus, **RQ1** and **RQ2** can be positively answered.

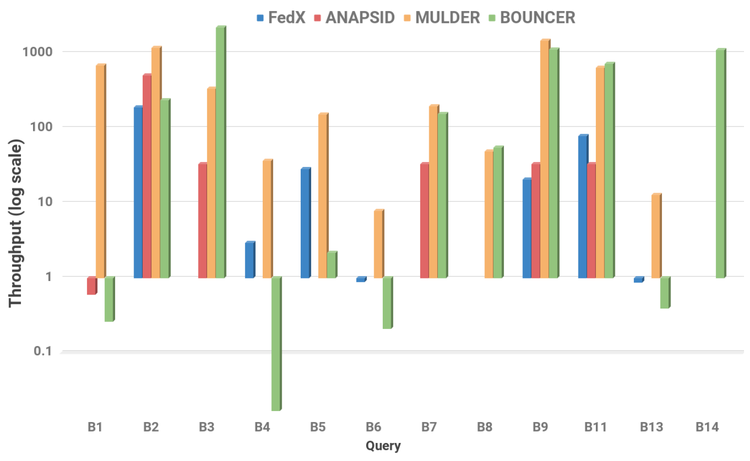


Fig. 7: **Efficiency of Query Plans**. Existing engines are compared based on throughput. ANAPSID plans are efficient but no valid. FedX and MULDER generate valid plans (by chance) but some are not efficient. BOUNCER generates both valid and efficient plans and overall execution time is reduced.

7 Related Work

The data privacy control problem has received extensive attention by the Database community; approaches by De Capitani et al. [6] and Bater et al. [3] are exemplars that rely on an authority network to produce valid plans. Albeit relevant, these approaches are not defined for federated systems; thus, the tasks of source selection and query decomposition are not addressed. BOUNCER also generates valid plans, but being designed for SPARQL endpoint federations, it also ensures that only relevant endpoints are selected to evaluate these valid plans. The Semantic Web community has also explored access control models for SPARQL query engines; RDF named graphs [5,8,12] and quad patterns [9] are used to enforce access control policies. Most of the work focuses on the specification of access control ontologies and enforcement on RDF data [5,12] stored in a centralized RDF store, while others explore access control specification and enforcement on distributed RDF stores [2,4] and federated query processing [8,10] scenarios. Costabello et al. [5] present SHI3LD, an access control framework for RDF stores accessed on mobile devices; it provides a pluggable filter for generic SPARQL endpoints that enforces context-aware access control at named graph level. Kirane et al. [9] propose an authorization framework that relies on stratified Datalog rules to enforce access control policies; RDF quad patterns are used to model permissions (grant or deny) on named graphs, triples, classes, and properties. Ubehauen et al. [12] propose an access control approach at the level of named graphs; it binds access control expressions to the context of RDF triples and uses a query rewriting method on an ontology for enabling the evaluation of privacy regulations in a single query. SAFE [8] is designed to query statistical RDF data cubes in distributed settings and also enables graph level access control. BOUNCER is a privacy-aware federated engine where policies are defined over RDF properties of PRDF-MTs; it also enables access control statements at source and mediator level. More important, BOUNCER generates query plans that both enforce privacy regulations and speed up execution time.

8 Conclusion and Future Work

We presented BOUNCER, a privacy-aware federated query engine for SPARQL endpoints. BOUNCER relies on privacy-aware RDF Molecule Templates (PRDF-MTs) for source description and guiding query decomposition and plan generation. Efficiency of BOUNCER was empirically evaluated, and results suggest that it is able to reduce query execution time and increase answer completeness by producing query plans that comply with the privacy policies of the data sources. In future work, we plan to integrate additional Web access interfaces, like RESTful APIs, and empower PRDF-MTs with context-aware access policies.

Acknowledgements This work has been funded by the EU H2020 RIA under the Marie Skłodowska-Curie grant agreement No. 642795 (WDAqua) and EU H2020 Programme for the project No. 727658 (IASIS).

References

1. M. Acosta, M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *ISWC*, 2011.
2. M. Amini and R. Jalili. Multi-level authorisation model and framework for distributed semantic-aware environments. *IET Information Security*, 4(4), 2010.
3. J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers. Smcql: secure querying for federated databases. *Proceedings of VLDB Endowment*, 10(6), 2017.
4. P. A. Bonatti and D. Olmedilla. Rule-based policy representation and reasoning for the semantic web. In *Reasoning Web International Summer School*, 2007.
5. L. Costabello, S. Villata, and F. Gandon. Context-aware access control for rdf graph stores. In *ECAI-20th European Conference on Artificial Intelligence*, 2012.
6. S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Authorization enforcement in distributed query evaluation. *JCS*, 19(4), 2011.
7. K. M. Endris, M. Galkin, I. Lytra, M. N. Mami, M. Vidal, and S. Auer. MULDER: querying the linked data web by bridging RDF molecule templates. In *DEXA*, 2017.
8. Y. Khan, M. Saleem, M. Mehdi, A. Hogan, Q. Mehmood, D. Rebholz-Schuhmann, and R. Sahay. SAFE: SPARQL Federation over RDF Data Cubes with Access Control. *Journal of Biomedical Semantics*, 8(1), 2017.
9. S. Kirrane, A. Abdelrahman, A. Mileo, and S. Decker. Secure manipulation of linked data. In *ISWC*. Springer, 2013.
10. M. Kost and J.-C. Freytag. Swrl-based access policies for linked data. 2010.
11. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *ISWC*, 2011.
12. J. Unbehauen, M. Frommhold, and M. Martin. Enforcing scalable authorization on SPARQL queries. In *SEMANTiCS (Posters, Demos, SuCCESS)*, 2016.
13. M. Vidal, E. Ruckhaus, T. Lampo, A. Martínez, J. Sierra, and A. Polleres. Efficiently joining group patterns in SPARQL queries. In *ESWC*, 2010.
14. V. Zadorozhny, L. Raschid, M. Vidal, T. Urhan, and L. Bright. Efficient evaluation of queries in a mediator for websources. In *ACM SIGMOD*, 2002.