Implementing content revocation using microformats and certificate revocation as building blocks

Henrik Niklaus

Diploma Thesis
at the Faculty of Mathematics, Informatics and Natural Sciences
University of Hamburg

submitted by
Henrik Niklaus

Hamburg, December 3, 2007
First supervisor: Prof. Dr. Joachim Posegga
Second supervisor: Prof. Dr. Norbert Ritter
Acknowledgement

I would like to thank Prof. Dr. Joachim Posegga and Prof. Dr. Norbert Ritter for supervising my thesis. Special thanks go to Henrich C. Pöhls for the inspiration to this topic and for the always helpful comments and discussions. I am grateful to my parents who supported me the whole time of my studies and last but not least I thank Mirja for her persevering and loving support.
Contents

1. Introduction 1

I. Fundamentals 3

2. Overview of annotation methods 5
   2.1. XHTML 5
      2.1.1. Semantic HTML 6
      2.1.2. Referencing 7
   2.2. RDF 7
      2.2.1. Basic concepts 7
      2.2.2. RDF syntax 8
      2.2.3. RDF semantic 10
   2.3. Microformats 11
      2.3.1. Syntax 12
      2.3.2. XMDProfiles 13
      2.3.3. GRDDL 13
   2.4. Assessment 14

3. Overview of cryptographic methods 17
   3.1. Certificate 17
      3.1.1. Certificate validation 17
      3.1.2. Certificate revocation 18
   3.2. PKI 21
      3.2.1. Certification authority 22
      3.2.2. Registration authority 23
      3.2.3. Repository 23
      3.2.4. Archive 23
   3.3. X.509 24
      3.3.1. Certificate fields 24
   3.4. Simple Public Key Infrastructure (SPKI) 30
   3.5. Pretty Good Privacy (PGP) 31
      3.5.1. Web of trust 31
      3.5.2. Key distribution and revocation 32
   3.6. XMLDSig 32
      3.6.1. Structure 32
IV. Conclusion 79
6. Conclusion and further work 81
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Overview of differences between ordinary X.509 v3 field values and the values of a content certificate</td>
<td>60</td>
</tr>
</tbody>
</table>
List of Figures

2.1. Example RDF Graph - “John Doe is the creator of http://www.example.org/”. 8

3.1. Overview of X.509 public key certificate fields. 25
3.2. Exemplary Merkle Hash Tree with binary node identifiers. 35

4.1. Example n-ary hash tree. The nodes are equipped with decimal identifiers in level-order. 54
4.2. Chosen hash tree format, filled with example data. The nodes are equipped with binary identifiers, e.g. ’011’ for the right-most leaf. Hash values are lexicographical sorted. Hash function is SHA-1. 56
4.3. Example hash tree with omitted sub-tree beneath the node ’01’. 56
4.4. Certificate revocation in a classical PKI. Revoking the certificate implies the revocation of all until then published signatures. 58
4.5. Revocation of content certificates. Each content can be revoked on its own without revoking other content. 59

5.1. Interaction of XUL and Gecko. Source: [Xul]. 65
5.2. Activity diagram of the prototype. 68
5.3. Relations between extension components. 69
5.4. Statechart for the signature’s internal states. 70
5.5. Screenshot of the SMAC prototype displaying multiple signatures within a document. On the left side without installed SMAC-extension, on the right with SMAC. 72
5.6. Server-component objects and their relations. 73
Listings

2.1. Example RDF/XML-Syntax -
"John Doe is the creator of http://www.example.org/". 8

2.2. Example Notation 3 -
"John Doe is the creator of http://www.example.org/". 9

2.3. Example N-Triple -
"John Doe is the creator of http://www.example.org/". 9

2.4. Example RDFa statement -
"John Doe is the creator of http://www.example.org/". 10

2.5. Compound hCard microformat filled with example data. 12

2.6. Different representation forms of the same microformat property. 13

3.1. XMLDSig skeleton - '?' denotes zero or one occurrence, '+' one or more and '*' zero or more occurrences. Elements between '(' and ')' are optional. 32

3.2. XMLDSig for a FOAF [Foaf] RDF document using a PGP signature. Source: [Kah05]. 37

4.1. Enveloped hsig with shortened example values. 46

4.2. Enveloping hsig with shortened example values. 47

4.3. Exemplary representation of a policy property with example policy expression. 48

4.4. Key property with example X.509 certificate in Base64 PEM-format. 48

4.5. Cert property with example certificate in Base64 PEM-format. 49

4.6. Example authdata properties with SHA-1 hash of the omitted value in hexadecimal format. 49

4.7. Schematron schema for the hsig microformat. 50

5.1. Example hcard-property email with stated subtypes. 74
1. Introduction

Today content is more than ever in the focus of the modern web. The so called user generated content (UGC) unhinges the classical relationship between the content’s author and the publisher. The initial source for published content which is under the author’s control is complemented by various platforms which process this content and re-publish it under a new type of service. Services like Google News [Goob], Bloglines [Blo] or digg [Dig] are popular examples of these so called re-publishers. Usually re-publishers add additional value to the external content like the possibility for the user to add comments or rate certain content, therefore they are denoted as publishers them-self. Additionally introducing the term owner denoting the person or organization which is in the possession of the content’s publishing-rights, is more accurate.

Classic security-goals as authenticity, integrity and confidentiality are a complex and often neglected topic in the context of modern web-publishing. Furthermore, after the owner publishes his content, he immediately looses control over it. He is not able to withdraw the publishing as a whole nor is he able to control the manner his content is re-published by third parties. A common procedure of re-publishers encompass the omitting of content-fragments owed to space-economization. The owner can not control which fragments are omitted and therefore his intention concerning related fragments may be violated. Of course the owner is also not able to be in control over the context in which his content gets re-published nor can he control which services provide access to his content. Legal means may certainly be applied, but owed to the web’s diversity and immanent openness usually fail¹.

The user on the other hand lacks important information if he receives the content over a different channel than directly from the owner. Without a costly research a user is usually not able to determine the content’s origin, verify its integrity, check the owner’s consent to the publishing and not least to check the owner’s consent to the manner of its publishing.

This thesis proposes a technical approach to face these issues. The security-goals authenticity and integrity are traditionally safeguarded by means of a signature. Confidentiality is a goal which applies mostly to closed environments and is not suitable for web-publishing. Safeguarding the confidentiality of content is in contrast to content-publishing and therefore not considered in the thesis. Furthermore an approach which

¹see e.g. http://www.ft.com/cms/s/2/08b10610-f8cd-11db-a940-000b5df10621.html and http://blog.digg.com/?p=74
attempts to enforce cryptographic systems upon the user like various Digital Rights Management (DRM) systems did in the past\textsuperscript{2}, does not seem to be feasible within the open nature of today's web. Trying to control which third party re-publishes the content or in which context the content is published is a complex task. Until the Semantic Web like Tim Berners Lee introduced it [TBLL01] is not existent, the publishing-context can not generally be determined. Regulating which third parties re-publish the content is likewise infeasible in respect of the vast amount of re-publishers nor is it commonly intended by the owner. Nevertheless an approach is possible which addresses the majority of the mentioned issues.

The thesis introduces a signature-format for web-content. The signature re-uses existing cryptographic technologies and annotation-methods to implement the mentioned requirements. The owner should thereby get the possibility to express his intention of how his content may be processed and to express his consent to the publishing as a whole. The re-publisher should be able to process and thus alter the content conforming to the owner's intention without invalidating the content's signature. The user or viewer of the content should be able to verify the origin, the integrity and the owner's consent to the re-publishing of the content. All of these measures should operate despite the absence of a direct trust-relationship between user and publisher.

In order to assess the approach's practical relevance a prototype-implementation is done as part of the thesis. The focus lies here especially on the user's perspective. His perspective is the most relevant one concerning complexity and the amount of recurring tasks in comparison to the other perspectives.

\textsuperscript{2}see e.g. CSS http://www.dvdcca.org/css/ and DeCSS http://en.wikipedia.org/wiki/DeCSS/
Part I.

Fundamentals
2. Overview of annotation methods

The vast majority of items indexed by internet-search-engines like yahoo! (Yahe) are (X)HTML¹-documents (Yahb). Considering a digital signature in the web-environment, one consequentially has to focus on signatures over content marked up in (X)HTML.

A digital signature assures the integrity and authenticity of arbitrary data. The data’s semantic is irrelevant for the signature-process to perform its task as long as the data is accessible. The accessibility of data in the web is provided by an Uniform Resource Locator (URL). This locator is not only able to reference data at a specific location, he may also reference arbitrary HTML document fragments with so called fragment or element identifiers. This method of referencing data would be sufficient for an ordinary digital signature in the web-environment.

Nevertheless, an additional layer of semantics would generate an explicit additional value for all participating parties. If for example an author publishes his address-data on his homepage and signs this data with an ordinary signature, a user is able to verify the integrity and authenticity of the data. If the raw data is additionally semantically annotated and afterwards signed, a user would be able to gather more information from the signed data. This semantical layer becomes even more useful by adding a content revocation mechanism as introduced in section 4.6 on page 57. Therewith the semantically annotated content may be equipped with an additional status indicating if the owner still consents to the publishing.

The sections in this chapter give an overview of current languages and technologies for referencing content(-fragments) and semantically annotating content(-fragments) in the web-environment.

2.1. XHTML

The Extensible HyperText Markup Language (XHTML) is a markup language for representing structural, presentational and semantic information in text-based documents. XHTML is standardized by the W3C and actually consists out of three standardization efforts: XHTML 1.0 as the current and most widespread XHTML W3C Recommendation [SP00] which is a reformulation of HTML 4 [DR99a] in XML 1.0 [TB06], XHTML

¹(Extensible) HyperText Markup Language
1.1 [MA01] and the working draft XHTML 2.0 [JA06]. A SGML® DTD⁵ is used to formalize HTML. XHTML in contrast is an application of XML which in turn is a smaller subset of SGML. All these recommendations and drafts are recommended readings to gain a deeper insight.

2.1.1. Semantic HTML

The original intention of HTML was not presentational markup but solely structural and semantic markup. This intention was diminished by introducing presentational-only tags like the <font>-tag beginning in 1993. With the introduction of CSS⁴ level 1 in 1996 [HWL96] and the successive standard recommendations HTML 4.0 strict and finally XHTML 1.1, the W3C’s movement towards the elimination of presentational components became obvious [All07].

In XHTML 1.1 the application of presentational tags is strictly banned, while the preceding XHTML 1.0 still allowed and currently allows the usage of presentational-tags-enabling DTDs [SP00, Appendix DTDs].

An overview of semantic and structural HTML-tags can be found at [Mfc] and certainly in the HTML 4.01 strict DTD [DR99g]. An adequate example for a presentational tag in HTML 4.01 is the font style element <b> [DR99b, pp. 15.2.1 Font style elements] which has no semantical meaning but only renders the enclosed text as bold text style. To emphasize a text portion HTML actually intended to use the tag <em> [DR99c, pp. 9.2.1 Phrase elements].

Besides the defined semantics-carrying tags, the HTML specification also provides several means to extend its semantic. Mainly four tags and attributes provide this functionality: the tags meta and link and the attributes profile, rel, rev and class. [Sud06]

The meta-tag [DR99d, pp. 7.4.4 The META element] is used to identify document properties like author or keywords, by specifying a property/value pair. The specification itself does not provide the definition of property-sets, but offers this possibility with the profile-tag [DR99d, pp. 7.4.1 The HEAD element]. A format for this meta data profile is XMDP⁵, as introduced in section 2.3.2 on page 13. Meta tags may also carry Dublin Core metadata elements [Dc] to express information resource descriptions.


---

²Standard Generalized Markup Language
³Document Type Definition
⁴Cascading Style Sheets
⁵XHTML Meta Data Profiles
with the current document [DR99e, pp. 12.2 The A element] and the `rev`-attribute describes exactly the reverse relationship. Both latter attributes may also be applied to the anchor tag. HTML 4.01 states, that additional link types not defined in the specification may be defined by authors [DR99f, pp. 6.12 Link types].

The `class`-attribute is commonly known as a selector for style-sheet information concerning the carrying HTML-element. The HTML 4.01 specification nevertheless defines a second role for the `class`-attribute: “For general purpose processing by user agents.” [DR99d, pp. 7.5.2 Element identifiers]. This second role enables the addition of element semantics through class-names. Microformats (introduced in 2.3 on page 11) embody a technology which utilizes this role.

### 2.1.2. Referencing

The locating of a web resource is realized through an URL as a subset of an URI [TBL98]. The former identifies resources via their primary access method like for example HTTP\(^6\) or FTP\(^7\).

Referencing a content fragment within a HTML-document is realized through the fragment identifier [TBL98, p. 14], which is appended to the URL consisting out of the crosshatch ("#") character and a trailing reference information. This fragment identifier, from the W3C also referred to as element identifier is interpreted by the user-agent after the resource has been retrieved. To reference a specific fragment of a HTML-document the attributes `id` and `class` are used [DR99d, pp. 7.5.2 Element identifiers]. These identifiers them-self provide no general namespace. Therefore they are only defined to be unique within the surrounding document.

### 2.2. RDF

The Resource Description Framework (RDF) is a collection of W3C Recommendations for making statements about web-resources. The RDF enables the machine processing and understanding of resources by defining RDF concepts and abstract syntax [GK04], the mathematical foundations [PH04], the RDF/XML syntax [DB04b] and a vocabulary description language [DB04a].

#### 2.2.1. Basic concepts

The basic data-model of every expression in RDF is a collection of subject-predicate-object triples, also referred to as an RDF Graph [GK04].

The subject specifies the resource and the predicate expresses the form of relationship between the subject and the object, while the latter is again a resource. Each part of the

---

\(^6\) Hypertext Transfer Protocol

\(^7\) File Transfer Protocol
Figure 2.1.: Example RDF Graph -
"John Doe is the creator of http://www.example.org/".

triple or binary relation respectively is identified by URI References [TBL98], a literal or a blank node. An object of a triple may be used as a subject in another triple, what leads to a directed labeled graph as depicted in figure 2.1 on page 8. The meaning of an RDF Graph is the logical conjunction of all triple-statements it contains. Additionally RDF is able to define RDF statements about other RDF statements with the so called RDF reification.

The RDF itself predefines no data-types except the one used for embedding XML in RDF, instead URIs are used to identify XML Schema data-types [Xmlb].

2.2.2. RDF syntax

An RDF Graph may be serialized in different formats. The RDF/XML syntax is also a W3C recommendation to serialize RDF with XML [DB04b]. Besides this main format N-Triples [JG04] and the Notation 3 [BL98] introduced by Tim Berners-Lee are variations of formats encoding RDF Graphs. In addition RDFa is a RDF/XML variation for embedding RDF directly in XHTML.

RDF/XML

RDF/XML is a serialization format for RDF based on XML. The root element of a RDF document is `<rdf:RDF>` and holds arbitrary many statements in form of `<rdf:Description>` elements. The `rdf:about`-attribute identifies the statement's subject, while the statement-element itself contains one or more predicates and objects. The example RDF document in listing 2.1 conveys the statement "John Doe is the creator of the resource http://www.example.org/".

```xml
<?xml version="1.0" ?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
>
    <rdf:Description rdf:about="http://www.example.org/">
        <dc:creator>John Doe</dc:creator>
    </rdf:Description>
</rdf:RDF>
```
2.2 RDF

Listing 2.1: Example RDF/XML-Syntax -
“John Doe is the creator of http://www.example.org/".

RDF additionally provides types and properties to represent groups of resources and
RDF statements, referred to as RDF containers. A RDF Container has the three sub-
classes <rdf:Bag> for unordered containers, <rdf:Seq> for a container with a sequence
of statements and <rdf:Alt> for a list of alternative statements.

Notation 3 (N3)

Notation 3 is an alternative encoding format for RDF Graphs introduced by Tim Berners-
Lee in [BL98]. It is a format which is more compact and provides a better human read-
ability.

```xml
@prefix dc: <http://purl.org/dc/elements/1.1/>.
<http://www.example.org/> dc:creator "John Doe";
```

Listing 2.2: Example Notation 3 -
“John Doe is the creator of http://www.example.org/".

N-Triples

N-Triples [JG04] is a plain-text format to encode a RDF Graph, while being a subset
of N3. N-Triples is the recommended format for exchanging RDF informations between
applications.

```xml
```

Listing 2.3: Example N-Triple -“John Doe is the creator of http://www.example.org/".

RDFa

RDFa is a syntax for encoding a RDF Graph in XHTML documents. RDFa is currently
under development and at the time of writing just a W3C working draft [BA07]. At
the current state RDFa is integrated in XHTML through additional meta-informational
attributes introduced in XHTML 2.0 [JA06], which is likewise a W3C working draft.
The Semantic Web Deployment Working Group (SWD-WG) nevertheless makes efforts
to-wards a valid integration into all existing (X)HTML specifications. These efforts have
lead to a DTD for the RDFa integration in XHTML 1.1 [Xhtl]. RDFa is actually a subset
of RDF.

In RDFa a RDF statement predicate is specified using a property-, rel- or rev-
attribute, the object is specified by the content- or href-attribute and the subject is
usually specified by the about-attribute as demonstrated in the XHTML document excerpt in listing 2.4

```xml
<div xmlns:dc="http://purl.org/dc/elements/1.1/"
  about="http://www.example.org/">
  <span property="dc:creator">John Doe</span>
</div>
```

Listing 2.4: Example RDFa statement -
"John Doe is the creator of http://www.example.org/".

Directly annotating the content in place has the drawback that a predicate's subjects and/or object is often represented by a literal and not an URI. This issue impedes the building of RDF-triple-chains and therefore the finding of new information through inferencing.

The Gleaning Resource Descriptions from Dialects of Languages (GRDDL) [Con07] is one way of extracting RDF from RDFa contained in a XHTML-document and is introduced in section 2.3.3 on page 13.

### 2.2.3. RDF semantic

Besides the syntactical understanding of statements about resources, it is certainly necessary for a machine to understand the meaning of these statements. Although syntactical correct statements may be formulated, they are not implicitly meaningful. Without a form of vocabulary functioning as an ontology [Gru93] for a specific domain, only a human is able to distinguish between meaningless and meaningful statements.

RDF Schema (RDFS) is a language for describing such a vocabulary. With RDFS it is possible to define data-types similar as in object-oriented programming languages. Relationships between resources can be expressed by classes. Each RDF statement can use the rdf:type declaration to indicate that a statement is an instance of the specified class. Inheriting is enabled through rdfs:subClassOf and rdfs:subPropertyOf, indicating for example that the class author is a sub-class of the class creator. Classes mainly exist to restrict what can be stated in a RDF document. In contrast to object-oriented programming languages like Java, attributes in RDFS are not defined in the class itself, for each predicate it is stated which class is its subject (referred to as domain) and which class is its object (referred to as range). This approach enables the subsequent definition of additional properties of a specific domain or range without modifying the class-description. This openness is one of the basic architectural principles of the (Semantic) Web. [DB04a]

The expressive power of RDF and RDFS is limited in some aspects. RDF is basically designed for binary relations, thus limiting the amount of possible arguments of predicates. RDFS is limited to a subclass hierarchy and a property hierarchy and therefore is just able to describe taxonomies. It is additionally not possible to define constraints
upon properties, but just on the domain and the range. Therefore a more expressive ontology language is required to fill the gaps in RDFS. The Web Ontology Language (OWL) introduced by the W3C [DLM04] is such a language, however this additional layer is out of the thesis' scope and therefore not described any further.

2.3. Microformats

The trick.... is to make sure that each limited mechanical part of the Web, each application, is within itself composed of simple parts that will never get too powerful.
-Tim Berners-Lee, Weaving The Web [BL00]

Microformats are an approach to extend the existing semantics of (X)HTML. Unlike RDF they do not define a new generic language for statements about resources, but utilize the existing (X)HTML syntax. One of the primary goals is to retain the human-readability of semantic extensions, while enabling the machine-readability. Microformats are not standardized through an authoritative organisation alike the W3C or the IETF, instead a community organised around the microformats website [Mf] is centrally responsible for issuing specifications and drafts for the provided formats. In short "Microformats are a codification of convention" [Mfd].

Microformats are based on the following basic design principles [All07]. Microformats

- solve a specific problem.

- start as simple as possible.

- are designed for humans first, machines second.

- reuse building blocks from widely adopted standards.

- are modular and embeddable.

- enable and encourage decentralized development, content, services.

It is a data-driven bottom-up approach where new formats emerge only in a way conforming to a pre-defined process [mic07b] which in turn reflects the mentioned design principles. Generally only formats with widespread use-cases could become microformats. Especially data which is commonly available either way on a web-page like address-data or calendar- and event-information, may be annotated with microformats. A complete list of specifications and drafts is located at [Mf b].
2.3.1. Syntax

Microformats syntactically conform to the (X)HTML syntax. They actually have no namespace and no specific underlying schema. Nevertheless (X)HTML provides a mechanism for defining property-value-pairs of meta-data through so called (X)HTML Meta Data Profiles (XMDProfiles) (see section 2.3.2 on page 13). These profiles may be used to define microformat’s semantic.

A microformat utilizes the offered extensibility of (X)HTML as introduced in section 2.1.1 on page 6. Primarily the class-attribute is used to convey specific formats. Listing 2.5 holds an example of a hCard format [mic07a] representing address-data of an entity.

```html
<div class="vcard">
  <span class="fn">John Doe</span>
  <a class="email" href="mailto:john.doe@example.com">john.doe@example.com</a>
  <div class="adr">
    <div class="street-address">street</div>
    <span class="locality">locality</span>
    <span class="region">region</span>
    <span class="postal-code">11111</span>
    <span class="country-name">country</span>
  </div>
</div>
```

Listing 2.5: Compound hCard microformat filled with example data.

There are two types of microformats, referred to as **elemental** and **compound**. Elemental microformats are the building blocks for compound ones. These formats usually consist of a single (X)HTML element and express a minimal solution to a single problem. Compound microformats consist of multiple elemental microformats or standard (X)HTML elements. Furthermore compound microformats may also function as building blocks for another compound format. [All07] [Sud06]

I further refine the definition of compound and elemental microformats to be able to describe the nature of microformat properties. Microformat properties are all elements beneath the microformat identifier in the DOM-tree, corresponding to the defined properties according to the actual specification. Beneath the hCard identifier vcard in listing 2.5, fn and postal-code are for example properties of hCard. I call properties **parent**, if there are sub-properties defined, as for example adr in hCard. Furthermore **atomic** properties are quite the same as elemental microformats. They directly consist of the identified value and do not contain further sub-properties.

Additionally I introduce the notion **container** as a generalization of all (X)HTML elements and attributes who are capable of providing microformat semantics. The class-attribute is the major method to provide the microformat property identifier, alongside for example rel and rev. The value of the property may be provided through different methods as exemplary demonstrated in listing 2.6 on page 13. Therefore a generic term
for microformat-providing structures seems to be appropriate.

```html
<br/>{<span class="tel">+49-40-428-83-2202</span>}<br/>

Listing 2.6: Different representation forms of the same microformat property.

The microformats community introduced so-called include-pattern which is a mechanism to include or reference content-fragments which are located in another subtree of the DOM\(^8\). This pattern circumvents the repetition of already in the document present microformat data. The hResume-microformat for example requires the fn property. The include-pattern may now be used to include the author’s name in all of his reviews without the need for repeating it.

For the sake of completeness it should be mentioned that microformats may also be provided through non-(X)HTML markup languages as RDF Site Summary (RSS). These structures should also be subsumed beneath the notion container.

2.3.2. XMDProfiles

XHTML Meta Data Profiles (XMDP) is a format based on (X)HTML for defining HTML meta data profiles [DR99d, pp. 7.4.4.3 Meta data profiles]. It uses the profile-attribute of the head-element in (X)HTML to point to the profile’s location. Actually XMDP defines no data-types as for example XML Schema. A format based on XMDP is just an informative description of properties, more human-readable than machine-readable, also referred to as a dictionary [DR99d, pp. 7.4.1 The HEAD element].

GRDDL (section 2.3.3 on page 13) can use such a profile to reference the GRDDL-profile and the corresponding transformation. According to [Dan07] usable profiles today only exist for the formats hCard, hCalendar and hReview, while profiles in general but not usable with GRDDL exist for the format specifications XFN, VoteLinks, XOXO, relLicense, relTag, rel-nofollow and the format drafts hAtom, hResume, Robots Exclusion, xFolk, rel-directory, rel-enclosure, rel-home and rel-payment. Currently not every on microformats.org mentioned microformat is in the possession of a XMD-profile.

2.3.3. GRDDL

Gleaning Resource Descriptions from Dialects of Languages (GRDDL) is a technique for obtaining RDF data from XML documents, especially (valid) XHTML documents. [HH07]

\(^8\)Document Object Model
It extracts RDF data mainly through XSLT, but theoretically other transformation methods are possible. Currently documented dialects for extraction are RDFa, Dublin Core Metadata in meta-tags and microformats. Effectively every XML dialect or XHTML document wherefor an XSLT style-sheet exists is certainly usable with GRDDL.

Technically a reference in the head-element's profile-attribute to the GRDDL-profile (http://www.w3.org/2003/g/data-view) indicates that within the document a link to transformations are present. These transformations conjuncted with the originating document result in the desired RDF data. Microformats use an indirection over their XMDProfile if existent. These profiles convey a reference to the GRDDL-profile and the corresponding transformation, usually a XSLT style-sheet.

Note: Trying to map RDF and microformats does not seem to be reasonable. Both technologies have totally different intentions and procedures. Microformats being a set of conventions and RDF being a whole framework for describing semantic relations between resources.

2.4. Assessment

(X)HTML provides sufficient means for referencing content or content-fragments. (X)HTML itself only provides marginal support for semantic annotations. Referencing content and providing extensibility to add further semantics, make (X)HTML to a major building block to superimpose richer semantics with RDF or microformats to web-documents.

RDF is a comprehensive framework to describe relations among resources. In combination with ontology languages like OWL, RDF is the de facto standard in relation to the Semantic Web. Except for RDFa all syntaxes of RDF add their metadata at a designated location, while not annotating the data directly in place. This characteristic adds unnecessary redundancy and complexity to web-documents.

Microformats have neither a namespace nor globally unique identifiers. This absence may lead to naming-conflicts within a certain document. Furthermore they lack the existence of a underlying schema, what generally makes them hard to validate. In addition microformats make extensive use of the possibility of an (X)HTML-element to provide multiple values for class-attributes, which complicates the validation even more. Nevertheless Schematron [Sch] is a schema language which is theoretically capable of validating microformats, but is based on XPath-expressions which in turn are not specified by origin of the microformat community. Microformats in contrast to RDF use a implicitly defined ontology for a specific narrowed domain and provide no potentials to define arbitrary ontologies to use. Consequently reasoning about a set of rules is not possible with microformats either.
Despite all these drawbacks, microformats have the major advantage of simplicity. They can be easily integrated into existing web-documents due to their unobtrusive (X)HTML-Syntax, they do not necessarily require the adjustment of a (X)HTML document's head element for enabling annotations and the learning curve for web-developers is quite flat. Microformats add their metadata directly within the document, surrounding the already displayed content, without using an external or global location. Furthermore microformats are generally compatible to all existing specifications of the HTML family. This simplicity and compatibility within the heterogeneous web makes microformats to a much more practise oriented approach as any other approach for the Semantic Web.

My subjective impression is that microformats are experiencing a relative broad distribution among web-documents compared to RDF. While a valid survey about this issue is quite difficult and to my knowledge not existent until today, there exists an informal survey about web-documents conducted by google back in 2005 [Gooc] which supports my impression. Additionally many major web-sites including yahoo! local [Yaha], Google Maps [Gooa] or LinkedIn [Lin] support microformats [Mf a].

As an overall conclusion microformats seem to be the most valuable solution concerning current markup languages in the web. Even though microformats do not provide the same expressiveness as RDF, they are a simple, bottom-up approach to extend HTML semantics where practical. Concerning a digital signature for content, microformats provide the necessary means to reference content and to add a certain amount of semantics to the signed content. Nevertheless RDFa is a good composition between microformats and RDF. If RDFa experiences more popularity and distribution among web-documents and becomes compatible to all HTML family members, it is a good candidate to fulfil the requirements for a building block for digital signatures in the web-environment.
2. Overview of annotation methods
3. Overview of cryptographic methods

This chapter gives an overview of selected cryptographic methods and concepts that may be utilized to achieve the goal of authenticity and integrity while publishing in the web environment. After an introduction to certificates and their validation and revocation mechanisms in general, the concept of a public key infrastructure (PKI) will be the focus. Appending to the PKI the X.509 standard and the associated X.509 public key certificate is contemplated in more detail. PGP\(^1\) and its basic trust model will be shortly introduced followed by an illustration of XMLDSig as a standard for digitally signing XML documents. Last but not least the concept of a Merkle Hash Tree as a valuable structure at helping to keep a signature valid while the signed data changes will be discussed.

3.1. Certificate

In an abstract view a public key certificate binds an entity name (and possibly additional attributes) to a corresponding public key [AL99]. Kohfelder [Koh78] was the first to introduce the concept of a certificate and a certificate revocation list in a PKI environment.

There exist different types of certificates. Besides the most common X.509 public key certificate and the Simple public key infrastructure (SPKI) certificates, the so called PGP certificates and Attribute Certificates [AL99] are available. The focus of this section lies on the X.509 public key certificate due to its widespread distribution in internet protocols and implementations like S/MIME\(^2\), TLS\(^3\)/SSL\(^4\) and IPsec\(^5\).

3.1.1. Certificate validation

Before considering a certificate as valid, the following checks have to be performed:

- The certificate is signed by a trusted CA. This step may include certificate path processing as introduced in the next subsection.

---

\(^1\)Pretty Good Privacy  
\(^2\)Secure/ Multipurpose Internet Mail Extensions  
\(^3\)Transport Layer Security  
\(^4\)Secure Sockets Layer  
\(^5\)IP security
• The certificate's integrity is sound. This indicates that the data, the calculated hash value and the signature of the certificate match.

• The certificate is within the stated validity period.

• The certificate has not been revoked.

• The certificate's usage conforms to the stated policies and use-restrictions.

Certificate path processing
The purpose of constructing a certificate path is to find a complete and sound path of certificates between a given starting certificate and a already trusted certificate. Thereby the binding between the subject and its public key stated in the examined certificate is verified. A path between two certificates exists if the subject of each certificate in the path is the issuer of the certificate underneath it. The first phase comprises the aggregation of all necessary certificates in the path, it is the so called path construction. The following path validation includes the appraisal of the certificate's trustworthiness. If every certificate in the path between the examined certificate and a trusted certificate (the so called trust anchor) is valid, the examined certificate's binding is verified. This procedure becomes complex if cross-certification trust models are used. In this case path construction may involve graph-theoretic path-finding algorithms and tree-traversal methods like pre-order, in-order, post-order or level-order.

3.1.2. Certificate revocation
The binding a certificate vouches for is normally valid until the certificate's validity period expires. However there are several reasons like a private key compromise or a loss of the password for the private key usage, for revoking the validity of a certificate before it expires. Therefore a reliable method for revoking certificates must be provided.

Different implementations of certificate revocation mechanisms exist. The most widespread mechanism is the periodic publication of a Certificate Revocation List (CRL). All details of the CRL fields including the extension can be found in the RFC 3280 [RH02]. Another approach is the Online Certificate Status Protocol (OCSP), which provides an online request/response mechanism to obtain current revocation information. The following publication methods mainly apply to X.509 certificates but could also be used by other certification standards.

Periodic publication mechanisms
Periodic publication of revocation information include the following mechanisms: Complete Certificate Revocation Lists (CRLs), Authority Revocation Lists (ARLs), CRL Distribution Points, Delta CRLs, Indirect CRLs, Redirect CRLs, Over-issued CRLs and Certificate Revocation Trees (CRTs). [AL99]
3.1 Certificate

Complete CRLs  Complete CRLs include the complete revocation information associated to a certain CA domain. This approach is appropriate for CAs which are limited in the amount of issued certificates. CAs interacting with a larger amount of certificates undergo performance and distribution problems concerning the growing size of the CRL. The size of the CRL is indirectly influenced by the number of overall issued certificates assuming a certain probability of certificate revocations. Other factors are the average certificate’s validity period and the size of the certificate’s serial number. Therefore the estimation of the exact threshold where a complete CRL is becoming too large is quite complex, but it is reasonable to conclude that many certificate using systems reside above this threshold. For this reason different alternatives exist which are introduced in the following paragraphs.

Authority Revocation Lists (ARLs)  Authority revocation lists include exclusively information about CAs and therefore provide no revocation information about end entity certificates. The ARL-issuer is normally a superior CA or a CA which issued a cross-certificate.

Revocation of CA certificates has dependently on the trust model’s hierarchical structure quite an impact. In a strict rooted hierarchy a revocation of a CA certificate impacts all subordinate CAs.

CRL Distribution Points  CRL distribution points allow the splitting of a CA’s revocation information into multiple CRLs. Partitioning a complete CRL divides the CRL information into more suitable parts and reduces the drawback of a complete CRL. While pointing directly to the location of the partitioned CRL the relying party does not require prior knowledge of the CRL’s location.

Delta CRLs  Delta CRLs only contain the differences between the last issued and the current status of the CRL. This update mechanism again provides a reduction of size and complexity towards a complete CRL. Delta CRLs are incremental, thus starting with a so called base CRL, every issued delta CRL contains the information the previously issued delta CRL contained, plus the updated revocation information. Delta CRLs are provided by the certificate’s freshest CRL extension (a.k.a. Delta CRL Distribution Point). [RH02, pp. 5.2.4 Delta CRL Indicator]

Indirect CRLs  Indirect CRLs facilitate the distribution of CRLs originating from multiple CAs by integrating revocation information about certificates issued by foreign CAs into one CRL. In a scenario where multiple CAs participate, this type of CRL could reduce the traffic load and costs. Furthermore a third party aggregation service could use the indirect CRL as a centralized service for arbitrary CA domains, often called a Indirect CRL Authority (ICRLA). In a strict hierarchical PKI, all CAs on the upper levels which certainly only issue CA certificates could delegate the CRL-issuing to a single ICRLA and therefore reducing the amount of interactions with otherwise frequently
requested CRL issuers. Certainly a trust relationship has to be established between the CAs and the ICRLA.

The Issuing Distribution Point extension for CRLs provide the mean for indirect CRLs by setting the indirectCRL boolean to true.

**Redirect CRLs**  Common CRL distribution points are fixed for the certificate's lifetime. A prior knowledge of the future partitioning concerning the complete revocation information is therefore required. Changes in the size of the partitioning or the current location of a (partitioned) CRL can not be realized.

Redirect CRLs offer a solution to this circumstance by adding an intermediate layer in form of a standard conform CRL with a scope statement extension which enables the alteration of the partitioning and size of the CRLs with hindsight.

**Over-issued CRLs**  To reduce the workload of CRL issuers, especially during the CRL's expiration period, the over-issued CRLs were introduced [Coo99]. The only difference to complete CRLs is the overlapping of validity periods. Publishing CRLs equally distributed over the common CRL validity period, while every CRL is equipped with the same length of the period, provides the workload reduction.

**Certificate Revocation Trees (CRTs)**  Certificate revocation trees (CRTs) are an alternative to CRL based distribution methods [Koc98]. Each revoked certificate of a specific CA is represented through an equation combined with additional information like revocation reason, revocation date etc. and functions as a leaf of a Merkle Hash Tree (see section 3.7 on page 35). By means of the equation a verifying process is instantly able to check the certificate's revocation status. The root hash value of the CRT is signed by the issuer and therefore the whole tree can be distributed to untrusted third parties. A verifying application therefore just requires the authentication path data and not the whole signed complete CRL.

**Online Certificate Status Protocol (OCSP)**  The Online Certificate Status Protocol (OCSP) provides means to immediately check the revocation status of a certificate. A so called OCSP responder therefore provides a simple online request/ response service for revocation information. The OCSP is in detail documented in the RFC 2560 [MM99]. Support for OCSP is provided by a certificate with the authority information access extension.

CRLs use the pair of issuer DN and certificate serial number to uniquely identify a certificate. An OCSP responder eventually handles more than one CA. Due to the absence of a global directory for CAs it is theoretically possible that two CAs carry the same DN. OCSP therefore provides a more complicated certificate ID specification,
which bases on the certainly unique CA's public key [MM99, pp. 4.1.1 Request Syntax].

An OCSP request is comprised of the protocol version, the requester's DN, one or more certificate identifiers and an optional signature over the whole request. The certificate identifier is a sequence of a hash algorithm identifier, the hash of the issuer's DN, the hash of the issuer's public key and the certificate serial number. These fields are embedded in an ASN.1\textsuperscript{6} [Asn] structure and submitted over an arbitrary and not by the RFC defined channel like the HTTP-Protocol to the OCSP responder. The responder answers with a simple message, providing the three options good, revoked or unknown. This response has to be signed in order to ensure the revocation information's trust and integrity.

While offering a real-time service, OCSP does not inevitably offer up-to-date revocation information. This depends solely on the CA's policy and the source of the information for the OCSP responder, which is quite often just a common CRL. OCSP just supplies information about revocation status of certificates and does not adopt certificate path validation or other services concerning the verification of certificates. More advanced services which implement protocols like the Server-based Certificate Validation Protocol (SCVP) are required for these purposes.

**Server-based Certificate Validation Protocol (SCVP)** SCVP enables a certificate-using application to delegate certification path construction and certification path validation to a dedicated online service [TF07]. These services are accomplished by using a centrally administrated validation policy. Delegating the path construction and validation to a SCVP service therefore reduces the complexity of certificate handling client applications and facilitate the management of validation policies.

### 3.2. PKI

"Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed."

– Sir James George Frazer [Fra03]

A public key infrastructure (PKI) facilitates the use of public key cryptography through the creation and distribution of public key certificates and certificate revocation lists (CRLs). A PKI provides the infrastructure for communication between participants while ensuring the communication's confidentiality, integrity and authenticity.

According to [HP01] exist four main components for the provision of a PKI:

- The certification authority

\textsuperscript{6}Abstract Syntax Notation One
The registration authority

The repository

The archive

3.2.1. Certification authority

The certificate authority (CA) is the main building block of a PKI. Basically a CA provides the four functionalities certificate issuing, maintaining certificate status information and issuing CRLs, publishing its current certificates and CRLs to users and maintaining archives of its revoked/expired certificates.

Certificate issuing

Issuing a certificate means that the CA asserts that the subject possesses the private key corresponding to the stated public key and that all other attributes in the certificate correspond to the subject as well. A CA may not only issue certificates to users but also to other CAs, whereby it asserts that issued certificates from the other CA are trustworthy. A CA has the following general responsibilities [HP01, p. 44], ordered by descending importance and resumed in the next paragraphs:

1. Safeguarding the CA's private key against disclosure.

2. Verifying the subject's credentials like identity, personal information and policy information.

3. Ensuring that all issued certificates and CRLs conform to the CA's profile.

Maintaining certificate status information and issuing CRLs

Every information concerning the certificate status has to be complete and sound. Especially the date of revocation and the associated reason of revocation have to be maintained.

4. Accurate maintenance of revoked and expired certificates including all provided information.

Publishing its current certificates and CRLs to users

Users should obviously be able to retrieve certificates and CRLs to implement their own security services.

5. Provision of the technical infrastructure for certificate and CRL distribution to the PKI participants.
3.2 PKI

Maintain archives of its revoked/expired certificates

6. Maintaining sufficient archival information to establish the validity of certificates after expiration.

A CA as an entity may focus on requirements with the highest priority requirements and delegating lower-priority requirements to other PKI components. Especially the verification of user-credentials before issuing a certificate and the responsibility for certificate revocation decisions is often delegated to the so called registration authority (RA). The task of distributing certificates and CRLs can be delegated to the repository and the storage of archival information can be assumed by an archive.

3.2.2. Registration authority

The main task of a registration authority is the verification of credentials which a certificate requesting entity submits. The verification procedure usually bases on non-electronic means as for example a drivers licence or similar identity information provided by trusted third parties.

Between the CA and its affiliated RAs exist a trust relationship. To safeguard this relationship it is naturally equally important as for a CA to securely protect the RA’s private key. Verification of entity credentials or the future certificate contents respectively has two possible models. The first model includes that the subject directly contacts the RA for a certificate and after the provided information was verified the RA assembles a certificate request which is immediately signed by the CA due to the its trust relationship with the RA. The second model describes the case that the subject primarily generated a certificate request and send the request to the CA. The CA now delegates the request to the RA and awaits a positive or negative answer to proceed. Within the latter model the CA is certainly able to verify the matching of the given public key and the signature over the request before delegating the request to the RA.

3.2.3. Repository

A repository handles the provision of certificates and CRLs upon request. Itself the repository does not require to be a trusted entity, because the provided certificates and CRLs are signed by the already trusted issuing CA. Updating stored information is nevertheless restricted, to prevent denial of service (DOS) attacks towards the repository. Providing a repository as a standalone entity has the advantage that its design is able to facilitate the optimization of availability and performance of its provided services.

3.2.4. Archive

An archive is responsible for long-term storage of certificate related information. The archive safeguards the integrity of expired certificates. It additionally provides information about the reason of expiration or revocation respectively. The archive can be
3. Overview of cryptographic methods

utilized to prove that a document's signature was sound at a given time.

3.3. X.509

A PKI requires a service for the publishing and conveying of public keys to the participants. In this context a public key certificate is mainly utilized to assure that the integrity of a public key is sound and that the key belongs to the specified owner.

X.509 is a ITU-T recommendation and part of the X.500 recommendations that define a directory service. X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. [Sta02]

The X.509 ITU-T recommendation defines certain requirements for the standard fields and the extensions within the certificate [X50b]. Due to its general form the standard has nevertheless to be refined in specific profiles, to realize and maintain the interoperability between implementations. The IETF PKIX Working Group [Pki] has specified such a profile in RFC 3280 [RH02]. Three versions of a X.509 public key certificate exist. Versions 1 is a subset of Version 2 which itself is a subset of Version 3. The differences between these versions are discussed further later. For the remainder of the thesis the terms certificate and X.509 certificate are synonymously used with a X.509 Version 3 public key certificate as profiled in the RFC.

3.3.1. Certificate fields

X.509 certificates consist of three nested components. The outermost component is the temper-evident envelope, inside this envelope is the basic certificate content which is in RFC3280 labeled as tbsCertificate and the latter optionally includes a set of certificate extensions. [HP01]

Certificate fields are defined by means of Object Identifiers (OIDs) within a ASN.1 definition structure. All temper-evident envelope fields of a X.509 certificate are explained in the following list:

---

7International Telecommunication Union
8ITU Telecommunication Standardization Sector
9Internet Engineering Task Force
10Public Key Infrastructure X.509
11Request for Comments
12to-be-signed Certificate
Figure 3.1.: Overview of X.509 public key certificate fields.

- **certificate**

- **tbsCertificate**
  
  - **version** Indicates the version of the certificate format. Default is 1. If issuerUniqueID or subjectUniqueID is present it must be v2 or v3.
  
  - **serialNumber** A positive integer value, unique within the issuing CA.
  
  - **signature** The type of algorithm used to sign the certificate, together with optional parameters depending on the algorithm. Must be the same as signatureAlgorithm and is therefore redundant.
  
  - **issuer** Identifies the entity who issued and signed this certificate, usually a CA. Represented through a distinguished name (DN) according to the X.501 [X50a] type name. A DN is a path through a X.500 [Rad94] Directory Information Tree (DIT) which uniquely identifies an entity. A DN consists of a sequence of RelativeDistinguishedNames (RDNs), for example common name (CN), organisation (O), organisational unit (OU) and country (C). Nevertheless, the RFC does not exactly define the DN’s format, but recommends that implementations must be prepared to receive the following standard attribute types: country, organization, organizational-unit, distinguished name qualifier,
state or province name, common name and serial number. The syntax and associated object identifiers (OIDs) for these attribute types are provided in the ASN.1 modules within the appendix of the RFC.

- **validity** Consists of a sequence of two dates which define the certificate’s validity period. Both fields (*notBefore* and *notAfter*) may be encoded as *UTCTime* or *GeneralizedTime* [RH02].

- **subject** The subject field provides the DN for the entity associated to the public key stated in the subjectPublicKeyInfo field.

- **subjectPublicKeyInfo** Provides the subject’s public key and an algorithm-identifier for the corresponding algorithm with which the key is used. The OIDs of the supported algorithms and encoding methods are specified in [WP02].

- **issuerUniqueID** Provides a unique identifier for the issuer in the case the issuer’s DN field has been re-used for different entities. Only version v2 and v3 of the certificate support this field. Nonetheless does the RFC not recommend the re-use of the issuer DN for different entities.

- **subjectUniqueID** Provides a unique identifier for the subject in the case the subject’s DN field has been re-used for different entities. Only version v2 and v3 of the certificate support this field. Nonetheless does the RFC not recommend the re-use of the subject DN for different entities.

- **extensions** Instead of extending the certificate’s standard field the v3 version of the certificate provides the means to extend the certificate with the flexible extension field. This field is able to hold arbitrary many extensions out of a set of pre-defined extension categories. The section 3.3.1 on page 26 provides more details on this on the extension field.

- **signatureAlgorithm** Contains the identifier for the signature algorithm used in the certificate signing process. Recommended are the algorithms listed in [WP02]. Must carry the same identifier as the signature field in the *tbsCertificate* sequence.

- **signatureValue** Contains the actual digital signature over the ASN.1 DER encoded *tbsCertificate* sequence.

### Extensions

Since the standard fields of a certificate do not meet all potential requirements of an organization, the X.509 recommendation introduced the extension field. With extensions a certificate is able to provide additional information associated to the issuer, the subject or the public key. An organization is able to define its own private extension to

---

13Distinguished Encoding Rules
meet its special requirements, but the standard extensions defined in the X.509 profile meet most of these requirements.

An extension consists of the three components **extension identifier**, **criticality flag** and the **extension value**. The extension identifier is an OID indicating the syntax and the semantic of the extension value. The criticality flag indicates if a certificate using system must recognize the extension or not prior to further processing. If the system does not recognize a critical extension, it has to reject its use.

The X.509 profile recommends that conforming CAs must at least support the extensions **key identifiers**, **basic constraints**, **key usage** and **certificate policies** and conforming applications must at least recognize **key usage**, **certificate policies**, the **subject alternative name**, **basic constraints**, **name constraints**, **policy constraints**, **extended key usage**, and **inhibit any-policy**.

The following summarized extension descriptions originate from the X.509 recommendation [X50b], the X.509 profile [RH02], the books *Planning for PKI* [HP01] and *Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations* [AL99].

According to [HP01] the extensions can be grouped into the categories **subject type**, **names and identity information**, **key attributes**, **policy information** and **additional information**.

**Subject type**

**Basic Constraints** The basic constraints extension indicates whether or not this is a CA certificate. If the latter applies, an additional value for the length of the certification path is provided, a length of zero for example indicates that only one more certificate may follow this certificate in a certification path (so only end-entities come next and no further CAs). The extension is commonly absent in end-entity certificates. The X.509 profile recommends to set the criticality flag if the certificate is a CA certificate.

**Names and identity information**

**Subject Alternative Name** This extension provides alternative name forms for the subject, as for example email-addresses or DNS-entries. If the subject's DN field is empty, this extension is mandatory and has to be marked critical. Usually it is not included in CA certificates.

**Issuer Alternative Name** The issuer alternative name extensions is semantically identical to the subject alternative name extension, except that the issuer's DN is addressed. Additionally the profile does not allow the emptiness of the issuer's DN field.
Name Constraints  The extension indicates required and excluded subtrees of a directory information tree (DIT). Subsequent CAs can herewith be constrained to issue only certificates within a certain subtree or name-space respectively. The extension is only used in CA certificates and according to the profile must be marked critical if present.

Key attributes

Key Usage  An entity is able to possess multiple private/public key pairs, each possibly intended for a different purpose. The key usage extension defines the intended purpose of the public key contained in the certificate. The supported purposes are digital signature, non-repudiation, key encipherment, data encipherment, key agreement, certificate signature, CRL signature, encipher only and decipher only. Neither the profile nor the standard addresses allowable key usage value combinations. Therefore possible contradictions like the encipher only and decipher only purpose together could occur. The profile recommends to let this extension be critical if present.

Extended Key Usage  Indicates additional key usage purposes on top of the regular key usage extension especially designed for end entity certificates. Supported purposes are Transport Layer Security (TLS) server authentication, TLS client authentication, code signing, email protection, time stamping and OCSP signing. Additionally a purpose named anyExtendedKeyUsage indicates that the usage of the key is not restricted, while still maintaining compatibility with a certificate processing system which requires certain extended key usage purposes to be present.

Private Key Usage Period  With this extension the certificate issuer is able to indicate a different validity period for the associated private key as for the public key stated in the according certificate field validity. The extension is mainly intended for use with digital signature keys, where the private key should for example only be used for a few month, but validating the signature should be possible for a longer period.

Authority Key Identifier  If the signing issuer is in the possession of multiple private/public key pairs, the authority key identifier extension provides the mean to identify the corresponding public key to the signing private key. This extensions aids in the construction of a certification path, by facilitating the verification of a certificate's signature.

Subject Key Identifier  Identifies uniquely certificates carrying particular public keys. Especially if the subject has multiple certificates and probably multiple CAs which signed these certificates, this extension aids in the building of the certification path, by quickly identifying the necessary certificates or the public key respectively.

Policy information
Certificate Policies  Before the advent of this extension, every CA was only able to provide exactly one policy which was additionally only referenced implicitly. The extension contains a sequence of policy information terms, each consisting of an OID and optional qualifiers. The latter are not recommended by the profile to retain interoperability between certificate using systems. Nevertheless two qualifiers are defined: an URI which points to the Certification Practice Statement (CPS) - a document which describes how the CA operates - and the user notice qualifier which provides a notice (or a URI to that notice) for the user of the certificate and for display in a certification path.

If the certificate is a CA certificate, these policy information terms limit the set of policies for certification paths. Within an end entity certificate these terms indicate the policy or policies under which the certificate was issued. If a CA does not want to limit the set of policies, it may include the any-policy OID. The any-policy OID is especially used when issuing certificates to a totally trusted CA.

The X.509 recommendation defines a Certificate Policy as “A named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements.”

Policy Mappings  Defines equivalent policy OIDs in different CA domains. The extension is only present in CA certificates.

Policy Constraints  Constraints path validation by inhibiting policy mapping for the remainder of the certification path or by requiring each certificate in a path to contain an acceptable policy identifier. The extension only applies to CA certificates.

Inhibit Any-Policy  Like the policy constraints extension the inhibit any-policy extension imposes additional limitations upon certification paths. It limits the use of the any-policy OID (see certificate policies extension) by introducing a counter of the number of certificates in the path where the any policy is still allowed. The extension only applies to CA certificates and if present it has to be marked as critical.

Additional information

CRL Distribution Points  In contrast to complete CRLs (see section 3.1.2 on page 18) the CRL distribution points extension allows the splitting of a CA's revocation information into multiple CRLs. It consists of a sequence of pointers to a certificate revocation list for this certificate. Each pointer contains either a general Name in form of a DN-Sequence or a general Name of type URI which points to the location of the CRL. More details to this issue are found in section 3.1.2 on page 18.
Freshest CRL  This extension is also known as the delta CRL distribution point extension. The extension identifies how delta CRL information is obtained. The format is identical to the CRL distribution point extension. More details to this issue, especially what delta CRLs are, are found in section 3.1.2 on page 18.

Authority Information Access  The authority information access extension indicates how to access CA information and services. Information and services may consist of online validation services and CA policy data, but no CRL locations because these are handled by the CRL distribution point extension. The extension provides access methods and access locations to indicate how and where to retrieve the information. Hereby exist two options:

1. CA issuer OID: Identifies other CAs which issued certificates to this CA and helps hereby in the construction of certification paths.

2. OCSP OID: Pointer to a OCSP responder the CA is running.

The profile states that the extension has to be non-critical.

Subject Information Access  Similar to the authority information access extension this extension describes information and services of the certificate subject. Two options are defined:

1. timestamping OID: Indicates that the end entity is a Time Stamp Authority (TSA).

2. CA repository OID: Indicates that the end entity is a CA and publishes its certificates and CRLs. to a repository

Subject Directory Attributes  Associates additional attributes to the certificate’s subject. Especially access control information is conveyed. The extension’s usage is not recommended by the profile due to the fact that the lifetime of access control information is usually shorter than the information contained in a certificate. Alterations to these informations directly lead to the revocation of the whole certificate. Additionally a CA is usually not in the responsibility of managing access control informations.

3.4. Simple Public Key Infrastructure (SPKI)

A different IETF working group addressed a simpler public key infrastructure named SPKI. In 1996 SPKI merged with Simple Distributed Security Infrastructure (SDSI) [Ell] and the work on the specifications have been discontinued.

SPKI’s main goal was to simplify the complicated X.509 standard. Nevertheless the published RFC drafts [Ell99] and [CE99] almost exclusively address the concerns of a
closed authorization infrastructure. A SPKI certificate is therefore referred to as an authorization certificate.

According to [AL99] SPKI has not gained the same amount of encouragement of corporate and governmental environments as X.509 did and will presumably only occupy a niche market in the future.

3.5. Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP) is an alternative technology implementing a public key infrastructure. PGP provides confidentiality and authentication services for email messages and file storage applications, while utilizing symmetric and public key cryptography. It was originally introduced by Phil Zimmerman in 1991 [Zim95]. The successive version of PGP has been published by the IETF in RFC 1991 [DA96], while the most current version of PGP is referred to as openPGP and standardized in RFC 2440 [JC98].

3.5.1. Web of trust

PGP is capable of X.509 certificates and PGP certificates. The latter consist of a version number, the holder’s public key, the holder’s identity information, the holder’s signature over the certificate, the validity period and the preferred symmetric encryption algorithm [Pgp]. PGP certificates are self-signed certificates. A PGP certificate is able to contain multiple signatures of other participants therewith expressing their trust in the stated certificate information.

The trust model used in PGP is a user-centric trust model. Every user is directly responsible for decisions concerning his trust in unknown public keys. On the one hand every user acts himself as a certification authority by certifying public keys he trusts and on the other hand lets his keys be certified by other participating users in the web of trust.

Certification authorities are build upon so called trust signatures [JC98, sec. 5.2.3.12]. A certificate’s signer does not only assert the certificate’s validity but also its trustworthiness at a specific level. The signer is able to appraise the trust he has in the public key certificate on a specific scale. If a signer trusts a certificate for example on the level 1, the signer will also and only trust all certificates directly signed by the former public key. The higher the level, the more adjacent signed certificates are trusted by the original signer.

The trust in this model derives from active and highly networked participants who have a broad understanding of public key technology and its consequences in a web of trust. Thus the user-centric trust model is not necessarily suitable for all application areas. Additionally corporate environments often require a centrally administered trust policy which is certainly not realizable with a user-centric trust model.
3.5.2. Key distribution and revocation

In consequence of the absence of a central repository for certificates, public key servers were introduced. Key servers solely provide the means for distributing public keys to users. If a private key gets lost or compromised there is no method of deleting the corresponding public key from public servers, instead a so called revocation signature signed by the private key could be uploaded to a key server, which marks this key as revoked. The common term revocation nevertheless usually refers to the revocation of a public key’s certificate, the offered PGP revocation mechanism therefore just assigns a revoked flag to the public key.

3.6. XMLDSig

A XML digital signature (XMLDSig) provides a digital signature in XML syntax over arbitrary data, typically XML documents. XML signatures apply to any resource which is reference-able by an URI\textsuperscript{14} [TBL98]. Even parts of XML documents can explicitly be signed. Furthermore data which is derived from an URI through certain transform algorithms can be signed by XML signatures. XMLDSig is standardized by a collaboration of the IETF and the W3C\textsuperscript{15} in the W3C Recommendation XML-Signature Syntax and Processing [MB02]. The security goals are integrity, message/ signer authentication and non-repudiation.

XMLDSig just provides the means for establishing an association between the signed data and an entity [Dou02]. Consequently trust semantics like certificate path validation lie outside the scope of XMLDSig and are established through the XML Key Management Specification (XKMS) [WF01].

Enveloped and enveloping signatures apply to data within the same XML document. In the enveloped case the signature is a child element of the referenced subtree within the XML document. In the enveloping case the signature is the parent element of the signed data. The third case is referred to as detached signature, it applies to data residing within a different subtree in the same XML document or to data at an external location referenced by an URI.

3.6.1. Structure

\begin{verbatim}
1  <Signature ID?>
2  <SignedInfo>
3    <CanonicalizationMethod/>
4    <SignatureMethod/>
5    (<Reference URI?>
6       (<Transforms>)?)
\end{verbatim}

\textsuperscript{14}Uniform Resource Identifier\textsuperscript{15}World Wide Web Consortium
3.6 XMLDSig

The following listing provides an overview over important XMLDSig elements and attributes. For a complete specification refer to the W3C Recommendation *XML-Signature Syntax and Processing* [MB02] and the underlying XML Schema [Xmla].

- **SignedInfo** Provides the information that is actually signed.

- **CanonicalizationMethod** Indicates by an URI which canonicalization algorithm is used. *Canonical XML Version 1.0* [Boy01] is the default algorithm for canonicalization in XMLDSig, but any adequate algorithm could be used. Canonical XML is defined as:

  "A method for generating a physical representation of an XML document that accounts for permissible changes that preserve logical equivalence." [Dou02]

  A simple example would be the equivalence of two XML documents just differing in the amount of dispensable white-spaces which could be transformed in equivalent physical representations by stripping these white-spaces.

- **SignatureMethod** Simply indicates the signature method used by an URI.

- **Reference** The Reference element may occur one or more times, while providing a digest method, a digest value and a transform algorithms to apply before digesting the data. Possible transform operations are canonicalization, encoding/decoding, XSLT, XPath, XML schema validation, or XInclude. The URI attribute of the Reference element references data objects, which are dereferenced by an application either into an octet stream or a XPath node-set. There exist the following exemplary options: URI="http://example.com/bar.xml", indicating a normal external resource, URI="http://example.com/bar.xml#chapter1" identifying a node-set with an element identifier at an external location, URI="" identifying the node-set of the signature-containing document and URI="#chapter1" identifying a node-set within the signature-containing document.

- **SignatureValue** Contains the actual signature value encoded in a base64 format.

- **KeyInfo** The optional KeyInfo element provides information about the key to validate the signature. It is able to carry the validation key itself, a name,
a certificate or other information associated to public key management. The predefined types like X509Data, PGPData or SPKIData may nevertheless be extended with application specific types.

- **Object** The optional object element may contain miscellaneous properties extending the information provided by XMLDSig. It acts therefore as a generic container and is related to the X.509 extension field. Examples for this arbitrary content are timestamps, or further structural manifest information (see [Dou02, p. 141]).

### 3.6.2. Signature generation

The following steps have to be taken to generate the signature:

**Reference generation**

1. After dereferencing the data, apply the stated transform operations to the data.
2. Calculate the hash value of the transformed data.
3. Create the according Reference element.

**Signature generation**

1. Create the SignedInfo element by adding signatureMethod, canonicalizationMethod and one or more References.
2. Canonicalize the data in SignedInfo and calculate the SignatureValue afterwards over it, based on the algorithms specified in SignedInfo.
3. Build the Signature element including the optional keyInfo and Object element(s).

### 3.6.3. Signature validation

The following steps have to be taken to perform the signature’s validation:

**Reference validation**

1. Use the CanonicalizationMethod from SignedInfo to canonicalize the SignedInfo element.
2. For all references in SignedInfo:
   a) Retrieve the data object which will be hashed. Dereferencing the URI and applying certain transform-algorithms are usually performed in this step.
   b) Use the in its reference specified DigestMethod to hash the retrieved data object.
c) Compare the resulting hash value against the DigestValue contained in the SignedInfo Reference. The validation fails if there is no match.

**Signature validation**

1. Retrieve the keying information from KeyInfo or from an external resource.

2. Retrieve the canonical form of the SignatureMethod using the Canonicalization-Method and use the result to verify the SignatureValue over the SignedInfo element.

### 3.7. Merkle Hash Tree

A Merkle tree is a complete binary tree where all leaves have an associated value and all ancestor nodes carry the value of a one-way hash function of the value of the node’s children. Merkle originally introduced this tree for producing multiple one-time signatures associated to a single public key. [Mer82]

Merkle trees find a wide range of applications due to their simplicity and versatility. Especially Distributed Hash Tables (DHTs) and their use in peer-to-peer systems are a widespread application.

![Exemplary Merkle Hash Tree with binary node identifiers.](image_url)

Due to the fact that Merkle trees require a relatively large amount of computational costs and storage, different suggestions exist for the optimization of the tree traversal.
A main contribution was made by Michael Szydlo in [Szy03] and [Jak03], who introduced a traversal algorithm which computes sequential tree leaves and authentication path data in time $2\log_2(N)$ and space less than $3\log_2(N)$. The following formalisms are based on Szydlo's suggestions.

The traversal of a complete binary tree $T$ with the height $H$ is facilitated by labeling each left child node $n_{\text{left}}$ with a “0” and each right node $n_{\text{right}}$ with a “1”. Each node in the tree now holds a binary identifier. Interpreting the identifier as a binary number and converting them into the decimal system, the leaves are naturally ordered in the range $\{0, 1, \ldots, 2^H - 1\}$.

Let

$$\Phi(n_{\text{parent}}) = \text{hash}(n_{\text{left}}||n_{\text{right}})$$

be the function that computes the value of the parent node from the values of the node's children. $\text{hash}$ in this case is an arbitrary one-way-hash-function such as SHA-1 [Sha]. The values of interior nodes are usually calculated by the same hash function used on the so called pre-images to fill the leaf-values.

An authentication path $\text{Auth}_h$, for each $h < H$ is a path from a leaf to the root including all necessary siblings on the way that are needed to calculate the next parent node. The authentication data is then the set $\{\text{Auth}_i \mid 0 \leq i \leq H\}$.

In common application areas like P2P-networks the user receives the root hash through a protected channel and is now able, together with the authentication data, to prove that a chosen pre-image that could be received through an unprotected channel was in the tree during the hash tree's generation.

Obviously a re-ordering of intermediary nodes in the tree after the order of the leaves is fixed, destroys the hash tree's semantic. Binary search trees and their standard traversal methods like pre-order, in-order, post-order or level-order are therefore of lower priority for hash trees. The focus lies here on the authentication path/data and the traversal performance according to time and space.

The standard algorithm for computing an authentication path of a given leaf is commonly named TREEHASH (listed in Algorithm 1), which will be used later as a basis for the algorithm of the prototype-implementation (see section 5 on page 63).

A hash tree solves the problem of omitting signed content-fragments and thereby invalidating the overall signature. Building a hash-tree with all content-fragments as leaf pre-images and afterwards signing only the tree's root-hash, enables the omitting of certain fragments without invalidating the signature. If the signature structure still
Algorithm 1 TREEHASH (start, maxheight), Source: [Szy03].

1. Set leaf = start and create empty stack

2. Consolidate if top 2 nodes on the stack are equal height:
   - Pop node value $\phi_{\text{right}}$ from stack.
   - Pop node value $\phi_{\text{left}}$ from stack.
   - Compute $\phi_{\text{parent}} = \text{hash}(\phi_{\text{left}} || \phi_{\text{right}})$.
   - If height of $\phi_{\text{parent}} = \text{maxheight}$, output $\phi_{\text{parent}}$ and stop.
   - Push $\phi_{\text{parent}}$ onto the stack

3. New leaf otherwise:
   - compute $\phi_l = \text{hash}(\text{leaf})$.
   - push $\phi_l$ onto stack.
   - increment leaf

4. Loop to step 2.

provides the substituting hash values of the omitted fragments, a verifying application is still able to build the authentication path and verify the signature of the root-hash.

3.8. Signing RDF

Signing RDF Graphs is a related topic of this thesis and is therefore shortly considered in this section.

There are generally two methods for implementing a signature for RDF Graphs. The first method involves the direct signing of the RDF Graph's document representation while the second method involves a signature over the graph representation of a RDF Graph. The first method does widely differ from other common signature processes involving a document representation of arbitrary data. An example for such a signature in XMLDSig format using a PGP key and a RDF document as input, conveys the following listing 3.2 on page 37.
Another option to implement the first method is to reference a digital signature with a RDF document using the Web Of Trust RDF Ontology (wot) [Wot].

The second method is described by J. J. Carroll in [Car03]. His solution explicitly rules out the method of signing the file-representation of a RDF Graph. He defines a class of canonicalizable RDF Graphs in which a digital signature for the graphs may be created and verified. He restricts the set of RDF Graphs to be able to solve the signature-creation and -verification problem in polynomial time. A full solution for RDF canonicalization in polynomial time is unlikely because of the implicated resulting solution to the graph isomorphism problem [For96]. He uses an algorithm which makes slight syntactic modifications to a RDF Graph without changing its meaning before signing it. The assumption is therefore that the signature only signs the semantic of the graph and not the graph itself.

3.9. Assessment

X.509 as a public key certificate format is a mature specification which provides comprehensive certificate management functionalities including certificate distribution and revocation mechanisms. Through its powerful extension mechanism it is possible to add certain additional information like the intended key usage for the corresponding private key. Virtually all currently existing browsers are capable to handle SSL-secured connections. Therefore many root-CA certificates are distributed through the browser’s certificate store, facilitating therewith the certificate verification process. Utilizing the already existing PKI-functionalities in common browsers while handling signatures and their certificates in the web-environment explored by browsers, seems to be an obvious solution.

SPKI’s focus essentially lies on certificates conveying authorization information associated to a public key holder. The fact that the SPKI RFCs never left the official status Experimental and a broad distribution of this certificate format in the near future is highly unlikely, disqualifies SPKI as applied certificate format.

PGP in principle has a promising implementation of fine grained trust relationships through its web of trust. It nevertheless lacks of a mature certificate revocation mechanism which is essential for achieving the goal of this thesis. Circumventing this drawback
may lead to good alternative signature or certificate format respectively.

Although X.509 has some advantages over other current certificate formats, the following signature specification should leave the actual possibilities for a format open, to retain its extendibility. The prototype described in section 5 on page 63 will be based upon X.509.
3. Overview of cryptographic methods
Part II.

Consolidation
4. Cryptographic annotation: a digital signature as microformat

This chapter shows on the one hand how microformats may be utilized to implement a content-signature and on the other hand how classic PKI-technology may be re-used to help signing the content and to implement a content-revocation-mechanism. The first sections will cover the specification of a newly introduced signature-microformat, while the later sections focus on content revocation and content certificates.

4.1. Introduction

To be able to extract signed content which is annotated with microformats, the signature itself has to be placed within the published content. Placing the signature inside the document requires in turn a mechanism to identify and extract the relevant signature-concerning informations. Since the signed content is already annotated with microformats it suggests itself to annotate the signature itself as a microformat as well. Thereby existing microformat parsers may be later utilized to extract all signature-related information.

Many publishers alter the owner's content by omitting parts of it before they publish the content. The here introduced signature is capable of managing this problem by utilizing a Merkle hash tree as introduced in section 3.7 on page 35. The owner is able to convert his intention into a policy that instructs an arbitrary publisher which parts of the content he is able to omit and which not. The publisher is certainly able to publish the content in a way not conforming to the owner's policy. This breach of the rules will then however result in an invalid signature. The policy thus is the basis for building the hash tree and the resulting signature of the root hash. Besides providing authenticity a classical signature safeguards the integrity of data. If just one bit of the signed data gets altered, the signature can not be verified anymore. This standard behavior becomes an issue in an area where omitting parts of content before publishing is everyday life. The solution for this problem is a hash tree structure, where omitted parts of the content can be replaced by hashes of the original parts. The calculation of the root hash, or the result of a calculation of an authentication path of a leaf of the tree respectively, has the identical result whether the content part itself or the substituting hash value is present.

Signing the root hash of a hash tree implicates therefore a great flexibility for the
publisher when it comes to altering content before publishing. Furthermore the owner saves costs since he just needs to sign his content once and is not in the need to sign all different possible variations of his content.

4.2. Referencing microformat properties

The hash tree is build upon the microformat properties mentioned in the policy expression. There are mainly three possible cases to reference these properties in a document conforming to the use of XML Signatures [MB02]: Enveloped, enveloping, and detached signatures.

The detached case requires an URI¹ [TBL98] for referencing the location of the corresponding content. Within a (X)HTML document the element identifier [DR99a] is responsible for referencing a fragment of the content by providing the id-attribute. If the URI or the element identifier changes after the content has been signed, the dereferencing and therefore the signature verification will fail. This circumstance is a major drawback of detached signatures.

Presuming that the URI and the element identifier is not changed within a document after the signature is generated, element identifiers seem to be a promising solution for identifying content fragments and therefore referencing microformat properties. These identifiers are defined as unique according to the document but not globally. In a republishing environment it is however a requirement to have globally unique identifiers to safeguard that signed content, originating from different sources, can be published within the same document. A duplicated element identifier causes on the one hand a failure of the (X)HTML validation and on the other hand the signature verification is not able to uniquely identify the signed data and would also fail. This implicates that document wide unique identifiers are insufficient means for the policy expression to reference microformat properties.

An improved solution are element identifiers that are simultaneously an UUID². UUIDs could prevent the collision of unintentionally identical identifiers. Regardless of the UUID, a malicious publisher or content owner is able to publish content with identical identifiers to provoke a denial of service of the verification mechanism.

To face these issues this specification constrains the possible cases of signature embedding in a document to enveloped and enveloping. Only these two guarantee the unique identification of the corresponding content while still allowing a publisher to omit arbitrary content fragments without restrictions and the placement of multiple signed microformats within one document. While using enveloped and enveloping signatures the referencing just requires the class identifiers microformats are using.

¹Uniform Resource Identifier
²Universally Unique IDentifier
4.2.1. Enveloped

An enveloped signature is a format where the referenced content directly encloses the signature. Referencing the microformat specified in the policy in this case is quite simple as a consequence of the hierarchical structure of (X)HTML documents. Starting at the location of the signature's identifier in the DOM³-tree, an application just has to traverse the DOM-tree up-wards to identify the properties by using always their first occurrences in the DOM-tree. If a property occurs more than once, the application certainly may use an arbitrary occurrence of the property. For efficiency-reasons only the first found occurrence will be considered. The search algorithm follows an upwards directed traversal in the DOM-tree. For each referenced property at first all the signature's ancestor nodes are inspected until the property's microformat root-identifier is found. Beginning with the microformat root-node its whole subtree is afterwards searched for the referenced property. These two steps are performed until all properties are found.

An enveloped signature is just able to sign microformat properties which reside in the subtree of the DOM where simultaneously the signature resides. All properties that reside in a different subtree of the DOM are out of the signature's scope. Microformats which are referenced by means of the include-pattern (introduced in section 2.3 on page 11), likewise reside in another subtree of the DOM.

Nested microformats, where microformats contain other microformats, are nevertheless still possible as long as the owner and the publisher consider that only the first occurrences of the properties during traversal are taken into account.

4.2.2. Enveloping

An enveloping signature encloses the referenced data. Referencing the microformat properties here works nearly the same way as in the case of an enveloped signature. The only difference is that the traversal of the DOM-tree happens down-wards until the first occurrences of the specified properties.

All microformat properties residing beneath the signature in the DOM-tree can be referenced. As in the enveloped case, only first occurrences of the specified properties will be regarded.

4.3. Hsig format specification

According to the naming-principles of the microformat community [mic07c] a new microformat should have the letter h as a prefix for the class-name of the microformat root-element. The newly introduced microformat specification therefore carries the name hsig. It is defined as a compound microformat which reuses the key-property of the

³Document Object Model
Cryptographic annotation

hcard-specification and is designed to be embedded and re-used by other microformats.

This specification is not a microformat in the common sense. It has not yet been approved by the microformat-community according to the process [mic07b] for new microformats. For simplification this specification is nevertheless called a microformat by name in the remainder of the thesis.

To keep the flexibility of the specification it is designed in a way that every possible past and future algorithms can be used to accomplish the goal of generating a signature over a hash value. The same certainly applies for the type of algorithm used to calculate the hash values.

With regard to the content revocation and the re-use of the certificate revocation mechanism introduced in section 4.6 on page 57, the use of X.509 compliant algorithms as listed in [WP02] seem to be appropriate. For this reason the following parts reduce the spectrum of possibilities to the mentioned X.509 compliant algorithms, and to keep it even more simple only RSA\(^4\) and SHA-1 are used in the following examples.

It is certainly possible to use PGP or any other system for digital signatures which is based on public-key cryptography.

The hsig specification basically consists of four properties: policy, key, cert and auth-data. Listing 4.1 on page 46 shows the enveloped case of a hsig microformat with some left out or shortened property-values for better readability and listing 4.2 on page 47 the same hsig in the enveloping case.

---

\(^4\) Algorithm for public key cryptography, invented by Rivest, Shamir and Adleman
4.3 Hsig format specification

4.3.1 Policy property

The policy property carries the owner's processing policy for the publisher. This policy expresses a processing rule in propositional logic which controls in which manner a publisher is allowed to alter the content before publishing. This policy and the underlying language is discussed in more detail in section 4.4 on page 51.

In the following exemplary policy representation in listing 4.3 on page 48 the policy is carried by the (X)HTML-Tag `<abbr>`, although the property could be provided by every type of container as introduced in section 2.3 on page 11. `<abbr>` indicates
an abbreviated form of an expression, while the title-attribute is used to provide the expanded form of the expression\(^5\). The actual value of the policy in this example is therefore provided by the title-attribute.

```
<abbr class="policy"
    title="(vcard:fn AND vcard:email) OR (vcard:fn AND vcard:org)">
    vcard:fn AND (vcard:email OR vcard:org)
</abbr>
```

Listing 4.3: Exemplary representation of a policy property with example policy expression.

The policy is a mandatory property and has exactly one occurrence in hsig.

To prevent forgery of the policy it is added to the hash tree as a leaf pre-image and is therefore signed.

4.3.2. Key property

The key property contains a certificate of the owner's public key. The certificate is used to authenticate the content's owner.

The representing container for the key requires an additional indicator for the applied type of certificate. The key in listing 4.4 on page 48 is exemplary provided by the (X)HTML-tag `<div>` which is a surrounding container for two `<span>`-tags carrying the type and value of the certificate. The fragmentation in type and value is quite usual to microformat properties which have a subtype.

```
<div class="key">
    <span class="type">x509</span>
    <span class="value">−−−−−BEGIN CERTIFICATE−−−−−
        MIID8GCCAqAwIBAgIGCgIBAwIBAJKoZIhvCNA [...]
        yITcxqDf
        −−−−−END CERTIFICATE−−−−−</span>
</div>
```

Listing 4.4: Key property with example X.509 certificate in Base64 PEM-format.

The key is a mandatory property and has exactly one occurrence in hsig.

**Note:** Every owner who is willing to generate content signatures has to be in the possession of a common public key certificate. Marking this property as mandatory therefore implies no additional constraints to the owner.

For the reason of extensibility the options for type are not limited to a specific string. The example uses a X.509 certificate because the X.509 v3 certificate [RH02] is the most widespread format for a certificate and provides the necessary characteristics. For these reasons X.509 is a recommended format to use. The key property is already defined

\(^5\)see http://www.w3.org/TR/html4/struct/text.html#edef-ABB
in the hcard specification [mic07a] and is therefore an example of the re-use pattern of microformats which is also applied in this specification.

The format of the value property is a base64 encoded PEM⁶ representation of the certificate.

### 4.3.3. Cert property

The cert property is open to any certificate format but again the recommendation for a X.509 format holds. In contrast to traditional X.509 certificates like the one in the key property, this certificate contains among other data a root hash of a hash tree and the owner’s public key which altogether is signed with the private key of the owner and thus results in a self signed certificate for the content. The differences and details of the hereby introduced content certificate are discussed in section 4.7 on page 58.

The container for cert possesses exactly the same characteristics as the container for the key (see section 4.3.2).

```
1 <div class="cert">
2   <span class="type">x509</span>
3   <span class="value">−−−−−BEGIN CERTIFICATE−−−−−</span>
4   MID8jCCA1wInABgAgCARwDQYwNADIAKoZih [...].y1xgDTcf
5   −−−−−END CERTIFICATE−−−−−</span>
6 </div>
```

Listing 4.5: Cert property with example certificate in Base64 PEM-format.

### 4.3.4. Authdata property

The authdata property provides the substituting hash values of each by the publisher omitted property. These hash values are later used for building the hash tree. Authdata is an abbreviation for authentication data which is defined in section 3.7 on page 35.

For each omitted property there exist one authdata property carrying two values, the binary identifier of the node in the hash tree and the actual hash value of that node. An exemplary container for authdata is listed in listing 4.6 on page 49, where the (X)HTML-tags `<div>` and `<span>` are utilized as a container. The `value` holds the hash value of the original property and the `type` holds the binary identifier of the node in the hash tree.

```
1 <div class="authdata">
2   <span class="type">010</span>
3   <span class="value">9e464fe4553c88c78be46a9cf7ea5e4d8df93d26</span>
4 </div>
```

⁶Privacy Enhanced Mail
Listing 4.6: Example authdata properties with SHA-1 hash of the omitted value in hexadecimal format.

The appearance of the authdata property in hsig depends on the fact whether or not the publisher omitted parts of the content. For each omitted part, which is mentioned in the policy expression, it is mandatory for the publisher to provide a substituting hash value. Without this substituting hash it would be unfeasible to calculate the authentication path for each mentioned leaf or the authentication data for the whole hash tree respectively.

Providing the node’s binary identifier in the hash tree has the advantage of contingently economizing the amount of authdata properties in the hsig, by potentially providing the ancestor node’s value of two or more omitted properties. More details to this economization can be found in section 4.5.1 on page 55.

The format of the SHA-1 hash value in the example is a hexadecimal 40 digit representation of the 160bit hash value. This format can vary depending on the type of the algorithm used.

4.3.5. Hsig validation

Validating a microformat is quite complex as already mentioned in section 2.4 on page 14. Validating the hsig microformat is even harder due to the additional tasks to perform, including policy evaluation, hash tree assembly and the signature verification. Nevertheless a Schematron schema [Sch] may be defined to validate the syntactical correctness of the hsig structure. Schematron issues a report about the in the schema stated assertions and may therefore be used as an indicator for syntactical correctness. Publishers may use the following schema in listing 4.7 on page 50 for simple validation purposes. All further validation tasks are performed by the implementing application.
4.4 Policy

The policy is a rule for processing the signed content. Publishers shall use this rule as a basis for the content processing. The rule defines which properties of the signed content can be omitted and which not in case of publishing the content. The owner has thereby the ability to impose a rule which reflects his intention for processing.

The syntax of the rule is a simple expression in propositional logic as e.g. introduced in [CL00].

The here introduced policy are words formed with the following alphabet:

\[ A = M \cup \{ NOT, AND, OR \} \cup \{ \}, () \]

While \( M \) is the finite set of all possible atomar microformat properties which act as propositional variables in the common-sense:

\[ M = \{ vcard : fn, vcard : tel, vcard : email, hreview : summary, \ldots \} \]

Each atomar property is prefixed by its microformat root-identifier and a ':'. \( M \) does not include parent microformat properties which themselves have no actual value, but
are a set of other atomic or parent properties, e.g. adr of hcard.

It is assumed that the set \( \{ NOT, AND, OR \} \) has no intersection with \( M \) and all elements of \( M \) are strings of lower case.

The set \( \{ NOT, AND, OR \} \) is equivalent to the set \( \{ \neg, \wedge, \vee \} \) and is hereby a set of functionally complete logical operators. This set of operators is a good trade-off between human readability of a policy expression and the most minimal functionally complete set \( \{ \| \} \) or \( \{ NAND \} \) respectively.

If the parser evaluates the expression to an outcome of the logical value \( true \), all microformat properties are specified in the document as the owner intended. An empty expression evaluates to \( false \).

The exemplary propositional expression \( a \ AND (b \ OR c) \) implicates that property \( a \) can not be omitted by the publisher, instead he can choose to omit \( b \) or \( c \). Omitting \( b \) and \( c \) together violates the policy.

To reduce the complexity of expressions and to avoid semantically equivalent but syntactically different expressions, the assumption is to note the expression in disjunctive normal form (DNF). A DNF has the additional advantage over a conjunctive normal form (CNF) of being easier evaluated to \( true \), especially if the expression is getting longer.

A grammar for these expressions in infix notation and in DNF is the following listing in extended Backus-Naur form (EBNF):

\[
exp ::= '(' , exp , ')' | exp , ',' , binaryOp , ',' , exp |
unaryOp , '(' , exp , ')' | lit ;
binaryOp ::= 'AND' | 'OR' ;
unaryOp ::= 'NOT' ;
lit ::= char , { char } [ { '-' , char , { char } } ] ;
char ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' ;
\]

The example policy expression above should therefore be transformed into the following notation in DNF:

\((a \ AND b) \ OR (a \ AND c)\)

In general microformats have at least one property which is mandatory. Identifying these mandatory properties especially within a compound microformat is quite demanding.
The microformat community has not yet published a processable and well formed definition of these mandatory properties. Brian Suda has published a so called cheatsheet [Sud07] that lists all properties of existing microformat specifications and associates a quantifier for each of the properties analogical to the quantifiers of regular expressions. The policy expression and the signature generation and verification process is independent of the validity of addressed microformats. Nevertheless it is recommended that an owner does not specify a policy which violates the validity by excluding (negating) a mandatory property. Such a policy would force every publisher of the content to publish invalid microformats in order to retain a valid signature.

4.5. Hash tree

The underlying data structure for the hash tree has important implications on the performance and the needed amount of storage space while calculating the authentication data. Besides the binary tree introduced by Merkle, one can think of a n-ary tree. The latter could be build upon given microformat structures. An example is the adr property of the hcard specification. Adr is a parent property which itself carries no explicit value but is a set of atomar properties. Adr semantically provides a structuring of address-data. The probability that a publisher wants to omit semantically close properties like the ones beneath adr seems quite large. A n-ary tree would take advantage out of this form of omitting values because of the decreased amount of substituted hash values which have to be provided.

The example in figure 4.1 on page 54 clarifies the approach of a n-ary hash tree. The node's are equipped with a decimal identifier in level-order.

Knowledge of the structure of all present and future microformats is a requirement for the n-ary data structure. Otherwise the application is not able to implicitly build the hash tree out of the policy expression. Using the given hierarchical structure of microformats as a basis for building the hash tree is not an option either. The problem arises if the policy just references an excerpt of the whole microformat structure. A publisher could therefore omit certain properties which are mandatory for building the hash tree but not mandatory for satisfying the policy.

A possibility to deal with this issue is to transport meta information in hsig to enable the building of the hash tree in the validating client prototype. However this would indicate the need for an unnecessary additional property in the specification and therefore violates the microformat principles. The signing part of the prototype would require a-priori knowledge of all microformats. This implicates an update mechanism to stay actual and constrains the autonomy of the prototype.

The data structure I have chosen is an almost complete left-balanced binary tree. This kind of structure has several advantages.

First of all Merkle originally introduced a binary tree as the underlying data structure for the Merkle hash tree. The optimization of the tree traversal algorithm for calculat-
ing the authentication path of a leaf in a Merkle hash tree has already been the focus of several publications (see [PDS01], [Jak03] and [Szy03]) and can be utilized in this approach. These optimizations significantly improve the response-time of the prototype implementation introduced in section 5 on page 63.

4.5.1. Tree assembly

The assembly of the hash tree must be regarded from three different perspectives. The content's owner while initially signing his content, the publisher while processing the content and the user while verifying the content. All three perspectives have the building of the basic hash tree structure in common, which has actually no assigned node values at all.
The hash tree is generated out of the policy expression. Every atomar microformat property which is specified in the expression is only considered once as a leaf pre-image. Also a negated property must be included.

First a binary tree structure is constructed, where all nodes have no associated value. The number of leafs is identical to the cardinality of the set of properties specified in the policy plus one. The amount of leafs in the example expression 
\((\text{vcard} : \text{fn} \ \text{AND} \ \text{vcard} : \text{email}) \ \text{OR} \ (\text{vcard} : \text{fn} \ \text{AND} \ \text{vcard} : \text{org})\) is therefore \(3 + 1 = 4\). The incrementation of one belongs to the addition of the policy property, that has to be secured as well.

All leafs are equipped with a binary identifier as explained in section 3.7 on page 35 and hereby receive an ascending order from left to right.

**Owner perspective**

After a simple whitespace normalization, all hash values of the leaf pre-images or microformat properties respectively are calculated. The type of applied hash function is freely selectable by the owner. Theses hashes are then sorted via a lexicographical relation and assigned to the prepared leafs from left to right.

The example expression 
\((\text{vcard} : \text{fn} \ \text{AND} \ \text{vcard} : \text{email}) \ \text{OR} \ (\text{vcard} : \text{fn} \ \text{AND} \ \text{vcard} : \text{org})\) has the leafs \(\text{fn}, \ \text{email}, \ \text{org}\) and \(\text{policy}\) and results in the hash tree with example property values in figure 4.2 on page 56.

**Publisher perspective**

If a publisher wants to alter the hsig-annotated content he received from the owner or another source, he has to build the hash tree similarly as the owner. If the publisher decides to publish the received content as is, there is certainly no need for action.

If the publisher omits properties he has the option to check whether the omitted properties' hash values form a sub-tree within the previously build hash tree. If the latter applies, the publisher can choose whether to omit the whole sub-tree except the hash value of the sub-tree's root node or to keep the approach of providing a hash value for each omitted atomar property. Omitting the sub-tree in such a case is recommended to reduce the computational costs concerning time and space at his own and the user's perspective.

Omitting a sub-tree and providing the hash value of its root node would result in a similar hash tree as in the following example figure 4.3 on page 56.
4. Cryptographic annotation

User perspective

The user’s perspective requires to assemble the hash tree in respect for the policy and the contingently present authdata values. This task is quite straightforward if the omitted properties are just atomar. The user or the verifying application respectively just has
4.6 Content revocation

In classical PKI a user usually signs all of his content with a designated private key. Signing emails with a key certified by a S/MIME-certificate is a common example for this scenario. An overview about the revocation-scheme in classical PKI with certificates is shown in figure 4.4 on page 58. A revocation of the corresponding certificate leads immediately to a revocation of all until then issued signatures and their content. Depending on the revocation-reason this outcome might be intended. Concerning the revocation of content to express the not anymore existing consent to its publishing, each content has to be able to be revoked on its own.

Utilizing a certificate which itself is nothing more than a signature over certain data, enables the issuing of “revocable signatures” for each published content. Additionally there is no need to generate a new public/private key-pair for each content. This here introduced and so called content certificate is pictured in more details in section 4.7 on page 58.

The revocation-scheme of content certificates is slightly different to classical revocation as depicted in figure 4.5 on page 59. The so called content CA is here responsible for managing all owner specific content certificates including a repository and an archive for these certificates. Providing a OCSP-Responder is recommended for being able to provide current revocation information for the certificates. If a user hosts his own content
4. Cryptographic annotation

Figure 4.4.: Certificate revocation in a classical PKI. Revoking the certificate implicates the revocation of all until then published signatures.

CA as demonstrated in the figure, or if he uses a trusted third party for this service, is secondary. The important point is that the owner is in the possession of his content certificates and the possibility of their revocation.

4.7. Content certificate

An ordinary X.509 certificate as introduced in section 3.3 on page 24 basically signs a defined set of data-fields including a subject entity, the corresponding public key of the subject and a validity period with a private key of a trusted third party as a CA. This concept is adopted here for a signature of the content's hash tree or the root hash value respectively. For a better differentiation between the ordinary X.509 v3 certificate and the here introduced certificate, the latter is entitled content certificate for the remainder of the thesis. The syntactical structure of the content certificate is nevertheless identical to a X.509 certificate while the semantical interpretation differs in some points.

The major difference is located in the subject's distinguished name (DN) field. The DN usually defines a unique path through a X.500 Directory Information Tree (DIT). The common name (CN) of the subject is a leaf node in the DIT hierarchy and usually contains the subject's for- and surname. In the content certificate this attribute contains
the root hash of the hash tree generated by means of the policy expression.

“There is nothing in any of these standards that would prevent me from including a 1 gigabit MPEG movie of me playing with my cat as one of the RDN components of the DN in my certificate.”
– Bob Jueneman on IETF-PKIX [Mpe]

This use of the CN attribute does not violate the X.509 standard. Actually there is no exact definition of the format for the DN. Nevertheless this alternative use of the CN as a container for a hash value does not conform to the hidden agenda of the X.509 recommendation to use a DN according to the X.500 directory service. Since the compatibility to X.500 or the related Lightweight Directory Access Protocol (LDAP) are of minor priority to achieve the goal of this thesis, this issue is disregarded. Common implementations of the X.509 standard like OpenSSL [Ope], allow arbitrary data to be included in the CN. The format of the CN is only restricted to a string of 64Byte length. The recommended hash algorithm of the hsig specification is SHA-1, which produces a hash of 160Bit length or 40Byte length in a hexadecimal representation. Furthermore, all one-way hash functions defined in [WP02] would have enough storage space in the CN field of the DN.
The DN’s remaining fields may be filled with arbitrary metadata at the owner’s choice. For compatibility reasons at least the fields country, location and organisation should be filled. If the owner intends to establish a form of hierarchy among all fields of the DN, he certainly may begin with the usual DN hierarchy one level above the CN. The owner’s CN would in this case be located in the Organisational Unit (OU) field etc. A content hash value is then logically sorted below the owner.

The subjectPublicKeyInfo field holds the public key of the content’s owner instead of the subject, as the content does not have a key-pair.

The signatureValue provides the signature generated by means of the private key of the owner and the stated signature algorithm. The content certificate is therefore a self signed certificate for the content’s root hash value, issued by the owner.

Table 4.1 gives an overview of the differences between the field values of an ordinary X.509 certificate and a content certificate.

<table>
<thead>
<tr>
<th>X.509 v3 FIELD</th>
<th>CONTENT CERTIFICATE VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>ordinary use</td>
</tr>
<tr>
<td>serialNumber</td>
<td>ordinary use</td>
</tr>
<tr>
<td>signature</td>
<td>ordinary use</td>
</tr>
<tr>
<td>issuer</td>
<td>ordinary use</td>
</tr>
<tr>
<td>validity</td>
<td>ordinary use</td>
</tr>
<tr>
<td>subject</td>
<td>CN=root hash, OU=arbitrary metadata, O=arbitrary metadata, L=arbitrary metadata, C=arbitrary metadata, ...</td>
</tr>
<tr>
<td>subjectPublicKeyInfo</td>
<td>public key of the owner</td>
</tr>
<tr>
<td>issuerUniqueID</td>
<td>ordinary use</td>
</tr>
<tr>
<td>subjectUniqueID</td>
<td>ordinary use</td>
</tr>
<tr>
<td>signatureAlgorithm</td>
<td>ordinary use</td>
</tr>
<tr>
<td>signatureValue</td>
<td>signature resulting from the owner’s private key and the signatureAlgorithm</td>
</tr>
</tbody>
</table>

Table 4.1.: Overview of differences between ordinary X.509 v3 field values and the values of a content certificate
Part III.
Implementation
5. Prototype implementation

This chapter provides an overview about the implemented signature prototype. After a short introduction, the core technologies used and the rationale to adopt them will be described. Afterwards the implementation-details will be contemplated followed by the occurred problems while implementing. The chapter ends with a section covering the prototype's limitations and possible further work for the prototype.

5.1. Introduction

An application for a signature mechanism for web-content as introduced in the foregoing chapters, should cover the three parties who are handling with the signature, the content's owner, the (re-)publisher and the user. The signature generation at the owner's perspective is certainly a one-time-task for each content the owner wants to sign. The publisher in case he wants to omit certain parts of the signed content has not much effort for this task, he just has to build the hash-tree once and add the appropriate authdata-values to the hsig. If the publisher wants to re-publish the content as is, he just has to pass-through the content without further modifications. The better part of the tasks lies at the user's perspective. He has to build the hash-tree, verify the signature and visualize the verification outcome again. These tasks have to be performed each time a signature is found within a document.

The introduced prototype implementation therefore focuses on an application for the user's perspective while the other perspectives are left open for further work. Nevertheless the client prototype's major functionalities may be re-used and easily adopted within the other perspectives. This possible re-use is the motivation for out-sourcing the part of the certificate-verification to a server-side component. All remaining parties who deal with certificates may re-use and extend this external component for their needs. For the prototype either the user must provide a hosting-environment or a trusted third party may later adopt the hosting for multiple users.

5.2. Technology

5.2.1. Client

As a result of the browser market-shares [Neta], currently available and used browsers may be roughly reduced to Microsoft's Internet Explorer, Mozilla Firefox and Safari. Similar to Firefox, Internet Explorer provides a mechanism to extend the browser's capabilities through its ActiveX-Technology and its so called Browser Helper Objects.
(BHOs). Safari provides no official extension-mechanism for adding additional functionality to the browser at all. The most substantial argument for the decision for Firefox besides its well documented, open and powerful extension-mechanism is its cross-platform-compatibility. Safari at the current state supports only Mac OS X and has published a public beta version for the Windows-platform. The current Internet Explorer 7 provides only exclusive support for Windows operating systems.

Because of the long-time tradition of JavaScript or ECMAScript respectively as client-side scripting-language in web-browsers, JavaScript, as the preferred language for implementing Firefox extensions, offers reliable methods for accessing and manipulating DOM-trees.

Both of the most widely used microformat-parser (Operator [Kap] and Tails [Bru]) are also implemented by means of a Firefox extension. Therefore it seems to be obvious to utilize and re-use already existing methods to extract microformats from (X)HTML-documents instead of re-implementing them. Another reason is the long-term objective to integrate the introduced hsig microformat and the corresponding signature verification process into existing and established microformat parsers. This integration will be easier achieved by using the same technology.

Mozilla Firefox

Mozilla Firefox or short Firefox is a web browser published by the Mozilla Cooperation [Mozc]. The current version is 2.0.0.11 and is available under the Mozilla Public License [Mozb] as open source. Currently Firefox has nearly 16% of the world’s market-share in Web browsers [Neta]. Firefox is one of several applications based on the Mozilla Application Framework [Moza].

Mozilla Application Framework

The Mozilla Application Framework is a framework for developing cross-platform applications. It consists mainly out of the components XUL (XML User Interface Language), Gecko, Necko and XPCOM (Cross Platform Component Object Model).

XUL is a XML-based language for defining cross-platform compatible graphical user interfaces (GUIs). Similar to DHTML¹ XUL uses standard technologies such as CSS, JavaScript, DTDs² and RDF. The rendering engine Gecko which Firefox uses, is capable of parsing XUL as well as (X)HTML and is therefore responsible for displaying the Firefox browser GUI and the rendered web content as depicted in figure 5.1 on page 65.

¹Dynamic HTML
²Document Type Definitions
Necko is Mozilla’s networking library which provides a platform-independent API for networking layers. Necko is built on XPCOM and provides implementations of common protocols such as HTTP, FTP or File i/o, a generic framework for fetching URLs, a generic cache service and DNS resolution.

XPCOM is a platform for component development similar to Microsoft’s COM³ [Com]. It provides the means to access Gecko and Necko libraries. The components may be implemented and used through different programming languages like C++, JavaScript, Java, Ruby or Python.

Firefox extension

A Firefox extension provides additional functionality for the Firefox browser. In contrast to the usual plug-in-mechanism which aids browsers in displaying advanced content like flash-animated web-pages, an extension is capable of adding GUI-elements and functionality to the whole browser-GUI. All Mozilla application use the same extension mechanism and an extension’s port to another application should be feasible, although not generally reasonable.

Extensions do not alter the existing application source-code, instead they use so called overlays to modify or add existing or new GUI-elements and their behavior. Mozilla additionally provides an integrated Add-on-manager for a simple installation- and updating-mechanism.

The main programming language used in extensions is JavaScript. It is possible to develop a XPCOM component by means of any of the already mentioned programming languages and to define its interface with the so called cross-platform Interface Definition

³Component Object Model
5. Prototype implementation

Language (XPIDL) [Xpi] to enable the inclusion and addressing from the extension with JavaScript.

**JavaScript**

JavaScript is a weakly typed, dynamic, object-based scripting-language primarily used in client-side applications. ECMAScript is the language's standardization while ECMAScript 262 3rd edition [Ecm] is the current valid standard. The term JavaScript is quite misleading, because it does not describe the standard itself but a standard implementation originating from the Mozilla Corporation. Another major implementation originates from Microsoft and is called JScript [Jsc]. JavaScript version 1.7 and JScript 5.7 implement the current ECMAScript 262 3rd edition standard. The ECMAScript 262 4th edition is under development and will add multiple features to the language such as classes, packages, namespaces and static typing.

### 5.2.2. Server component

The client's server-component should be able to process the client's request and to accomplish the verification of the extracted signature-related information. OpenSSL is providing a comprehensive cryptographic library with a command-line interface. Besides the certificate-verification including the OCSP-client- and -server-functionalities it is also possible to generate key-pairs, certificate requests and to feature the advanced functionalities of a Certificate Authority (CA). Consequently OpenSSL is a well-founded choice for a certificate or signature verifying component while still keeping in mind the possible re-use in the owner's or publisher's perspective.

For assuring the communication with the extension, providing an interface to the OpenSSL-commandline and adding additional features such as the comparison of the signed root-hash in the certificate and the calculated root-hash from the content, a supplementary layer is needed. This layer has to be capable of providing a secured connection to the extension.

Assuming that the required capabilities are accomplished, many possibilities for implementing the out-sourced component exist. A HTTPS-Server including a runtime-environment for an application seems to be an obvious choice. The prototype uses the Apache HTTPS-Server and the scripting-language PHP which are shortly introduced in the remainder of this section. Nevertheless also such combinations as Apache/Tomcat-/Java or even IIS/ASP may lead to a comparable solution.

The chosen data-exchange format for the server-response is JSON (introduced in subsection 5.2.2 on page 67). To reduce the overall overhead for the here introduced process and ease the integration with JavaScript, JSON is preferred to other exchange-formats such as XML or SOAP/XML.
Apache

The Apache Software Foundation develops and maintains an open source HTTP server project commonly known as Apache [Apache]. Apache has a current market-share of approximately 50% [Netb] on the worlds web server market. The Apache web server provides a simple add-on mechanism for certain programming-language or other functionalities in form of dynamically loaded modules or based on a CGI. Apache supports also the HTTP over SSL (HTTPS) protocol which is later used to secure the communication between the extension and the server.

PHP

PHP [Php] is a dynamically and weakly typed object-oriented scripting language released under an open source license. PHP is a recursive acronym for "PHP: Hypertext Pre-processor" and is especially suited for web-development in conjunction with (X)HTML. Currently PHP is according to [Apabi] the most popular module for Apache web servers. As of today PHP has the version 5.2.5.

OpenSSL

OpenSSL [Open] is an open source toolkit implementing the SSL and TLS protocols, besides providing a comprehensive cryptography library including Certificate Authority (CA) functionality as well as OCSP client- and server-functionalities.

JSON

The JavaScript Object Notation (JSON) [Cro06] is a simple data-format suited for data-exchange. Similar to XML JSON provides a hierarchical structure for representing structured data but with a smaller amount of overhead and is therefore often used as lightweight data-exchange format in multiple application areas. JSON is derived from the ECMAScript Programming Language Standard. Implementations of ECMAScript such as JavaScript provide a simple eval()-Function to convert a JSON string-representation into a JavaScript-Object.

5.3 Prototype details

The prototype is named SMAC which is an acronym for Signature for Microformat Annotated Content. It consist out of a Firefox extension and an out-sourced server-side component. Conceptually the extension is responsible for extracting the signature-related information from a (X)HTML-document, delegating the extracted information to the server-side component which performs the signature-verification and visualizing the outcome of the overall verification-process in the browser again.

---

4Common Gateway Interface
A more detailed overview about the conducted activities gives figure 5.2 on page 68 which are described together with all components in the following subsections.

![Activity diagram of the prototype.](image)

5.3.1. Extension

The extension is compatible with Firefox versions beginning at 1.5 up to version 2.x. The extension is composed out of the following components or JavaScript-Objects respectively: Smac, HsigAggregator, Hsig, Parser, XPath, Comm, Preferences and Debug.

Figure 5.3 on page 69 provides an overview about the relations between the here introduced components.

Smac

This object is initialized by Firefox on the browser’s startup. Smac is responsible for the overall embedding and initialization of the extension’s components. It registers event-listeners to intercept the successful loading of a new document in the browser-window and to intercept all possible DOM-tree modifications. These modifications have to be intercepted, to circumvent subsequent manipulation of the DOM-tree by means of the loaded document. An attacker could thereby inject wrong content without changing the prior verification outcome. This issue is discussed in more detail later in section 5.4 on page 73.
**HsigAggregator**

This object works mainly as an aggregator for all Hsig-Objects. It parses the current loaded document and searches for all occurring hsig microformat class-identifiers. For each found hsig-identifier it initializes an Hsig-Object.

![Diagram](image)

*Figure 5.3.: Relations between extension components.*

**Hsig**

Hsig is the extension’s major component. It handles all necessary tasks including extraction, verification and visualization. Hsig includes two further prototyped objects named “HashTree” and “State”. HashTree encapsulates the whole process of generating and building the hash-tree-structure on basis of the extracted content. State on the other hand provides a structure containing all relevant status-information of the signature-verification process. A detailed state-chart of the State-object is shown in figure 5.4 on page 70.

The following tasks are performed by Hsig:

1. Extracting the hsig-specific properties namely `policy`, `cert`, `key` and `authdata`.
2. Validating on this basis the syntax of the hsig microformat.
3. Extracting the in the policy referenced microformat-properties from the document.
4. Evaluating the policy-expression to an outcome of `true` or `false` depending on the extracted microformat-properties.
5. Building of the hash-tree considering all given authdata-properties and the microformat-properties. The prototyped object hashTree handles this task.

6. Communication with the server-side component for signature- or certificate-verification respectively.

7. Managing the internal State-object reflecting the state of each step in the verification process and an overall outcome of the verification.

8. Visualizing the overall outcome, documented in the State-Object, in the browser-GUI.

Figure 5.4.: Statechart for the signature’s internal states.
5.3 Prototype details

Parser

The *Parser*-object supplies methods for extracting values corresponding to a given node in the DOM-tree and a property's class-identifier. The extraction methods are mainly based on the methods used in Michael Kaply's microformat parser Operator [Kap].

XPath

*XPath* functions as a wrapper for all XPath-related functionalities. All used XPath-expressions within the application are aggregated here. XPath-expressions are mainly used to locate the DOM-node of given microformat-class-identifiers.

Comm

*Comm* provides an interface for the communication with the out-sourced server-component of the client. The communication is implemented by means of the XMLHttpRequest-Object which enables to trigger asynchronous and synchronous HTTP(S)-Requests and to read their response. The request to the server-component is a usual POST-command with the necessary parameters while the retrieved answer is in JSON-format. *Comm* provides a simple converting-mechanism to translate JSON in a JavaScript object-representation.

Debug

*Debug* is a simple wrapper for accessing the build-in Firefox JavaScript-Console and to print out debugging information. This behavior is controlled by global settings in the browser's preferences. The debugging-object is used by nearly all parts of the extension, but mainly Hsig uses it to print out its State-object.

Preferences

The *Preferences*-object provides set- and get-methods to the Firefox global preferences system, which is usually accessed by typing 'about:config' in the address-location-bar. In the SMAC preferences the user is able to control the output of debugging-information on the JavaScript-console and to specify the URL for the verifying server-component.

Visualization

The extension transports the signature's verification-result in two ways to the user. After the result from the server-component has been received, an image indicating this result is added to the DOM-node carrying the hsig-microformat. Additionally a sidebar is opened which lists for each hsig the referenced content and the signature's state. Highlighting an entry in the sidebar leads to a CSS-modification of the original (X)HTML-document. The user may thereby re-check the signed content. Figure 5.5 on page 72 shows a screen-shot of this behavior.

A discussion of the spoof-ability of this visualization is found in section 5.4 on page 73.
5.3.2. Server component

The server component of the client is responsible for verifying the signature over the submitted certificates and also verifying existing certificate chains. Afterwards a comparison of the submitted root-hash, which was the result of the hash-tree build in the extension, and the extracted root-hash from the content certificate's CN-field is accomplished. At last the validity period is verified followed by a check of the revocation-status of each certificate by means of an OCSP-Request on basis of the certificate's authority information access extension. The outcome of all these performed tasks is returned as a server-response to the extension which then proceeds with the process. The server component manages the internal state content certificate in figure 5.4 on page 70.

The server-side component is itself composed out of the following PHP-Classes/Objects: Controller, Verifier, Certificate and OpenSSL. A short overview of their relations is depicted in figure 5.6 on page 73.

Controller

Works as the central controller for the application similar to the Front-Controller pattern [Fro]. The controller validates the request-parameters and delegates them afterwards for the verification process to the Verifier. After receiving the verification-result from the Verifier the Controller converts it into a JSON-string and submits the response back to the extension.
5.4 Limitations and problems

Verifier

The Verifier instantiates a Certificate-object for each received certificate. Within one request there is a maximum of two certificates, the content certificate and the owner certificate. The Verifier extracts the root-hash by means of the Certificate-object's method and compares it with the one received by the extension. After again using Certificate-methods to verify the validity period, the certificate's integrity and signature and receiving its revocation status, the Verifier delegates the outcome back to the Controller.

![Diagram of server-component objects and their relations.](image)

Figure 5.6.: Server-component objects and their relations.

Certificate

The Certificate-object encapsulates the whole certificate-handling. For all methods to succeed the OpenSSL-object is required. The Certificate-object can verify the issuer's signature for the certificate and build a certificate chain to find a trusted CA as a root-anchor. Furthermore it extracts the CommonName-field of the subjects Distinguished-Name where the root-hash of the content is located. For checking the revocation-status of the certificate a OCSP-Request is send to a service which is specified in the CRL Distribution Point of the certificate.

OpenSSL

OpenSSL encapsulates the access to the openssl-command-line and provides methods for building specific openssl-commands to be executed.

5.4. Limitations and problems
5.4.1. Overhead

A general issue is the overhead a signature-mechanism is adding to the generation, publishing and viewing of content in the web. With the proposed solution the content’s owner has to define a policy controlling the way his content may be published, build a hash-tree based on this policy, generate a content certificate and distribute this altogether on the web. Managing his content-certificates and therewith providing a OCSP-responder are additional tasks. The publisher has to assure that his content-re-publishing conforms to the policy defined by the owner and eventually re-calculate the content’s hash-tree. The user of the signed content for whom the prototype is implemented experiences the most overhead.

The signature processing does not interfere with the browser’s page-rendering of a website. The processing just starts right after the web-page has been completely loaded. The familiar browsing-experience is therefore not disturbed as a user does not need to wait for the complete page to be loaded. Nevertheless the current prototype adds a significant experienced delay to the page-loading the more signatures in the document exist.

A comparison between a document without a hsig microformat and a document with one, two, three, five and ten hsig microformats revealed that the process, independently of the overall amount of hsigs, for each found signature takes an average of 0.74 seconds to finish. This determined delay does not reflect a scientifically valid survey, it solely indicates a tendency and proves that a significant delay may be measured. This circumstance is further discussed in the further work section 5.5 on page 76.

5.4.2. Restrictions of microformats

The possibility to sign generally all content-fragments that are annotated with microformats has to be restricted. If for example a hcard-microformat contains multiple occurrences of the sub-property url, it is only possible to sign the URL which is contained in the first node found in the DOM-tree. There is no way with microformats to directly address or reference the second occurrence. Microformat-parser like Operator overcome this issue by aggregating all found URLs into an array, this behaviour is obviously not suitable for a signature for microformat annotated content. Additionally it is impossible to sign the subtype of a microformat-property. The hcard-property email for example has a subtype indicating the type of email-address stated, which may be for example the email-address at work or at home. The example in listing 5.1 on page 74 reveals that.

```html
<di
```
5.4 Limitations and problems

Listing 5.1: Example hcard-property email with stated subtypes.

The value of the node with the class-identifier type may carry arbitrary descriptions of the type of email-address. Generally, referencing the value of properties is done by stating their class-identifier (e.g. type) and usually extracting the node's text value (e.g. work). If now somebody wants to sign his email-address he is using at work, a possible solution could be a more hierarchical policy expression like vcard:email:type AND vcard:email:value. For the extraction-process of the prototype it is nevertheless impossible to distinguish between the content provided by the first occurrence of the subtype (work) and the second one (home) because both of them just contain arbitrary data for the process. Microformats provide additionally a scheme to assign related type/value-pairs. In the example 5.1 on page 74 the signature-process is therefore just able to guess the pairing of work and john.doe@example.com on basis of their sequence in the DOM-tree.

These issues arise from the insufficient addressing-scheme microformats use and consequently the prototype inherits this insufficiency.

5.4.3. Spoof-able GUI

The result of a signature verification should naturally not be spoof-able. The problem with microformats in general and therefore existing microformat-parsers as well, is that only the content is important. The styling of the content with CSS or as well by (X)HTML-means is not considered. As long as the content can be found by a parser it can be extracted. If the content's display is hidden or altered by means of CSS for example through a style-attribute containing display:none or not, is out of the scope of microformats. According to the microformat principle "Microformats are designed for humans first, machines second", the hiding of microformat annotated content is discouraged but not prohibited nor can this principle be enforced.

Displaying the result of the signature verification within the content-containing document therefore runs the risk of being spoofed. It is certainly possible with CSS to hide the complete signed content, displaying totally different content and still not invalidating the signature for the content. Additionally an attacker could take the same (X)HTML which visualizes the outcome and place it in the original document, without providing any hsig-related information at all and therefore tricking the user's perception concerning a valid signature. Displaying an un-spoof-able result of the signature verification is generally out of the scope of this thesis. To nevertheless counter this issue, the extension follows a dual approach by displaying the verification-result within the original document and additionally in the browser sidebar as demonstrated in figure 5.5 on page 72.
The Firefox sidebar is itself a privileged page and may only be accessed through privileged code such as JavaScript within an extension. Nevertheless it may be possible for an attacker to emulate the sidebar by means of non-privileged code coming from the attacker's web-site. The sidebar of the SMAC-extension is triggered by the on-click-event on an icon in the browser's status-bar (see again figure 5.5 on page 72). Therefore spoofing the status-bar including the icon indicating an SSL-secured connection is a related problem. Its solution is up to the browser vendor in this case the Mozilla Corporation and as already mentioned is out of the scope of this thesis.

5.4.4. General

The prototype is at the current state not capable of handling multiple browser-tabs. If the user opens multiple tabs at a time which all contain the hsig-microformat it is only possible to launch the verification-process by reloading the tab with the current focus. This limitation could not been resolved until this thesis was finished but should not cause a significant amount of modifications at the code-basis.

5.5. Further work

For becoming a comprehensive solution in the area of content signatures an implementation regarding the owner's and the publisher's perspective is mandatory. For the owner an integration of the signature-generating process in existing publishing and development environments is desirable. Popular web-development tools are for example Eclipse\(^5\) and Dreamweaver\(^6\). The latter already provides a plug-in for microformat support. An integration in publishing tools alike Content Management Systems (CMS) or blog-tools such as Wordpress\(^7\) or Serendipity\(^8\) is also for (re-)publishers an additional value. For being able to conveniently manage and revoke content certificates, the owner could also use a graphical interface which improves the usability of his content CA.

For the publisher a mechanism would be interesting which enables the direct marking of signed content within the browser and which copies the content into a certain publishing-tool thereby retaining the signature-relevant information. A Firefox extension utilizing the prototype's functionality would be an appropriate mean to implement this mechanism.

Reducing the delay each signature adds to the browsing-experience is a necessary task to process. Integrating the verifying server-component into the extension will certainly lead to much shorter response-times. For a productive version of the prototype the

---

\(^5\) see http://www.eclipse.org/
\(^6\) see http://www.adobe.com/products/dreamweaver/
\(^7\) see http://wordpress.org/
\(^8\) see http://www.s9y.org/
5.5 Further work

internal debugging-mechanism has to be removed which probably will also lead to a reduction of the processing-time. Additional fields of improvement are the manner of visualization for the verification result and a possible move to-wards an implementation of the extension's main functionality in a pre-compiled XPCOM component.

The current prototype parses the complete DOM-tree in order to find signatures. Another possibility to optimize the prototype's performance is therefore to consider and parse only the part of the DOM-tree the user actually perceives. This optimization operates under the assumption that the signatures are uniformly distributed within the document.

The prototype focuses on microformats for content-annotation and on X.509 PKI functionalities on the cryptographic side. Concerning this a more flexible approach is desirable to enable the plugging-in of other technologies for the markup (as for example RDFa) and other certificate formats alike PGP-certificates as already intended in the hsig-specification.

The prototype's browser-part is limited to Mozilla-based applications. To become a well accepted solution with a broad distribution among users it is mandatory to implement such a browser-extension at least for Microsoft's Internet Explorer.

An improvement of the integrity and authenticity of the extension itself could be achieved by signing the extension with an official Code Signing Certificate which is issued by several root CAs such as the Mozilla Corporation. By installing an extension signed with this certified private key, the Firefox Add-on-manager will automatically trust the installed extension. An attacker might thereby be prevented from distributing malicious code under the same extension's name.

An integration of the SMAC-functionalities into existing microformat-parser should be regarded in the future. This will centralize all microformat-concerned extensions and may utilize more of the already existing parser- and GUI-functionalities.
Part IV.

Conclusion
6. Conclusion and further work

This work introduced an approach to a solution for the majority of the problems of published content in the internet. These problems arise from a lack of authenticity, integrity and a narrowed control of the owner over his content, as mentioned in the introduction. With this solution an owner is able to digitally sign his produced content and simultaneously retain a certain degree of control. The user on the other hand is thereby able to check the content’s integrity and to approve the content’s origin or its authenticity respectively. He can verify that the owner acknowledged a special manner of publishing and likewise still consents to the overall publishing. As the signature also signs the semantic information contained in the microformat annotated content, the user receives an additional value. Integrating the verification-process in existing microformat-parser which already visualize semantics in (X)HTML-documents, is thus a recommendation concerning future developments.

The introduced approach is voluntary for all participants and keeps the information free for copying while not preserving the signature-information. The control for the owner is therefore certainly not guaranteed. Nevertheless besides the owner also the user receives additional value, so the approach relies on the “power of the crowds”. As soon as owners and a notable amount of users support this type of publishing also republishers gain an additional value. They receive a certain amount of reliability and credit which could in turn lead to a greater amount of users. These additional values may lead to a broader distribution and thus a working solution.

The initial decision to use microformats as annotation method turned out to be complicated. Microformats are a simple and unobtrusive method for semantic annotation, but this simplicity - although intended - is simultaneously their drawback. They lack of expressive power and extensibility, they currently have no namespace nor a underlying schema for validation and a comprehensive addressing scheme as illustrated in section 5.4.2 on page 74 is also missing. A further development of the approach should hence consider other annotation methods such as RDFa (see section 2.2.2 on page 9) or at least support for choose-able annotations.

Trust as it is known from real world scenarios like social relationships, is difficult to transfer to the internet. A non-existing fully distributed Public Key Infrastructure impedes this issue even more. However nearly every common web browser is today equipped with a variety of pre-installed X.509 public key certificates. These certificates belong to so called root-anchors of the classic PKI, also known as root CAs. Every certificate which is beneath such a root certificate in a certificate path is implicitly trusted.
A real trust-relationship like in the non-digital world applied to published content is not realizable with classic PKI, but an implicit trust in the content's authenticity may be reached. A usage of these pre-installed root certificates should therefore be regarded in further development of the prototype.

PGP with its web of trust is generally a much better model of the mentioned real world trust-relationship. Despite PGP's insufficient revocation-mechanism, more research effort should be spent for this kind of PKI. Especially a combination of PGP with X.509 certificates as mentioned in section 3.5.1 on page 31 seems promising.

The chosen data-structure for the hash-tree is a complete left-balanced binary tree. As already discussed in section 4.5 on page 53, an underlying data-structure which reflects the hierarchical structure of microformats would use the advantages of a hash-tree more efficiently. Re-publishers will more likely omit semantical close properties and thus such a n-ary hash-tree will economize space and reduce computational time concerning its build.

Another consideration touches the area of semantics. It is conceivable that by the advent and a wider distribution of RDF/RDFa a more fine grained semantic description of content-fragments and their relations is possible. A policy which expresses the owner's intention for semantically related fragments on a higher and more abstract level than direct fragment-addressing, could safeguard that a re-publisher only omits fragments which do not alter the overall meaning of the content.
Bibliography


Bibliography


Bibliography


Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

_________________________, Hamburg den 03. Dezember 2007