Query-based Linked Data Anonymization

Rémy Delanaux, Angela Bonifati, Marie-Christine Rousset, Romuald Thion

To cite this version:

Rémy Delanaux, Angela Bonifati, Marie-Christine Rousset, Romuald Thion. Query-based Linked Data Anonymization. The 17th International Semantic Web Conference (ISWC 2018), Oct 2018, Monterey, United States. pp.530-546, <http://iswc2018.semanticweb.org/>. <hal-01896276>

HAL Id: hal-01896276
https://hal.archives-ouvertes.fr/hal-01896276

Submitted on 16 Oct 2018

Query-based Linked Data Anonymization

Remy Delanaux[1], Angela Bonifati[1], Marie-Christine Rousset[2,3], and Romuald Thion[1]

[1] Université Lyon 1, LIRIS CNRS, Villeurbanne, France
[name].[surname]@univ-lyon1.fr
[2] Université Grenoble Alpes, CNRS, INRIA, Grenoble INP, Grenoble, France
[name].[surname]@imag.fr
[3] Institut Universitaire de France, Paris, France

Abstract. We introduce and develop a declarative framework for privacy-preserving Linked Data publishing in which privacy and utility policies are specified as SPARQL queries. Our approach is data-independent and leads to inspect only the privacy and utility policies in order to determine the sequence of anonymization operations applicable to any graph instance for satisfying the policies. We prove the soundness of our algorithms and gauge their performance through experiments.

1 Introduction

Linked Open Data (LOD) provides access to continuously increasing amounts of RDF data that describe properties and links among entities referenced by means of Uniform Resource Identifiers (URIs). Whereas many organizations, institutions and governments participate to the LOD movement by making their data accessible and reusable to citizens, the risks of identity disclosure in this process are not completely understood. As an example, in smart city applications, information about users' journeys in public transportation can help re-identify the individuals if they are joined with other public data sources by leveraging quasi-identifiers.

The main problem for data providers willing to publish useful data is to determine which anonymization operations must be applied to the original dataset in order to preserve both individuals' privacy and data utility. For all these reasons, data providers should have at their disposal the means to readily anonymize their data prior to publication into the LOD cloud. The majority of the solutions proposed so far and mainly devoted to relational legacy systems rely on variants of differential privacy as surveyed in [14] or k-anonymity proposed by [18]. Differential privacy offers strong mathematical guarantees of non-disclosure of any factual information by adding noise to the data, with as a counterpart a low utility of answers returned by precise queries (as opposed to statistical queries). In a similar fashion, several k-anonymization methods have been developed that transform the original dataset into clusters containing at least k records with indistinguishable values over quasi-identifiers. When taken into account, the utility loss is defined by a metric that measures and minimizes the information loss between the original dataset and the output of the anonymization algorithm. All these approaches fall short in empowering the data providers with the capability of *specifying their own privacy and utility policies*, and in managing anonymization operations as *update operations* on the data.

In this paper, we present a novel declarative framework that *(i)* allows the data providers to specify as queries both the privacy and utility policies they want to enforce, *(ii)* checks whether the specified policies are compatible with each other, and *(iii)*, based on a set of basic update queries, automatically builds candidate sets of anonymization operations that are guaranteed to transform any input dataset into a dataset satisfying the required privacy and utility policies. We provide an algorithm that implements this *data-independent method* starting from privacy and utility policies only, and we prove its soundness.

We believe that our framework is tailored for data publishing in the LOD in which it is important to strike a balance between non-disclosure of information (likely to serve as quasi-identifiers when interconnected with LOD links) and utility preservation for end-users querying the LOD. Our framework is carefully designed to meet all these requirements by leveraging at the same time the expressive power of SPARQL queries and the maturity and effectiveness of SPARQL query engines. It builds on the common wisdom that queries from data providers are more and more available through online SPARQL endpoints [3]. Such queries are a valuable resource to understand the real utility needs of users on publicly available data and to guide the data providers in safe RDF data publishing. For all these reasons, our approach paves the way to a democratization of privacy-preserving mechanisms for LOD data.

The paper is organized as follows. Section 2 reviews related work. Section 3 summarizes preliminaries for explaining the query-based model of privacy and utility that we propose in Section 4. Section 5 and Section 6 describe respectively our query-based static anonymization method and its experimental assessment. Finally, we conclude the paper in Section 7 with further research directions.

2 Related work

Privacy preserving data publishing (PPDP) has been a long-standing research goal for several research communities, as witnessed by a flurry of work on the topic [7]. A rich variety of privacy models have been proposed, ranging from k-anonymity [18] and l-diversity [15] to t-closeness [13] and ϵ-differential privacy [6]. For each of the aforementioned methods, one or more attack models (such as record linkage, attribute linkage, table linkage and probabilistic attack) are considered, amounting to make two fundamental assumptions: (i) what an attacker is assumed to know about the victim and (ii) under which conditions a privacy threat occurs.

Most of these models, first conceived for relational databases, have been recently extended to the setting of the Semantic Web [12]. Among them, the privacy model that is definitely the closest to our work is k-anonymity that has been recently adapted to RDF graphs in [17, 10]. These works focus on defining operations of generalization, suppression or perturbation to apply to values in the range of properties known to be quasi-identifiers for persons identification, along with metrics to measure the resulting loss of information. Our approach is more generic than theirs and also fully declarative since, by leveraging the logical foundations of PPDP for Linked Data in [8], it allows the definition of fine-grained privacy policies specified by queries, and to obtain candidate sets of anonymization operations allowing to practically enforce the requested privacy without loosing the desired utility.

Contrarily to [8], which focuses on the computational complexity of checking whether privacy requirements are fulfilled in Linked Data, we leverage utility policies as queries for which anonymization operations must preserve the original answers, and we employ the interactions of privacy and utility policies in a static analysis method. To the best of our knowledge, our framework is the first to provide practical algorithms for building candidate sequences of atomic operations, described as an open research challenge in [8].

An alternative approach to anonymization for protecting against privacy breaches consists in applying access control methods to Linked Data ([16, 19, 11]). In the Semantic Web setting, when data are described by description logics ontologies, preliminary results on role-based access control have been obtained in [1] for the problem of checking whether a sequence of role changes and queries can infer that an anonymous individual is equal to a known individual. Compared to access control techniques that perform verification at runtime, we focus on a static analysis approach executed only once and guaranteeing that the published datasets do not contain sensitive information.

3 Preliminaries

We introduce the standard notions and concepts for RDF graphs and SPARQL queries. Let $\mathbf{I}$, $\mathbf{L}$ and $\mathbf{B}$ be countably infinite pairwise disjoint sets representing respectively *IRIs*, *literal values* (or *literals*) and *blank nodes*. IRIs (Internationalized Resource Identifiers) are standard identifiers used for denoting any Web resource described in RDF within the LOD. We denote by $\mathbf{T} = \mathbf{I} \cup \mathbf{L} \cup \mathbf{B}$ the set of *terms*, in which we distinguish *constants* (IRIs and literal values) from blank nodes. We also assume an infinite set $\mathbf{V}$ of variables disjoint from the above sets. In the examples, variables in $\mathbf{V}$ are prefixed with a question mark as in the SPARQL language.

Definition 1 (RDF graph). *An* RDF graph *is a finite set of* RDF triples (s, p, o)*, where* $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{B})$.

IRIs appearing in position p into triples denote properties composing the *schema* of the RDF graph.

The queries we consider in our work are built on *graph patterns* that are made of triples with constants and variables (blank nodes are not allowed).

Definition 2 (Graph pattern). *A* triple pattern *is a triple* $(s, p, o) \in (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$*. A* graph pattern *is a finite set of triple patterns.*

We can now define the two types of queries under study along with their answers. The first type of queries correspond to the standard notion of *conjunctive queries*, while the second type corresponds to *counting queries* that are the basis for simple analytical tasks.

Definition 3 (Conjunctive query). *A* conjunctive query *Q is defined by an expression* `SELECT` $\bar{x}$ `WHERE` $G(\bar{x}, \bar{y})$ *where $G(\bar{x}, \bar{y})$ is a graph pattern and $\bar{x} \cup \bar{y}$ is the set of its variables, among which $\bar{x}$ are the result (also called the* distinguished*) variables. A conjunctive query Q is alternatively written as $\langle \bar{x}, G\rangle$.*

The evaluation of a query $\langle \bar{x}, G\rangle$ over an RDF graph DB consists in finding mappings μ assigning the variables in G to terms such that the set of triples, denoted $\mu(G)$, obtained by replacing with $\mu(z)$ each variable z appearing in G, is included in DB. The corresponding answer is defined as the tuple of terms $\mu(\bar{x})$ assigned by μ to the result variables.

Definition 4 (Evaluation of a conjunctive query). *Let Q be a conjunctive query defined by $\langle \bar{x}, G\rangle$, and let DB an RDF graph. The* answer set *of Q over DB is defined by :* $\mathsf{Ans}(Q, DB) = \{\mu(\bar{x}) \mid \mu(G) \subseteq DB\}$

Definition 5 (Counting query). *Let Q be a conjunctive query. The query* $\mathsf{Count}(Q)$ *is a* counting query*, whose answer over a graph DB is defined by:* $\mathsf{Ans}(\mathsf{Count}(Q), DB) = |\,\mathsf{Ans}(Q, DB)|$

4 Query-based policies and anonymization operations

Following [8], a privacy policy, represented by a set of conjunctive queries, satisfies the anonymization process if none of the sensitive answers holds in the resulting dataset. This is achieved by letting the privacy queries return no answer or, alternatively, answers with blank nodes, as shown in the remainder. We also model utility policies by sets of queries that can be either conjunctive queries or counting queries useful for data analytics. For satisfying an utility policy, the anonymization process must preserve the answers of all the specified utility queries. We now formally define privacy and utility policies.

Definition 6 (Privacy and utility policies). *Let DB be an input RDF graph, a* privacy *(resp.* utility*)* policy *$\mathcal{P}$ (resp. $\mathcal{U}$) is a set of conjunctive queries (resp. conjunctive or counting queries). Let* $\mathsf{Anonym}(DB)$ *be the result of an anonymization process of the graph DB by a sequence of anonymization operators.*

A privacy policy $\mathcal{P}$ is satisfied *on* $\mathsf{Anonym}(DB)$ *if for every $P \in \mathcal{P}$ and for any tuple of constants $\bar{c}$, it holds that: $\bar{c} \notin \mathsf{Ans}(P, \mathsf{Anonym}(DB))$. An utility policy $\mathcal{U}$ is* satisfied *on* $\mathsf{Anonym}(DB)$ *if for every $U \in \mathcal{U}$ it holds that:* $\mathsf{Ans}(U, \mathsf{Anonym}(DB)) = \mathsf{Ans}(U, DB)$.

As usual, we call $|\mathcal{P}|$ (resp. $|\mathcal{U}|$) the *cardinality* of the policy. For a policy $\mathcal{P}$ (resp. $\mathcal{U}$) made of n queries $P_i = \langle \bar{x}_i^P, G_i^P\rangle$ (resp. m queries $U_i = \langle \bar{x}_i^U, G_i^U\rangle$) we call the sum of the cardinalities of their bodies the *size of the policy* defined by $\sum_{i=1}^{n} |G_i^P|$ (resp. $\sum_{i=1}^{m} |G_i^U|$).

The following running example shows that privacy and utility policies might impose constraints on overlapping portions of a dataset.

Example 1. Consider a privacy policy $\mathcal{P} = \{P_1, P_2\}$ on data related to public transportation in a given city, and defined by the two following conjunctive queries written in concrete SPARQL syntax. The first privacy query expresses that travelers' postal addresses are sensitive and shall be protected, and the second privacy query specifies that the disclosure of users identifiers associated with geolocation information (like latitude and longitude as given by the user ticket validation) may also pose a risk (for re-identification by data linkage with other LOD datasets).

```
# Privacy query P1
SELECT ?ad
WHERE {
  ?u  a    tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

```
# Privacy query P2
SELECT ?u ?lat ?long
WHERE {
  ?c  a    tcl:Journey.
  ?c  tcl:user       ?u.
  ?c  geo:latitude   ?lat.
  ?c  geo:longitude  ?long.
}
```

As a consequence, any query displaying either users' addresses or users' identifiers together with their geolocation information would infringe this privacy policy, violating the anonymization of the underlying dataset to be published as open data. The counterpart utility policy is the set of queries $\mathcal{U} = \{U_1, U_2\}$. This set states that users' ages and location related to journeys are to be preserved.

```
# Utility query U1
SELECT ?u ?age
WHERE {
  ?u  a   tcl:User.
  ?u  foaf:age  ?age.
}
```

```
# Utility query U2
SELECT ?c ?lat ?long
WHERE {
  ?c  a    tcl:Journey.
  ?c  geo:latitude   ?lat.
  ?c  geo:longitude  ?long.
}
```

Regarding anonymization operations, we extend the notion of suppression functions considered in [8] that replace IRIs with blank nodes by allowing also triple deletions. The anonymization operations that we consider correspond to **update queries** (Definition 7): when evaluated against an RDF graph DB, the update query `DELETE` $D(\bar{x})$ `INSERT` $I(\bar{y})$ `WHERE` $W(\bar{x}, \bar{z})$ suppresses all occurrences of $D(\bar{x})$ in DB such that $W(\bar{x}, \bar{y})$ can be mapped to a subgraph of DB, and inserts triples corresponding to the pattern $I(\bar{y})$. Note that $\bar{y}$ may contain *existential variables*, i.e., variables that do not appear in $W(\bar{x}, \bar{z})$, and thus cannot be mapped to terms present in DB. This would lead to add triples with blank nodes.

Definition 7 (Update query). *An* update query *(or* update operation*)* Q_{upd} *is defined by* `DELETE` $D(\bar{x})$ `INSERT` $I(\bar{y})$ `WHERE` $W(\bar{x}, \bar{z})$ *where* D *(resp.* I*,* W*) is a graph pattern which set of variables is* $\bar{x}$ *(resp.* $\bar{y}$*,* $\bar{x} \cup \bar{z}$*). The result of its evaluation over an RDF graph* DB *is defined by:*

$$\mathsf{Result}(Q_{upd}, DB) = DB \backslash \{\mu(D(\bar{x})) | \mu(W(\bar{x}, \bar{y})) \subseteq DB\} \cup \{\mu'(I(\bar{y})) | \mu(W(\bar{x}, \bar{z})) \subseteq DB\}$$

where μ' *is an extension of* μ *to fresh blank nodes, i.e. a mapping such that*

$$\mu'(x) = \begin{cases} \mu(x) \ \textit{when } x \in \bar{x} \cup \bar{z} \\ b_{new} \in \mathbf{B} \ \textit{otherwise} \end{cases}$$

A **deletion query** Q_{del} is a particular case of update query where the insertion pattern $I(\bar{y})$ is empty.

In the following section, we will focus on two kinds of **atomic anonymization operations** that correspond respectively to **triple deletions** (i.e., particular case of Definition 7 where $D(\bar{x})$ is reduced to a triple pattern) and **replacement of IRIs by blank nodes** (i.e., particular case of Definition 7 where $D(\bar{x})$ and $I(\bar{y})$ are triple patterns that differ just by the fact that one bound variable of $D(\bar{x})$ is replaced with an existential variable in $I(\bar{y})$).

These two atomic anonymization operations are illustrated in Examples 2 and 3 respectively. From now, by slight abuse of notation w.r.t Definition 7, we will use the SPARQL standard notation `[]` for denoting single existential variables.

Example 2. In the setting of Example 1 related to transportation data, the following query specifies the operation deleting the addresses of users.

```
DELETE  { ?u  vcard:hasAddress   ?ad. }
WHERE   { ?u  a tcl:User.
          ?u  vcard:hasAddress   ?ad.}
```

Example 3. In the same context, this query replaces users' identifiers related to a ticket validation by a blank node.

```
DELETE  { ?c  tcl:user   ?u. }
INSERT  { ?c  tcl:user   []. }
WHERE   { ?c  a tcl:Journey.
          ?c  tcl:user       ?u.
          ?c  geo:latitude   ?lat.
          ?c  geo:longitude  ?long. }
```

5 Finding candidate sets of anonymization operations

Given privacy and utility policies, the problems of interest that we address in this paper are named Compatibility and EnumOperations. Both problems are generic as they are essentially built on the query-based definition of policy satisfaction, hence they are applicable to larger classes of operations and queries.

Problem 1. The Compatibility problem.
Input : $\mathcal{P} = \{P_i\}$ a privacy policy and $\mathcal{U} = \{U_j\}$ a utility policy
Output: `True` if *there exists* a sequence of operations O such that $O(DB)$ satisfies both $\mathcal{P}$ and $\mathcal{U}$ for any DB and `False` otherwise.

Problem 2. The EnumOperations problem.
Input : $\mathcal{P} = \{P_i\}$ a privacy policy and $\mathcal{U} = \{U_j\}$ a utility policy
Output: The set $\mathcal{O}$ of all sequences of operations O such that $O(DB)$ satisfies both $\mathcal{P}$ and $\mathcal{U}$ for any DB.

An algorithm that solves the EnumOperations problem solves the Compatibility problem as well, by checking whether its output is $\emptyset$.

The rest of this section is devoted to the design of Algorithm 2 that solves the EnumOperations problem using update operations (Definition 7) when $\mathcal{P}$ and $\mathcal{U}$ are defined by conjunctive queries (Definition 3). We also define an intermediate step dealing with unitary privacy policies, with Algorithm 1. Note that Algorithm 2 produces a set of *sets* of operations and not a set of *sequences*. As we guarantee that the *sets* of operations hereby computed solve the problem, any sequence obtained by reordering these sets would work as well. Hence, the difference between sets and sequences of operations is fairly immaterial.

If the answer set of Q is preserved by an anonymization process so does its cardinality, implying that any solution for a non-counting query Q is also a solution for its counting counterpart $\mathsf{Count}(Q)$. Similarly, if a utility query Q is satisfied, then its counting counterpart $\mathsf{Count}(Q)$ is also satisfied. Therefore, we focus on non-counting queries in Algorithm 1. However, the opposite implication does not hold, hence we may miss some operation that may guarantee a utility counting query $\mathsf{Count}(Q)$ without guaranteeing a utility non-counting query Q.

5.1 Finding candidate sets of operations for unitary privacy policies

We start with the case where the privacy policy is unitary, i.e. when it is reduced to a singleton $\mathcal{P} = \{P\}$. Intuitively, Algorithm 1 tries to find edges that are in the graph pattern G^P of the privacy policy $\mathcal{P}$ but in none of the utility policy graph patterns G^U_j. For each such an edge, a delete operation is constructed, and possible update operations are considered. Update operations take place in two manners: either the subject of the triple is replaced with a blank node, or its object is replaced with a blank node if it is an IRI. In both cases, the algorithm looks for three alternatives:

- The triple is part of a path of length ≥ 2 in the privacy graph pattern G^P, and therefore the update operation breaks the path thus satisfying the privacy policy P;
- The replaced subject (resp. object) is also the subject (resp. object) of another triple in the privacy query graph G^P and the update operation breaks the link between these triples, hence satisfying the privacy policy P;

- The replaced subject (resp. object) of the triple is also part of the distinguished variables $\bar{x}$ of the privacy policy query, leading to a blank value in the query results.

The soundness of this algorithm is encapsulated in Theorem 1. Due to space constraints, proofs are available in an online appendix.[4] We define the following helper functions that check if update operations are possible:

$$\begin{aligned}\texttt{check-subject}((s,p,o),G) = {} & \exists(s',p',s) \in G \vee \\ & (\exists(s,p',o') \in G \wedge \nexists\sigma\ (\sigma(s,p',o') = \sigma(s,p,o))) \\ \texttt{check-object}((s,p,o),G) = {} & \exists(o,p',o') \in G \vee \\ & (\exists(s',p',o) \in G \wedge \nexists\sigma\ (\sigma(s',p',o) = \sigma(s,p,o)))\end{aligned}$$

Algorithm 1: Find update operations to satisfy a unitary privacy policy

```
Input  : a unitary privacy policy $\mathcal{P} = \{P\}$ with $P = \langle \bar{x}^P, G^P \rangle$
Input  : a utility policy $\mathcal{U}$ made of $m$ queries $U_j = \langle \bar{x}^U_j, G^U_j \rangle$
Output: a set of operations $O$ satisfying both $\mathcal{P}$ and $\mathcal{U}$
function find-ops-unit($P$,$\mathcal{U}$):
    Let $H$ be the graph $G^P$ with all its variables replaced by fresh ones[a];
    Let $O := \emptyset$;
    forall $(s,p,o) \in H$ do
        Let $c :=$ true;
        forall $G^U_j$ do
            forall $(s',p',o') \in G^U_j$ do
                if $\exists\sigma\ (\sigma(s',p',o') = \sigma(s,p,o))$ then
                    $c :=$ false;
        if $c$ then
            $O := O \cup \{$DELETE $\{(s,p,o)\}$ WHERE $H\}$;
            if check-subject$((s,p,o),H) \vee s \in \bar{x}^P$ then
                $O := O \cup \{$DELETE $\{(s,p,o)\}$ INSERT $\{($[]$,p,o)\}$ WHERE $H\}$;
            if $o \in \mathbf{I} \wedge ($check-object$((s,p,o),H) \vee o \in \bar{x}^P)$ then
                $O := O \cup \{$DELETE $\{(s,p,o)\}$ INSERT $\{(s,p,$[]$)\}$ WHERE $H\}$;
    return $O$;
```

[a] i.e., with variables that do not appear in any G^U_j

Theorem 1 (Soundness of Algorithm 1). *Let $\mathcal{P}$ be a privacy policy consisting of a single query and let $\mathcal{U}$ be a utility policy. Let $O =$*`find-ops-unit`$(\mathcal{P},\mathcal{U})$ *computed by Algorithm 1. For all $o \in O$, for all RDF graph DB, $\mathcal{P}$ and $\mathcal{U}$ are satisfied by $o(DB)$ obtained by applying the update operation o to DB.*

The behavior of Algorithm 1 is illustrated in the following Example 4.

[4] See https://liris.cnrs.fr/~rdelanau/papers/ISWC2018_appx.pdf

Example 4 (Example 1 cont'd). Consider the policies $\mathcal{P} = \{P_1, P_2\}$ and $\mathcal{U} = \{U_1, U_2\}$ given in Example 1 with bodies G_1^P, G_2^P, G_1^U and G_2^U, respectively. Let us consider two different runs of Algorithm 1. The call to `find-ops-unit(`P_1`,`$\mathcal{U}$`)` produces the following set O_1 of operations whereas the call to `find-ops-unit(`P_2`,`$\mathcal{U}$`)` produces O_2:

$$O_1 = \{\texttt{DELETE}\ \{(?u, \texttt{vcard:hasAddress}, ?ad)\}\ \texttt{WHERE}\ G_1^P,$$

$$\texttt{DELETE}\ \{(?u, \texttt{vcard:hasAddress}, ?ad)\}\ \texttt{INSERT}\ \{(\texttt{[]}, \texttt{vcard:hasAddress}, ?ad)\}\ \texttt{WHERE}\ G_1^P,$$

$$\texttt{DELETE}\ \{(?u, \texttt{vcard:hasAddress}, ?ad)\}\ \texttt{INSERT}\ \{(?u, \texttt{vcard:hasAddress}, \texttt{[]})\}\ \texttt{WHERE}\ G_1^P\}$$

$$O_2 = \{\texttt{DELETE}\ \{(?c, \texttt{tcl:user}, ?u)\}\ \texttt{WHERE}\ G_2^P,$$

$$\texttt{DELETE}\ \{(?c, \texttt{tcl:user}, ?u)\}\ \texttt{INSERT}\ \{(\texttt{[]}, \texttt{tcl:user}, ?u)\}\ \texttt{WHERE}\ G_2^P,$$

$$\texttt{DELETE}\ \{(?c, \texttt{tcl:user}, ?u)\}\ \texttt{INSERT}\ \{(?c, \texttt{tcl:user}, \texttt{[]})\}\ \texttt{WHERE}\ G_2^P\}$$

Indeed, there is only one way to satisfy P_1, U_1 and U_2: delete or update the address $?ad$ of each user $?u$ as shown in O_1. This goes by either deleting it, replacing the address value by a blank node in the `hasAddress` triple (possible since $?ad$ is also a distinguished variable), or replacing the user with a blank node (possible since there is another triple originating from the user variable $?u$ in the policy query body). Notice that the update or deletion of the triple `{?u a tcl:User}` is not authorized, because U_1 would not be satisfied.

The only acceptable operations for P_2, U_1 and U_2 as shown in O_2, are either to delete the link between users and their journeys, or replace each argument of this relation with a blank node. Replacing the subject of the considered triple (the journey variable $?c$) is possible since it is also featured as the subject of other triples in the query body, while replacing the object (the user variable $?u$) is possible since it is a distinguished variable of the privacy query.

5.2 Finding candidate sets of operations for general privacy policies

We now extend the previous algorithm to the general case where $\mathcal{P}$ is a set of n queries. The idea is to compute operations that satisfy each P_i using Algorithm 1 and then to distribute the results. The soundness of this algorithm is encapsulated in Theorem 2 and its associated Corollary 1.

Algorithm 2: Find update operations to satisfy policies

```
Input  : a privacy policy $\mathcal{P}$ made of $n$ queries $P_i = \langle \bar{x}_i^P, G_i^P \rangle$
Input  : a utility policy $\mathcal{U}$ made of $m$ queries $U_j = \langle \bar{x}_j^U, G_j^U \rangle$
Output: a set of sets of operations $Ops$ such that each sequence
        obtained from ordering any $O \in Ops$ satisfies both $\mathcal{P}$ and $\mathcal{U}$
function find-ops($\mathcal{P}$,$\mathcal{U}$):
    Let $Ops = \{\emptyset\}$;
    for $P_i \in \mathcal{P}$ do
        Let $ops_i :=$ find-ops-unit($P_i$,$\mathcal{U}$);
        if $ops_i \neq \emptyset$ then $Ops := \{O \cup \{o'\} \mid O \in Ops \land o' \in ops_i\}$;
    return $Ops$;
```

Theorem 2 (Soundness of Algorithm 2). *Let $\mathcal{P}$ be a privacy policy and let $\mathcal{U}$ be a utility policy. Let $\mathcal{O} = \texttt{find-ops}(\mathcal{P},\mathcal{U})$ and let DB be an RDF graph. For any set of operations $O \in \mathcal{O}$, and for any ordering S of O, $\mathcal{P}$ and $\mathcal{U}$ are satisfied by $S(DB)$ obtained by applying to DB the sequence of operations in S.*

Theorem 2 guarantees the soundness of *all* sequences of operations that can be built from the output of Algorithm 2. Corollary 1 leverages this result for the Compatibility problem.

Corollary 1. *Let $\mathcal{P}$ be a privacy policy and let $\mathcal{U}$ be a utility policy made of counting and non-counting queries. If $\texttt{find-ops}(\mathcal{P},\mathcal{U}) \neq \emptyset$ then the* Compatibility *problem has* `True` *as a solution.*

Algorithm 2 guarantees the same robustness to linking attacks as [8]: for any anonymization $DB' = \mathit{Anonym}(DB)$ produced using Algorithm 2, its union with any RDF graph G satisfying the same privacy policy $\mathcal{P}$ will also satisfy $\mathcal{P}$. The reason is that the IRIs possibly common to DB' and G cannot be the images of a mapping from any privacy query. Indeed, the operations that have produced DB' have either deleted triples corresponding to triples in a privacy query or have replaced IRIs involved in mappings from privacy queries to DB with blank nodes (which cannot be joined with blank nodes in G).

The sets of operations produced by Algorithm 2 are *not* equivalent in the sense that they may delete *different sets* of triples in the dataset. Moreover, even for a *given* set of operations, the choice of a possible reordering of its operations may have different effects on the dataset. Indeed, deletions and modifications of triples are not commutative operations but due to the soundness of the algorithm, every obtained solution *satisfies* the privacy and utility policies.

Regarding the complexity of Algorithm 1, its result $O = \texttt{find-ops-unit}(P,\mathcal{U})$ grows linearly with the size of P. Indeed, each triple in the body G^P of P produces at most one delete operation and two update operations. However, regarding the overall complexity of Algorithm 2, if each set O of operations $O \in \mathcal{O} = \texttt{find-ops}(\mathcal{P},\mathcal{U})$ has cardinality $|\mathcal{P}|$ by construction, the distribution of the results obtained by `find-ops-unit` on line 4 induces an exponential blowup on the size of $\mathcal{O}$ due to the cartesian product on Line 5. In our experimental assessment (Section 6), we will show that in practice the utility and privacy queries in $\mathcal{P}$ and $\mathcal{U}$ oftentimes overlap, thus decreasing drastically the actual number of sequences output by Algorithm 2, possibly to none.

6 Experimental Study

In this section, we present an empirical study devoted to gauge the efficiency of our main algorithm (Algorithm 2) and measure various factors that determine the impact of the overlap and size of the policy queries on its output. The experimental study is organized into three main parts: (1) experimental analysis of the risk of incompatibility between privacy and utility policies; (2) experimental evaluation of the impact of the privacy and utility policies on the number of anonymizations alternatives produced by Algorithm 2 ; (3) experimental evaluation of Algorithm 2 runtime performance.

Setup and Implementation. We adopted gMark [2], a schema-based synthetic graph and query workload generator, as a benchmark for our experimental study. We used gMark to define the schema of public transportation data, by including types and properties observed in real-world smart city open data platforms[5]. Due to the static nature of our approach, we only need to use such a schema to generate query workloads without the need of generating actual graph instances.

Precisely, we defined a schema with 13 data types and 12 properties capturing information regarding users (including personal data and subscription data for cardholders), ticket validations and user rides (such as geographic coordinates of ticket validations and optional subscription-related data), and information on the transportation network (such as maps). Using gMark, we then built a sample of 500 randomly generated conjunctive queries upon the aforementioned schema, each one containing between 1 and 6 distinguished variables with a size ranging between 1 and 6 triples. As shown in a recent study [3], queries of such size are the most frequent ones in a large corpus of real-world query logs extracted from SPARQL endpoints. This further corroborates our assumption that our query sample is representative of real-world queries formulated by end-users. To account for the structural variability of real-world queries, experiments were performed on workloads using different shapes of queries: chain queries, star queries, star-chain queries and a random mix of star-chain and star queries. For space reasons, we present the results for star-chain queries only. The full list of experiments is available in a notebook at the project's GitHub repository[6].

To generate privacy and utility policies, we fix a number of conjunctive queries to be part of the privacy and utility policies. Then, we randomly pick as many queries as necessary in the query sample to build the policies based on this cardinality, while avoiding duplicates in the same policy and in between both kinds of policies.

In all our experiments, we have opted for a balanced cardinality between privacy and utility policies: we have set the policy cardinality equal to 3 for the experiments in Sections 6.1 and 6.2, whereas Section 6.3 features a more extreme case for performance testing with policy cardinality equal to 10. Depending on the experiment, policy size (i.e. the sum of the sizes of the conjunctive queries defining it) may vary since the picked queries have a varying size from 1 to 6.

The *overlap degree* between privacy and utility policies plays an important role in our experiments as a factor likely to impact the results of Algorithm 2. We define it as the ratio between the number of triples appearing in privacy queries that can be mapped to a triple appearing in a utility query and the total size of the privacy policy. More formally, let $\mathcal{P} = \{P_i\}$ and $\mathcal{U} = \{U_j\}$ be privacy and utility policies. The overlap degree between $\mathcal{P}$ and $\mathcal{U}$ is a real number in $[0 \ldots 1]$ defined as:

[5] Notably, the Grand Lyon data website and datasets: https://data.grandlyon.com/

[6] https://github.com/RdNetwork/Declarative-LOD-Anonymizer

$$\frac{\sum_{i=1}^{n} |\{t \in G_i^P \mid \exists j \; \exists t' \in G_j^U \; \exists \mu \; \mu(t) = \mu(t')\}|}{\sum_{i=1}^{n} |G_i^P|}$$

Algorithm 2 returns $\emptyset$ as output when it is applied to privacy and utility policies having an overlap degree equal to 1, which are thus incompatible. In Section 6.1, we will measure the risk of incompatibility between randomly generated privacy and utility policies by counting the number of cases where this complete overlap occurs.

All our tests have been performed under Windows 10 on a Intel® Core™ i5-6300HQ CPU machine running at 2.30GHz and 8GB of RAM. We have implemented our algorithms using Python 2.7. The code of our working prototype along with the datasets and results of our experiments are made open-source and available at the aforementioned project's GitHub repository.

6.1 Measuring compatibility between privacy and utility policies

Our goal is to measure the incompatibility rate of privacy and utility policies randomly generated with a fixed cardinality of 3 and a varying size.

We have performed two experiments where we vary the size of the privacy (resp. utility) policy from 6 to 12, which corresponds to privacy (resp. utility) queries having between 2 and 4 triples, while keeping the size of the utility (resp. privacy) policy fixed to 9, which corresponds to utility (resp. privacy) queries with 3 triples. In the first (resp. second) experiment, for each of the 7 privacy (resp. utility) policy sizes, we launch 200 executions of Algorithm 2 and we count the number of executions returning $\emptyset$, which allows to compute the proportion of incompatible policies. For space reasons, we omit the corresponding histograms (available in our online notebook) and we describe the obtained results in the following.

In both experiments, we observed that only **49.2% and 49.3%** of the 1400 (i.e., corresponding to 200 runs multiplied by 7 data points) executions exhibit compatible policies. This result clearly shows the necessity of designing an algorithm which automatically verifies policy incompatibility prior to the the anonymization process. It also reveals that even small policy cardinalities (equal to 3 for privacy and utility queries) can already substantially prevent possible anonymizations.

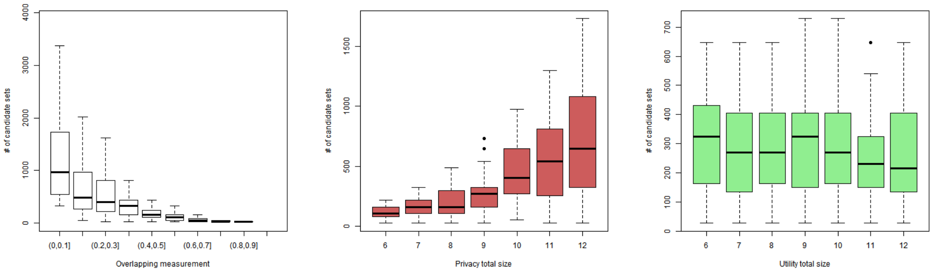
We also noted in the first experiment that the compatibility rate between privacy and utility policies tends to grow with the privacy policy size. This behavior is in clear contrast with the intuition that the more privacy policy is constrained, the less flexibility we have in satisfying them. The explanation however is that increasing the size of the privacy policy decreases the risk that all its triples are mapped with triples in the (fixed size) utility policy, and thus augments the possibilities of satisfying the privacy and utility policies by triple deletions.

We observe the opposite trend in the second experiment: the compatibility rate between privacy and utility policies decreases with the utility policy size.

The reason is that requiring more utility for end-users restrains the possibilities of deleting data for anonymization purposes.

6.2 Measuring the number of anonymization alternatives

When applied to compatible privacy and utility policies, Algorithm 2 computes the set of all the candidate sets of update operations that satisfy the input policies. In the worst case, the number of candidate sets corresponds the product of the sizes of the privacy queries. In this experiment, we want to evaluate how this number evolves in practice depending on (1) the overlap between privacy and utility policies, and (2) the total size of the privacy and utility policies.



(a) Depending on overlap (b) Depending on privacy size (c) Depending on utility size

Fig. 1: Candidate set length based on policy overlap, privacy size and utility size

Algorithm 2 has been run on 7000 randomly generated combinations of privacy and utility policies, thus covering a wide spectrum of combinations exhibiting various overlap degrees with various types of queries. For each execution, we compute the overlap degree between the input privacy and utility policies and group results in clusters of 10% before plotting as a boxplot the number of candidate sets in executions featuring the given overlap degree (Figure 1a). This provides a representation of how many alternatives our algorithm provides for anonymizing a graph, depending on the policies overlap. The boxplot allows to visualize both extreme values and average trends, given that the randomization can easily create extreme cases and outlier values.

We can observe that the number of candidate sets quickly decreases when overlapping grows even slightly. This is easy to understand, given that increasing overlap degree induces that less deletion operations are permitted by the algorithm. As soon as the overlap degree reaches an high value, our algorithm provides very few anonymization alternatives since no possible operation exists to satisfy the given policies.

We use the same experimental settings as in Section 6.1 to evaluate how the number of candidate sets evolves as a function of policy size.

Figure 1b displays the results of this experiment when varying privacy size with a fixed utility size of 9 triples. We can observe a steady increase of the number of candidate sets with the privacy size. The explanation for this behavior is that increasing privacy size (with fixed utility size) provide more possible operations for the anonymization.

On the other hand, when varying utility size (with fixed privacy size), the number of candidate sets almost stagnates when increasing the utility size (Figure 1c). This means that increasing the size of utility queries, without increasing the number of queries itself, does not significantly reduce the anonymization opportunities.

In short, this experiment emphasizes the faint influence of utility policies on possible anonymizations sets, along with the crucial role of privacy policies in shaping possible anonymization operations.

6.3 Runtime performance

One of the benefits of dealing with a query-driven static method for anonymization is to avoid dealing with the size of an input graph, which could impact performance by increasing runtime. Our static approach only deals with policy size when looking for candidate anonymization sets, which is likely to make the algorithm simple and efficient in general.

To confirm this, we ran the Algorithm 2 for a batch of 100 executions corresponding to input privacy and utility policies of 10 queries each, and we measured the average running time. As a result, we have obtained an average runtime of **0.843 seconds** over all executions, which turns to be satisfactory in practice.

We can conclude that this static approach provides a fast way to enumerate all the candidate sets of anonymization operations.

7 Conclusion and Future work

We presented in this paper a novel query-based approach for Linked Open Data anonymization under the form of delete and update operations on RDF graphs. We consider policies as sets of privacy and utility specifications, which can be readily written as queries by the data providers. We further designed a data-independent algorithm to compute sets of anonymization operations guaranteed to satisfy both privacy and utility policies on any input RDF graph. Our proof-of-concept open-source implementation confirms the intuition that (i) the larger is the utility policy, the lesser anonymization operations are available; (ii) the opposite holds for privacy policy but with a stronger impact on the number of candidate anonymization operations; (iii) the more privacy and utility policies are interleaved, the lesser is the number of candidate operations.

Our query-based approach can be combined with ontology-based query rewriting and thus can support reasoning for first-order rewritable ontological languages such as RDFS [20], DL-Lite [5] or EL fragments [9]. More precisely, given a pair of privacy and utility policies made of conjunctive queries defined over an ontology, each set of anonymization operations returned by Algorithm 2 applied to the two sets of their corresponding conjunctive rewritings (obtained using existing query rewriting algorithms [4, 5, 9]) will produce datasets that are guaranteed to satisfy the policies.

It is also important to emphasize that our approach can be combined with other anonymization approaches (k-anonymity techniques, differential privacy) after the transformation of an input RDF graph by the application of a sequence of operations output by Algorithm 2.

We are planning several orthogonal research directions for future work. A first research direction consists in extending the expressivity of the queries considered in this paper both for defining the policies and the anonymization operations. More expressive privacy and utility queries (with FILTER, NOT EXISTS, aggregate functions) fits in our general framework (Section 4) but requires extensions of the Section 5 algorithms. In addition to triple deletion and IRI replacement with blank nodes, other anonymization operations can be considered such as value replacement in triples involving datatype properties and IRI aggregation. The point is that each of these operations can be defined with (possibly complex) queries by leveraging SPARQL 1.1 aggregate and update queries as well as calls to built-in functions.

Another future extension is the study of data-dependent solutions, as opposed to the *data-independent* approach introduced in this paper. The proof of Theorem 1 relies on the fact that all instances of the utility queries are completely left unmodified by the deletions operations. However, it may happen that *some* instances common to the privacy and utility queries are suppressed without impacting the answers of the utility queries evaluated over a *given dataset*. An alternative is thus to consider data-dependent solutions, at the cost of running the algorithm on the (possibly huge) dataset. Such an approach could be adopted when no data-independent solution can be found.

Another research direction we envision is to consider an optimization problem that extends the EnumOperations problem defined in Section 5. The optimization problem consists in finding *optimal* sequences of anonymization operations and not all sequences, where optimality can be defined as minimality w.r.t. a partial order over sequences of anonymization operations (e.g., their size, or a distance between the original and resulting datasets.)

Acknowledgements

This work has been supported by the Auvergne-Rhône-Alpes region through the ARC6 research program for funding Remy Delanaux's PhD, by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01), the SIDES 3.0 project (ANR-16-DUNE-0002) funded by the French Program Investissement d'Avenir and the Palse Impulsion 2016/31 programme (ANR-11-IDEX-0007-02) at UDL.

References

1. Baader, F., Borchmann, D., Nuradiansyah, A.: Preliminary results on the identity problem in description logic ontologies. In: Description Logics. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017)

2. Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G.H.L., Lemay, A., Advokaat, N.: gMark: Schema-driven generation of graphs and queries. IEEE Transactions on Knowledge and Data Engineering 29(4), 856–869 (2017)
3. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. PVLDB 11(2), 149–161 (2017)
4. Bursztyn, D., Goadoué, F., Manolescu, I.: Reformulation-based query answering in rdf: alternatives and performance. PVLDB 8 (2015)
5. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. Journal of Automated Reasoning 39(3), 385–429 (2007)
6. Dwork, C.: Differential privacy. In: ICALP (2). LNCS, vol. 4052, pp. 1–12. Springer (2006)
7. Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: A survey of recent developments. ACM Comput. Surv. 42(4), 14:1–14:53 (2010)
8. Grau, B.C., Kostylev, E.V.: Logical foundations of privacy-preserving publishing of linked data. In: AAAI. pp. 943–949. AAAI Press (2016)
9. Hansen, P., Lutz, C., Seylan, I., Wolter, F.: Efficient query rewriting in the description logic el and beyond. In: IJCAI (2015)
10. Heitmann, B., Hermsen, F., Decker, S.: k - rdf-neighbourhood anonymity: Combining structural and attribute-based anonymisation for linked data. In: PrivOn@ISWC. CEUR Workshop Proceedings, vol. 1951. CEUR-WS.org (2017)
11. Kirrane, S., Mileo, A., Decker, S.: Access control and the resource description framework: A survey. Semantic Web 8(2), 311–352 (2017)
12. Kirrane, S., Villata, S., d'Aquin, M.: Privacy, security and policies: A review of problems and solutions with semantic web technologies. Semantic Web Journal 9(2), 153–161 (2018)
13. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: ICDE. pp. 106–115. IEEE Computer Society (2007)
14. Machanavajjhala, A., He, X., Hay, M.: Differential privacy in the wild: A tutorial on current practices & open challenges. PVLDB 9(13), 1611–1614 (2016)
15. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: *L*-diversity: Privacy beyond *k*-anonymity. TKDD 1(1), 3 (2007)
16. Oulmakhzoune, S., Cuppens-Boulahia, N., Cuppens, F., Morucci, S.: Privacy policy preferences enforced by SPARQL query rewriting. In: ARES. pp. 335–342. IEEE Computer Society (2012)
17. Radulovic, F., García-Castro, R., Gómez-Pérez, A.: Towards the anonymisation of RDF data. In: SEKE. pp. 646–651. KSI Research Inc. (2015)
18. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(5), 557–570 (2002)
19. Villata, S., Delaforge, N., Gandon, F., Gyrard, A.: An access control model for linked data. In: OTM Workshops. LNCS, vol. 7046, pp. 454–463. Springer (2011)
20. W3C: RDF schema 1.1. http://www.w3.org/TR/rdf-schema/ (2004)