Access Control for HTTP Operations on Linked Data

Luca Costabello, Serena Villata, Oscar Rodriguez Rocha, Fabien Gandon

Access Control for HTTP Operations on Linked Data

Luca Costabello[1], Serena Villata[1],
Oscar Rodriguez Rocha[2], and Fabien Gandon[1]

[1] INRIA Sophia Antipolis, France
firstname.lastname@inria.fr
[2] Politecnico di Torino, Italy
oscar.rodriguezrocha@polito.it

Abstract. Access control is a recognized open issue when interacting with RDF using HTTP methods. In literature, authentication and authorization mechanisms either introduce undesired complexity such as SPARQL and ad-hoc policy languages, or rely on basic access control lists, thus resulting in limited policy expressiveness. In this paper we show how the Shi3ld attribute-based authorization framework for SPARQL endpoints has been progressively converted to protect HTTP operations on RDF. We proceed by steps: we start by supporting the SPARQL 1.1 Graph Store Protocol, and we shift towards a SPARQL-less solution for the Linked Data Platform. We demonstrate that the resulting authorization framework provides the same functionalities of its SPARQL-based counterpart, including the adoption of Semantic Web languages only.

1 Introduction

In scenarios such as Linked Enterprise Data, access control becomes crucial, as not all triples are openly published on the Web. Solutions proposed in literature protect either SPARQL endpoints or generic RDF documents and adopt Role-based (RBAC) [20] or Attribute-based (ABAC) [18] models. The Semantic Web community is recently emphasizing the need for a substantially *"Web-like"* interaction paradigm with Linked Data. For instance, the W3C Linked Data Platform[3] initiative promotes the use of read/write HTTP operations on triples, thus providing a *basic profile* for Linked Data servers and clients. Another example is the SPARQL 1.1 Graph Store Protocol[4], a set of guidelines to interact with RDF graphs with HTTP operations. Defining an access control model for these scenarios is still an open issue[5]. Frameworks targeting HTTP access to RDF resources rely on access control lists, thus offering limited policy expressiveness [13,14,17,19] (e.g., no location-based authorization). On the other hand,

[3] http://www.w3.org/TR/ldp/
[4] http://www.w3.org/TR/sparql11-http-rdf-update/
[5] http://www.w3.org/2012/ldp/wiki/AccessControl

existing access control frameworks for SPARQL [1,4,10] add complexity rooted in the query language and in the SPARQL protocol, often introducing ad-hoc policy languages, thus requiring adaptation to the HTTP-only scenario.

In this paper, we answer the research question: *How to design an authorization framework for HTTP-based interaction with Linked Data?* This research question breaks down into the following sub-questions: (i) how to define an authorization model featuring expressive policies based on standard Web languages only and (ii) how to adopt this model in HTTP-based interaction with Linked Data scenarios like the Graph Store Protocol (GSP) and the Linked Data Platform (LDP).

We adapt the Shi3ld authorization framework for SPARQL [4] to a SPARQL-less scenario. We choose Shi3ld because its features satisfy our requirements: (i) it adopts attribute-based access policies ensuring expressiveness, and (ii) exclusively uses Semantic Web languages for policy definition and enforcement.

We illustrate Shi3ld-GSP, an intermediate version designed for the SPARQL 1.1 Graph Store HTTP Protocol. We progressively shift to the Linked Data Platform context, a scenario where SPARQL is no longer present. We have developed two solutions for this scenario: (i) an authorization module embedding a hidden SPARQL engine, and (ii) a framework where we completely get rid of SPARQL. In the latter case, the Shi3ld framework adopts a SPARQL-less subgraph matcher which grants access if client attributes correspond to the declared policy graphs. For each framework, we evaluate the response time and we show how the authorization procedure impacts on HTTP operations on RDF data.

The key features of our attribute-based authorization frameworks for HTTP-based interaction with Linked Data are (i) the use of Web languages only, i.e., HTTP methods and RDF, without ad-hoc languages for policies definition, (ii) the adoption of attribute-based access conditions enabling highly expressive policies, and (iii) the adaptation to the GSP and LDP scenarios as a result of a progressive disengagement from SPARQL. Moreover, Shi3ld is compatible and complementary with the WebID authentication framework[6].

In this paper, we focus on *authorization* only, without addressing the issues related to authentication and identity on the Web. Although we discuss state-of-the-art anti-spoofing techniques for attribute data, the present work does not directly address the issue.

The remainder of this paper is organized as follows. Section 2 summarizes the related work, and highlights the requirements of an authorization model for our scenario. Section 3 describes the main insights of Shi3ld, and presents the three proposed solutions to adapt the framework to HTTP operations on RDF. An experimental evaluation of response time overhead is provided in Section 4.

[6] `http://www.w3.org/2005/Incubator/webid/spec/`

2 Related Work

Many access control frameworks rely on *access control lists* (ACLs) that define which users can access the data. This is the case of the Web Access Control vocabulary (WAC)[7], that grants access to a whole RDF document. Hollenbach et al. [13] present a system where providers control access to RDF documents using WAC. In our work, we look for more expressive policies that can be obtained without leading to an increased complexity of the adopted language or model.

Similarly to ACLs, other approaches specify *who* can access the data, e.g., to which roles access is granted. Among others, Giunchiglia et al. [12] propose a Relation Based Access Control model (*RelBAC*) based on description logic, and Finin et al. [9] study the relationship between OWL and RBAC [20]. They briefly discuss possible ways of going beyond RBAC such as Attribute Based Access Control, a model that grants access according to client attributes, instead of relying on AC lists.

The ABAC model is adopted in the Privacy Preference Ontology (PPO)[8] [19], built on top of WAC, where consumers require access to a given RDF file, e.g., a FOAF profile, and the framework selects the part of the file the consumer can access, returning it. In our work, we go beyond the preference specification based on FOAF profiles. Our previous work [4] adopts ABAC for protecting the accesses to SPARQL endpoints using Semantic Web languages only.

Other frameworks introduce a *high level syntax* for expressing policies. Abel et al. [1] present a context-dependent access control system for RDF stores, where policies are expressed using an ad-hoc syntax and mapped to existing policy languages. Flouris et al. [10] present an access control framework on top of RDF repositories using a high level specification language to be translated into a SPARQL/SerQL/SQL query to enforce the policy. Muhleisen et al. [17] present a policy-enabled server for Linked Data called PsSF, where policies are expressed using a descriptive language based on SWRL[9]. Shen and Cheng [21] propose a Context-Based Access Control Model (SCBAC) where policies are expressed using SWRL.

Access control models may consider not only the information about the consumer who is accessing the data, but also the *context* of the request, e.g., time, location. Covington et al. [5] present an approach where the notion of role proposed in RBAC is used to capture the environment in which the access requests are made. Cuppens and Cuppens-Boulahia [6] propose an Organization Based Access Control model (OrBAC) that contains contextual conditions. Toninelli et al. [22] use context-awareness to control access to resources, and semantic technologies for policy specification. Corradi et al. [3] present UbiCOSM, a security middleware adopting context as a basic concept for policy specification and enforcement.

[7] `http://www.w3.org/wiki/WebAccessControl`

[8] `http://vocab.deri.ie/ppo`

[9] `http://www.w3.org/Submission/SWRL/`

Table 1 summarizes the main characteristics of the related work described above[10]: none of the presented approaches satisfies all the features that we require for protecting HTTP operations on Linked Data, i.e.: absence of ad hoc policy languages, CRUD permission model, protection granularity at resource-level, and expressive access control model to go beyond basic access control lists.

	Web-based	AC model	Policy language	Protection granularity	Permissio model	Context Awareness	Conflict verificatior	Eval.
WAC[7]	YES	RBAC	RDF	RDF document	R/W	N/A	N/A	N/A
Abel et al. [1]	YES	ABAC	Custom	triples	R	YES	N/A	YES
Finin et al. [9]	YES	RBAC	OWL/RDF	resources	N/A	N/A	N/A	N/A
RelBAC [12]	YES	relation	DL	resources	N/A	N/A	N/A	N/A
Hollenbach[13]	YES	RBAC	RDF	RDF document	R/W	N/A	N/A	YES
Flouris et al. [10]	YES	RBAC	Custom	triples	R	N/A	YES	YES
PeLDS [17]	YES	RBAC	SWRL	RDF document	R/W	N/A	N/A	YES
PPO [19]	YES	ABAC	RDF, SPARQL	RDF doc(part)	R/W	N/A	N/A	N/A
SCBAC [21]	YES	context	SWRL	resources	N/A	YES	YES	N/A
Shi3ld-SPARQL[4]	YES	ABAC	RDF, SPARQL	named graphs	CRUD	YES	N/A	YES
Covington [5]	NO	RBAC	Custom	resources	R/W	YES	YES	N/A
CSAC [14]	NO	gen. RBAC	XML	resources	R	YES	N/A	N/A
Proteus[22]	NO	context	DL	Resources	N/A	YES	YES	YES
OrBAC [6]	NO	organizatior	Datalog	resources	R/W	YES	YES	N/A
UbiCOSM [3]	NO	context	RDF	resources	N/A	YES	YES	YES

Table 1: A summarizing comparison of the related work.

3 Restricting HTTP operations on Linked Data

Before discussing how we modified the Shi3ld original proposition [4] to obtain a SPARQL-less access control framework for HTTP operations on Linked Data, we provide an overview of the original Shi3ld authorization model for SPARQL endpoints. Shi3ld [4] presents the following key features:

Attribute-based paradigm. Shi3ld is an *attribute-based* authorization framework, i.e., authorization check is performed against a set of attributes sent by the client along the query that targets the resource. Relying on attributes provides broad access policy expressiveness, beyond the access condition lists adopted by RBAC frameworks. That means, among all, creating location-based and temporal-based access policies.

Semantic Web languages only. Shi3ld uses access policies defined with Semantic Web languages only, and no additional policy language needs to be defined. In particular, the access conditions specified in the policies are SPARQL ASK queries.

CRUD permission model. Access policies are associated to specific permissions over the protected resource. It is therefore possible to specify rules satisfied only when the access is in *create*, *read*, *update* and *delete* mode.

Granularity. The proposed degree of granularity is represented by named graphs, allowing protection from triples up to whole dataset.

[10] We use N/A when the feature is not considered in the work.

The HTTP-based interaction with Linked Data requires some major modifications to the above features: although we keep the attribute-based paradigm and the CRUD permission model, the new versions of Shi3ld satisfy also the following requirements:

Protection of HTTP access to resources. Protected resources are retrieved and modified by clients using HTTP methods only, without SPARQL querying[11].

RDF-only Policies. In the SPARQL-less scenario, access conditions are RDF triples with no embedded SPARQL.

Granularity. The atomic element protected by Shi3ld is a *resource*.

In this paper, we rely on the definition of resource provided by the W3C Linked Data Platform Working Group: LDP resources are HTTP resources queried, created, modified and deleted via HTTP requests processed by LDP servers[12]. Linked Data server administrators adopting Shi3ld must define a number of *access policies* and associate them to protected resources. Access policies and their components are formally defined as follows:

Definition 1. *(Access Policy) An Access Policy (P) is a tuple of the form* $P = \langle ACS, AP, R \rangle$ *where (i)* ACS *is a set of Access Conditions to satisfy, (ii)* AP *is an Access Privilege, and (iii)* R *is the resource protected by* P*.*

Definition 2. *(Access Condition) An Access Condition (AC) is a set of attributes that need to be satisfied to interact with a resource.*

Definition 3. *(Access Privilege) An Access Privilege (AP) is the set of allowed operations on the protected resource.* $AP = \{Create, Read, Update, Delete\}$*.*

The lightweight vocabularies used by Shi3ld are `s4ac`[13] for defining the policy structure, and `prissma`[14] for the client attributes[15]. Client attributes include user profile information, device features, environment data, or any given combination of these dimensions, in compliance with the widely-accepted definition by Dey [7] and the work by Fonseca et al.[16] (we delegate refinements and extensions to domain specialists, in the light of the Web of Data philosophy). The main classes and properties of these vocabularies are visualized in Figure 1. Shi3ld offers a double notation for defining access conditions: with embedded SPARQL (Figure 2a) for SPARQL-equipped scenarios and in full RDF (Figure 2b), adopted in SPARQL-less environments.

[11] This is in compliance with the LDP specifications.

[12] An LDP server is an "*application program that accepts connections in order to service requests by sending back responses*" as specified by HTTP 1.1 definition.

[13] `http://ns.inria.fr/s4ac`

[14] `http://ns.inria.fr/prissma`

[15] Although this vocabulary provides classes and properties to model context-aware attributes, it is not meant to deliver yet another contextual model: instead, well-known Web of Data vocabularies and recent W3C recommendations are reused. For more details, see Costabello et al. [4].

[16] `http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/`

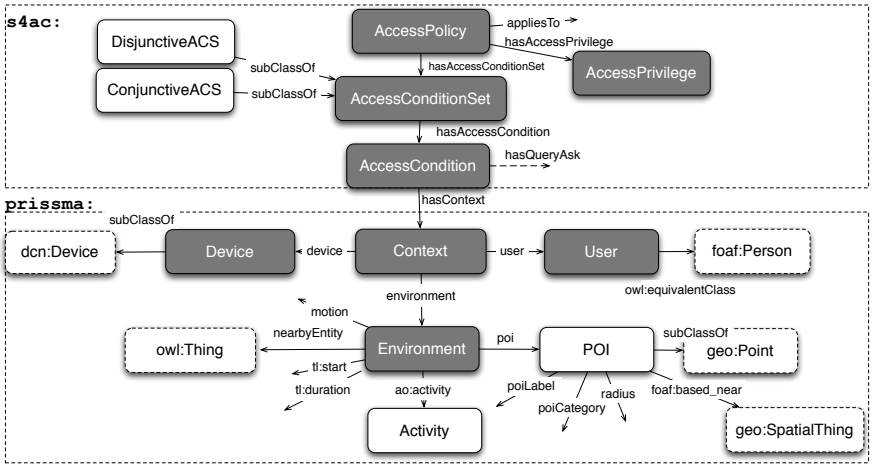


Fig. 1: Interplay of `s4ac` and `prissma` vocabularies for Shi3ld access policies.

Example 1. Figure 2 presents two sample access policies, expressed with and without SPARQL. The policy visualized in Figure 2a allows *read-only* access to the protected resource exclusively by a specific user and from a given location. The policy in Figure 2b authorizes the *update* of the resource by the given user, only if he is currently near Alice.

```
:policy1 a s4ac:AccessPolicy;                          PROTECTED RESOURCE
         s4ac:appliesTo :protected_res;                ACCESS PRIVILEGE
         s4ac:hasAccessPrivilege s4ac:Read;
         s4ac:hasAccessConditionSet :acs1.

:acs1 a s4ac:AccessConditionSet;
        s4ac:ConjunctiveAccessConditionSet;
        s4ac:hasAccessCondition :ac1.

:ac1 a s4ac:AccessCondition;                           ACCESS CONDITION TO VERIFY
     s4ac:hasQueryAsk
"""ASK
{?ctx a prissma:Context.
?ctx prissma:environment ?env.
?ctx prissma:user <http://johndoe.org/foaf.rdf#me>.
?env prissma:currentPOI ?poi.
?poi prissma:based_near ?p.
?p geo:lat ?lat; geo:lon ?lon.
FILTER(((?lat-45.8483) > 0 && (?lat-45.8483) < 0.5
|| (?lat-45.8483) < 0 && (?lat-45.8483) > -0.5)
&& ((?lon-7.3263) > 0 && (?lon-7.3263) < 0.5
|| (?lon-7.3263) < 0 && (?lon-7.3263) > -0.5 ))""".
```

(a) SPARQL-based

```
:policy1 a s4ac:AccessPolicy;                          PROTECTED RESOURCE
         s4ac:appliesTo :protected_res;                ACCESS PRIVILEGE
         s4ac:hasAccessPrivilege s4ac:Update;
         s4ac:hasAccessConditionSet :acs1.

:acs1 a s4ac:AccessConditionSet;
        s4ac:ConjunctiveAccessConditionSet;
        s4ac:hasAccessCondition :ac1.

:ac1 a s4ac:AccessCondition;                           ACCESS CONDITION TO VERIFY
     s4ac:hasContext :ctx1.

:ctx1 a prissma:Context;
   prissma:user <http://johndoe.org/foaf.rdf#me>.
   prissma:environment :env1

:env1 a prissma:Environment;
      prissma:nearbyEntity <http://alice.org#me>.
```

(b) SPARQL-less

Fig. 2: Shi3ld access policies, expressed with and without SPARQL.

Whenever an HTTP query is performed on a resource, Shi3ld runs the authorization algorithm to check if the policies that protect the resource are satisfied or not. The procedure verifies the matching between the client attributes sent with the query and the access policies that protect the resource.

Shi3ld deals with authorization only. Nevertheless, authentication issues cannot be ignored as the trustworthiness of client attributes is critical for a reliable access control framework. Shi3ld supports heterogeneous authentication strategies, since the attributes attached to each client request include heterogeneous data, ranging from user identity to environment information fetched by device sensors (e.g. location). The trustworthiness of user identity is achieved thanks to the WebID[6] compatibility: in Shi3ld, user-related attributes are modelled with the `foaf` vocabulary[17], thus easing the adoption of WebID. Authenticating the attributes fetched by client sensors is crucial to prevent tampering. Hulsebosch et al. [14] provide a survey of verification techniques, such as heuristics relying on location history and collaborative authenticity checks. A promising approach is mentioned in Kulkarni and Tripathi [16], where client sensors are authenticated beforehand by a trusted party. To date, no tamper-proof strategy is implemented in Shi3ld, and this is left for future work.

Moreover, sensible data, such as current location must be handled with a privacy-preserving mechanism. Recent surveys describe strategies to introduce privacy mainly in location-based services [8,15]. Shi3ld adopts an *anonymity-based* solution [8] and delegates attribute anonymisation to the client side, thus sensitive information is not disclosed to the server. We rely on partially encrypted RDF graphs, as proposed by Giereth [11]. Before building the RDF attribute graph and sending it to the Shi3ld-protected repository, a partial RDF encryption is performed, producing RDF-compliant results, i.e., the encrypted graph is still RDF (we use SHA-1 cryptographic hash function to encrypt RDF literals). On server-side, every time a new policy is added to the system, the same operation is performed on attributes included in access policies. As long as literals included in access conditions are hashed with the same function used on the client side, the Shi3ld authorization procedure still holds[18].

We now describe the steps leading to a SPARQL-less authorization framework for HTTP operations on Linked Data. Our first proposal is a Shi3ld authorization framework for the SPARQL 1.1 Graph Store Protocol (Section 3.1). In Sections 3.2 and 3.3 we describe two scenarios tailored to the Linked Data Platform specifications, the second being completely SPARQL-less. Our work is grounded on the analogies between SPARQL 1.1 functions and the HTTP protocol semantics, as suggested by the SPARQL Graph Store Protocol specification[4].

3.1 Shi3ld for SPARQL Graph Store Protocol

The SPARQL 1.1 HTTP Graph Store Protocol[4] (GSP) provides an alternative interface to access RDF stored in SPARQL-equipped triple stores. The recommendation describes a mapping between HTTP methods and SPARQL queries,

[17] `http://xmlns.com/foaf/spec/`

[18] The adopted technique does not guarantee full anonymity [15]. Nevertheless, the problem is mitigated by the short persistence of client-related data inside Shi3ld cache: client attributes are deleted after each authorization evaluation. Encryption is not applied to location coordinates and timestamps, as this operation prevents geo-temporal filtering.

thus enabling HTTP operations on triples. The Graph Store Protocol can be considered as an intermediate step towards an HTTP-only access to RDF datastores, since it still needs a SPARQL endpoint.

Figure 3a shows the architecture of the authorization procedure of Shi3ld for GSP-compliant SPARQL endpoints (Shi3ld-GSP). Shi3ld-GSP acts as a module protecting a stand-alone SPARQL 1.1 endpoint, equipped with a Graph Store Protocol module. First, the client performs an HTTP operation on a resource. This means that an RDF attribute graph is built on the client, serialized and sent with the request in the HTTP `Authorization` header[19]. Attributes are saved into the triple store with a SPARQL 1.1 query. Second, Shi3ld selects the access policies that protect the resource. The access conditions (SPARQL `ASK` queries, as in Figure 2a) included in the policies are then executed against the client attribute graph. Finally, the results are logically combined according to the type of access condition set (disjunctive or conjunctive) defined by each policy. If the result returns *true*, the HTTP query is forwarded to the GSP SPARQL engine, which in turns translates it into a SPARQL query. If the access is not granted, a HTTP 401 message is delivered to the client.

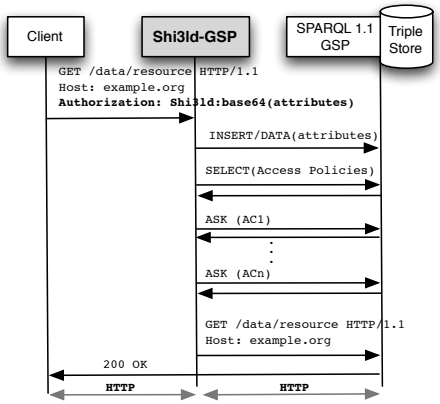
3.2 Shi3ld-LDP with Internal SPARQL Engine

The Linked Data Platform Initiative (LDP) proposes a simplified configuration for Linked Data servers and Web-like interaction with RDF resources. Compared to the GSP case, authorization frameworks in this scenario must deal with a certain number of changes, notably the absence of SPARQL and potentially the lack of a graph store.

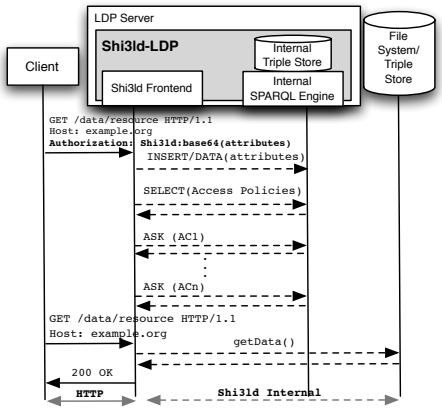
We adapt Shi3ld to work under these restrictions (Shi3ld-LDP). The framework architecture is shown in Figure 3b. Shi3ld-LDP protects HTTP operations, but it does not communicate with an external SPARQL endpoint, i.e. there are no intermediaries between the RDF repository (the filesystem or a triple store) and Shi3ld. To re-use the authorization procedure previously described, we integrate an internal SPARQL engine into Shi3ld, along with an internal triple store. Although SPARQL is still present, this is perfectly legitimate in a Linked Data Platform scenario, since the use of the query language is limited to Shi3ld internals and is not exposed to the outside world[20]. Despite the architectural changes, the Shi3ld model remains unchanged. Few modifications occur to the authorization procedure, as described in Figure 3a: clients send HTTP requests to the desired resource. HTTP headers contain the attribute graph, serialized as previously described in Section 3.1. Instead of relying on an external SPARQL endpoint, attributes are now saved internally, using an `INSERT DATA` query. The access policies selection and the access conditions execution remain substantially unchanged, but the whole process is transparent to the platform administrator, as the target SPARQL endpoint is embedded in Shi3ld.

[19] We extend the header with the ad-hoc `Shi3ld` option. Other well-known proposals on the web re-use this field, e.g. the OAuth authorization protocol.

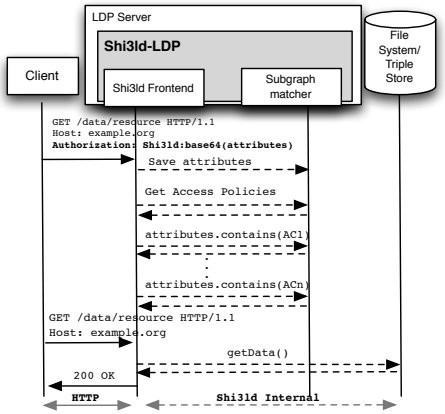
[20] SPARQL is still visible in Access Policies (Figure 2a).



(a) Shi3ld-GSP



(b) Shil3d-LDP (internal SPARQL engine)



(c) Shi3ld-LDP (SPARQL-less)

Fig. 3: Shi3ld Configurations

3.3 SPARQL-less Shi3ld-LDP

To completely fulfill the Linked Data Platform recommendations, thus achieving a full-fledged *basic profile* for authorization frameworks, we drop SPARQL from the Shi3ld-LDP framework described in Section 3.2. Ditching SPARQL allows RDF-only access policies definition, and a leaner authorization procedure. To obtain a SPARQL-less framework, we re-use the access policy model and the logical steps of the previously described authorization procedure, although conveniently adapted (Figure 3c). First, Shi3ld-LDP policies adopt RDF only, as shown in Figure 2b: attribute conditions previously expressed with SPARQL ASK queries (Figure 2a) are expressed now as RDF graphs. Second, the embedded SPARQL engine used in Section 3.2 has been replaced: its task was testing whether client attributes verify the conditions defined in each access policy. This operation

boils down to a *subgraph matching problem*. In other words, we must check if the access conditions (expressed in RDF) are contained into the attribute graph sent along the HTTP client query. Such subgraph matching procedure can be performed without introducing SPARQL in the loop. To steer clear of SPARQL, without re-inventing yet another subgraph matching procedure, we scrap the SPARQL interpreter from the SPARQL engine [2] used in Section 3.2, keeping only the underlying subgraph matching algorithm[21].

To understand how the SPARQL-less policy verification procedure works and comprehend the complexity hidden by the SPARQL layer, we now provide a comprehensive description of the adopted subgraph matching algorithm, along with an overview of the RDF indexes used by the procedure. The algorithm checks whether a query graph Q (the access condition) is contained in the reference graph R (the client attributes sent along the query).
The reference graph R is stored in two key-value indexes (see example in Figure 4): index I_s stores the associations between property types and property subjects, and index I_o stores the associations between property types and property objects. Each RDF property type of R is therefore associated to a list of property subjects S_p and a list of property objects O_p. S_p contains URIs or blank nodes, O_p contains URIs, typed literals and blank nodes. Blank nodes are represented as anonymous elements, and their IDs are ignored.
The query graph Q, i.e., the access condition attributes, is serialized in a list L of subject-property-object elements $\{s_i, p_i, o_i\}$[22]. Blank nodes are added to the serialization as anonymous s_i or o_i elements.
The matching algorithm works as follows: for each subject-property-object $\{s_i, p_i, o_i\}$ in L, it looks up the indexes I_s and I_o using p_i as key. It then retrieves the list of property subjects S_p and the list of property objects O_p associated to p_i. Then, it searches for a subject in S_p matching with s_i, and an object in O_p matching with o_i. If both matches are found, $\{s_i, p_i, o_i\}$ is matched and the procedure moves to the next elements in L. If no match is found in either I_s or I_o, the procedure stops. Subgraph matching is successful if all L items are matched in the R index. Blank nodes act as wildcards: if a blank node is found in $\{s_i, p_i, o_i\}$ as object o_i or subject s_i, and O_p or S_p contains one or more blank nodes, the algorithm continues the matching procedure recursively, backtracking in case of mismatch and therefore testing all possible matchings.
The example in Figure 4 shows a matching step of the algorithm, i.e., the successful matching of the triple "`_:b2 p:nearbyEntity http://alice.org/me`" against the client attributes indexes I_s and I_o. The highlighted triple is successfully matched against the client attributes R.

Note that policies might contain location and temporal constraints: Shi3ld-GSP (Section 3.1) and Shi3ld-LDP with internal SPARQL endpoint (Section 3.2) handle these conditions by translating RDF attributes into SPARQL `FILTER`

[21] Third-party SPARQL-less Shi3ld-LDP implementations might adopt other off-the-shelf subgraph matching algorithms.

[22] A preliminary step replaces the query graph Q intermediate nodes into blank nodes. Blank nodes substitute SPARQL variables in the matching procedure.

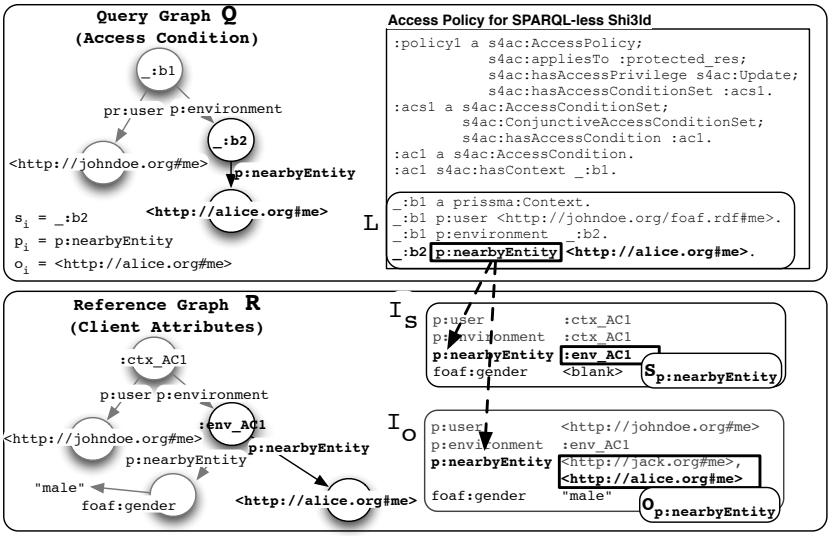


Fig. 4: Example of subgraph matching used in the SPARQL-less Shi3ld-LDP.

clauses. The subgraph matching algorithm adopted by SPARQL-less Shi3ld-LDP does not support geo-temporal authorization evaluation yet.

The three Shi3ld configurations described in this Section use the `Authorization` header to send client attributes. Even if there is no limit to the size of each header value, it is good practice to limit the size of HTTP requests, to minimize latency. Ideally, HTTP requests should not exceed the size of a TCP packet (1500 bytes), but in real world finding requests that exceed 2KB is not uncommon, as a consequence of cookies, browser-set fields and URL with long query strings[23]. To keep size as small as possible, before base-64 encoding, client attributes are serialized in turtle (less verbose that N-triples and RDF/XML). We plan to test the effectiveness of common lossless compression techniques to reduce the size of client attributes as future work. Furthermore, instead of sending the complete attribute graph along all requests, a server-side caching mechanism would enable the transmission of attribute graph deltas (i.e. only newly updated attributes will be sent to the server). Sending differences of RDF graphs is an open research topic[24] and we do not address the issue in this paper.

[23] `https://developers.google.com/speed/docs/best-practices/request`
[24] `http://www.w3.org/2001/sw/wiki/How_to_diff_RDF`

4 Evaluation

We implemented the three scenarios presented in Section 3 as Java standalone web services[25]. The Shi3ld-GSP prototype works with the Fuseki GSP-compliant SPARQL endpoint[26]. The Shi3ld-LDP prototype with internal SPARQL endpoint embeds the KGRAM/Corese[27] engine [2]. Our test campaign assesses the impact of Shi3ld on HTTP query response time[25]. We evaluate the prototypes on an Intel Xeon E5540, Quad Core 2.53 GHz machine with 48GB of memory. In our test configuration, Shi3ld-GSP protects a Fuseki SPARQL server, while Shil3d-LDP scenarios secure RDF resources saved on the filesystem. First, we investigate the relationship between response time and the number of access conditions to verify. Second, we test how access conditions complexity impacts on response time. Our third test studies the response time with regard to different HTTP methods. We execute five independent runs of a test query batch consisting in 50 HTTP operations (tests are preceded by a warmup run). Each query contains client attributes serialized in turtle (20 triples). The base-64 turtle serialization of the client attributes used in tests[25] is 1855 bytes long (including prefixes). Tests do not consider client-side literal anonymization (Section 3).

Our first test shows the impact of the access conditions number on HTTP GET response time (Figure 5a and 5b). Each policy contains one access condition, each including 5 triples. We progressively increased the number of access conditions protecting the target RDF resource. Not surprisingly, the number of access conditions defined on the protected resource impacts on response time. In Figure 5a we show the results for Shi3ld-LDP scenarios: data show a linear relationship between response time and access conditions number. We tested the system up to 100 access conditions, although common usage scenarios have a smaller number of conditions defined for each resource. For example, the 5 access condition case is approximately 3 times slower than unprotected access. Nevertheless, ditching SPARQL improved performance: Figure 5a shows that the SPARQL-less configuration is in average 25% faster than its SPARQL-based counterpart, due to the absence of the SPARQL interpreter. As predicted, the delay introduced by Shi3ld-GSP is higher, e.g., 7 times slower for resources protected by 5 access policies (Figure 5b). This is mainly due to the HTTP communication between the Shi3ld-GSP module and Fuseki. Further delay is introduced by the Fuseki GSP module, that translates HTTP operations into SPARQL queries. Moreover, unlike Shi3ld-LDP scenarios, Shi3ld-GSP uses a shared RDF store for protected resources and access control-related data (client attributes and access policies). This increases the execution time of SPARQL queries, thus determining higher response time: in Figure 5b, we show the behaviour of Shi3ld-GSP with two Fuseki server configurations: empty and with approximately 10M triples, stored in 17k graphs (we chose the "4-hop expansion Timbl crawl" part of the

[25] Binaries, code and complete evaluation results are available at:
`http://wimmics.inria.fr/projects/shi3ld-ldp`

[26] `http://jena.apache.org/documentation/serving_data`

[27] `http://tinyurl.com/corese-engine`

Billion Triple Challenge 2012 Dataset[28]). Results show an average response time difference of 14%, with a 27% variation for the 5 access condition case (Figure 5b). The number and the distribution of triples in the RDF store influence Shi3ld-GSP response time. Results might vary when Shi3ld-GSP is coupled with SPARQL endpoints adopting different indexing strategies or with different triple number and graph partitioning.

In Figure 5c, we show the impact of access conditions complexity on HTTP GET response time. The requested resource is protected by a single access condition, with growing complexity: we added up to 20 triples, and we assess an access condition containing a FILTER clause (for SPARQL-based scenarios only). Results show no relevant impact on response time: this is because of the small size of the client attributes graph, over which access conditions are evaluated (in our tests, client attributes include 20 triples). Although attribute graph varies according to the application domain, it is reasonable that size will not exceed tens of triples.

The third test (Figure 5d) shows the delay introduced by Shi3ld for each HTTP operation. The figure displays the difference between response time with and without access control. We executed HTTP GET, POST, PUT and DELETE methods. Each HTTP method is associated to a 5-triple access condition. As predicted, the delay introduced by Shi3ld is independent from the HTTP method.

In Section 2, we addressed a qualitative comparison with respect to the related work. On the other hand, addressing a quantitative evaluation is a tricky point: among the list in Table 1, only few works explicitly designed for the Web come with an evaluation campaign [1,4,10,13,17]. Moreover, although some of these works provide a response time evaluation, the experimental conditions vary, making the comparison difficult.

5 Conclusions

We described an authorization framework for HTTP operations on Linked Data. The framework comes in three distinct configurations: Shi3ld-GSP (for the SPARQL 1.1 Graph Store Protocol) and Shi3ld for the Linked Data Platform (with and without the internal SPARQL endpoint). Our solutions feature attribute-based access control policies expressed with Web languages only. Evaluation confirms that Shi3ld-GSP is slower than the Shi3ld-LDP counterparts, due to the HTTP communication with the protected RDF store. Shi3ld-LDP with internal SPARQL endpoint introduces a 3x delay in response time (when resources are protected by 5 access conditions). Nevertheless, under the same conditions, the SPARQL-less solution exhibits 25% faster response times. We show that response time grows linearly with the number of access conditions, and that the complexity of each access condition does not relevantly impact on the delay.

[28] `http://km.aifb.kit.edu/projects/btc-2012/`

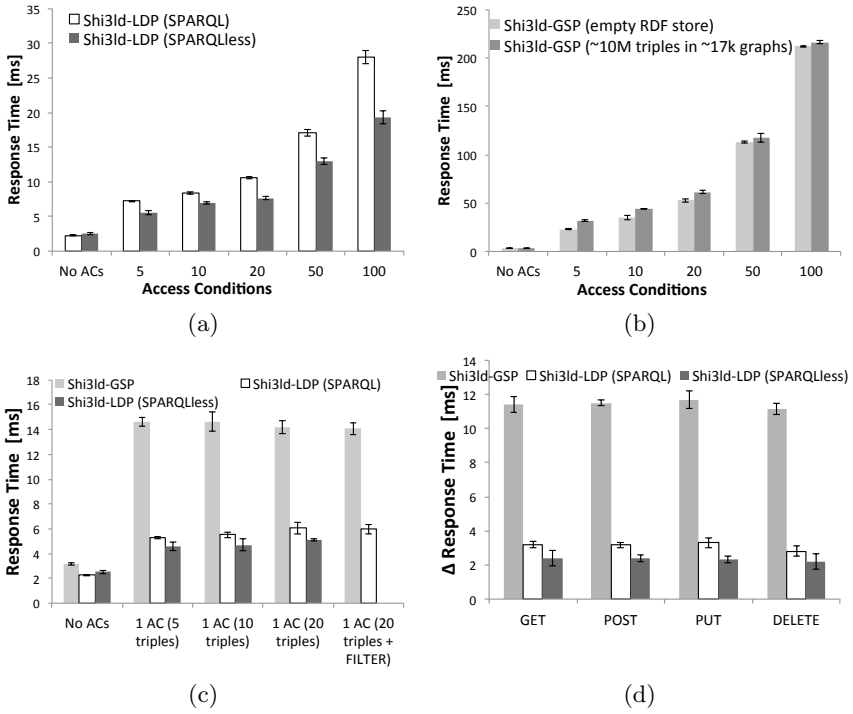


Fig. 5: Shi3ld response time evaluation.

Future work includes ensuring the trustworthiness of attributes sent by the client. Furthermore, a caching mechanism for client attributes must be introduced, to speed up the authorization procedure. The caching mechanism must be coupled with an efficient strategy to send attributes updates, to reduce the average size of HTTP requests. Finally, an effective administration interface to define access policies has to be designed, as user interaction issues should not be underestimated.

References

1. F. Abel, J. L. De Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla. Enabling Advanced and Context-Dependent Access Control in RDF Stores. In *Procs of ISWC, LNCS 4825*, pages 1–14, 2007.
2. O. Corby and C. Faron-Zucker. The kgram abstract machine for knowledge graph querying. In *Web Intelligence*, pages 338–341. IEEE, 2010.
3. A. Corradi, R. Montanari, and D. Tibaldi. Context-based access control management in ubiquitous environments. In *Procs of NCA*, pages 253–260. IEEE Computer Society, 2004.
4. L. Costabello, S. Villata, and F. Gandon. Context-aware access control for rdf graph stores. In *Procs of ECAI*, 2012.

5. M. J. Covington, W. Long, S. Srinivasan, A. K. Dey, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Procs of SACMAT*, pages 10–20, 2001.
6. F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. *Int. J. Inf. Sec.*, 7(4):285–305, 2008.
7. A. K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5:4–7, 2001.
8. M. Duckham. Moving forward: location privacy and location awareness. In *Procs of SPRINGL*, pages 1–3, 2010.
9. T. W. Finin, A. Joshi, L. Kagal, J. Niu, R. S. Sandhu, W. H. Winsborough, and B. M. Thuraisingham. R*OWL*BAC: representing role based access control in *OWL*. In *Procs of SACMAT*, pages 73–82. ACM, 2008.
10. G. Flouris, I. Fundulaki, M. Michou, and G. Antoniou. Controlling Access to RDF Graphs. In *Procs of FIS, LNCS 6369*, pages 107–117. Springer, 2010.
11. M. Giereth. On partial encryption of rdf-graphs. In *Procs of ISWC*, pages 308–322, 2005.
12. F. Giunchiglia, R. Zhang, and B. Crispo. Ontology driven community access control. In *Procs of SPOT*, 2009.
13. J. Hollenbach, J. Presbrey, and T. Berners-Lee. Using RDF Metadata To Enable Access Control on the Social Semantic Web. In *Procs of CK-2009*, 2009.
14. R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben, and J. Reitsma. Context sensitive access control. In *Procs of SACMAT*, pages 111–119. ACM, 2005.
15. J. Krumm. A survey of computational location privacy. *Personal Ubiquitous Comput.*, 13(6):391–399, Aug. 2009.
16. D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In *Procs of SACMAT*, pages 113–122, 2008.
17. H. Muhleisen, M. Kost, and J.-C. Freytag. SWRL-based Access Policies for Linked Data. In *Procs of SPOT*, 2010.
18. T. Priebe, E. B. Fernández, J. I. Mehlau, and G. Pernul. A pattern system for access control. In *Procs of DBSec*, pages 235–249, 2004.
19. O. Sacco, A. Passant, and S. Decker. An access control framework for the web of data. In *Proc. of TrustCom, IEEE*, pages 456–463, 2011.
20. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
21. H. Shen and Y. Cheng. A semantic context-based model for mobile web services access control. *I. J. Computer Network and Information Security*, 1:18–25, 2011.
22. A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *Procs of ISWC, LNCS 4273*, pages 473–486. Springer, 2006.