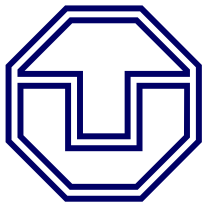


TECHNISCHE UNIVERSITÄT DRESDEN

Belegarbeit

Untersuchung von Möglichkeiten zur Datensicherheit in RDF

31. Oktober 2005

Fakultät Informatik
Institut für Systemarchitektur

Verfasser:	Sabrina Gerbracht
Betreuer(in):	Dipl.-inf. Sebastian Clauß
Verantwortlicher Hochschullehrer:	Prof. Dr. rer. nat. Andreas Pfitzmann

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Resource Description Framework (RDF) wird heute immer öfter und in immer mehr Bereichen der Datenverarbeitung eingesetzt. Unter anderem auch in Bereichen, in denen sensitive Daten gespeichert werden. Bislang gibt es jedoch keine Möglickeit, um diese sensitiven Daten auch innerhalb des Resource Description Frameworks zu schützen.

Um diesem Umstand Abhilfe zu schaffen, wird in diesem Beleg eine Untersuchung der Möglichkeit zur Datensicherheit in RDF vorgenommen. Diese Analyse basiert sowohl auf einer Suche nach bestehenden, als auch aus dem Neuentwurf von Möglichkeiten.

Die Untersuchung konzentriert sich insbesondere auf die Konzelation.

Kapitel 2

Grundlagen der Datensicherheit

„Datensicherheit bezeichnet den Schutz von Daten hinsichtlich gegebener Anforderungen an deren Vertraulichkeit, Verfügbarkeit und Integrität.“[1]

2.1 Schutzziele

In [Pfi00] werden diese Schutzziele für Daten benannt und genauer definiert. So wird hier die Vertraulichkeit als der Zustand beschrieben, in dem Informationen ausschließlich Berechtigten bekannt gegeben werden. Unter Integrität wird die Richtigkeit, Vollständigkeit und Aktualität von Informationen oder die Erkennbarkeit, dass dies nicht der Fall ist, verstanden. Die Verfügbarkeit wiederum definiert sich durch die zeitliche und räumliche Zugänglichkeit der Informationen.

2.2 Verschlüsselung

Um diese Schutzziele zu erreichen gibt es verschiedene Möglichkeiten, die in [Pfi00] ausführlich beschrieben werden. Die gängigste Möglichkeit um Vertraulichkeit zu erreichen ist die Verschlüsselung von Informationen. Hierbei wird die betreffende Information für Dritte unleserlich gemacht, so dass nur Berechtigte, in diesem Fall alle Personen mit dem richtigen Schlüssel, die Information wieder auslesen können.

In diesem Beleg werden insbesondere zwei Verfahren zur Verschlüsselung in Anspruch genommen. Zum Einen das symmetrische Verschlüsselungsverfahren Triple-DES und zum Anderen das asymmetrische Verfahren RSA. Unter symmetrischen Verfahren versteht man, dass zur Verschlüsselung und zur Entschlüsselung identische Schlüssel verwendet werden. Bei der asymmetrischen Verschlüsselung werden für die beiden Vorgänge unterschiedliche Schlüssel verwendet[2].

[1] Zitat aus dem Glossar des [bsi].
[2] [Pfi00]

2.2.1 Blockchiffre

Eine Variante der Verschlüsselung sind Blockchiffren. Hierbei wird ein Klartext in Blöcke gleicher Länge unterteilt, zum Beispiel bei DES in Blöcken zu 64 bit, und Block für Block verschlüsselt.

2.3 Hashfunktionen

In diesem Beleg werden insbesondere kollisionsresistente Einweg-Hashfunktionen verwendet, wie zum Beispiel Tiger. Eine Hashfunktion bildet eine Zeichenkette auf eine Andere ab, wobei eine minimale Änderung im Originalbereich eine möglichst große Änderung im Bildbereich bewirken sollte. Wichtigste Eigenschaft einer Hashfunktion ist, dass ein und dieselbe Zeichenkette immer auf die gleiche Zeichenkette im Bildbereich abgebildet wird.

Unter Einweg-Hashfunktionen werden Funktionen verstanden, bei der die Abbildung vom Original- in den Bildbereich möglich ist, jedoch eine Umkehrfunktion nicht existiert.

Da eine Hashfunktion immer von einem großen Originalbereich auf einen kleineren Bildbereich abbildet sind so genannte Kollisionen, Abbildungen zweier Zeichenketten aus dem Originalbereich auf ein und die selbe Zeichenkette des Bildbereichs möglich.

Schwach kollisionsresistente Hashfunktionen sind so konstruiert, dass es ein schweres Problem darstellt zu einem gegebenen Text T, der mit einer Hash-Funktion h verarbeitet wurde, einen zweiten Text T' zu finden, der das selbe Ergebnis der Hashfunktion liefert.

Bei stark kollisionsresistenten Hashfunktionen ist es schweres Problem, zwei Texte T und T' zu finden, die mit einer Hash-Funktion h verarbeitet das selbe Ergebnis liefern.

Die Wahrscheinlichkeit einer Kollision wird durch diese beiden Definitionen sehr gering.

2.4 Signierung

Um das Schutzziel der Integrität zu erreichen kann eine Information signiert werden. Auf dieses Verfahren wird in diesem Beleg jedoch nicht weiter eingegangen, da in [Car03] bereits eine Methode zum Signieren von RDF-Graphen untersucht wurde. Auch die Verfügbarkeit, das dritte Schutzziel der Datenssicherheit, wird hier nicht weiter untersucht.

Kapitel 3

Einführung in RDF

Das Resource Description Framework (RDF) stellt eine Sprache dar, mit deren Hilfe Beziehungen zwischen Daten und Metadaten abgebildet werden können. Sie wird durch das World Wide Web Consortium (W3C) gestützt und empfohlen. Die genauen Spezifikationen[1] sind auf der Website des W3C[2] zu finden.

RDF basiert auf zwei grundlegende Ideen. Die erste Idee ist, dass alle Elemente der Sprache durch sogenannten Uniform Resource Identifiers[3], kurz URIs, representiert werden. Diese können als Uniform Resource Names (URN) und als Uniform Resource Locators (URL) auftreten. URNs identifizieren eine Ressource mittels eines Namens, während URLs Ressourcen durch ihren Speicherort bestimmen. Die zweite Idee widmet sich der Anordnung der Daten. Jeder Datensatz in RDF muss aus einem Subjekt, einem Prädikat und einem Objekt bestehen. Dadurch wird eine Darstellung von Beziehungen erst möglich. Auf Grund dieser Form werden die Datensätze auch Tripel genannt und wie folgt in der „Notation 3“, kurz N3, niedergeschrieben.

```
<http://www.example.org/chapter1.html>
<http://www.example.net/partOf>
<http://www.example.org/book>.
```

Hierbei ist der URL $http://www.example.org/chapter1.html$ das Subjekt,
der nachfolgende URL $http://www.example.net/partOf$ das Prädikat und
der letzte URL $http://www.example.org/book$ das Objekt des Tripels.

Folglich lässt sich dieses Tripel auch als Satz darstellen.

[1] [MM04, KC04, Hay04, BG04, GB04, Bec04]

[2] www.w3.org

[3] Eine genaue Erläuterung der URIs ist unter http://de.wikipedia.org/wiki/Uniform_Resource_Identifier zu finden.

```
The site http://www.example.org/chapter1.html
is http://www.example.net/partOf the adress
http://www.example.org/book.
```

Eine weitere Möglichkeit ist die Darstellung als Graph, welche bei kleinen Modellen bei weitem die Übersichtlichste darstellt.

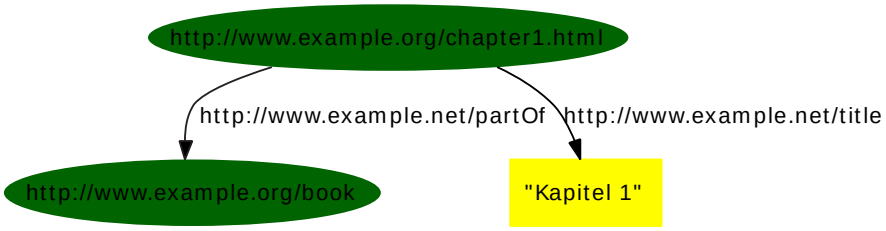


Abbildung 3.1: Ein simpler RDF-Graph mit Literal.

In der Graphennotation werden die Subjekte und Objekte als Ellipsen, die Prädikate als gerichtete Pfeile und Literale (auf die später noch näher eingegangen wird) als Rechtecke dargestellt.

Der Übersichtlichkeit halber verwendet man bei diesen drei Darstellungsformen oftmals eine Kurzform, welche den Namespaces bei XML sehr ähnlich ist. Es werden in jedem Dokument Präfixe für die verschiedenen URIs definiert und in den folgenden Angaben des Dokuments verwendet.

```
@prefix ex1: <http://www.example.org/>.
@prefix ex2: <http://www.example.net/>.

ex1:chapter1.html     ex2:partOf     ex1:book.
```

Es gibt in RDF eine Reihe von Standard-URIs, welche immer wieder verwendet werden. Ihnen sollte in jedem RDF Dokument die gleichen Präfixe zugewiesen werden.

Das Standard-RDF-Vokabular:

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

das RDF-Schema-Vokabular:

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

das Vokabular des Dublin Core:

@prefix dc: <http:purl.org/dc/elements/1.1/>.

die Web-Ontology-Language:
@prefix owl: <http://www.w3.org/2002/07/owl#>.
und zu guter Letzt das XML-Schema:
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

Im weiteren Verlauf werden diese Präfixe vermehrt verwendet.

3.1 Konzepte

Es gibt drei verschiedene Arten von Konzepten, die es zu unterscheiden gilt: Die grundlegenden Konzepte, die Schema-Definitionskonzepte (zum Einführen von neuen Vokabeln) und die Hilfskonzepte.

Alle Konzepte haben einen eigenen URI und werden in W3C-Dokumenten definiert. Die Dokumente, die die folgenden Konzepte definieren sind unter [W3C99] und [W3C01] zu finden. Da RDF und RDFS eng miteinander verbunden sind und aufeinander referenzieren ist es sinnvoll immer beide Konzepte zu verwenden. Aus diesem Grund werden beide Konzepte gleichgestellt behandelt.

3.1.1 Grundlegende Konzepte

rdfs:Resource

URI: http://www.w3.org/2000/01/rdf-schema#Resource

RDF dient dazu Ressourcen zu beschreiben. Ressourcen sind Quellenangaben, ähnlich dem URL. Jeder URI in RDF vertritt eine Quelle. Somit gehören sowohl Subjekte und Prädikate als auch Objekte zu den rdfs:Resources.

rdf:Property

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#Property

rdf:Property ist eine Unterklasse von rdfs:Resource und beschreibt die Eigenschaften eines Tripels, welche durch das Prädikat beschrieben sind. Dadurch wird die Beziehung der beiden Ressourcen Subjekt und Objekt hergestellt.

rdf:type

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#type

Durch rdf:type ist es möglich, noch einmal ausdrücklich anzugeben, von welchem Typ bzw. welcher Klasse eine Ressource ist.

Reification: rdf:Statement, rdf:subject, rdf:predicate und rdf:object

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#object

Mithilfe des rdf:Statement ist es möglich Aussagen über Tripel zu machen. Solche Aussagen werden im Englischen „reification“ genannt und bestehen aus vier Tripeln. Hierzu wird dem zu beschreibenden Tripel ein eigener URI zugewiesen, der dann mit den Prädikaten rdf:type, rdf:subject, rdf:predicate und rdf:object beschrieben wird.

```
ex1:chapter1.html     ex2:partOf     ex1:book.

ex1:triple321     rdf:type          rdf:Statement.
ex1:triple321     rdf:subject       ex1:chapter1.html.
ex1:triple321     rdf:predicate     ex2:partOf.
ex1:triple321     rdf:object        ex1:book.
```

Natürlich ist es auch möglich weitere Aussagen zu diesem Tripel zu machen, wie zum Beispiel, wer dieses Tripel geschrieben hat oder wann es geschrieben wurde.

3.1.2 Schema-Definitionskonzepte

rdfs:Literal

URI: http://www.w3.org/2000/01/rdf-schema#Literal

Auch im RDF ist es möglich Literale zu verwenden, um Text mit in die Relation einzubeziehen. Literale können jedoch nur als Objekt eines Tripels auftreten und werden in der N3 mit Anführungszeichen gekennzeichnet.

```
ex1:book        ex2:ID       ex1:book123.
ex1:book123     dc:title     "Informatik-Bibel".
```

rdf:XMLLiteral

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral

Eine gesonderte Form der Literale stellen die XMLLiterale dar, die es ermöglichen XML-Fragmente in eine RDF-Beziehung einzubinden.

rdfs:Datatype

URI: http://www.w3.org/2000/01/rdf-schema#Datatype

Da RDF selber kein eigenes Verständnis von Fakten aufweist, ist es notwendig auftretenden Zahlen einen Datentyp zuzuweisen. Erst hierdurch sind verschiedene Angaben für Rechner und zum Teil auch für den Menschen interpretierbar.

```
ex1:book123    ex2:pages    "350"^^xsd:integer.
```

Es ist ebenfalls möglich ungetypte Literale zu verwenden, jedoch ist dies zumeist nur bei Textliteralen sinnvoll.

rdfs:subPropertyOf

URI: http://www.w3.org/2000/01/rdf-schema#subPropertyOf

Wie zuvor schon beschrieben betiteln die Properties die Relationen zwischen zwei Ressourcen. Mit rdfs:subPropertyOf ist es möglich Unterrelationen zu deklarieren, die impliziet die Hauptrelation vorraussetzen.

```
ex2:shortTitle    rdfs:subPropertyOf    dc:title.
ex1:book123       ex2:shortTitle        "RDF–Primer".
```

Dieses Beispiel sagt implizit aus, dass „RDF-Primer“ ein Titel für das Buch ist.

rdfs:Class

URI: http://www.w3.org/2000/01/rdf-schema#Class

Ähnlich wie bei Java gibt es auch in RDF eine hierarchische Klassenverwaltung. Jede Ressource kann genau einer Klasse (oder einer der Oberklassen jener Klasse) zugeordnet werden oder eine eigene Klasse darstellen. Alle Klassen sind direkte Unterklassen von rdf:Resource und können beliebig viele eigene Unterklassen besitzen.

rdfs:subClassOf

URI: http://www.w3.org/2000/01/rdf-schema#subClassOf

Mit rdfs:subClassOf kann eine Ressource als Unterklasse einer anderen Klasse bestimmt werden. Die Unterklasse erbt dabei die Eigenschaften der Oberklasse.

```
ex1:hardcover    rdfs:subClassOf    ex1:book.
ex1:book         ex2:material       "paper".
ex1:book123      ex2:bookType       ex1:hardcover.
```

In diesem Beispiel erbt das Buch ex1:book123 vom Typ ex1:hardcover die Eigenschaft der Oberklasse ex1:book, dass das verwendete Material Papier ist.

rdfs:domain

URI: http://www.w3.org/2000/01/rdf-schema#domain

Die Domäne eines Tripels gibt an, von welcher Klasse die Instanz des Subjektes in einer Relation ist.

```
ex1:book123    rdf:type       ex1:book.
rdf:type       rdfs:domain    rdfs:Resource.
```

rdfs:range

URI: http://www.w3.org/2000/01/rdf-schema#range

Der Rang eines Tripels gibt an, von welcher Klasse die Instanz des Objektes in einer Relation ist.

```
ex1:book123    rdf:type       ex1:book.
rdf:type       rdfs:range     rdfs:Class.
```

3.1.3 Hilfskonzepte

rdfs:member

URI: http://www.w3.org/2000/01/rdf-schema#member

rdfs:member ist das Oberproperty aller Elemente, die in Containern verwendet werden. Diese Elemente sind dann die ContainerMembershipProperties.

rdfs:Container

URI: http://www.w3.org/2000/01/rdf-schema#Container

Als Container werden in RDF Datensammlungen betrachtet, wobei es hier keinerlei Regeln über die Art und Häufigkeit der Daten gibt, so dass auch doppelte Elemente vorkommen können.

Diese Datensammlungen können je nach Typ des Containers geordnet oder ungeordnet sein oder aus Alternativen bestehen. rdfs:Container ist die Oberklasse aller Containertypen.

rdf:Bag

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag

rdf:Bag wird für eine ungeordnete Datenmenge benutzt. Die Reihenfolge der einzelnen Elemente spielt also keine Rolle.

rdf:Seq

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq

Um eine geordnete Datenmenge darzustellen wird rdf:Seq verwendet. Die Reihenfolge der einzelnen Elemente ist hierbei signifikant.

rdf:Alt

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt

Um alternative Angabe, zum Beispiel ein und denselben Sachverhalt in verschiedenen Sprachen, zusammenzufassen, kann rdf:Alt verwendet werden. Hier wird immer nur eins der Elemente exklusiv verwendet.

rdfs:ContainerMembershipProperty

URI: http://www.w3.org/2000/01/rdf-schema#ContainerMembershipProperty

Die Elemente in einem Container sind die ContainerMembershipPropertys. Diese sind ein Unterproperty des Propertys rdfs:member. Die ContainerMembershipPropertys sind von der Form rdf:_1, rdf:_2, ...

rdf:List

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#List

Es ist in RDF möglich verkettete Listen anzulegen. Eine Liste besteht immer mindestens aus einem Kopf- und einem Endelement, wobei das Kopf- gleich dem Endelement sein, und beliebig viele Listenelemente zwischen Kopf und Ende besitzen kann. Wie bei den Containern gibt es auch bei der Liste keine Bedingungen hinsichtlich des Auftretens von Elementen, so dass diese mehrfach enthalten sein können.

rdf:first, rdf:rest und rdf:nil

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#first
URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#rest
URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#nil

Listen werden mit den Propertys rdf:first und rdf:rest definiert. Durch rdf:first wird das aktuelle Element beschrieben, mit rdf:rest auf die restliche Liste verwiesen.

```
ex1:bookList    rdf:first    ex1:book1.
ex1:bookList    rdf:rest     ex1:bookList1.
```

Mit dem rdf:nil-Element kann der Liste ein Ende hinzugefügt werden.

```
ex1:bookList1    rdf:first    ex1:book2.
ex1:bookList1    rdf:rest     rdf:nil.
```

Die kürzeste mögliche Liste besteht aus nur einer Zeile und entspricht der leeren Liste.

```
ex1:Liste rdf:first rdf:nil.
```

rdfs:seeAlso

URI: http://www.w3.org/2000/01/rdf-schema#seeAlso

Mit rdfs:seeAlso können Referenzen auf alternative oder ergänzende Quellen angegeben werden.

rdfs:isDefinedBy

URI: http://www.w3.org/2000/01/rdf-schema#isDefinedBy

Quellen zu Definitionen von einer Ressource können durch rdfs:isDefinedBy benannt werden.

rdfs:label

URI: http://www.w3.org/2000/01/rdf-schema#label

rdfs:label ermöglicht es, einer Ressource ein Label bzw. Namen, beizufügen.

rdfs:comment

URI: http://www.w3.org/2000/01/rdf-schema#comment

Durch rdfs:comment können ergänzende Kommentare zu einer Ressource hinzugefügt werden.

3.1.4 Blank Nodes

Eine Besonderheit sind Blank Nodes. Sie ermöglichen erst Container und Collections, indem an leere Knoten mehrere Properties angehangen werden. Anonyme Blank Nodes sind jedoch nur einmal verwendbar, da sie keinen Identifier besitzen, mit dem sie wieder aufgerufen werden können.

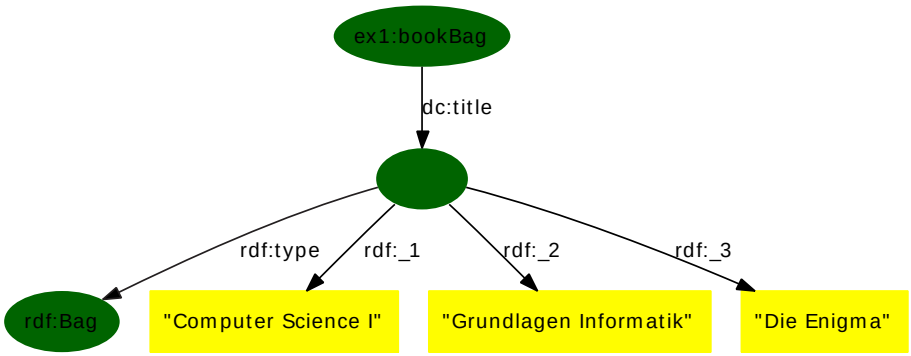


Abbildung 3.2: Ein RDF-Graph mit Blank Node.

Zu eben diesem Zweck gibt es jedoch die Blank Node Identifier, wie sie bei der Notation 3 verwendet werden.

```
ex1:bookBag     dc:title    _:blankNode1.
_:blankNode1    rdf:type    rdf:Bag.
_:blankNode1    rdf:_1      "Computer Sience I".
_:blankNode1    rdf:_2      "Grundlagen Informatik".
_:blankNode1    rdf:_3      "Die Enigma".
```

3.2 RDF und XML

Eine weitere gebräuchliche Darstellungsform von RDF ist XML. Die RDF-Vokabularien rdf:, rdfs:, dc: und Andere sind ebenso in der Präfix-Form (Namespaces) verwendbar, wenn die Präfixe zuvor deklariert werden. Hierzu bedient man sich eines umschließenden RDF-Tags:

```
<rdf:RDF
    xmlns:rdf=
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
```

```
<!-- Content -->

</rdf:RDF>
```

Leider können in XML die Blank Nodes nicht komplett dargestellt werden, da diese keinen Bezeichner besitzen. Für eine ausführliche Beschreibung der RDF-Darstellung in XML ist die Veröffentlichung [MM04], zu empfehlen.

Kapitel 4

Mögliche Modelle zur Verschlüsselung von RDF

Bei der Verschlüsselung von RDF muss unterschieden werden, in welchen Einheiten verschlüsselt werden soll. Die einfachste Möglichkeit ist den ganzen Graphen zu verschlüsseln. Hierfür reicht es, wenn das komplette Dokument verschlüsselt wird.

Eine andere Variante ist nur einzelne Ressourcen zu verschlüsseln. Zu dieser Variante werden in diesem Kapitel einige Ansätze genauer erläutert.

Das prinzipielle Vorgehen bei der Verschlüsselung von RDF unterscheidet sich nicht vom Verschlüsseln jedes anderen Textes. Es ist denkbar, dass alle kryptographischen Algorithmen verwendbar sind.

Da RDF Daten in Abhängigkeit zu anderen Daten abspeichert, muss die Verschlüsselung gewährleisten, dass diese Abhängigkeiten für einen Angreifer nicht erkennbar sind, um Schlussfolgerungen auf die Daten zu unterbinden.

So kann zwischen abhängigkeitserhaltender und abhängigkeitszerstörender Verschlüsselung unterschieden werden.

4.1 Verschlüsselung von Relationen

Im Folgenden werden Verschlüsselungsregeln für verschiedene Fälle beschrieben. Für alle Regeln wird das Beispiel aus Abbildung 4.1 als Grundlage genutzt. Die Verschlüsselungsfunktion wird als f(a, z) dargestellt, wobei a die zu verschlüsselnde Ressource und z eine Zufallszahl ist.

4.1.1 Abhängigkeitserhaltende Verschlüsselung

Ziel der abhängigkeitserhaltenden Verschlüsselung ist es, nur den Dateninhalt, jedoch nicht die Relationen zwischen den Daten zu schützen. Es ist daher die einfachste Form der Verschlüsselung

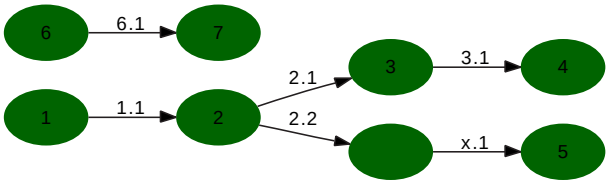


Abbildung 4.1: Ein RDF-Graph.

von RDF, da nur die einzelne Ressource verschlüsselt wird.

In Abbildung 4.2 wird der Knoten 2, welcher das Objekt der Relation 1.1 bzw. das Subjekt der Relation 2.1 ist, abhängigkeitserhaltend verschlüsselt.

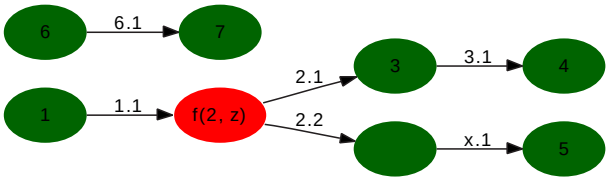


Abbildung 4.2: Verschlüsselung eines Subjektes/Objektes: f(2, z).

Die Abbildung 4.3 zeigt, wie die Prädikate eines Tripels in der abhängigkeitserhaltenden Variante zu verschlüsseln sind. Hierbei wird das zu verschlüsselnde Property 1.1 durch das verschlüsselte Property f(1.1, z) ersetzt.

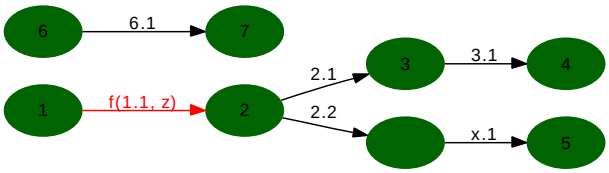


Abbildung 4.3: Verschlüsselung eines Propertys: f(1.1, z).

Problematisch sind Blank Nodes im Graphen, die verschlüsselt werden sollen, da diese ohne Identifier keinen Inhalt zum Verschlüsseln besitzen und bei der Entschlüsselung für Fehlinterpretationen sorgen. Die Verschlüsselung ist erst dann möglich, wenn allen Blank Nodes je ein Identifier zugewiesen wird. In [Car03] tritt das Problem der Blank Nodes bei der Signierung von Graphen auf. Es wird ein entsprechender Algorithmus vorgestellt, der One-step Deterministic Labelling Algorithm, der in den meisten Fällen Abhilfe verschafft. Dieser Algorithmus macht die Blank Nodes auf der Basis der direkt angrenzenden Knoten identifizierbar.

Sind die Blank Nodes ersteinmal mit Identifiern ausgestattet, können sie wie alle anderen Knoten behandelt und verschlüsselt werden.

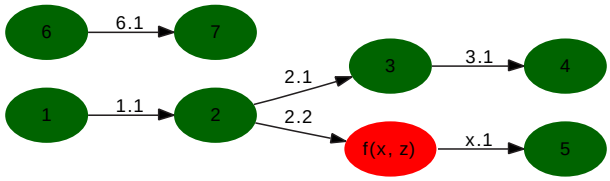


Abbildung 4.4: Verschlüsselung eines Blank Node, ggf. zuvor Zuweisung eines Identifiers x: f(x, z).

Fraglich ist jedoch, ob bei der abhängigkeitserhaltenden Verschlüsselung das Verschlüsseln einer einzelnen Blank Node sinnvoll ist. Dem Blank Node wird nur ein neuer, in diesem Fall verschlüsselter Identifier gegeben. In den allermeisten Fällen wird jedoch aus dem Kontext heraus ersichtlich, dass dieser Knoten ein Blank Node ist. Sinnvoll ist es nur dann, wenn alle umliegenden Knoten und Properties verschlüsselt werden und durch die Angleichung der äußeren Form erreicht werden soll, dass ein Außenstehender nicht erfährt, dass es sich um ein Blank Node handelt.

4.1.2 Abhängigkeitszerstörende Verschlüsselung

Bei der Abhängigkeitszerstörenden Verschlüsselung sollen alle Relationen, die auf die zu verschlüsselnde Ressource Schlussfolgerungen zulassen, zerstört werden. Um dies zu erreichen, müssen einzelne Fälle unterschieden werden.

Es soll ein Knoten verschlüsselt werden, wobei die drei Relationen 1.1, 2.1 und 2.2 im Anschluss nicht mehr in einen Zusammenhang gebracht werden können. Abbildung 4.5 zeigt die einfache Verschlüsselung eines Knotens, indem der betreffende Knoten für jedes Statement einzeln mit einer anderen Zufallszahl verschlüsselt wird. Dies garantiert, dass jeder dieser verschlüsselten Knoten im Anschluss in jedem Statement anders aussieht und somit die Knoten nicht mehr in Verbindung gebracht werden können.

Da den Blank Nodes mithilfe des One-step Deterministic Labelling Algorithm[1] Identifier zugeordnet werden, können die Blank Nodes an dieser Stelle wie normale Knoten behandelt werden.

Bei der Verschlüsselung der Properties sind je nach Einbindung von Subjekt und Objekt in den Gesamtgraphen mehrere Fälle zu unterscheiden.
Der einfachste Fall ist der, dass sowohl das Subjekt, als auch das Objekt mehreren Relationen angehört. Durch die Verschlüsselung von allen drei Ressourcen dieses einen Tripels ist die Abhängigkeit innerhalb des Graphen bezüglich dieses speziellen Prädikats zerstört (siehe Abbildung 4.6).

Ist einer der beiden Knoten des Statements ein Wurzel- (Fall II 4.7(a)) oder ein Blattknoten (Fall III 4.7(b)) innerhalb des Graphen, so reicht es das Prädikat und den Knoten zu verschlüsseln,

[1] [Car03]

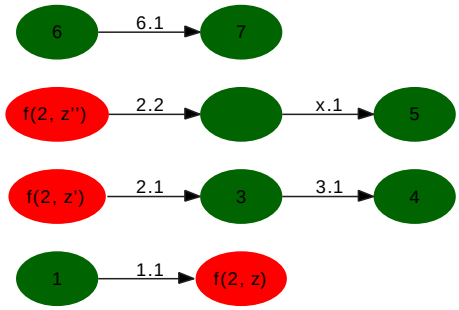


Abbildung 4.5: Verschlüsselung eines Subjektes/Objektes: f(2, z), f(2, z') und f(2, z").

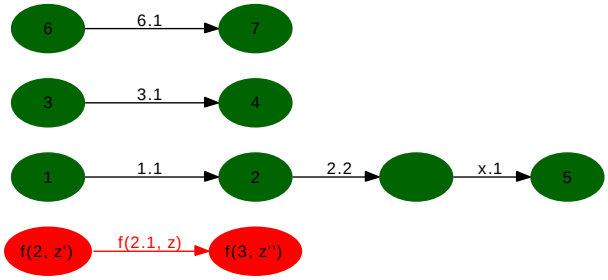


Abbildung 4.6: Verschlüsselung eines Property (Fall I): f(2.1, z), f(2, z'), f(3,z").

der mehreren Relationen im Graph angehört (Abbildung 4.7).

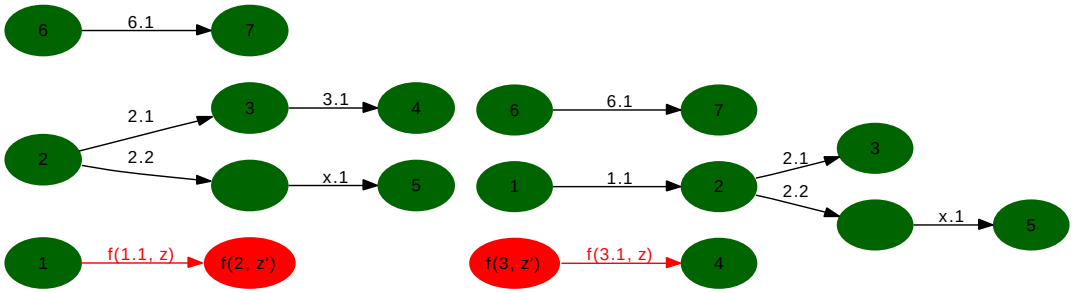
Wenn das Tripel sogar komplett unabhängig vom restlichen Graph ist, muss die Relation in zwei Relationen aufgesplittet werden. Hierzu wird das Tripel dupliziert. Eins der beiden Statements wird dann nach Fall II, das Andere nach Fall III behandelt. Dieser Vorgang wird als Fall IV (Abbildung 4.8) zusammengefasst.

4.2 Auswahlregeln

In einigen Fällen, wie zum Beispiel der Reification, dem Container und der Collection, ist es sinnvoll nicht nur einzelne Elemente zu verschlüsseln, sondern die komplette Struktur. Diese Regeln sind unabhängig von der Art der Verschlüsselung und können somit sowohl abhängigkeitserhaltend, als auch abhängigkeitszerstörend durchgeführt werden.

4.2.1 Reification

Wird bei der Reification wie bisher nur eine einzelne Ressource verschlüsselt, können Probleme auftreten. Wird zum Beispiel nur das Prädikat eines Tripels, über das eine Aussage getroffen wird, nach den zuvor beschriebenen Regeln verschlüsselt, so verraten die durch die Reification



(a) (Fall II): f(1.1, z), f(2, z'). (b) (Fall III): f(3.1, z), f(3, z').

Abbildung 4.7: Verschlüsselung eines Property mit Wurzel- oder Blattknoten.

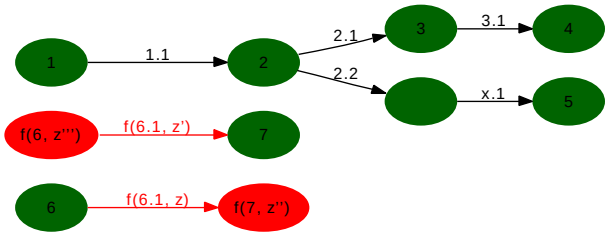


Abbildung 4.8: Verschlüsselung eines Property (Fall IV): f(6.1, z), f(6.1, z'), f(7, z"), f(6, z"').

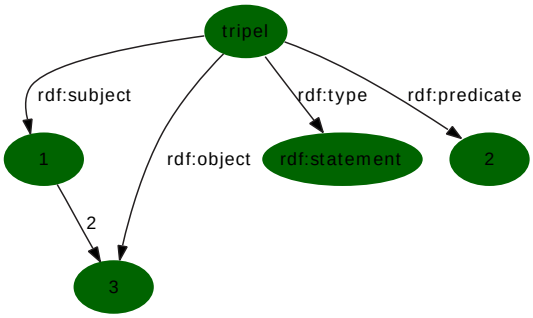
neu geschaffenen Tripel alles über das ursprüngliche Statement (Abbildung 4.9).

Das Tripel als solches ist zwar verschlüsselt, aber die Daten noch immer klar ablesbar.

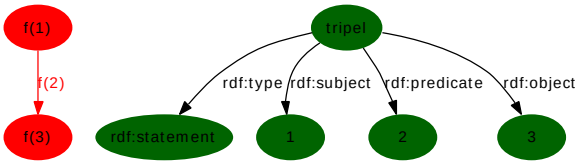
Um zu gewährleisten, dass die Daten vollständig verschlüsselt werden, muss zuvor festgestellt werden, ob das betreffende Tripel Basis für eine Reification ist. Dies ist zu erreichen, indem festgestellt wird, ob in dem Graphen Reifications verwendet wurden und gegebenenfalls, auf welche Tripel sie sich beziehen.

1. Stelle fest, ob Knoten „rdf:Statement“ vorhanden.
2. Verfolge Property „rdf:type“ zurück und lege Liste mit Statements an.
3. Speichere zu jedem Statement die Objekte der Properties „rdf:Subject“, „rdf:Predicate“ und „rdf:Object“.
4. Suche alle Tripel mit den entsprechenden Ressourcen und speichere diese ebenfalls in der Liste in Abhängigkeit zu den Statements.

Wird ein Basistripel verschlüsselt, dann muss die zugehörige Reification mit verschlüsselt werden. Hierzu werden die Ressourcen der vier Grundtripel einer Reification und bei zusätzlich an das reifizierte Statement angehangenen Tripeln zumindest das Subjekt verschlüsselt, so dass keinerlei Informationen mehr über das Basistripel zu erhalten sind.



(a) Unverschlüsselt



(b) Verschlüsselt nach bereits bekannten Regeln

Abbildung 4.9: Beispiel einer Reification

4.2.2 Collection und Container

Auch bei den beiden Strukturen Collection und Container verrät der Kontext der zu verschlüsselnden Ressource viel über die Ressource selber. Bereits das die Elemente miteinander in irgendeiner Beziehung stehen, ist eine Information, die schützenswert ist. Wird jedoch automatisch die komplette Struktur verschlüsselt, wenn ein Element der Struktur ausgewählt wurde, so werden auch Informationen verschlüsselt, die unter Umständen frei zugänglich bleiben sollen.

Es muss also ein Mittelmaß gefunden werden, dass es erlaubt alle Informationen zu verschlüsseln, mit deren Hilfe Rückschlüsse möglich sind, welches aber auch ermöglicht, dass Informationen weiterhin frei zugänglich sind.

Dieses Mittelmaß ist erreicht, wenn alle Relationen zwischen den Elementen der Struktur aufgetrennt wurden. Dies bedeutet, dass bei der Container Struktur die Relation vom Listenkopf (dem Node, der der Struktur den Namen gibt) zum Blank Node, der Blank Node und alle von dem Blank Node abgehenden Relationen verschlüsselt werden müssen. Bei der Collection ist es ähnlich. Hier muss ebenfalls die Relation vom Containerkopf, dem namengebenden Knoten, zum Blank Node, der Blank Node selber und die beiden abgehenden Relationen verschlüsselt werden. Weiterhin ist es bei der Collection wichtig, dass jeder weitere Blank Node, und seine abgehenden Relationen auch verschlüsselt werden.

Diese Prozedur kann automatisiert werden, indem bei Auswahl eines Elementes der Struktur der komplette Kern der Struktur verschlüsselt wird. Als Kern der Struktur wird die komplette Struktur ohne den Listenkopf und die Elemente bezeichnet. Es eignet sich jedoch nicht jedes Element für diese Automatisierung, da es zum Beispiel möglich ist, dass nur ein einzelnes Datum

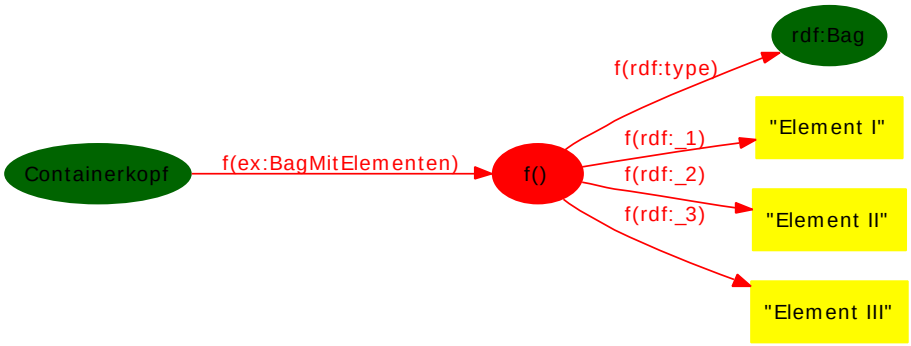


Abbildung 4.10: Abhängigkeitserhaltende Verschlüsselung des Propertys ex:BagMitElementen: f(ex:BagMitElementen), f(), f(rdf:type), f(rdf:_1), f(rdf:_2), f(rdf:_3).

aus der Struktur gelöst werden soll. Das einzige Element, dass sich tatsächlich eignet ist die Relation zwischen dem Listenkopf und dem Blank Node, da an dieser Stelle der Nutzer mit hoher Wahrscheinlichkeit genau das erreichen will (Abbildung 4.10). Bei der Collection ist zusätzlich zu überlegen, was passieren soll, wenn eines der rdf:rest-Propertys zwischen zwei Blank Nodes angewählt wird. Hier gibt es zwei Alternativen, was verschlüsselt wird:

- von der ausgewählten Relation ab alle Kernelemente bis zum Listenende
- genau die ausgewählte Relation

Die Automatisierung sollte nicht auf unterster Ebene passieren, sondern auf die Grundverschlüsselung aufgesetzt werden. Auf diese Art kann der Nutzer selbst entscheiden, ob er die Automatisierung verwenden will oder ob er die Verschlüsselung lieber manuell steuert.

4.3 Kompakte Verschlüsselung größerer zusammenhängender Teilgraphen

Um zusammenhängende Teilgraphen wie zum Beispiel Container verschlüsseln zu können, ist es von Vorteil, wenn diese im Anschluss in einer kompakteren Form, zum Beispiel einem einzigen Knoten vorliegen. Hierdurch würde nicht nur der Dateninhalt, sondern auch sämtliche Relationen sowie die Anzahl der Knoten und Relation innerhalb des Teilgraphen geschützt. Bei der Durchführung ist zu beachten, dass RDF-Graphen in den meisten Fällen keine Bäume sind und daher die zu verschlüsselnden Teilgraphen auch keine Äste. Dies bedeutet, dass der zu verschlüsselnde Teilgraph mehrere Relationen zu seiner Umgebung besitzen kann. Um die Verschlüsselung zu bewerkstelligen, gibt es daher verschiedene Ansätze.

4.3.1 Verschlüsselung mit anschließender Einbindung in den Graphen

Bei diesem Ansatz wird der komplette Teilgraph in einen Knoten verschlüsselt, der im Anschluss anstelle des Teilgraphens in den Gesamtgraph eingebunden wird.

Problematisch ist, dass die Relationen, die zuvor mit den einzelnen Elementen des Teilgraphens verknüpft waren jetzt nur noch auf eine einzige Ressource verweisen, den verschlüsselten Teilgraphen. Es entsteht also ein Datenverlust bei diesem Schritt. Dies ist bei der Verschlüsselung noch kein Problem, jedoch bei der Entschlüsselung, da die Relationen nicht mehr den verschiedenen Knoten zugewiesen werden können.

Daraus folgt, dass eben diese Tripel, bei denen der Datenverlust auftritt, dem verschlüsselten Teilgraphen als verschlüsselte Kopie hinzugefügt werden müssen, um bei der Entschlüsselung die Relationen wieder eindeutig zuweisen zu können.

4.3.2 Verschlüsselung mit Heraustrennung aus dem Hauptgraphen

Ein weiterer Ansatz ist, den verschlüsselten Teilgraphen nicht wieder in den Hauptgraphen einzubinden, sondern mit Hilfe eines neu erschaffenen Tripels an anderer Stelle anzufügen. Hierbei ist es egal, ob der verschlüsselte Teilgraph in einer neuen Datei gepeichert wird und das neu erschaffene Tripel auf diese Datei verweist oder ob der verschlüsselte Teilgraph direkt in der RDF-Datei eingefügt wird. Bei der Speicherung in einer Fremddatei ergeben sich geringe Probleme, da bei einer Speicherung im Internet zur Entschlüsselung eine Onlineverbindung bestehen muss und bei einer lokalen Speicherung auf der Festplatte die Datei an genau der Stelle liegen muss, auf die der URL des neuen Tripels verweist.

Die Relationen, die zuvor mit den Elementen des Teilgraphen verknüpft waren, sind nun eines Knoten beraubt und somit unvollständig und in RDF nicht verwendbar. Um diesem Problem zuvor zu kommen, werden diese Elemente vor dem Heraustrennen des Graphens dupliziert und verschlüsselt. So sind diese Relationen wieder vollständig und in RDF gültig. Durch das Duplikat innerhalb des Teilgraphen können die Knoten nach der Entschlüsselung wieder eindeutig einander zugeordnet werden, so dass der Graph in seiner ursprünglichen Form wieder hergestellt ist.

4.4 Einbindung der Verschlüsselung in RDF

Für die im Folgenden vorgestellten Modelle wird ein vereinfachter Beispielgraph (Abbildung 4.11) verwendet. Die oben beschriebenen Verschlüsselungsregeln sind alle auf diese Modelle anwendbar.

Abbildung 4.11: Grundgraph für die folgenden Beispiele.

4.4.1 Verschlüsselung unter Verwendung von URIs

Die grundlegende Idee bei dieser Variante ist, dass die verschlüsselten Zeichenketten wieder als URIs in den Graphen eingebunden werden. URIs sind so definiert, dass sie aus einem Protokoll gefolgt von einem Doppelpunkt und einer Zeichenkette bestehen. Dies lässt nun die Möglichkeit offen, vor dem Doppelpunkt als Protokoll alle Eigenschaften der Verschlüsselung, wie zum Beispiel Algorithmus und Schlüssellänge, anzugeben und hinter dem Doppelpunkt die Zeichenkette mit den verschlüsselten Daten anzuhängen. Dies ist in der Abbildung 4.12 zu sehen.

Abbildung 4.12: Verschlüsselung: Algorithmus:f(2, z).

Vorteil dieser Variante ist, dass alle Ressourcen, Knoten und Properties, gleichermaßen zu handhaben sind.

4.4.2 Verschlüsselung in Literalen

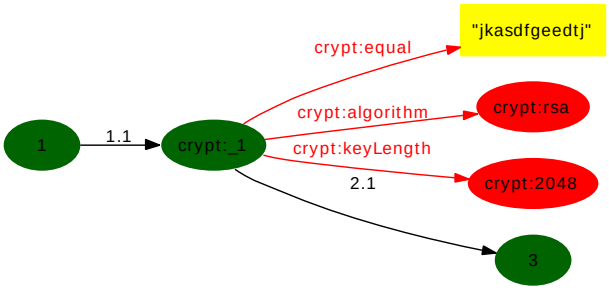
Bei dieser Variante werden die verschlüsselten Zeichenketten als Literale in den Graph eingebunden. Um dem Literal Metainformationen (Verschlüsselungsalgorithmus, Schlüssellänge) mitgeben zu können, muss es als Subjekt verwendbar sein. Um dies zu erreichen, bekommt jedes Literal einen Unique Identifier (UID) zugewiesen. Dieser UID können durch weitere Relationen Informationen zur Verschlüsselung zugefügt werden.

Da bei dieser Variante neue Properties verwendet werden, die bei jeder Verschlüsselung gleich sind, ist es angebracht dafür ein eigenes Vokabular (zum Beispiel „crypto“) aufzusetzen, welches die gängigsten Verschlüsselungseigenschaften und die Vergabe von UIDs unterstützt.

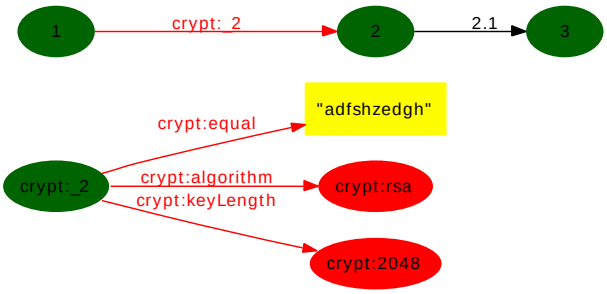
Zum Beispiel könnte dieses Vokabular wie in Abbildung 4.13 dargestellt aussehen.

Doppelte Verschlüsselung

Soll eine UID ein weiteres Mal verschlüsselt werden, ergeben sich Schwierigkeiten. Die UID wird durch eine neue UID ersetzt und erhält eine Relation mit dem Property „crypto:identic“, die das Literal mit der verschlüsselten UID als Objekt besitzt. Die bereits vorhandenen Relationen werden an die neue UID angehängt, so dass im Anschluss zwei Relationen mit dem selben Property „crypto:identic“ vorhanden sind. Zum Einen ergeben sich daraus Probleme bei der Entschlüsselung, da die Literale in einer bestimmten Reihenfolge zu entschlüsseln sind, diese jedoch nicht ersichtlich ist. Zum Anderen bringt die Verschlüsselung einer UID keine Erhöhung der Sicherheit, da der ursprünglich verschlüsselte Knoten nur einfach verschlüsselt als Literal anhängt. Um diese beiden Probleme zu lösen, bietet es sich an, die komplette Verschlüsselungsstruktur, also die UID mit anhängendem Literal und Zusatzinformationen, als Teilgraph zu verschlüsseln.



(a) Knoten: f(2, z)



(b) Property: f(1.1, z)

Abbildung 4.13: Verschlüsselung als Literal, weitere Details durch neue Relationen.

Vorteil dieses Modells ist es, dass zu jeder verschlüsselten Zeichenkette beliebig viele Angaben gemacht werden können und diese nochmals verschlüsselbar sind.

Kapitel 5

Bewertung der Datensicherheitsmodelle und Auswahl

5.1 Bewertung der Modellvarianten

In diesem Kapitel werden die verschiedenen Modellvarianten unter den Aspekten der Vertraulichkeit, Integrität und Verfügbarkeit bewertet, ohne jedoch dem Anspruch einer vollen Sicherheitsanalyse nachzukommen.

5.1.1 Abhängigkeitserhaltende versus abhängigkeitszerstörende Verschlüsselung

Abhängigkeitserhaltende Verschlüsselung

Durch die abhängigkeitserhaltende Verschlüsselung kann die Vertraulichkeit einer RDF-Ressource bis zu einem gewissen Maß erreicht werden. Ein potentieller Angreifer kann, bei geringer Anzahl an verschlüsselten Ressourcen innerhalb eines Graphens, durch die der verschlüsselten Ressource anhängenden Relationen mitunter genug über die Ressource erfahren, dass er sie entschlüsseln kann. Es gilt also bei diesem Modell je höher die Anzahl der verschlüsselten Ressourcen, desto höher der Grad der Vertraulichkeit.

Der Grad der Integrität ist bei diesem Verschlüsselungsmodell kaum höher als bei einem unverschlüsselten Graphen. Einem Angreifer wäre es ein Leichtes Teile des Graphen zu entfernen oder zu verändern. Dem geschädigtem Nutzer würde es lediglich auffallen, wenn der Angreifer aus der Verschlüsselungsstruktur beim Verschlüsseln in Literalen die notwendigen Metainformationen teilweise löschen oder verändern würde. Das unbefugte und unbemerkte Entfernen von Informationen kommt somit einem Angriff auf die Verfügbarkeit gleich. Die Verschlüsselung

bietet jedoch keinerlei Varianten um den Schutz der Verfügbarkeit zu gewährleisten, da dieser größtenteils von der Speicherung der Daten abhängig ist.

Abhängigkeitszerstörende Verschlüsselung

Generell gilt für die abhängigkeitszerstörende Verschlüsselung bezüglich Integrität und Verfügbarkeit das Selbe wie für die abhängigkeitserhaltende Verschlüsselung. Unterschiede gibt es lediglich bei der Vertraulichkeit. Die abhängigkeitszerstörende Verschlüsselung bringt durch ein wenig mehr Aufwand ein hohes Maß an zusätzlicher Vertraulichkeit. Ein möglicher Angreifer muss nun nicht mehr nur den Inhalt eines einzelnen Literals in Erfahrung bringen, um die zugehörigen Relationen zu finden, sondern sämtliche verschlüsselten Literale knacken, um ein volles Bild vom Graphen zu bekommen. Ohne die zugehörigen Relationen erschweren sich auch Rückschlüsse auf benachbarte verschlüsselte Ressourcen.

Negativ zu bewerten ist jedoch die Tendenz dieses Modells das Datenvolumen sehr schnell zu erhöhen.

5.1.2 Mehr Vertraulichkeit durch Auswahlregeln

Die Auswahlregeln bewirken genau das, was der abhängigkeitserhaltende Verschlüsselung als großes Manko angerechnet wird. Die Regeln sollen aus der Struktur des RDF-Graphen heraus sinnvoll die Liste der zu verschlüsselnden Ressourcen erweitern und damit einem Angreifer die Entschlüsselung einer Ressource unter Zuhilfenahme der angrenzenden Relationen erschweren oder gar unmöglich machen.

Auch bei der abhängigkeitszerstörenden Verschlüsselung sind diese Regeln für den Anwender sehr hilfreich, da ihm auf diese Art bei der Auswahl der zu verschlüsselnden Ressourcen sinnvolle Vorschläge gemacht werden können.

5.1.3 Reduzierung des Datenvolumens durch Verschlüsselung von Teilgraphen

Die Verschlüsselung von Teilgraphen bietet den höchsten Grad an Vertraulichkeit bei der Verschlüsselung von RDF. Da hier ganze Teile eines Graphens kompakt verschlüsselt werden ist es einem Angreifer nicht möglich die Relationen zwischen einzelnen verschlüsselten Ressourcen zu betrachten und auf diese Art Schlüsse zu ziehen.

Bezüglich der Integrität verhält sich diese Variante jedoch wie eine einzeln verschlüsselte Ressource. Wenn ein Angreifer die komplette Verschlüsselungsstruktur (Verschlüsselung in Literalen) löscht, so bemerkt der Nutzer dieses Fehlen nicht. In diesem Fall wäre somit ein kompletter Teil des Originalgraphen unbemerkt abhanden gekommen. Lediglich wenn besagter Angreifer nur Teile der Struktur löscht oder die Struktur ändert kann es vom Nutzer bemerkt werden.

Weiterhin hat diese Verschlüsselungsvariante den Vorteil, dass viele Ressourcen gemeinsam in einem Knoten verschlüsselt werden können und nicht wie bei der abhängigkeitszerstörenden

Verschlüsselung jede Ressource mehrfach neu verschlüsselt werden muss. Dadurch reduziert sich das Datenvolumen innerhalb des Graphen.

5.1.4 Verschlüsselung in URIs versus Verschlüsselung in Literalen

Die Verschlüsselung in Literalen birgt gegenüber der Verschlüsselung in URIs einige entscheidende Vorteile. So können bei dieser Methode Metadaten über den Verschlüsselungsvorgang in geordneter Form der jeweiligen Ressource mitgegeben werden. Weiterhin ist die Erkennbarkeit der verschlüsselten Ressourcen innerhalb des Gesamtgraphen in höherem Maße gegeben, da jede verschlüsselte Ressource durch eine eigene Ontologie („CryptOntology“) explizit markiert wird.

5.2 Auswahl einer Modellvariante

Aufgrund der zuvor aufgeführten Bewertungen der verschiedenen Varianten wurde die abhängigkeitszerstörende Verschlüsselung in Literalen mit Auswahlregeln und Verschlüsselung von Teilgraphen implementiert. In dieser Kombination ist es dem Anwender möglich das höhere Datenvolumen, welches durch die abhängigkeitszerstörende Verschlüsselung entsteht, mithilfe der Teilgraphenverschlüsselung auszugleichen. Weiterhin wird dem Anwender so der größtmögliche Grad an Vertraulichkeit geboten.

Kapitel 6

Umsetzung des Modells - RDFcrypto

RDFcrypto ist eine erste, als Beispiel gedachte Umsetzung des Modells zur Verschlüsselung von RDF. In RDFcrypto wurden zur Verschlüsselung der symmetrische Verschlüsselungsalgorithmus DES und der asymmetrische Algorithmus RSA verwendet.

Das Java-Programm liest eine bestehende RDF-Datei, die im Format RDF/XML oder N3 vorliegen muss, ein und wandelt sie intern in ein Model. Das Model wird dann in der GUI in Form von Listen ausgegeben.

Aus den Listen kann der Nutzer nun die zu verschlüsselnden Ressourcen auswählen. RDFcrypto prüft im Anschluss mithilfe der durch den Nutzer ausgewählten Regeln, ob es ratsam ist zusätzlich zu den selektierten Ressourcen weitere zu verschlüsseln und schlägt dieses dem Nutzer in einer Preview vor. Nach der Verschlüsselung wird das veränderte Model wieder in der GUI ausgegeben. Der Nutzer kann zu jedem Zeitpunkt das Model in den Formaten RDF/XML, N3 oder DOT[1] in eine Datei ausgeben lassen.

Zur Entschlüsselung wird das Model im Ganzen entschlüsselt.

6.1 Die Benutzeroberfläche

Die Benutzeroberfläche ist einfach gehalten. Sie ist unterteilt in einen Bereich für die Verschlüsselung (Karteireiter mit dem Namen „Encryption“) und einen Bereich für die Entschlüsselung (Karteireiter mit dem Namen „Decryption“).

6.1.1 Encryption

Der Karteireiter der Verschlüsselung besteht aus einem Bereich zur Anzeige (links) und einem Bereich für die Einstellungen und zur Ausführung (rechts).

[1] DOT ist ein Datenformat, das unter anderem von GraphvizA verwendet wird und gerichtete Graphen beschreibt

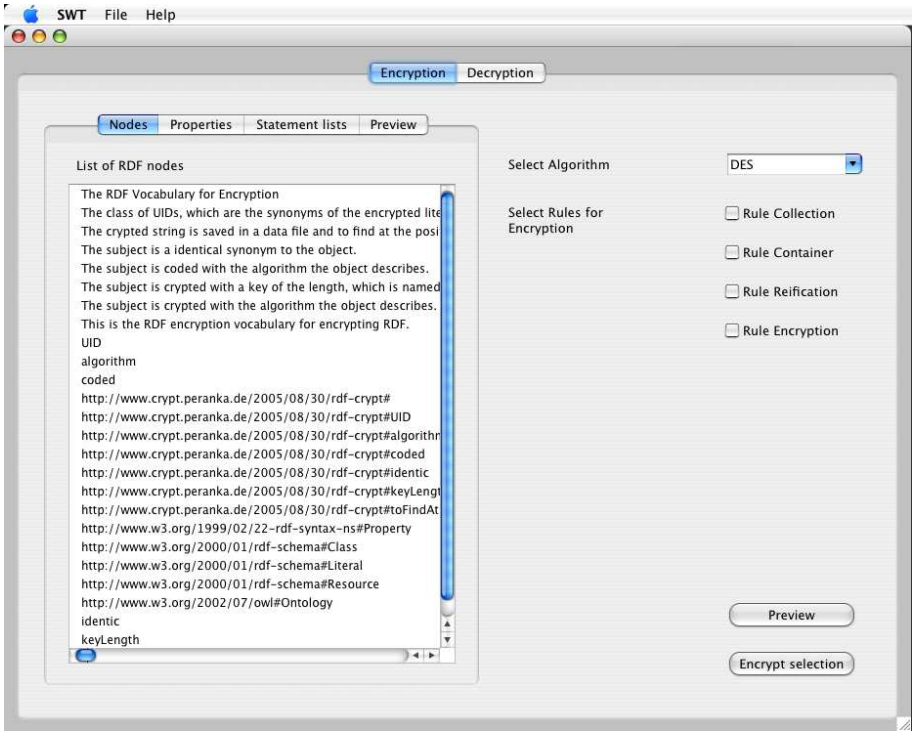


Abbildung 6.1: RDFcrypto: Ansicht des Verschlüsselungsbereichs

Als Anzeigen stehen vier Listen zur Verfügung: RDF Nodes, Propertys in Form von Trippeln, Statements und die Preview. Die Liste der RDF Nodes listet alle Subjekte und Objekte innerhalb des Graphen auf. Die Liste der Propertys zeigt alle Prädikate aus dem Graphen, welche zusammen mit dem kompletten Statement ausgegeben werden, um den Kontext für den Nutzer einsehbar zu machen. Die Liste der Statements wiederum birgt eine komplette Aufzählung der Statements des Graphen, jedoch werden diese später als Teilgraph verschlüsselt. Aus diesen drei Listen kann der Nutzer eine Auswahl der zu verschlüsselnden Komponenten treffen. In der Previewliste wird dann die zuvor getroffene Auswahl nach Durchlaufen des Regelwerks angezeigt.

Zu den Einstellungen gehören die Auswahl des Verschlüsselungsalgorithmuses und die gewünschten Auswahlregeln.

Zur Ausführung stehen zwei verschiedene Methoden zur Verfügung:

- die Preview und
- die Verschlüsselung.

Nach Auswahl seiner Verschlüsselungselemente kann sich der Nutzer durch das Betätigen der Schaltfläche „Preview“ anzeigen lassen, welche Elemente nach Durchlaufen des Regelwerks

verschlüsselt werden, ohne das die Verschlüsselung statt findet. Durch betätigen der Schaltfläche „Encrypt selection“ wird die Auswahl der Verschlüsselungselemente unter Zuhilfenahme der ausgewählten Regeln verschlüsselt. Im Anschluss zeigen die Listen RDF Nodes, Propertys und Statements den Graphen nach dem Verschlüsselungsvorgang an.

6.1.2 Decryption

Die Entschlüsselung bietet weniger Funktionen. Dem Nutzer werden in zwei Listen alle Nodes und alle Statements des Graphen angezeigt. Er hat die Möglichkeit den Entschlüsselungsalgorithmus zu wählen. Die Entschlüsselung des Modells geschieht im Ganzen nach Betätigung des Buttons „Decrypt Model“. Im Anschluss zeigen auch hier die beiden Listen den entschlüsselten Graphen an.

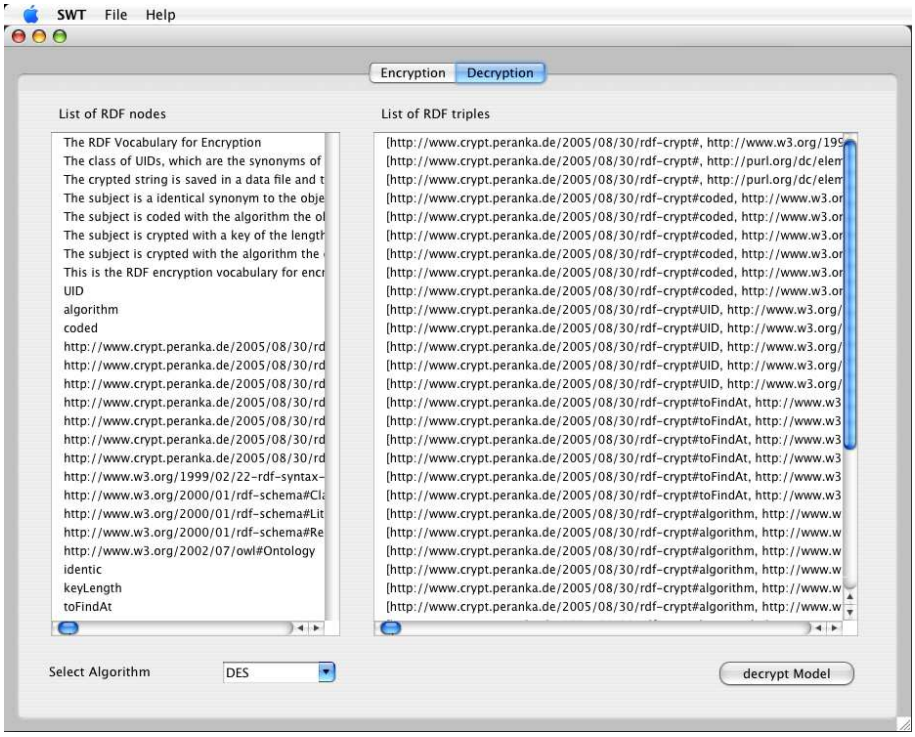


Abbildung 6.2: RDFcrypto: Ansicht des Entschlüsselungsbereichs

6.2 Der Verschlüsselungsprozess

Der Verschlüsselungsprozess wird bewerkstelligt durch:

- die Klasse Encryptor,

- die Klasse Decryptor und
- dem Interface ICipher.

6.2.1 Encryptor

Dem Encryptor wird die zu verschlüsselnde Ressource übergeben. Die Klasse ermittelt den Typ der Ressource und die für die abhängigkeitszerstörende Verschlüsselung notwendigen Duplikate. Im Anschluss reicht der Encryptor für jede zu verschlüsselnde Ressource den Typ und den Inhalt der Ressource an den ICipher weiter und erhält eine verschlüsselte Zeichenkette zurück. Dieser wird von der Klasse in eine neue Ressource vom Typ Literal eingebunden. Weiterhin wird eine Ressource mit einer gehashten UniqueID erstellt, welche als Subjekt in ein neu erstelltes Trippel eingeht. Das Literal wird als Objekt mit dem Prädikat „crypto:identic“ an das Subjekt angehangen. Darüber hinaus erstellt der Encryptor alle weiteren Trippel für die Verschlüsselungsstruktur. Zudem werden in allen Trippeln die zu verschlüsselnden Ressourcen durch die Ressource mit der UID ersetzt.

6.2.2 Decryptor

Der Decryptor ermittelt alle verschlüsselten Ressourcen eines Modells und lässt sie vom ICipher entschlüsseln. Aus dem Klartext entnimmt er den Typ der Ressource. Der Decryptor legt im Anschluss eine neue Ressource dieses Typs an, falls diese noch nicht existiert. Als Inhalt wird der Ressource der Klartext (ohne die Typangabe) übergeben. Alle Vorkommen der UID in Trippeln werden ersetzt durch die neue Klartextressource.

6.2.3 ICipher

ICipher ist ein Interface, dass die Methoden zur Ver- und Entschlüsselung vorgibt. Implementierungen des ICipher sind die beiden Klassen TripleDESCipher und RSACipher. In diesen beiden Klassen wird die jeweilige Ver- und Entschlüsselung implementiert.

Beim Trippel DES geschieht die Verschlüsselung, indem in das zu verschlüsselnde Bytearray zuerst Zufallszahlen an jeder zweiten Stelle eingebettet werden. Im Anschluss wird der mit Zufall aufgefüllte Klartext mithilfe von java.security und BouncyCastle[2] mit einem dem ICipher übergebenen Schlüssel verschlüsselt. Das daraus resultierende Bytearray wird anschließend Base64[3] kodiert und als String zurückgegeben.

Die Entschlüsselung läuft umgekehrt dazu ab. Der Cipher erhält einen String, den er Base64 dekodiert und anschließend entschlüsselt. Als Letztes wird der beim Verschlüsseln hinzugefügte Zufall entfernt.

[2] Siehe Appendix A.

[3] Eine kompakte Erläuterung des Kodierungsalgorithmuses Base64 ist unter http://de.wikipedia.org/wiki/Base64 zu finden.

Die Verschlüsselung von RSA läuft generell genauso ab, jedoch muss der Klartext, nachdem der Zufall eingebunden wurde, in Blöcke der Länge *(Schlüssellänge in Byte - 1)* zerlegt werden, bevor er verschlüsselt werden kann. Die einzelnen Bytearrays der Länge *(Schlüssellänge in Byte)* werden wieder zusammengefügt und danach Base64 kodiert.

Bei der Entschlüsselung müssen die Selben Schritte in umgekehrter Reihenfolge vorgenommen werden. Nach der Base64 Dekodierung muss das Bytearray in Blöcke der Länge *(Schlüssellänge in Byte)* zerlegt und diese dann entschlüsselt werden. Die daraus resultierenden Arrays der Länge *(Schlüssellänge in Byte - 1)* werden wieder zusammengefügt. Zum Abschluss wird der eingefügte Zufall entfernt. Der Klartext wird dann zurückgegeben.

6.2.4 Problembereiche

An einigen Stellen haben sich bei der Implementierung sicherheitskritische Probleme gezeigt, die zuvor einfach übersehen wurden. In allen Fällen konnten diese Probleme durch die nachfolgenden Methoden leicht gelöst werden.

Verwendung von Zufallszahlen bei der Verschlüsselung - der RandomMaker

Üblicherweise wird ein Klartext mit einem Schlüssel immer auf genau einen Schlüsseltext abgebildet. Da bei der abhängigkeitszerstörenden Verschlüsselung eine Klartext-Ressource auf mehrere Schlüsseltexte abgebildet werden soll, ist es notwendig Zufallszahlen in den Klartext einzubinden.

DES - Da der DES eine Blockchiffre ist müssen die Zufallszahlen, um eine Wirkung zu erzielen, in jeden Block eingespeist werden. Daher wird in den Klartext an jeder zweiten Stelle, mit der ersten Stelle in der neuen Zeichenkette angefangen, eine Zufallszahl eingeschoben, so dass sich die Länge des Klartextes verdoppelt.

RSA - Bei RSA ist die maximale Klartextlänge abhängig von der verwendeten Schlüssellänge. Alle längeren Klartexte müssen demnach auch in Blöcke zerlegt und in diesen verschlüsselt werden. Daher kann hier die selbe Methode zum Einbinden des Zufalls benutzt werden wie beim DES.

Abhängigkeiten durch Schlüsseltextlängen erkennbar

Sowohl DES als auch RSA füllen beim Verschlüsseln den Klartext nur auf Blocklänge auf. Um nun nicht Abhängigkeiten zwischen den Ressourcen aufgrund der verschiedenen Schlüsseltextlängen ausmachen zu können, ist es notwendig die Klartexte zuvor auf eine definierte Länge aufzufüllen. Die Länge ist als Primzahl zu wählen, da bei einer durch die Blocklänge des DES teilbaren Länge die letzten Stellen des Schlüsseltextes identisch sind, da der letzte Block Informationen zum Padding beinhaltet. Durch die Primzahl sind versehentliche Abhängigkeiten zur

Blocklänge ausgeschlossen. Bei der Umsetzung in RDFcrypto wurde eine Auffülllänge von 997 Byte gewählt.

Rückschlüsse durch die UID

Unique IDs sind so angelegt, dass sie mit sehr hoher Wahrscheinlichkeit einmalig auf der Welt vergeben wurden. Um dies zu erreichen verwenden die meisten Algorithmen Daten, die der Rechner ihnen liefern kann, wie zum Beispiel die ProzessorID, die Uhrzeit und das Datum. Dadurch ergeben sich jedoch zwei Probleme. Erstens ist es möglich, dass durch die UID der Rechner, mit dem sie erstellt wurden, ermittelt wird und somit unter Umständen auch derjenige, der verschlüsselt hat. Zum Zweiten erstellt der Algorithmus bei zu schnell aufeinander folgenden Anfragen sehr ähnliche UIDs, aus denen die Reihenfolge der Verschlüsselung und somit mögliche Abhängigkeiten der Ressourcen zueinander zu erkennen sind.

Um diese beiden Probleme zu beseitigen müssen die UIDs mit einer kollisionsresistenten Hashfunktion bearbeitet werden. Für die Umsetzung wurde die Hashfunktion Tiger[4] verwendet. Ein Angreifer müsste somit erst die Hashfunktion Tiger brechen, um dann die UID zu erhalten. Nachteilig ist jedoch, dass durch das Hashen sich die Wahrscheinlichkeit einer Wiederholung erhöht, da auch kollisionsresistente Hashfunktionen nur mit einer sehr hohen Wahrscheinlichkeit nicht kollidieren. Somit erhöht sich die Wahrscheinlichkeit einer Kollision nach folgender Formel: $Kollisionswahrscheinlichkeit UID - (1 - Kollisionswahrscheinlichkeit UID) * Kollisionswahrscheinlichkeit Hash$ Diese neue Wahrscheinlichkeit ist jedoch noch immer klein genug um die gehashte UID als eindeutig anzunehmen.

6.3 Umsetzung der Auswahlregeln

Die Auswahlregeln sind in RDFcrypto als Aufsatz auf die eigentliche Verschlüsselung und somit als Erweiterung der Basisverschlüsselung realisiert. Daher sind die Regeln komplett austauschbar. Als Richtlinie für die Implementierung der Regeln dient das Interface IRule, in dem die verschiedenen Methoden aufgelistet sind. Eine abstrakte Implementierung des Interfaces stellt AbstractRule dar. Diese Klasse implementiert eine Basisfunktionalität. Die einzelnen Auswahlregeln sind von AbstractRule abgeleitet, so dass nur die Methoden, die für die Regel wirklich gebraucht werden überschrieben werden müssen.

Das Interface IRule ist so angelegt, dass ein einzelnes, zu verschlüsselndes Element übergeben wird und eine oder mehrere Listen von Elementen zurückgegeben werden. Da die Ergebnisse, die eine Regel zurückgibt, Elemente enthalten können, die wiederum Grundlage für eine Auswahlregel sind, muss für die Regeln eine Ansteuerung existieren, die die Abarbeitung organisiert. Dies geschieht durch den Controller.

Der Controller führt intern alle zu verschlüsselnden Elemente und die Rules. Nacheinander werden alle Regeln auf alle Elemente angewandt. Wichtig ist, dass auch auf die Ergebnisse der

[4] [AB]

Regeln die Regeln erneut angewandt werden.

6.4 Verwendete Hilfsmittel

Für die Umsetzung der ausgewählten Modellvariante wurde Java J2SE 1.5.0 verwedendet. Weiterhin wurde das Framework Jena in der Version 2.2 für die RDF-spezifischen Belange und das Framework BouncyCastle in der Version 1.30 für die Verschlüsselungselemente benutzt. Die GUI wurde mit dem Plugin Jigloo in der Version 3.52 erstellt. In Appendix A sind die entsprechenden Links und Lizenzen zu finden.

Kurz vor Abschluss des Projekts wurde ein Update des Frameworks Jena auf die Version 2.3 notwendig, woraus sich neue Probleme ergaben. Rückschauend ist zu sagen, dass Jena für dieses Projekt zu umfangreich ist und es vermutlich die bessere Variante gewesen wäre ein eigenes kleines Framework zu schreiben, welches die Anforderungen besser und schneller verarbeiten hätte können.

6.5 Systemanforderungen

Für RDFcrypto werden folgende Bibliotheken und Frameworks verwendet:

- J2SE 1.5.0
- Jena - RDF Bibliothek
  - ANTLR - Parser Generator
  - Jakarta commons-logging - Protokollierungsbibliothek
  - concurrent - Unterstützung für parallele Java Programmierung
  - ICU4J - Unicode Unterstützung
  - jakarta-oro - Perl5 Parser
  - jUnit - Test Framework
  - log4j - Protokollierungsbibliothek
  - xercesImpl - Java Parser
  - xml-apis - XML-Schnittstelle für Datenbanken
- BouncyCastle - Kryptografie Provider
- SWT - Standard Widget Toolkit, Oberflächengestaltung

6.6 Ausblick für die Zukunft

Das Programm RDFcrypto stellt einen Prototyp für die Verschlüsselung von RDF dar. Im Laufe der Implementierung hat sich gezeigt, dass an einigen Stellen Mängel in der Wahl der Frameworks und im Aufbau der Struktur bestehen. Für eine Neuimplementierung gibt es daher folgende Punkte zu empfehlen.

Die Basis

Als Basis für die RDF-spezifischen Datentypen sollte auf Jena verzichtet werden, da dies zurzeit noch aus nachfolgenden Gründen ein Problem darstellt.

Nach dem Löschen von Statements aus dem Modell sind die ungenutzten Nodes noch immer Bestandteil des Modells und somit potentiell auch noch auslesbar.

Umspeichern des Modells

Es hat sich im Laufe der Implementierung herausgestellt, dass es von Vorteil gewesen wäre, nicht das komplette Modell zu nehmen und Teile daraus einzeln abzuändern, sondern ein leeres Modell zu nehmen und in dieses neue Modell die Verschlüsselungen einzutragen und den Rest des Originalmodells zu kopieren.

Auf diese Weise würde die Verschlüsselung geordneter und fehlerfreier ablaufen, da sie atomar wäre. Zum Beispiel würden keine Probleme mehr mit der Verschlüsselung von Propertys auftreten. Zur Zeit ist es so, dass ein Node, der in mehreren Statements integriert ist bei der Verschlüsselung aller Statements (Propertys) hinterher einmal unverschlüsselt im Graphen auftritt, da das zuletzt verschlüsselte Property nach einem anderen Fall verschlüsselt wird als die Vorhergehenden. Das Problem betrifft zur Zeit in erster Linie Blank Nodes in Collections und Containern.

Integrität

RDFcrypto bietet bislang noch keine Möglichkeiten die Integrität der Daten zu schützen. Daher wäre bei einer Neuimplementierung eine Funktion zur Signierung des Graphen wünschenswert.

Kapitel 7

Zusammenfassung

Bei der Untersuchung der Möglichkeiten von RDF hat sich gezeigt, dass es drei grundlegende Ansätze gibt, um Vertraulichkeit zu erreichen:

- Verschlüsselung von Teilgraphen,
- die anhängigkeitserhaltende Verschlüsselung und
- die abhängigkeitszerstörende Verschlüsselung.

Es hat sich gezeigt, dass die abhängigkeitszerstörende Verschlüsselung die Vertraulichkeit in höherem Maße schützt als die anhängigkeitserhaltende Verschlüsselung. Weiterhin konnte gezeigt werden, dass die Verschlüsselung von Teilgraphen sowohl mit der anhängigkeitserhaltende Verschlüsselung als auch mit der abhängigkeitszerstörende Verschlüsselung kombinierbar ist.

Es konnte auch gezeigt werden, dass zum maximalen Schutz der Vertraulichkeit Auswahlregeln für die verschiedenen Konstrukte wie Collection und Container notwendig sind.

Auch für das Erreichen von Integrität wurde mit der Ausarbeitung „Signing RDF Graphs“ von Jeremy J. Carroll[1] eine Möglichkeit gefunden.

Für das Einbinden der verschlüsselten Daten und ihren Metadaten wurden weiterhin zwei mögliche Varianten gezeigt:

- Verschlüsselung in URIs und
- Verschlüsselung in Literalen.

Es wurde herausgestellt, dass die Verschlüsselung in Literalen das größere Potential bezüglich der Speicherung von Metadaten bietet.

Abschließend wurde ein Prototyp für die abhängigkeitszerstörende Verschlüsselung in Literalen in RDF implementiert.

[1] [Car03]

Anhang A

Verwendete Software

Graphviz

Link:
http://www.gaphviz.org

License:
http://www.graphviz.org/License.php

Eclipse SDK 3.1.1

Link:
http://www.eclipse.org/

License:
http://www.eclipse.org/org/documents/epl-v10.html

Java J2SE 1.5.0

Link:
http://java.sun.com/

License:
http://java.sun.com/j2se/1.5.0/docs/relnotes/license.html

Jena 2.2 und 2.3

Link:
http://jena.sourceforge.net/

License:

Weitere Angaben:

Jena includes software developed by the Apache Software Foundation (http://www.apache.org/).

Jena is built on top of other sub-systems which we gratefully acknowledge: details of these systems and their version numbers.

* Xerces

* JUnit

* Jakarta ORO

* ICU4J

* util.concurrent from Doug Lea

BouncyCastle 1.30

Link:

http://bouncycastle.org/ License:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the „Software“), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED „AS IS“, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Jigloo 3.52

Link:

http://cloudgarden.com/jigloo/

License Terms:

By downloading, installing and using Jigloo you are agreeing to the following license terms:

* Jigloo SWT/Swing GUI Builder is free for NON-COMMERCIAL use only.

* „Commercial use“ is defined as use by an employee of a business, corporation or institute (including academic institutes) to carry out the work for which they are employed. (Note: students of academic institutes are NOT defined as commercial users, unless they are being employed by their institute to carry out work which requires Jigloo).

* If Jigloo is used (after a successful evaluation period) for the work of any business, corporation or institute, a Professional License must be purchased for each employee using Jigloo.

* The code must not be disassembled or reverse engineered, or copied in any way.

* This software is provided AS IS and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

Anhang B

CryptOntology

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">

<owl:Ontology rdf:about=
       "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#">
    <dc:title>The RDF Vocabulary for Encryption</dc:title>
    <dc:description>This is the RDF encryption vocabulary for
       encrypting RDF.</dc:description>
</owl:Ontology>

<rdf:Property rdf:about=
       "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#identic">
    <rdfs:isDefindedBy rdf:resource=
       "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#"/>
    <rdfs:label>identic</rdfs:label>
    <rdfs:comment>The subject is a identical synonym to the object.
       </rdfs:comment>
    <rdfs:domain rdf:resource=
       "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#UID"/>
    <rdfs:range rdf:resource=
       "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about=
       "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#algorithm">
```

```
    <rdfs:isDefindedBy rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#"/>
    <rdfs:label>algorithm</rdfs:label>
    <rdfs:comment>The subject is encrypted with the algorithm the
        object describes.</rdfs:comment>
    <rdfs:domain rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#UID"/>
    <rdfs:range rdf:resource=
        "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#coded">
    <rdfs:isDefindedBy rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#"/>
    <rdfs:label>coded</rdfs:label>
    <rdfs:comment>The subject is encoded with the algorithm the object
        describes.</rdfs:comment>
    <rdfs:domain rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#UID"/>
    <rdfs:range rdf:resource=
        "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#keyLength">
    <rdfs:isDefindedBy rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#"/>
    <rdfs:label>keyLength</rdfs:label>
    <rdfs:comment>The subject is encrypted with a key of the length,
        which is named by the object.</rdfs:comment>
    <rdfs:domain rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#UID"/>
    <rdfs:range rdf:resource=
        "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdfs:Class rdf:about=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#UID">
    <rdfs:isDefindedBy rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#"/>
    <rdfs:label>UID</rdfs:label>
```

```
    <rdfs:comment>The class of UIDs, which are the synonyms of the
        encrypted literals.</rdfs:comment>
    <rdfs:subClassOf rdf:resource=
        "http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdfs:Class>

<rdf:Property rdf:about=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#toFindAt">
    <rdfs:isDefindedBy rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#"/>
    <rdfs:label>toFindAt</rdfs:label>
    <rdfs:comment>The encrypted string is saved in a data file and
        to find at the position, the object specifies.</rdfs:comment>
    <rdfs:domain rdf:resource=
        "http://www.crypto.peranka.de/2005-10-31/rdf-crypto#UID"/>
    <rdfs:range rdf:resource=
        "http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Property>

</rdf:RDF>
```

Abbildungsverzeichnis

Literaturverzeichnis

[AB] Ross Anderson and Eli Biham. Tiger: A Fast New Cryptographic Hash Function (Designed in 1995). http://www.cs.technion.ac.il/ biham/Reports/Tiger/.

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised), Februar 2004. http://www.w3.org/TR/rdf-syntax-grammar/.

[BG04] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, Februar 2004. http://www.w3.org/TR/rdf-schema/.

[bsi] BSI Schulung IT-Grundschutz - Glossar. https://ncc.uni-mannheim.de/bsi-webkurs/gsschul/gskurs/seiten/glossar/gloss_ah.htm.

[Car03] Jeremy J. Carroll. Signing RDF Graphs, Juli 2003. http://www.hpl.hp.com/techreports/2003/HPL-2003-142.pdf.

[Cha01] Pierre-Antoine Champin. RDF Tutorial, April 2001. http://www710.univ-lyon1.fr/ champin/rdf-tutorial/rdf-tutorial.pdf.

[ER02] Donald Eastlaek and Joseph Reagle. XML Encryption Syntax and Processing, Dezember 2002. http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[GB04] Jan Grant and Dave Beckett. RDF Test Cases, Februar 2004. http://www.w3.org/TR/rdf-testcases/.

[Hay04] Patrick Hayes. RDF Semantics, Februar 2004. http://www.w3.org/TR/rdf-mt/.

[HIM02] Merlin Hughes, Takeshi Imamura, and Hiroshi Maruyama. XML Encryption Syntax and Processing, Dezember 2002. http://www.w3.org/TR/2002/REC-xmlenc-decrypt-20021210/.

[KC04] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, Februar 2004. http://www.w3.org/TR/rdf-concepts/.

[MBD05] Brian McBride, Daniel Boothby, and Chris Dollin. An Introduction to RDF and the Jena RDF API, Oktober 2005. http://jena.sourceforge.net/tutorial/RDF_API/index.html.

[MM04] Frank Manola and Eric Miller. RDF Primer, Februar 2004. http://www.w3.org/TR/rdf-primer/.

[Pfi00] Andreas Pfitzmann. Sicherheit in Rechnernetzen: Mehrseitige Sicherheit in verteilten und durch verteilte Systeme, Oktober 2000. http://dud.inf.tu-dresden.de/ pfitza/D-SuKrypt.pdf.

[W3C99] W3C. RDF Definition, Februar 1999. http://www.w3.org/1999/02/22-rdf-syntax-ns#.

[W3C01] W3C. RDFS Definition, Januar 2001. http://www.w3.org/2000/01/rdf-schema#.

[Wik] Wikipedia - Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Hauptseite.

Erklärung

Hiermit versichere ich, dass ich die vorliegende schriftliche Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen verwendet habe.

Dresden, den 1. November 2005

Sabrina Gerbracht