On Partial Encryption of RDF-Graphs

Mark Giereth

Institute for Intelligent Systems, University of Stuttgart,
70569 Stuttgart, Germany
giereth@iis.uni-stuttgart.de

Abstract. In this paper a method for Partial RDF Encryption (PRE) is proposed in which sensitive data in an RDF-graph is encrypted for a set of recipients while all non-sensitive data remain publicly readable. The result is an RDF-compliant self-describing graph containing encrypted data, encryption metadata, and plaintext data. For the representation of encrypted data and encryption metadata, the XML-Encryption and XML-Signature recommendations are used. The proposed method allows for fine-grained encryption of arbitrary subjects, predicates, objects and subgraphs of an RDF-graph. An XML vocabulary for specifying encryption policies is introduced.

1 Introduction

Giving information a well-defined meaning is on one hand the basis for intelligent applications in an emerging Semantic Web, but on the other hand can have profound consequences when considering privacy, security, and intellectual property rights issues. In the Semantic Web vision agents automatically gather and merge semantically annotated data, infer new data and re-use the data in different contexts [6]. However seemingly harmless pieces of data could reveal a lot of information when combined with others. In the Semantic Web there will also be the need of integrating data which is sensitive in some contexts.

Therefore, methods for specifying *who* is allowed to use *which* data are important in the next step towards the Semantic Web. There are two approaches to achieve this. The first is to specify access rights, to control the data access and to secure the communication channel when the data is transferred. The second attempt is to use cryptographic methods to protect the sensitive data itself.

There has been a considerable amount of work about access control for the Web [4, 26]. However, all these approaches need trustworthy infrastructures for specifying and controlling the data access. If sensitive data is stored in (potentially) insecure environments, such as public web-spaces, shared desktop systems, mobile devices, etc. the only way to do this is to locally encrypt the data before uploading or storing it. The ability to merge distributed data and to re-use the data have been important design aspects for the Semantic Web. From that perspective, partial encryption – where only the sensitive data are encrypted while all other data remain publicly readable – is desirable.

A common practice for encrypting sensitive data in an RDF-graph is to cut the data from the original graph, store the data in a separate file, encrypt the file

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 308–322, 2005.

and finally link the encrypted file to the original graph [12]. This approach has some shortcomings: (1) the original RDF-graph is separated into different physical resources, (2) the encrypted files are not RDF-compliant and therefore could not be consistently processed by common RDF frameworks, (3) the linking has to be done manually, and (4) no rules are given for re-integrating the data into the RDF-graph after decryption. Another practice for encrypting RDF-graphs is to serialize them in XML and to use XML-Encryption [13] and XML-Signature [14] based security frameworks. One problem with this approach, is the structural difference between the tree-based XML Information Set data model [11] and the graph-based RDF Abstract Syntax data model [16]. Another problem is that this approach only allows to handle XML serializations of RDF-graphs.

To address these problems, we propose a method for *partial RDF encryption (PRE)* which allows for fine-grained encryption of arbitrary fragments of an RDF-graph without creating additional resources. Both encrypted data and plaintext data are represented in a single RDF-compliant model together with the metadata describing the encryption parameters. PRE uses the XML-Encryption and XML-Signature standards to represent the encryption metadata.

The rest of this paper is organized as follows. In the next section a brief introduction to RDF-graphs is given. Section 3 gives an overview of the partial encryption process for RDF-graphs. The subsequent sections look at important realization aspects: encryption and decryption of RDF fragments (section 4), description of graph transformations necessary to keep encrypted graphs RDF-compliant (section 5), a graph-pattern based method for dynamic selection fragments to be encrypted and a notion of encryption policies (section 6). The last section summarizes and gives an outlook to future work. In the appendix the namespaces used in the examples are listed.

2 Triple Sets and Graphs

RDF is an assertional language. Each assertion declares that certain information about a resource is true. An assertion is modeled as a $\langle s, p, o \rangle$-triple where s (subject) identifies the resource the assertion is about, p (predicate) is a property of the resource, and o (object) is the value of p. A triple is an element of $(U \cup B) \times U \times (U \cup B \cup L)$, where U denotes the set of URIs [5], B denotes the set of blank node identifiers, and L denotes the set of RDF literals [16]. A triple set can be interpreted as a Directed Labeled Graph (DLG) with the subjects and objects as nodes and node labels, the triples as arcs, and the predicates as arc labels ($\boxed{s} \xrightarrow{p} \boxed{o}$). A subgraph is a subset of the corresponding triple set. In this paper the term *RDF-graph* is used as a synonym for the term *triple set*. A triple set (or any subset) can be serialized in different languages, such as RDF/XML, N-Triples, N3, etc. The result is a sequence of words over an alphabet defined by the particular RDF serialization language.

A DLG encodes two different types of information: structural information and label information. Encrypting structural information means to hide the topology of the graph, whereas encrypting the label information means to hide individual

node and arc values. With regard to RDF-graphs, label information is encoded by the URI-references and literals of subjects, predicates and objects. Structural information is encoded in terms of triples. It should be noted that blank nodes only provide structural information but no label information.

RDF-graphs can be interpreted as restricted DLGs having the following properties: (1) structural and label information of nodes are both encoded in terms of URI-references and literals – changing a node label also changes the structure of the graph; (2) node labels can be distributed over several triples. Thus, changing a node label can cause several triples of an RDF-graph to be changed; (3) all nodes are connected by at least on arc – there are no isolated nodes. Thus, the encryption of a triple, can cause the encryption of the connected subject and object nodes if they are only connected by that triple; (4) RDF makes the constraint, that subject and predicate labels have to be URIs. Encrypted labels are not words of the URI language. Therefore, encrypted labels have to be represented as objects which can have arbitrary literal values. As a consequence, graph transformations have to be performed in order to keep an encrypted graph RDF-compliant.

The following three encryption types for RDF-graphs can be distinguished: (1) encryption of subjects and objects (= encryption of node labels) (2) encryption of predicates (= encryption of arc labels) (3) encryption of triples (= encryption of nodes, arcs and subgraphs. An arc is represented by a single triple, a node by a set of triples having the node label either as subject or object, and a subgraph can be any subset of a triple set).

3 Partial RDF Encryption

Partial RDF Encryption (PRE) is a transaction which is composed of the six steps showed in Fig. 1. We will briefly describe each step.

1. **Fragment Selection:** The first step is the selection of the RDF fragments to be encrypted. RDF fragments can either be subjects, predicates, objects, or triples. The selected fragments are called *encryption fragments* and the remaining fragments are called *plaintext fragments*. Selection can be done, for example, by explicitly enumerating the encryption fragments (static selection), by specifying selection patterns which check specific properties (dynamic selection), by random selection, etc. This step is described in more detail in section 6.
2. **Encryption:** In this step, each encryption fragment is serialized and encrypted. The result of this step is a data structure containing both, encrypted data and encryption metadata. We will call this structure an *Encryption Container (EC)*. An encryption container can be serialized and represented as literal value. This step is described in more detail in section 4.
3. **Encryption Transformations:** All encryption fragments are replaced by their corresponding encryption containers. The result is a single self-describing RDF-compliant graph containing three different kinds of components: (1) encrypted data, (2) encryption metadata and (3) plaintext fragments. In order to fulfill RDF well-formedness constraints – in particular

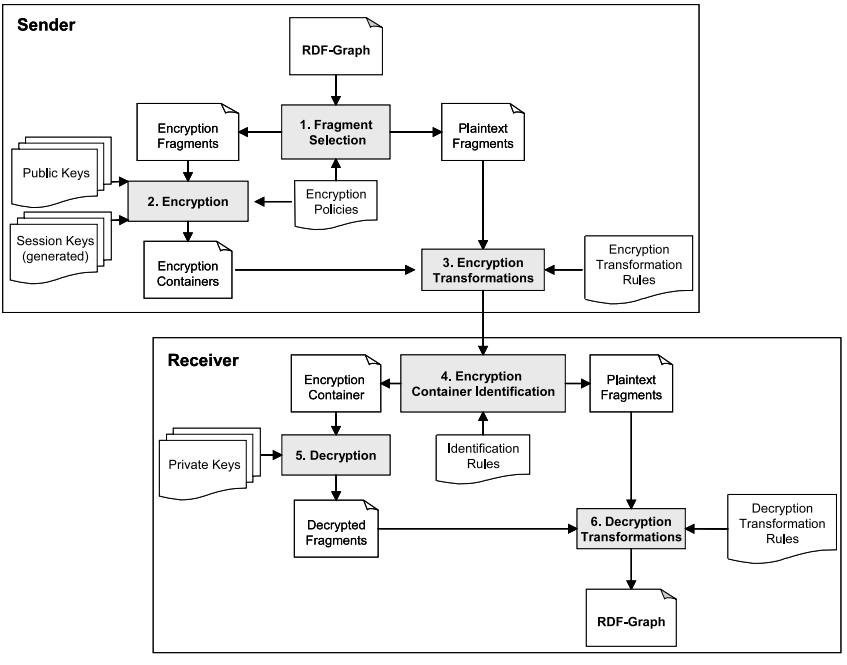

Fig. 1. Partial Encryption Process

the constraint that literals are only allowed as the object of a triple – graph transformations have to be performed. This step is described in more detail in section 5.

4. **Encryption Container Identification:** Encryption containers and encryption metadata are identified and extracted. This can be done by using an RDF query language.
5. **Decryption:** In this step, the encryption containers are decrypted according to the parameters specified in the encryption metadata. If a receiver does not have an appropriate decryption key, the decryption fails.
6. **Decryption Transformations:** The last step is the re-construction of the RDF-graph by replacing the encryption containers with the corresponding decrypted values. Graph transformations have to be performed which are inverse to the encryption transformations in step three. If a recipient has the keys to decrypt all encryption containers, then the re-constructed RDF-graph is identical to original RDF-graph. In the case that keys are missing, there will be remaining encryption containers in the RDF-graph.

4 Encryption of RDF Fragments

A cryptosystem can formally be described as a tuple (P, C, K, E, D), where P is a set of plaintexts, C is a set of ciphers, K is a set of keys, $E = \{e_k : k \in K\}$

is a family of encryption functions $e_k : P \rightarrow C$ and $D = \{d_k : k \in K\}$ is a family of decryption functions $d_k : C \rightarrow P$. For all $k_e \in K$ there is a $k_d \in K$ so that $d_{k_d}(e_{k_e}(p)) = p$ holds for all $p \in P$. A cryptosystem is called symmetric if $k_e = k_d$. It is called asymmetric if $k_e \neq k_d$. Examples of symmetric cryptosystems are Triple-DES [20] and AES [21]. An example of an asymmetric cryptosystem is RSA [24].

4.1 Encryption Schemes

For secure key transport and in consideration of performance, plaintexts are usually encrypted by using a session-key scheme which combines symmetric and asymmetric encryption (Fig. 2). The sender encrypts a plaintext m using a symmetric encryption function f parameterized with a randomly generated session key k. The result is a cipher c_m. To transmit the session key to the recipient in a secure way, k is encrypted with an asymmetric encryption function g parameterized by the public key *pub* of the recipient. The result is a cipher c_k. Then the ciphers c_m and c_k are transmitted. The recipient recovers the session key k by decrypting c_k using the decryption function g^{-1} parameterized with its private key *priv*. Finally, the recipient computes the plaintext m from c_m using f^{-1} parameterized with k.

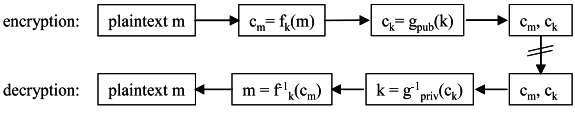

Fig. 2. Session-Key Scheme

We can extend the above session-key scheme to be able to encrypt a set of messages for a set of recipients. Let $M = \{m_1, \ldots, m_m\}$ be a non-empty set of messages to be encrypted, $P = \{pub_1, \ldots, pub_n\}$ be a non-empty set of public keys, and $P_i \subseteq P$ be a non-empty set of public keys representing the recipients of message $m_i \in M$. For each message m_i a new session key k_i is generated. m_i is encrypted using the symmetric function f paramerized by k_i. Then k_i is encrypted $|P_i|$-times using the asymmetric functions g parameterized by $pub_i \in P_i$ (Fig. 3). The encryption of M takes $|M|$ symmetric and $\sum_{i=1}^{|M|} |P_i| \leq |M| \cdot |P|$ asymmetric encryption function calls.

For each message, the extended session-key scheme creates a set of key ciphers $c_{k_1}, \ldots, c_{k_n}$ of which at the most one can be correctly decrypted using a given private key. A naive approach would be to decrypt sequentially each key cipher and to check the integrity of the decrypted values. Providing additional information about the public keys used for encryption (such as finger prints, certificate serial number, etc.) helps to identify the corresponding private key in advance. Thus, key information is an important class of encryption metadata.

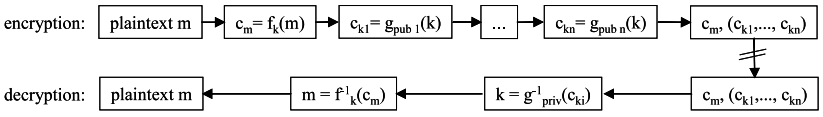

Fig. 3. Extended Session-Key Scheme

4.2 Digests

When using cryptosystems, a method to ensure the data integrity is needed. A common approach for this problem, is to use one-way hash functions, for example SHA-1 [19] or MD5 [23]. A hash or digest is a sequence of bytes that represents the input in a unique way and usually is smaller than the input. The sender computes the digest d_m of a message m using a one-way hash function h. Both, the digest d_m and the cipher c_m are transferred to the recipient. The recipient decrypts the cipher (let m' be the decrypted cipher) and computes the digest $d_{m'} = h(m')$. If $d_{m'} = d_m$ then $m' = m$ holds.

An important idea in PRE is using hash values for merging RDF-graphs, similar to the inverse functional property *mbox_sha1sum* defined in the FOAF vocabulary [9]. *mbox_sha1sum* contains the digest of an email to prevent publishing the email but to allow for merging based on the email. Partially encrypted fragments of an RDF-graph can be used for merging, if they (1) are object fragments, are inverse functional, and provide a direct hash value and (2) are subject fragments and provide a direct hash value.

There are cases in which it is not secure to use direct hash values. For example when the range of a property only contains few values. When using a direct hash for a 4-digit bank account PIN, it takes less than 1000 tests to know the correct PIN by comparing the hash values. In this case a randomization of the value before computing the digest is necessary. Randomized hash values provide a higher security. They still can be used for testing the data integrity but cannot be used for merging. So it is a trade-off between security and data integration.

For the representation of randomized values, we use a simple XML-based method. The original fragment serialization is embedded as the content of a `FragmentValue` element and can be retrieved using a simple XPath expression. `FragmentValue` is a child of `RandomizedValue` which contains randomly generated bytes as text. The structure is described by the following schema fragment.

```
<xs:complexType name='RandomizedValue' mixed='true'>
 <xs:choice><xs:element name='FragmentValue' type='xs:string'/></xs:choice>
</xs:complexType>
```

4.3 Encryption Metadata

To allow for a abstract definition of the encryption process, encryption metadata has to be specified, such as the encryption algorithms and their parameters, the computed hash values, key information for public key identification, canonicalization methods, transformation to be performed, etc. The encryption metadata

is stored together with the ciphers in a single data structure – the *Encryption Container (EC)*. There are different approaches to integrate encryption containers into RDF-graphs. We take the approach of serializing the encryption containers into XML and including the serializations as XML literals.

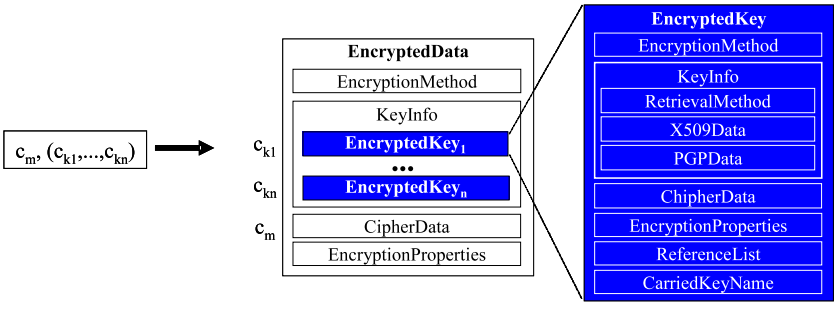

Fig. 4. Overall Encryption Container Structure

The general EC structure is shown in Fig.4. The key ciphers $c_{k_1}, \ldots, c_{k_n}$ are each stored in an *EncryptedKey* slot and the message cipher c_m is stored in an *EncryptedData* slot. Both, *EncryptedKey* and *Encrypted Data* have a similar structure. The XML-Encryption recommendation [13] provides a detailed description about the structure. The *EncryptionMethod* slot specifies the encryption algorithm. Each algorithm has a unique URI (cf. [13]). The *KeyInfo* slot provides information about the key used for encrypting the cipher. When using the extended session-key scheme, the *KeyInfo* slot inside *EncryptedData* contains a sequence of *EncryptedKey* slots, whereas the *KeyInfo* slot inside *EncryptedKey* contains information about the public key, for example a certificate or a certificate reference. The *CipherData* slot stores the concrete cipher value computed by the encryption function as Base64 encoded string. The *EncryptionProperties* slot contains additional information such as the digest value, the digest algorithm, data type information, the language used for serializing the data, etc.

Example 1: Alice has annotated the resource `http://www.xy.de/alice.htm` in RDF. To access the resource, a username and password is needed. Alice wants to store the access data together with other annotations in the same RDF-graph, so that only Bob and Chris can read the access data while all other annotations are publicly readable. Alice has the X.509 certificates of Bob and Chris and wants to encrypt the following RDF triples. AES (with 128-bit key size), RSA and SHA-1 is to be used.

```
<http://www.xy.de/alice.htm> <http://xy.de/schema#username> "alice" .
<http://www.xy.de/alice.htm> <http://xy.de/schema#password> "secret" .
```

First, the triples are serialized using an RDF language (N-Triples [15] in this example). Second, the SHA-1 digest is computed. Then, the data is AES encrypted

(in CBC mode) with a generated 128-bit session key k. Then k is RSA encrypted twice using the RSA public keys contained in the X.509 certificates of Bob and Chris. Finally, the ciphers, the digest, the certificate, and the algorithm names and parameters are combined in an encryption container. An XML-Encryption and XML-Signature conforming serialization looks like:

```
<xenc:EncryptedData>
  <xenc:EncryptionMethod Algorithm="&xenc;#aes128-cbc"/>
  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm="&xenc;#rsa-1_5"/>
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate>MIICQjCCAasCBE...</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>rrOC4FYSNogKsi...</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
    <xenc:EncryptedKey>encrypted key of Chris...</xenc:EncryptedKey>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>37++haErMYLidG...</xenc:CipherValue>
  </xenc:CipherData>
  <xenc:EncryptionProperties>
    <xenc:EncryptionProperty>
      <ds:DigestMethod Algorithm="&ds;#sha1"/>
      <ds:DigestValue>/84Cdz6BdYd6kY9zSa6sT1IjLoo=</ds:DigestValue>
    </xenc:EncryptionProperty>
  </xenc:EncryptionProperties>
</xenc:EncryptedData>
```

5 Transformations

Since in RDF only the objects can represent literal values, encrypted subjects and predicates cannot directly be replaced by their corresponding encryption container serializations. Instead, graph transformations have to be performed. An overview of the transformations for integrating the encrypted content is given in Fig. 5 (literals containing the encryption container serialization are marked with a 'lock' icon). We will briefly describe each transformation.

1. *Subject Transformation*: In order to encrypt a subject S, a new triple $\langle B,\ \textit{renc:encNLabel},\ EC_S\rangle$ is added to the graph. EC_S contains the XML serialization of the encryption container of S. All references to S are replaced by references to B. Therefore all triples containing S either as object or subject have to be changed.

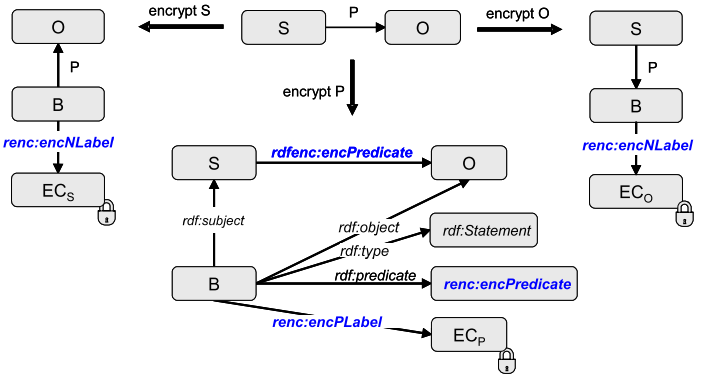

Fig. 5. Subject, Object and Predicate Transformations

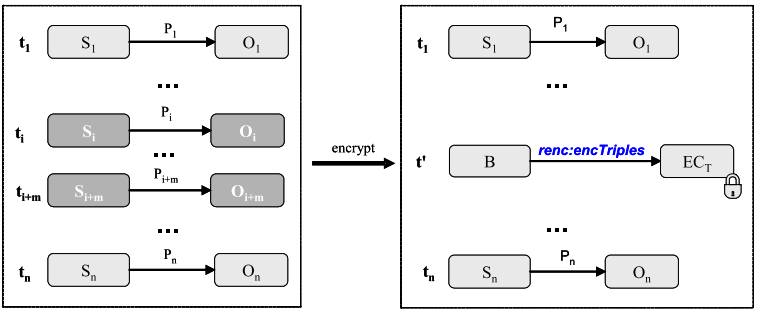

Fig. 6. Triple-Set Transformation

2. *Object Transformation*: Objects could directly be replaced by their encryption container serializations. But this would also change the datatype into `rdf:XMLLiteral`. Therefore, a blank node is introduced which replaces the original object node. A new triple $\langle B,\ renc{:}encNLabel,\ EC_O \rangle$ is added to the graph. EC_O contains the XML serialization of the encryption container of O including the original datatype information. All references to O have to be replaced by references to B.
3. *Predicate Transformation*: Since in RDF only URI references are allowed as predicates, blank nodes cannot be used for bridging between arcs and their encrypted label data. Instead a RDF reification [18] based approach is used. The transformation is carried out in three steps. First, the predicate P of the original triple t is replaced by the URI reference `renc:encPredicate`. Second, a new reification quad is added for identifying t. Finally, a new property $renc{:}encPLabel = EC_P$ is added to the reification quad stating that the real predicate of t is encrypted in EC_P.
4. *Triple-Set Transformation*: The encryption of a non-empty triple set $T_{enc} = \{t_i, \ldots, t_{i+m}\}$ takes the following steps. First, T_{enc} is serialized into a string s

using an RDF serialization language. Second, an encryption container EC_T is constructed containing the encrypted string s together with the encryption metadata. Third, a new triple $\langle B,\ renc{:}encTriples,\ EC_T \rangle$ is added to the graph. Finally, all triples in T_{enc} are removed from the graph. The transformation for triple sets are showed in figure 6.

5.1 Handling of Blank Nodes

The described transformations can be directly applied to RDF-graphs that do not contain blank nodes (ground graphs). As noted earlier, a blank node identifier is not regarded as node label and thus cannot be encrypted. However blank nodes may be contained in triple sets that are to be encrypted. Blank node identifiers have to be unique in one RDF-graph. They are not required to be globally unique and may be changed to some internal representation by RDF frameworks. In order to be able to encrypt triples containing blank nodes, additional information is needed to uniquely identify the blank nodes after decryption, since their identifiers might have changed.

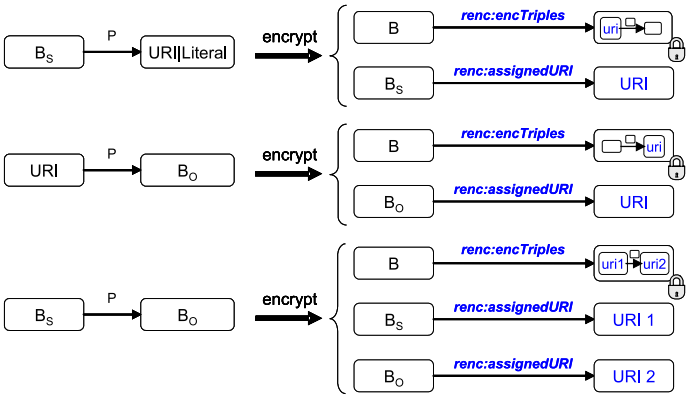

Fig. 7. Graph Transformations for Blank Nodes

Therefore, a unique UUID [17] is generated for each blank node contained in a triple to be encrypted. The UUID is assigned to the blank node as URI value of the `renc:assignedURI` property. The blank nodes of the triples to be encrypted are then replaced by the generated URIs. During decryption, the generated URIs are used for identifying the original blank nodes. Blank nodes can occur as the subject of a triple, as the object of a triple or both. Fig. 7 gives an overview and shows the corresponding transformations.

Example 2: Alice wants to encrypt the `foaf:knows` relation between her and Bob expressed by triple t_{enc}. Since persons have no adequate URI representation, blank nodes are used for bundling properties about the person which are the

email addresses in this example. Fig. 8 shows the result of the encryption. The triple t_{enc} is removed. Three new triples are added: two triples for identifying the blank nodes (t_1 and t_2) and one triple containing the encrypted data (t_3). The blank node identifiers for B_1 and B_2 are replaced by the generated UUIDs ($uri1$ and $uri2$) before the encryption. During decryption the t_3 is decrypted, parsed, and removed. Let T_{dec} denote the decrypted triples. In a second step, the objects of all triples having an `renc:assignedURI` predicate are tested against the subject and object URIs of the triples in T_{dec}. If a correspondence is detected (the object of t_1 with $uri1$ and the object of t_2 with $uri2$), the URI references are replaced by the corresponding blank nodes and the identification triples (t_1, t_2) are removed.

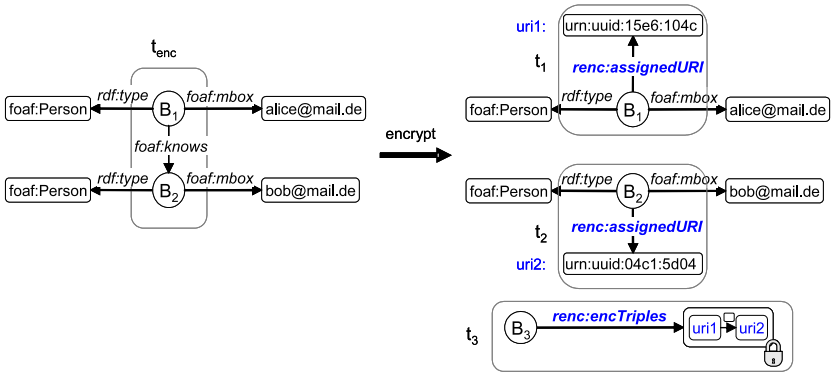

Fig. 8. Blank Node Example

6 Encryption Policies

Encryption policies for RDF-graphs define *which* fragments to encrypt and *how* to encrypt them. The *PRE Policy Language (PRE-PL)* uses a graph pattern based approach that allows for dynamic selection of encryption fragments. PRE-PL uses the RDQL [25, 2] query language. The result of a query can be interpreted as a set of fragments which are instances of the same 'category' defined by the search pattern. Each category is encrypted in the same way (the same keys, algorithms, etc.). RDQL mainly defines a list of triple patterns which are mapped to concrete triples in an RDF-graph. A triple pattern generally has the form

```
TriplePattern ::= '(' (Var|URI) (Var|URI) (Var|Const) ')'
```

where `Var` are variables, `URI` are URI references and `Const` are URI references or (typed) literals. The result of a query is a set of bindings, in which the variables are bound to concrete RDF items (subjects, predicates or objects).

RDQL has been adapted in a way that it returns a set of triples bound to each triple pattern instead of returning variable bindings. Based on the ordered triple pattern sequence, the encryption fragments are identified by using the markers s, p, o, or t. The marker s (p, o) will cause the encryption of the subjects (predicates, objects) of the bound triple set. The marker t will cause the encryption of each triple in the set. This mechanism allows the encryption of fragments which are not bound to variables, e.g. named values. Additionally, it has to be specified how to encrypt the selected fragments, i.e. which encryption method, keys, parameters, etc. to use. PRE-PL is described in more detail in on the RDF Encryption Project site [3]. We will give a short example here.

Example 3: The rule *„encrypt the email addresses of all persons"* using Triple-DES as block cipher algorithm and the RSA keys provided in the certificates of Bob and Alice can be formulated in PRE-PL as follows:

```
<pre:PREPolicy>
  <ds:KeyInfo>
    <ds:X509Data id="alice">...</ds:X509Data>
    <ds:X509Data id="bob">...</ds:X509Data>
  </ds:KeyInfo>
  <pl:DefaultEncryptionScheme>
    <pl:Symmetric>
      <xenc:EncryptionMethod Algorithm="xenc:tripledes-cbc"/>
    </pl:Symmetric>
    <pl:Asymmetric>
      <xenc:EncryptionMethod Algorithm="xenc:rsa-1_5"/>
    </pl:Asymmetric>
    <pl:Digest type="pl:directDigest">
      <ds:DigestMethod Algorithm="ds:sha1"/>
    </pl:Digest>
    <pl:RDFLanguage name="pl:N-Triples"/>
    <pl:DefaultKeys><pl:KeyRef id="alice"/></pl:DefaultKeys>
  </pl:DefaultEncryptionScheme>
  <pl:GraphPattern>
    <pl:TriplePattern subj="?x" pred="rdf:type" obj="foaf:Person"/>
    <pl:TriplePattern subj="?x" pred="foaf:mbox" obj="?y">
      <pl:Encryption target="o"><KeyRef id="bob"/></pl:Encryption>
    </pl:TriplePattern>
  </pl:GraphPattern>
</pl:PREPolicy>
```

Each `PREPolicy` has a `KeyInfo` section for key definition. Each child element provides key material which is referenced in the `GraphPattern` sections. Note, that the external keys can be referenced using the XML-Signature reference mechanism [14]. Each PRE policy also defines one `DefaultEncryptionScheme` section which defines the default encryption parameters: the symmetric and asymmetric algorithms, the digest algorithm and additional randomization, the RDF serial-

ization language for triples and the default keys for each fragment. Additional encryption schemes can be defined which can be referenced in `Encryption` elements. Each `GraphPattern` section has a list of triple patterns and optional constraints which are mapped to an RDQL query. For each `TriplePattern` it can be defined how to encrypt the bound fragments. In the above example, the object of the second triple pattern (the email address) is encrypted using the default encryption scheme and the additional key with the ID 'bob'.

7 Conclusions and Future Work

A method to partially encrypt RDF-graphs has been presented. It differs from other approaches in that the result is a single self-describing RDF-compliant graph containing both, encrypted data and plaintext data. The method allows for fine-grained encryption of subjects, objects, predicates and subgraphs of RDF-graphs. Encrypted fragments are included as XML literals which are represented using the XML-Encryption [13] and XML-Signature [14] recommendations. Graph transformations necessary to keep the encrypted RDF-graph well-formed have been described. The proposed method is adoptable for different algorithms and processing rules by using encryption metadata. We have motivated the usage of randomized digests for high-sensitive data (such as credit card number, passwords, etc.) and direct digests for low-sensitive data (such as email, phone number, etc.) in order to allow a trade-off between security and application integration needs. We have also introduced the idea of encryption policies for RDF and the PRE-PL policy language which uses RDQL queries for dynamic selection of encryption fragments. In future work we will integrate SPARQL [22] concepts, such as optional pattern matching, in PRE-PL. A prototypical implementation of PRE, PRE4J [3], is available under LGPL for the Jena Framework [2].

PRE heavily relies on a public key infrastructure or on a web of trust. There are RDF vocabularies, such as the Semantic Web Publishing Vocabulary [10, 7] or the WOT Vocabulary [8], for integrating certificates into the Semantic Web and in particular into FOAF [9] profiles. Therefore, it is planned to extend FOAF enabled browsers, such as the Foafscape browser [1], to be able to use the certificates provided in profiles.

As with all partial encryption methods, encrypted data has a certain context which can be used for 'guessing' the corresponding plaintext data. Semantic Web applications also typically make use of ontologies. An ontology formulates a strict conceptual scheme about a domain containing the relevant classes, instances, properties, data types, cardinalities, etc. This information can be used for attacks or even inferring encrypted content. Property definitions for example can dramatically reduce the search space for 'guessing' the plaintexts and can be used for systematically checking the hash value provided in the encryption container. Concerning the confidentiality of encrypted data, it is also crucial to know if the data to be encrypted is inferable. This topic has not been evaluated in detail, yet.

References

1. Foafscape Project Homepage. http://foafscape.berlios.de.
2. Jena Semantic Web Framework. http://jena.sourceforge.net.
3. RDF Encryption Project Homepage. http://rdfenc.berlios.de.
4. L. Bauer, M. Schneider, and E. Felten. A general and flexible access-control system for the web. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, Aug 2002.
5. T. Berners-Lee, R. Fielding, and L. Masinter. *RFC 2396 – Uniform Resource Identifiers (URI): Generic Syntax*. IETF, August 1998. http://www.isi.edu/in-notes/rfc2396.txt.
6. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 34–43, May 2001.
7. C. Bizer, R. Cyganiak, O. Maresch, and T. Gauss. TriQL.P - Trust Architecture. http://www.wiwiss.fu-berlin.de/suhl/bizer/TriQLP/.
8. D. Brickley. WOT RDF Vocabulary, 2002. http://xmlns.com/wot/0.1/.
9. D. Brickley and L. Miller. FOAF Vocabulary Specification, 2005. http://xmlns.com/foaf/0.1/.
10. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named Graphs, Provenance and Trust. Technical report, HP Laboratories Bristol, 2004. HPL-2004-57R1.
11. J. Cowan and R. Tobin, editors. *XML Information Set (Second Edition)*. W3C Recommendation, February 2004. http://www.w3.org/TR/xml-infoset/.
12. E. Dumbill. PGP Encrypting FOAF Files, 2002. http://usefulinc.com/foaf/encryptingFoafFiles.
13. D. Eastlake and J. Reagle, editors. *XML Encryption Syntax and Processing*. W3C Recommendation, December 2002. http://www.w3.org/TR/xmlenc-core/.
14. D. Eastlake, J. Reagle, and D. Solo, editors. *XML-Signature Syntax and Processing*. W3C, February 2002. http://www.w3.org/TR/xmldsig-core/.
15. J. Grant and D. Beckett, editors. *RDF Test Cases*. W3C Recommendation, http://www.w3.org/TR/rdf-testcases/, February 2004.
16. G. Klyne and J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, http://www.w3.org/TR/rdf-concepts/, February 2004.
17. P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace, July 2005.
18. F. Manola and E. Miller, editors. *RDF Primer*. W3C Recommendation, http://www.w3.org/TR/rdf-primer/, February 2004.
19. National Institute of Standards and Technology (NIST). Secure Hash Standard (SHA-1). Technical report, April 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm.
20. National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Technical report, October 1999. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.
21. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). Technical report, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
22. E. Prud'hommeaux and A. Seaborne, editors. *SPARQL Query Language for RDF*. W3C Working Draft, October 2004. http://www.w3.org/TR/rdf-sparql-query/.
23. R. Rivest. The MD5 Message-Digest Algorithm, RFC 1321. Technical report, April 1992. http://www.faqs.org/rfcs/rfc1321.html.

24. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM 21,2*, 1978.
25. A. Seaborne, editor. *RDQL - A Query Language for RDF*. W3C Member Submission, January 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
26. D. Weitzner, J. Hendler, T. Berners-Lee, and D. Connolly. Creating a policy-aware web: Discretionary, rule-based access for the world wide web. Hershey, PA (forthcoming), 2004.

Appendix: Namespaces

Prefix	Namespace
ds	http://www.w3.org/2000/09/xmldsig#
foaf	http://xmlns.com/foaf/0.1/
pl	http://rdfenc.berlios.de/pre-pl#
renc	http://rdfenc.berlios.de/pre#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
xenc	http://www.w3.org/2001/04/xmlenc#
xs	http://www.w3.org/2001/XMLSchema