UNIVERSITÄT
KOBLENZ · LANDAU
Institut für Wirtschafts- und Verwaltungsinformatik

FB 4
Informatik

Iterative Signing of RDF(S) Graphs, Named Graphs, and OWL Graphs: Formalization and Application

Andreas Kasten
Ansgar Scherp

Nr. 3/2013

Arbeitsberichte aus dem Fachbereich Informatik

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Herausgeber / Edited by:

Der Dekan:
Prof. Dr. Grimm

Kontaktdaten der Verfasser

Andreas Kasten, Ansgar Scherp
Institut für Wirtschafts- und Verwaltungsinformatik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
E-Mail: andreas.kasten@uni-koblenz.de, ansgar@informatik.uni-mannheim.de

Iterative Signing of RDF(S) Graphs, Named Graphs, and OWL Graphs: Formalization and Application

Andreas Kasten[1] and Ansgar Scherp[2]

[1] University of Koblenz, 56070 Koblenz, Germany,
andreas.kasten@uni-koblenz.de,
[2] University of Mannheim, 68131 Mannheim, Germany,
ansgar@informatik.uni-mannheim.de

Abstract. When publishing graph data on the web such as vocabularies using RDF(S) or OWL, one has only limited means to verify the authenticity and integrity of the graph data. Today's approaches require a high signature overhead and do not allow for an iterative signing of graph data. This paper presents a formally defined framework for signing arbitrary graph data provided in RDF(S), Named Graphs, or OWL. Our framework supports signing graph data at different levels of granularity: minimum self-contained graphs (MSG), sets of MSGs, and entire graphs. It supports for an iterative signing of graph data, e. g., when different parties provide different parts of a common graph, and allows for signing multiple graphs. Both can be done with a constant, low overhead for the signature graph, even when iteratively signing graph data.

1 Introduction

Exchanging trusted graph data on the Semantic Web is only possible to a limited extend today. On the contrary, the amount of graph data published and shared has tremendously increased in the last years. Thus, it becomes inherently necessary to be able to verify the authenticity and integrity of graph data published on the web in order track its provenance and building trust networks for knowledge-based systems using that data. Authenticity and integrity are basic security requirements which ensure that graph data is really created by the party who claims to be its creator and that any modifications on the data are only carried out by authorized parties. Signing graph data allows for verifying the provenance and trustworthiness of, e. g., assertional knowledge provided as RDF graphs or terminological knowledge published in form of vocabularies defined in RDFS or OWL. To the best of our knowledge, the only solution for signing graph data so far is the work by Tummarello et al. [1]. It provides a simple graph signing function for so-called minimum self-contained graphs (MSGs). An MSG is defined over statements. It is the smallest subgraph of the complete RDF graph that contains a statement and the statements of all blank nodes associated either directly or recursively with it. Statements without blank nodes are an MSG

on their own. Tummarello et al. provide an important early step for signing graph data. However, it has significant shortcomings regarding the functionality provided and overhead required for representing the graph signature: First, the signing function can be applied on MSGs only. To this end, the signature is attached to the MSG by using the RDF Statement reification mechanism. This requires significant overhead for representing the signature statements. Second, it cannot be applied on, e. g., sets of statements like ontology design patterns or graphs as a whole. The approach does not support the signing of Named Graphs and cannot be used to sign multiple graphs at the same time. Finally, the approach by Tummarello et al. does not allow for an iterative signing of graph data. The signature statements created for each signing step become part of the same MSG. There is no explicit relationship between the signature and the signed statements. This makes it practically impossible to verify the integrity and authenticity of the graph data.

In this work, we present a formal framework for signing arbitrary graph data. The framework can be configured, e. g., to optimize the signing process towards efficiency or minimizing the signature overhead. The resulting signature graph is assembled with the signed graph and can be published on the web. As input graph data, one can use RDF(S), Named Graph, or OWL. Our framework supports different levels of granularity of signing graph data. It can be used to sign a minimum self-contained graph (MSG), a set of MSGs, entire graphs, and multiple graphs at once. In addition, the graphs can be distributed over the web and can contain assertional knowledge as well as terminological knowledge. Finally, we require only a low overhead for signing graphs, which is constant even when iteratively signing graph data.

The following scenario motivates the need for iteratively signing different types of graph data. The related work is presented in Section 3. In Section 4, we derive the requirements for signing graph data from the scenario and the related work. A formal definition of our graph signing framework is given in Section 5 and three different configurations are discussed in Section 6. Finally, we present an example implementation of our framework in Section 7.

2 Scenario: Trust Network for Content Regulation

The ability to publish arbitrary content on the Internet also imposes the ethical and legal obligation to regulate access to content that is, e. g., inappropriate to minors or threatens public peace. In the scenario depicted in Fig. 1, we consider building a trust network for Internet regulation in Germany. The information about what kind of content is to be regulated is encoded as graph data, which is provided by different authorities. An authority receives signed graph data from another authority. It adds its own graph data to the received one, digitally signs both, and publishes it again on the web.

Due to Germany's history in the second World War, until today the access to neo-Nazi material on the Internet is prohibited by German law (Criminal Code, §86 [2]). The German Federal Criminal Police Office (Bundeskriminalamt, BKA)

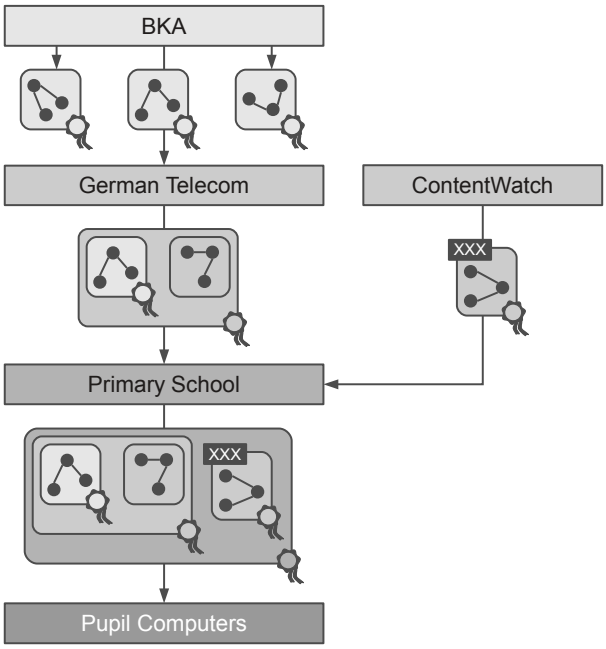

Fig. 1. Trust network for content regulation.

provides a set of formally defined ontologies **(ii)** making use of ontology design patterns [3]. The patterns represent knowledge such as wanted persons, recent crimes, and regulation information for Internet communication like it is required by §86. In addition, the BKA provides a blacklist of web sites to be blocked according to §86. It signs both the ontologies and the blacklist **(iii)** and publishes the ontologies on the web. Internet service providers (ISPs) such as the German Telecom receive the regulating information from the BKA. By verifying its authenticity **(a)** and integrity **(b)**, the ISPs can trust the BKA's regulation data. This data only describes what is to be regulated and not how it is regulated. Thus, ISPs like the German Telecom interpret the data received from the BKA and add concrete details such as the proxy servers and routers used for blocking the web sites. As shown in Fig. 1, the ISP compiles its technical regulation details as RDF graph which is based on the BKA's ontology pattern. It digitally signs the BKA's blacklist **(iv)** together with its own regulation graph **(i)** and sends it to its customers. The customers such as the primary school depicted in Fig. 1 are able to verify the authenticity and integrity of the regulating information. The school has to ensure that its pupils cannot access illegal neo-Nazi content. The iterative signing of the regulation data allows the school to check which party is responsible for which parts of the data. Thus, it can track the provenance of the regulation's creation. In addition, the school has to ensure that adult content cannot be accessed by the pupils. To this end, it receives regulation information for adult content from private authorities such as

ContentWatch (`http://www.contentwatch.com`), which offers regulation data as Named Graphs **(ii)** to protect children from Internet pornography and the like. Thus, different regulation information **(v)** from multiple sources **(vi)** is incorporated by the school. Finally, the primary school digitally signs the incorporated regulation information **(iv)** before providing it to the client computers located in the school. This ensures that the pupils using these computers access the Internet only after passing the predefined regulation mechanisms.

3 Related Work

The related work is structured along the process of signing data: First, the data is normalized using a *canonicalization function*. This function rearranges the data's structure to a unique representation. Second, the canonicalized data is transformed into a sequential representation by applying a *serialization function*. If the canonicalized data is already in a sequential form, this step can be omitted. Third, a cryptographic hash value is computed on the serialized data. A *hash function* transforms a sequential representation of arbitrary length to one of fixed length [4]. Fourth, the actual signature value is computed using a *signature function*. The signature is created by combining the hash value of the serialized data with a signature key [4]. The signature key is the secret key of an asymmetric key pair such as an RSA pair [5]. The combined results of all aforementioned functions actually make up the graph signing function. Fifth, the *assembly function* creates a signature graph which contains all data for verifying the graph's integrity and authenticity, which is the last step.

3.1 Canonicalization Functions for Graphs

A canonicalization function assures that the in principle arbitrary identifiers of a graph's blank nodes do not affect the graph's signature. Some canonicalization functions also ensure a unique ordering of the graph's statements. A formal definition of canonicalization functions is given in Section 5.2. Carroll [6] presents a canonicalization function for RDF graphs that replaces all blank node identifier with a uniform place holder, sorts all statements of the graph based on their N-Triples [7] representation, and renames the blank nodes according to the order of their statements. If this results in two blank nodes having the same identifier, additional statements are added for these blank nodes. Carroll's canonicalization function uses Unix's sort algorithm that has a runtime complexity of $O(n \log n)$ and a space complexity of $O(n)$ with n as the number of statements in the graph [6]. Fisteus et al. [8] provide a canonicalization function for datasets serialized in N3 [9]. The function requires a hash value for each statement based on a hash function of the same authors described in Section 3.3. The hash function assures that the blank nodes do not affect the statements' hash values. The canonicalization function sorts the statements according to their hash values. Due to the sorting process, the runtime complexity of the canonicalization function is $O(n \log n)$ and its space complexity is $O(n)$. Finally, Sayers and Karp [10]

provide a canonicalization function for RDF graphs, which stores the identifier of each blank node in an additional statement. If the identifier is changed, the original one can be recreated using this statement. Since this does not require sorting the statements, the runtime complexity of the function is $O(n)$. In order to detect already handled blank nodes, the function maintains a list of additional statements created so far. This list contains at most b entries with b as the total number of additional statements. Thus, the space complexity of the function is $O(b)$.

3.2 Serialization Functions for Graphs

A serialization function transforms an RDF graph into a sequential representation such as a bit string or a set of bit strings. This representation is encoded in a specific format such as statement-based N-Triples [7] and N3 [9] or XML-based RDF/XML [11] and OWL/XML [12]. TriG [13] is a statement-based format built upon N3, which allows for expressing Named Graphs. HDT [14] is a binary format for encoding RDF graphs in a compact form, which requires less storage space than ASCII-based formats. When signing RDF graphs, statement-based formats are often preferred to XML-based notations due to their simpler structure. Section 5.3 gives a formalization of serialization functions. If a serialization function does not utilize the full expressiveness of its serialization format like sorting the statements, it can be implemented with a runtime complexity of $O(n)$ and a space complexity of $O(1)$.

3.3 Hash Functions for Graphs

Computing the hash value of a graph is often based on computing the hash values of its statements and combining them into a single value. Computing a statement's hash value can be done by hash functions such as MD5 [15] or SHA-2 [16]. Section 5.4 provides both a formalization of such hash functions and a formal definition of hash functions for graphs. Melnik [17] uses a simple hash function for RDF graphs. A statement's hash value is computed by concatenating the hash value of its subject, predicate, and object and hashing the result. The hash values of all statements are sorted, concatenated, and hashed again to form the hash value of the entire RDF graph. Due to the use of a sorting algorithm, the function's runtime complexity is $O(n \log n)$ and its space complexity is $O(n)$. Fisteus et al. [8] suggest a hash function for N3 datasets. First, all blank nodes are associated with the same identifier. Secondly, the statements' hash values are computed like with Melnik's approach [17]. If two statements have the same hash value, new identifiers of the blank nodes are computed by combining the hash values of the statements in which they occur. This process is repeated until there are no collisions left. Finally, the hash value of a graph is computed by combining the hash values of its statements. Colliding hash values are detected by sorting them, which leads to a runtime complexity of $O(n \log n)$ and a space complexity of $O(n)$. In the worst case, the runtime complexity is $O(n^2)$ due to multiple re-hashing processes. Carroll [6] uses a graph-hashing function which

serializes all statements, sorts the serialized representations, concatenates the result into a bit string, and hashes this bit string using a simple hash function such as SHA-2 [16]. As the function uses Unix's sort algorithm, it has a runtime complexity of $O(n \log n)$ and a space complexity of $O(n)$. Finally, Sayers and Karp [10] compute a hash value of an RDF graph by incrementally multiplying the hash values of the graph's statements modulo a prime number. Since this operation is commutative and associative, sorting the statements' hash values is not required. Thus, the runtime complexity of the hash function is $O(n)$. Due to the multiplication, the space complexity is $O(1)$.

3.4 Signature Functions

A signature function computes the actual signature by combining the graph's hash value with a secret key. A formalization for signature functions is given in Section 5.5. Possible signature functions are DSA [18], ElGamal [19], and RSA [5]. Since the graph's hash value is independent from the number of statements, the signature is as well. Thus, the runtime complexity and the space complexity of all signature functions are $O(1)$.

3.5 Graph Signing Functions

A graph signing function creates a signature for a graph by combining all aforementioned functions. A formal definition of graph signing functions is given in Section 5.6. Tummarello et al. [1] present a graph signing function for fragments of RDF graphs. These fragments are minimum self-contained graphs (MSGs) and are defined over statements. An MSG of a statement is the smallest subgraph of the entire RDF graph which contains this statement and the statements of all blank nodes associated with it. Statements without blank nodes are an MSG on their own. The graph signing function of Tummarello et al. is based on Carroll's canonicalization function and hash function [6]. The resulting signature is stored as a set of six statements, which are added to the signed MSG. These signature statements are linked to the MSG via RDF Statement reification of one of the MSG's statements. The approach of Tummarello et al. is based on signing one MSG at a time. Signing multiple MSGs requires multiple signatures. Individually signing MSGs with only one statement creates a high overhead of six signature statements. Furthermore, the approach by Tummarello et al. does not allow for iterative signing of graph data. The signature statements created for each signing step become part of the signed MSG. Signing this MSG again also signs the included signature statements. This makes it impossible to relate a set of signature statements to the corresponding signed graph data. Thus, verifying the signature becomes practically impossible.

Signing a full graph can also be accomplished by signing a document containing a serialization of the graph [20]. For example, a graph can be serialized using an XML-based format such as RDF/XML [11] or OWL/XML [12]. This results in an XML document which can be signed using the XML signature standard [21].

If the graph is serialized using a plain text-based format such as the statement-based serializations N-Triples [7] or N3 [9], also standard text document signing approaches may be used [22]. However, signing the graphs on the granularity levels of single MSGs or sets of MSGs is not possible by these approaches. Iterative signing of the documents is also not supported. Most significant drawback, however, is that the signature is inextricably linked with the concrete document containing the graph [20]. This means that the created signature can only be verified with the very specific serialization of the graph contained in the document. For example, if the serialized graph data is transferred from the signed document into a triple store and then retrieved back, it is not possible anymore to verify the authenticity and integrity of the graph data with the document's signature.

Finally, another option for ensuring the authenticity and integrity of the graph data would be to use secure communication channels like an SSL connection [23]. Indeed, when transmitting the graph data over a secure channel like SSL the recipient of the data can verify for the graph's authenticity and integrity. However, once the communication channel is closed after all data is transmitted, there is no chance to verify the graph data again without retrieving it once more from the provider. Verifying the authenticity and integrity of the graph data again might be necessary when the data is stored on not fully-trusted services like cloud computing or when it is transmitted further to other parties (which like to check the authenticity and integrity themselves).

3.6 Assembly Function

An assembly function creates a detailed description of how a graph's signature can be verified. This description may then be added to the signed graph data or be stored at a separate location. Section 5.7 provides a formal definition of assembly functions. Tummarello et al. [1] present a simple assembly function which adds additional statements to a signed MSG. These statements contain the signature value and a URL to the signature key used to compute the value. Information about the graph signing function and its subfunctions is not provided. Once the URL to the signature key is broken, i. e., the signature key is not available anymore at this URL, the signature can no longer be verified. Even if a copy of the signature key is still available at a different location, the verifier finally cannot check the true authenticity of the signature key as the issuer is only implicitly encoded in the key itself.

In order to describe a signing function, the XML signature standard [21] provides an XML schema containing all details of an XML signature. These details comprise the names of the used canonicalization function, the hash function, and the signature function used for computing the signature value. Furthermore, the XML schema also allows for describing a distinguished name of the signature key issuer, the key's serial number, and further information.

4 Requirements

Based on the scenario in Section 2 and the related work discussed in Section 3, we derive the following functional requirements for a generic solution for signing graph data:

REQ-1: Signing at different levels of granularity: Allow a party to sign graph data at different levels of granularity starting from single statements and MSGs, respectively, sets of statements (like ontology design patterns), and entire graphs. This allows for a most flexible use of the graph signing approach. In the scenario, the BKA signs ontology patterns and the German Telecom signs its entire regulation graph (see **(i)**).

REQ-2: Signing different kinds of graph data: Allow a party to sign graph data provided in RDF(S), OWL, and Named Graphs. This allows for the graph signing approach to be used in Linked Open Data[3] contexts as well as for signing, e. g., foundational ontologies like DOLCE+DnS Ultralite [24]. In the scenario, the BKA provides OWL ontology patterns and ContentWatch provides Named Graphs (see **(ii)**).

REQ-3: Signing T-box and A-box knowledge: Allow a party to sign both assertional (A-box) knowledge and terminological (T-box) knowledge. This allows the graph signing approach to be used for signing vocabularies issued by, e. g., standardization bodies. In addition, parties publishing their own instance data using those vocabularies can sign their assertional knowledge as well. In the scenario, the BKA signs both its ontologies and its regulation data (see **(iii)**).

REQ-4: Iterative signing of graph data: Allow a party to sign graph data which is already signed. This allows for provenance tracking as each party signs its own data together with the provided data and its attached signatures. In the scenario, the German Telecom signs the BKA's regulation data, which is already signed by the BKA itself (see **(iv)**).

REQ-5: Signing multiple, distributed graphs: Allow a party to sign multiple graphs at the same time which are distributed over different locations. In the scenario, the primary school retrieves and signs two different graphs from the German Telecom and ContentWatch at once (see **(v)**).

In addition to these functional requirements, a generic framework for signing graph data shall allow for verifying the authenticity and integrity of the provided data. This allows for establishing a trust network between the involved communication parties. Thus, the following general security requirements must be fulfilled:

REQ-A: Verifying the authenticity of the graph data: Allow a party to check if a received graph was really created by the party who claims to be its creator [25]. In the scenario, the German Telecom can verify that the graph received from the BKA was really created by the BKA and not by a third party (see **(a)**).

REQ-B: Verifying the integrity of the graph data: Allow a party to check whether or not a received graph was modified by an unauthorized

[3] `http://www.w3.org/DesignIssues/LinkedData`, last accessed: 4/9/2013

party [25]. Thus, a party can identify any changes on the graph by verifying its integrity. In the scenario, the German Telecom can ensure that the graphs received from the BKA were not modified by a third party (see **(b)**).

5 Graph Signing Formalization

This section first defines RDF graphs as they are the basic data structure to be signed. This definition is then extended to Named Graphs. Subsequently, all functions of the signing process introduced in Section 3 are formally defined. These are the canonicalization function κ_N, the serialization function ν_N, the hash function λ_N, and the signature function ϵ. Using these functions, the graph signing function σ_N for Named Graphs is defined, followed by a definition of the signature graph S and the assembly function ς_N. The section is concluded with a description of the signature verification procedure. Fig. 2 depicts the process of signing graph data. The graph signing function is basically a combination of the functions used in the first four steps.

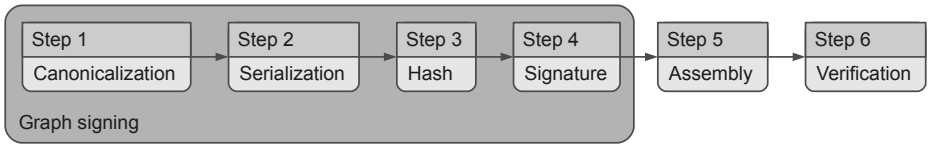

Fig. 2. The process of signing and verifying graph data. The graph signing function corresponds to the first four steps. In the fifth step, the signature graph is created. Finally, the sixth step is applied to verify the authenticity and integrity of the graph data.

5.1 Definition of Graphs

An RDF graph G is a finite set of RDF triples t. The set of all RDF triples is defined as $\mathbb{T} = (\mathbb{R} \cup \mathbb{B}) \times \mathbb{P} \times (\mathbb{R} \cup \mathbb{B} \cup \mathbb{L})$ with the pairwise disjoint sets of resources $\mathbb{R}$, blank nodes $\mathbb{B}$, predicates $\mathbb{P}$, and literals $\mathbb{L}$. Thus, it is $t = (s, p, o)$ with $s \in \mathbb{R} \cup \mathbb{B}$ being the subject of the triple, $p \in \mathbb{P}$ being the predicate, and $o \in \mathbb{R} \cup \mathbb{B} \cup \mathbb{L}$ being the object [26, 27]. An OWL graph can be mapped to an RDF graph [28]. Thus, in the following we will only denote RDF graphs and include OWL graphs mapped to RDF graphs. The set of all possible RDF graphs is $\mathbb{G} = 2^{\mathbb{T}}$.

A Named Graph extends this notion of RDF graphs and associates a unique name in form of a URI to a single RDF graph [29] or set of RDF graphs. This URI can be described by further statements, which form the so-called *annotation graph* of the RDF graph. The RDF graph is also called the *content graph*. A Named Graph $NG \in \mathbb{G}_N$ is defined as $NG = (a, A, \{C_1, C_2, \ldots, C_l\})$ with $a \in$

$\mathbb{R} \cup \{\varepsilon\}$ being the name of the graph, $A \in \mathbb{G}$ being the annotation graph, and $C_i \in \mathbb{G}_N$ being the content graphs with $i = 1 \ldots l$. If a Named Graph does not explicitly specify an identifier, ε is used as its name. This corresponds to associating the graph with a blank node. In this case, the graph A is empty, i. e., $A = \emptyset$. Any RDF graph $G \in \mathbb{G}$ can be defined as Named Graph C using the notation above as $C = (\varepsilon, \emptyset, G)$. The set of all Named Graphs $\mathbb{G}_N$ is recursively defined as $\mathbb{G}_N = (\mathbb{R} \times \mathbb{G} \times 2^{\mathbb{G}_N}) \cup \{(\varepsilon, \emptyset, G)\}$ with $G \in \mathbb{G}$.

5.2 Canonicalization Function

The canonicalization function κ transforms a graph $G \in \mathbb{G}$ into its canonical form $\hat{G} \in \hat{\mathbb{G}}$ with $\hat{\mathbb{G}} \subset \mathbb{G}$ being the set of all canonical graphs. Example canonicalization functions for RDF graphs are given in [6, 8, 10] and further described in Section 3.1.

$$\kappa : \mathbb{G} \to \hat{\mathbb{G}}, \quad \kappa(G) := \hat{G} \tag{1}$$

For Named Graphs, the canonicalization function κ_N is recursively defined. It computes a canonical representation of a Named Graph $NG = (a, A, \{C_1, \ldots, C_l\})$ by computing the canonical representations $\hat{A}$ and $\hat{C}_i$ of its annotation graph A and its content graphs C_i. The result is a canonical representation $\hat{NG} \in \hat{\mathbb{G}}_N$ with $\hat{\mathbb{G}}_N \subset \mathbb{G}_N$ being the set of all canonical Named Graphs.

$$\kappa_N : \mathbb{G}_N \to \hat{\mathbb{G}}_N, \quad \kappa_N(NG) := \hat{NG} \tag{2}$$

$$\kappa_N(NG) := \begin{cases} (\varepsilon, \emptyset, \hat{G}) & \text{if } NG = (\varepsilon, \emptyset, G),\ G \in \mathbb{G} \\ (a, \hat{A}, \{\hat{C}_1, \ldots, \hat{C}_l\}) & \text{if } NG = (a, A, \{C_1, \ldots, C_l\}) \end{cases}$$

5.3 Serialization Function

The serialization function ν transforms a graph $G \in \mathbb{G}$ into a set of bit strings $\overline{G} \in 2^{\{0,1\}^*}$. A bit string represents a statement in the graph G. The concrete characteristics of the bit strings in $\overline{G}$ as well as its length depend on the used serialization format. As outlined in Section 3.2, possible serializations for RDF graphs include Turtle [30] and RDF/XML [11].

$$\nu : \mathbb{G} \to 2^{\{0,1\}^*}, \quad \nu(G) := \overline{G} \tag{3}$$

The serialization function ν can be extended to the function ν_N for Named Graphs $NG \in \mathbb{G}_N$. The result of ν_N is a set of o bit strings $\overline{NG} \in 2^{\{0,1\}^*}$ with $\overline{NG} = \{b_1, b_2, \ldots, b_o\}$. The function is recursively defined as follows:

$$\nu_N : \mathbb{G}_N \to 2^{\{0,1\}^*}, \quad \nu_N(NG) := \overline{NG} \tag{4}$$

$$\nu_N(NG) := \begin{cases} \overline{G} & \text{if } NG = (\varepsilon, \emptyset, G),\ G \in \mathbb{G} \\ \{a\} \cup \overline{A} \cup \overline{C}_1 \cup \ldots \cup \overline{C}_l & \text{if } NG = (a, A, \{C_1, \ldots, C_l\}) \end{cases}$$

5.4 Hash Function

The hash function λ computes a hash value h of arbitrary bit strings $b \in \{0,1\}^*$. The resulting hash value h has a fixed length $d \in \mathbb{N}$, i. e., $h \in \{0,1\}^d$. Example hash functions are MD5 [15] and SHA-2 [16].

$$\lambda : \{0,1\}^* \rightarrow \{0,1\}^d, \quad \lambda(b) := h \tag{5}$$

The hash function λ_N computes a hash value h_N of a serialized Named Graph $\overline{NG} \in 2^{\{0,1\}^*}$ with $\overline{NG} = \{b_1, b_2, \ldots, b_o\}$ and is build upon the function λ. The function λ_N computes a hash value of each bit string $b_i \in \overline{NG}$ with $b = 1 \ldots o$ and combines the results into a new bit string $h_N \in \{0,1\}^d$ using a combining function ϱ. The function ϱ is defined as follows:

$$\varrho : 2^{\{0,1\}^d} \rightarrow \{0,1\}^d, \quad \varrho(\{h_1, h_2, \ldots, h_o\}) := h_N \tag{6}$$

Using the functions λ and ϱ, the hash function λ_N for Named Graphs is defined as follows:

$$\lambda_N : 2^{\{0,1\}^*} \rightarrow \{0,1\}^d, \quad \lambda_N(\overline{NG}) := h_N \tag{7}$$
$$\lambda_N(\overline{NG}) := \varrho(\{\lambda(b_1), \lambda(b_2), \ldots, \lambda(b_o)\})$$

Example hash functions λ_N are presented in [17, 8, 6, 10] and further described in Section 3.3. Example combining functions ϱ are discussed in [10].

5.5 Signature Function

A signature function ϵ computes the signature value of a graph based on the graph's hash value. It requires a bit string $b \in \{0,1\}^d$ and a cryptographic key as input. The keyspace, i. e., the set of all asymmetric, cryptographic keys is defined as $\mathbb{K} = \mathbb{K}_p \times \mathbb{K}_s$ with $\mathbb{K}_p$ being the set of public keys and $\mathbb{K}_s$ being the set of secret keys. For computing signatures, a secret key $k_s \in \mathbb{K}_s$ is used. As outlined in Section 3.4, example signature functions are DSA [18], ElGamal [19], and RSA [5]. Using $\tilde{b} \in \{0,1\}^{d'}$ as an identifier for the resulting bit string, the signature function is defined as follows:

$$\epsilon : \mathbb{K}_s \times \{0,1\}^d \rightarrow \{0,1\}^{d'}, \quad \epsilon(k_s, b) := \tilde{b} \tag{8}$$

5.6 Graph Signing Function

The graph signing function σ_N for Named Graphs is defined by using the different functions introduced above. The function σ_N allows for signing multiple graphs at once. It requires a secret key k_s and a set of m Named Graphs NG_i as input with $NG_i = (a_i, A_i, \{C_{1_i}, \ldots, C_{l_i}\})$ and $i = 1, \ldots, m$. The resulting signature is the bit string $s \in \{0,1\}^{d'}$. An example for a graph signing function is Tummarello et al. [1], which is described in Section 3.5.

$$\sigma_N : \mathbb{K}_s \times 2^{\mathbb{G}_N} \rightarrow \{0,1\}^{d'}, \quad \sigma_N(k_s, \{NG_1, \ldots, NG_m\}) := s \tag{9}$$
$$\sigma_N(k_s, \{NG_1, \ldots, NG_m\}) := \epsilon(k_s, \lambda_N(\nu_N(\kappa_N(NG_1)) \cup \ldots \cup \nu_N(\kappa_N(NG_m))))$$

5.7 Assembly Function

An assembly function ς_N creates the signature graph $S \in \mathbb{G}$ and includes it in another Named Graph NG_S. The specific contents and structure of S depend on the implementation of the function ς_N. The signature graph contains information about how a graph's signature was created and how it can be verified. This includes the used canonicalization function κ_N, the serialization function ν_N, the hash function λ_N, the signature function ϵ, the corresponding public key k_p of the used secret key k_s, the identifiers a_i of the signed Named Graphs, and the signature value s. A possible structure of a signature graph is shown in the example in Section 7. The graph NG_S contains the signature graph S as its annotation graph and the signed Named Graphs NG_i as its content graphs. In order to allow for an iterative signing of Named Graphs, the result of the assembly function ς_N is also a Named Graph.

$$\varsigma_N : \mathbb{K}_s \times 2^{\mathbb{G}_N} \rightarrow \mathbb{G}_N \tag{10}$$

$$\varsigma_N(k_s, \{NG_1, \ldots, NG_m\}) := (a_S, S, \{NG_1, \ldots, NG_m\})$$

An example signature graph for signed RDF graphs is used in [1] and an XML fragment for signed XML documents is defined in [21].

5.8 Verification Function

The verification of a signature is similar to its creation. A verification function γ_N also requires a canonicalization function κ_N, a serialization function ν_N, and a hash function λ_N. Furthermore, it requires a signature verification function δ, which is inverse to the signature function ϵ. The signature verification function δ requires a bit string $\tilde{b} \in \{0,1\}^{d'}$ and a public key $k_p \in \mathbb{K}_p$ as input. δ is defined as follows with $b \in \{0,1\}^d$ being the identifier of the resulting bit string. It holds $\delta(k_p, \epsilon(k_s, b)) = b$ with the secret key k_s.

$$\delta : \mathbb{K}_p \times \{0,1\}^{d'} \rightarrow \{0,1\}^d, \quad \delta(k_p, \tilde{b}) := b \tag{11}$$

The verification function γ_N checks whether or not a given signature is a valid signature of a set of Named Graphs. The function requires a public key k_p, a signature value s, and the set of signed Named Graphs $\{NG_1, \ldots, NG_m\}$. All values can be taken from the signature graph S. The key k_p is the public counterpart of the secret key k_s, which was used for creating the signature value s. The function γ_N combines the signature value s with the public key k_p and computes the hash value h' of the Named Graphs NG_i. The signature is valid iff both computed values are equal. It is $h' = \lambda_N(\nu_N(\kappa_N(NG_1)) \cup \ldots \cup \nu_N(\kappa_N(NG_m)))$.

$$\gamma_N : \mathbb{K}_p \times 2^{\mathbb{G}_N} \times \{0,1\}^* \rightarrow \{TRUE, FALSE\} \tag{12}$$

$$\gamma_N(k_p, \{NG_1, \ldots, NG_m\}, s) := \begin{cases} TRUE & \text{if } \delta(k_p, s) = h' \\ FALSE & \text{otherwise} \end{cases}$$

6 Three Graph Signing Functions

The complexity of a signature solely depends on the graph signing function σ_N and its subfunctions. The signature overhead depends on the additional statements created by these functions and on the size of the signature graph created by the assembly function ς_N. Table 1 summarizes the complexity of different implementations of κ_N, ν_N, λ_N, and ϵ as described in Section 3. In the table, n refers to the number of statements to be signed and b is the number of statements added by the canonicalization function. Thus, b corresponds to the number of blank nodes in the graph.

Table 1. Complexity of the functions used by the graph signing function σ_N. n is the number of statements and b is the number of blank nodes in the statements.

Function	Example	Runtime	Space
Canonicalization κ_N	Carroll [6]	$O(n \log n)$	$O(n)$
	Fisteus et al. [8]	$O(n \log n)$	$O(n)$
	Blank Node Labeling [10]	$O(n)$	$O(b)$
Serialization ν_N	N-Triples [7]	$O(n)$	$O(1)$
	N3 [9]	$O(n)$	$O(1)$
	TriG [13]	$O(n)$	$O(1)$
	RDF/XML [11]	$O(n)$	$O(1)$
	OWL/XML [12]	$O(n)$	$O(1)$
Hash λ_N	Melnik [17]	$O(n \log n)$	$O(n)$
	Fisteus et al. [8]	$O(n \log n)$	$O(n)$
	Carroll [6]	$O(n \log n)$	$O(n)$
	Incremental digest [10]	$O(n)$	$O(1)$
Signature ϵ	DSA [18]	$O(1)$	$O(1)$
	ElGamal [19]	$O(1)$	$O(1)$
	RSA [5]	$O(1)$	$O(1)$

Table 2. Possible configurations of a signing function σ_N.

Configuration	Canonicalization	Hash
A) Tummarello et al. [1]	Carroll [6]	Carroll [6]
B) Min. Signature Overhead	Fisteus et al. [8]	Fisteus et al. [8]
C) Min. Runtime Complexity	Blank Node Labeling [10]	Incremental Hash [10]

Table 3. Complexity and signature overhead of different signing functions σ_N.

Configuration	**Complexity of σ_N**		**Signature overhead**
	runtime	**space**	**of σ_N and ς_N**
A) Tummarello et al. [1]	$O(n \log n)$	$O(n)$	$b_h + 6r$ statements, $b_h \leq b$, $r \leq n$
B) Min. Signature Overhead	$O(n \log n)$	$O(n)$	$0 + 19$ statements
C) Min. Runtime Complexity	$O(n)$	$O(n)$	$b + 19$ statements

Table 2 shows three possible configurations of the signing process applied on a single graph. Table 3 depicts their complexity and signature overhead. Following the properties of the O-notation, the complexity of a graph signing function σ_N is defined by the highest complexity of all its subfunctions. To ease comparability, each configuration uses N-Triples for serialization and RSA as signature function ϵ. The configurations differ only in the canonicalization function κ_N and hash function λ_N. In the following, we discuss the configurations in more detail.

A) Tummarello et al [1] use the canonicalization function and hash function of Carroll [6]. Due to the complexity of these functions, the runtime complexity of the graph signing function is $O(n \log n)$ and its space complexity is $O(n)$. Carroll's canonicalization function handles blank node identifiers by sorting all of a graph's statements. If different blank nodes share the same identifier, an additional statement is created for each node. With $b_h \leq b$ being the number of such statements, the canonicalized graph contains b_h more statements than the original graph. The approach by Tummarello et al. only allows for signing a single MSG at a time. The signature is stored using six additional statements. Signing a graph with r MSGs requires r different signatures. The overhead created by the assembly functions is then $6r$ statements. The total overhead is thus $b_h + 6r$ statements. Since each MSG contains at least one statement, it is $r \leq n$. In the worst case, each MSG contains exactly one statement and it is $r = n$. The signature overhead is then $6n$.

B) Minimum Signature Overhead Using the canonicalization function and the hash function of Fisteus et al. [8] leads to a signing process with a minimum signature overhead. Both functions have a runtime complexity of $O(n \log n)$ and a space complexity of $O(n)$. The runtime complexity of the singing function σ_N is thus $O(n \log n)$ and the space complexity is $O(n)$. Since neither the canonicalization function nor the hash function create any additional statements, the signature overhead is solely determined by the signature graph S. The structure and size of the graph are defined by the ontology used for modeling it. Using a signature graph S as in the example depicted in Fig. 4 results in a signature overhead of 19 statements. When m graphs are signed at the same time, the m

graphs are arranged using RDF bag. This results in a signature graph of $19+2m$ statements.

C) Minimum Runtime Complexity Using the blank node labeling approach and incremental hash function of Sayers and Karp [10] leads to a minimum runtime complexity. Since both functions have a runtime complexity of $O(n)$, the runtime complexity of the graph signing function is also $O(n)$. Each statement of a graph can contain no, one, or two blank nodes. A blank node is part of at least one statement. Thus, the graph can contain at most twice as many blank nodes as statements. Thus, it is $b \leq 2n$. This leads to a space complexity of $O(n)$ of the graph signing function. The total overhead of the signing process consists of b statements added by the blank node labeling algorithm and 19 statements created by the assembly function for the signature graph S.

7 Implementation and Use of the Signing Framework

This section describes how the previously defined graph signing function σ_N and assembly function ς_N are applied. The description is structured along the scenario given in Section 2. The implementation of our signing framework is conducted in Java and uses as output format an extension of the TriG syntax for named graphs [13].

In Section 7.1, we present the application of the mathematical functions defined in Section 5. Subsequently, we present extensive examples based on TriG for signing graph data. The examples are presented along the functional requirements defined in Section 4: The signing of OWL graphs is shown in Section 7.2. The section also demonstrates signing graph data at different levels of granularity (**REQ-1**) and signing A-box and T-box statements (**REQ-3**). In Section 7.3, the signing of Named Graphs is shown (**REQ-2**) and Section 7.4 describes iterative signing of graph data (**REQ-4**). The signing of multiple and distributed graphs (**REQ-5**) is demonstrated in Section 7.5. Overall, the examples show that our approach fulfills all functional requirements stated in Section 4. The support for the two non-functional security requirements on the authenticity of the graph data (**REQ-A**) and the integrity of the graph data (**REQ-B**) is argued in Section 7.6.

7.1 Using the Signing Function σ_N and Assembly Function ς_N

In order to sign an OWL graph such as $G_{\texttt{bka-1}} \in \mathbb{G}$ from the BKA, it has to be interpreted as a Named Graph $NG_{\texttt{bka-1}} = (\varepsilon, (\emptyset, \emptyset), G_{\texttt{bka-1}})$ with $NG_{\texttt{bka-1}} \in \mathbb{G}_N$. Signing the graph $G_{\texttt{bka-1}}$ with the graph signing function σ_N and the secret key $k_{s_{\texttt{bka}}}$ is $\sigma_N(k_{s_{\texttt{bka}}}, \{NG_{\texttt{bka-1}}\}) = \sigma_N(k_{s_{\texttt{bka}}}, \{(\varepsilon, (\emptyset, \emptyset), G_{\texttt{bka-1}})\}) = s_{\texttt{bka-1}}$. The signature value $s_{\texttt{bka-1}}$ is added to a signature graph $S_{\texttt{bka-1}} \in \mathbb{G}$ by applying the assembly function ς_N on the key $k_{s_{\texttt{bka}}}$ and the graph $NG_{\texttt{bka-1}}$. It is $\varsigma_N(k_{s_{\texttt{bka}}}, \{NG_{\texttt{bka-1}}\}) = (\texttt{sg-bka-1}, S_{\texttt{bka-1}}, \{NG_{\texttt{bka-1}}\}) = NG_{\texttt{sg-bka-1}}$ with $S_{\texttt{bka-1}} = (V_{S_{\texttt{bka-1}}}, E_{S_{\texttt{bka-1}}})$ and $s_{\texttt{bka-1}} \in V_{S_{\texttt{bka-1}}}$. The resulting Named Graph $NG_{\texttt{sg-bka-1}} \in$

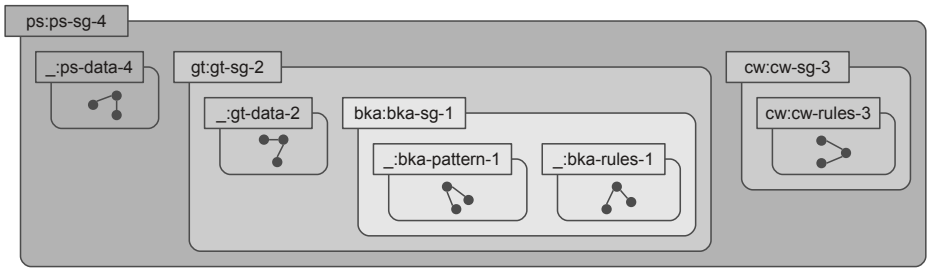

Fig. 3. Examples of iteratively signed graphs.

$\mathbb{G}_N$ is identified by `sg-bka-1`. It has the signature graph $S_{\texttt{bka-1}}$ as annotation graph and $NG_{\texttt{bka-1}} = (\varepsilon, (\emptyset, \emptyset), G_{\texttt{bka-1}})$ as content graph. Signing a Named Graph like $NG_{\texttt{cw-1}} \in \mathbb{G}_N$ from ContentWatch with the key $k_{s_{cw}}$ results in the signature value $s_{\texttt{cw-1}} = \sigma_N(k_{s_{cw}}, \{NG_{\texttt{cw-1}}\})$. Again, $s_{\texttt{cw-1}}$ is included in a signature graph $S_{\texttt{cw-1}}$, which is part of a new Named Graph $NG_{\texttt{sg-cw-1}}$. Using ς_N, it is $\varsigma_N(k_{s_{cw}}, \{NG_{\texttt{cw-1}}\}) = (\texttt{sg-cw-1}, S_{\texttt{cw-1}}, \{NG_{\texttt{cw-1}}\}) = NG_{\texttt{sg-cw-1}}$.

Iteratively signing a Named Graph corresponds to signing the result of the assembly function ς_N. Signing both the Named Graph $NG_{\texttt{sg-bka-1}}$ from above and the RDF graph $G_{\texttt{gt-1}} \in \mathbb{G}$ from the German Telecom with the secret key $k_{s_{gt}}$ results in the signature value $s_{\texttt{gt-1}} = \sigma_N(k_{s_{gt}}, \{NG_{\texttt{sg-bka-1}}, (\varepsilon, (\emptyset, \emptyset), G_{\texttt{gt-1}})\})$. Applying ς_N leads to a new Named Graph $NG_{\texttt{sg-gt-1}} = (\texttt{sg-gt-1}, S_{\texttt{gt-1}}, \{NG_{\texttt{sg-bka-1}}, (\varepsilon, (\emptyset, \emptyset), G_{\texttt{gt-1}})\})$ with $S_{\texttt{gt-1}} \in \mathbb{G}$ being the signature graph of the signature value $s_{\texttt{gt-1}}$. Signing multiple Named Graphs at once is done by applying the signature function on a set of Named Graphs. Signing both the Named Graphs $NG_{\texttt{sg-gt-1}}$ and $NG_{\texttt{sg-cw-1}}$ with the secret key $k_{s_{ps}}$ leads to $\sigma_N(k_{s_{ps}}, \{NG_{\texttt{sg-gt-1}}, NG_{\texttt{sg-cw-1}}\}) = s_{\texttt{ps-1}}$. Applying the assembly function leads to a new Named Graph $NG_{\texttt{sg-ps-1}}$ which contains both the graphs $NG_{\texttt{sg-gt-1}}$ and $NG_{\texttt{sg-cw-1}}$ as its content graphs and a new signature graph $S_{\texttt{ps-1}} \in \mathbb{G}$ as its annotation graph. It is $NG_{\texttt{sg-ps-1}} = (\texttt{sg-ps-1}, S_{\texttt{ps-1}}, \{NG_{\texttt{sg-gt-1}}, NG_{\texttt{sg-cw-1}}\})$.

The following examples are based on an extension of TriG [13] that supports nested Named Graphs. This allows for directly mapping the recursive definition of Named Graphs given in Section 5.1 to an intuitive format. An alternative for using TriG would be storing each signed graph in a separate file and referring to it by its URL. However, this would complicate the signature's verification. Fig. 3 shows the graph created in the scenario of Section 2. The graph has different parts signed by different parties. Each part is created by applying the graph signing function and the assembly function. In the following, we demonstrate the signing process for each party. Since the complete graph forms a nested structure, the examples are based on each other. The result after each signing iteration is shown using an excerpt of their representation in TriG. The complete example is available from our homepage referenced in the conclusion. The examples are based on configuration B) from Section 6, which results in a minimum signature overhead.

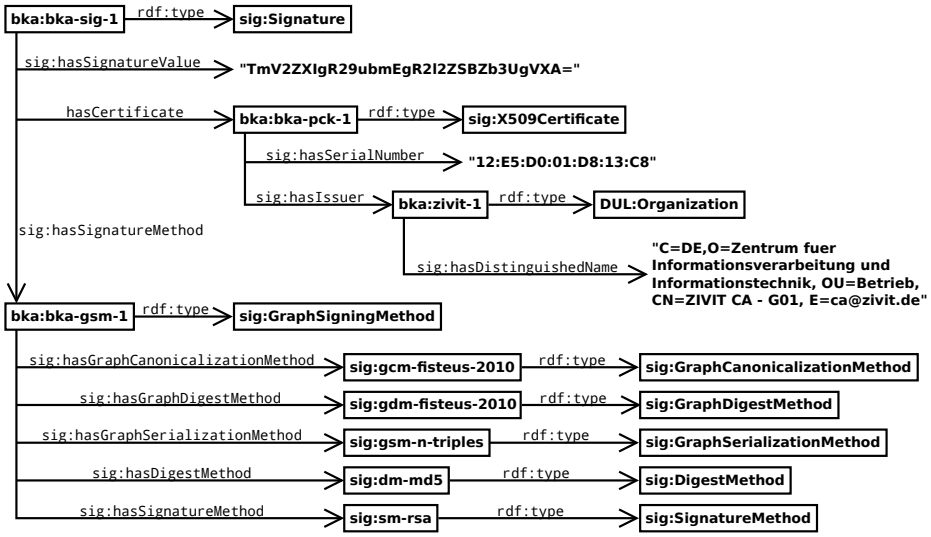

Fig. 4. Example signature graph following the XML signature standard [21].

7.2 Example 1: Signing an OWL Graph

In the first step of the scenario, the BKA creates an ontology design pattern (**REQ-1**) for describing web sites to be blocked according to §86 of the German Criminal Code. Using this pattern, the BKA compiles a list of particular web sites and encodes it as an OWL graph (**REQ-2**). It then signs the list along with the used regulation ontology design pattern. Listing 1 depicts a fragment of the resulting graph. The graph contains the regulation ontology design pattern, the list of blocked web sites, and a signature graph. The regulation ontology design pattern covers T-box knowledge of the BKA (**REQ-3**). It is modeled as a separate graph `_:bka-pattern-1` and shown in lines 27 to 49. On the other hand, the list of blocked web sites covers the A-box knowledge of the BKA. It is also modeled as a separate graph `_:bka-rules-1` and shown in lines 51 to 71. Signing both `_:bka-pattern-1` and `_:bka-rules-1` results in the Named Graph `bka:bka-sg-1` and a signature graph. `bka:bka-sg-1` contains the graphs `_:bka-pattern-1` and `_:bka-rules-1` as its content graphs and the signature graph as its annotation graph. The graph `bka:bka-sg-1` is shown in lines 10 to 72 and the signature statements are shown in lines 12 to 25. `bka:bka-sg-1` and its two content graphs `_:bka-pattern-1` and `_:bka-rules-1` are also shown in Fig. 3 as part of the graph `ps:ps-sg-4`.

The complete signature graph created by the assembly function ς_N is depicted in Fig. 4. The signature is defined in a vocabulary following the XML signature standard [21]. The vocabulary is available from our homepage, referenced in the conclusion. The signature graph stores the computed signature `bka:bka-sig-1` and its signature value. Furthermore, the signature graph also stores all parameters of the graph signing function σ_N required for verifying this value. In

the signature graph, the function σ_N is identified as `bka:bka-gsm-1` and linked to all its subfunctions. As explained in Section 5.8, this includes the names of the used graph canonicalization function `sig:gcm-fisteus-2010`, the graph serialization function `sig:gsm-n-triples`, the hash function (also called digest function) `sig:dm-md5`, the graph hashing function `sig:gdm-fisteus-2010`, and the signature function `sig:sm-rsa`. In order to be able to verify the signature, the signature graph contains a reference to the BKA's public key certificate. The certificate contains the corresponding public key of the secret key k_s, which was used as the signature key. The certificate is represented as `bka:bka-pck-1` and corresponds to an X.509 certificate [31]. It was issued by an organization identified as `bka:zivit-1`. X.509 certificates are uniquely identified by their serial number and the distinguished name [32] of their issuer. Thus, the signature graph contains these identifiers of the BKA's certificate.

```
@prefix bka: <http://icp.it-risk.iwvi.uni-koblenz.de/policies/bka-graph#> .
@prefix DUL: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> .
@prefix flow: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/flow_control.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix proxy: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/proxy_flow_control.owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sig: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/signature.owl#> .
@prefix tec: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/technical_regulation.owl#> .

bka:bka-sg-1 {

  bka:bka-sig-1
    sig:hasGraphSigningMethod bka:bka-gsm-1 ;
    sig:hasSignatureValue "TmV2ZXIgR29ubmEgR2l2ZSBZb3UgVXA=" ;
    sig:hasVerificationCertificate bka:bka-pck-1 ;
    a sig:Signature .

  bka:bka-gsm-1
    sig:hasDigestMethod sig:dm-md5 ;
    sig:hasGraphCanonicalizationMethod sig:gcm-fisteus-2010 ;
    sig:hasGraphDigestMethod sig:gdm-fisteus-2010 ;
    sig:hasGraphSerializationMethod sig:gsm-n-triples ;
    sig:hasSignatureMethod sig:sm-rsa ;
    a sig:GraphSigningMethod .
  ...

  _:bka-pattern-1 {

  proxy:URLBlockingRuleMethod
    a owl:Class ;
    rdfs:subClassOf flow:DenyingFlowControlRuleMethod, [
      a owl:Restriction ;
      owl:allValuesFrom proxy:URLBlockingRuleSituation ;
      owl:onProperty DUL:isSatisfiedBy
    ], [
      a owl:Restriction ;
      owl:onProperty DUL:defines ;
      owl:someValuesFrom [
        a owl:Class ;
        owl:intersectionOf (tec:EnforcingSystem
          [
            a owl:Restriction ;
            owl:onProperty DUL:classifies ;
            owl:someValuesFrom tec:ProxyServer
          ]
        )
      ]
    ] .
```

```
49  }

51  _:bka-rules-1 {

53    bka:naq-1
54      DUL:hasRegion bka:ipr-1 ;
55      a tec:NetworkAddressQuality .

57    bka:uq-1
58      DUL:hasRegion bka:ur-1 ;
59      a tec:URLQuality .

61    bka:ur-1
62      tec:hasURL "http://www.stormfront.org/" ;
63      DUL:hasSetting bka:ri-1 ;
64      a tec:URLRegion .

66    bka:wst-1
67      DUL:hasQuality bka:uq-1 ;
68      DUL:hasSetting bka:ri-1 ;
69      a tec:WebSite .
70    ...
71  }
72 }
```

Listing 1. Example of a signed RDF graph.

7.3 Example 2: Signing a Named Graph

In the scenario, ContentWatch compiles a blacklist of web sites providing adult content. This list is used for regulating access to the web sites and encoded as Named Graph (**REQ-2**). Signing a Named Graph is similar to signing an RDF/OWL graph. Listing 2 depicts the signed Named Graph created by ContentWatch. The blacklist of web sites is identified as `cw:cw-rules-3` and shown in lines 10 to 30. Signing `cw:cw-rules-3` results in several signature statements which are shown in lines 3 to 8. The statements cover the used graph signing function `cw:cw-gsm-3` (line 4), the created signature value (line 5), and ContentWatch's public key certificate `cw:cw-pck-3` (line 6). The signature statements and the Named Graph `cw:cw-rules-3` are part of the newly created Named Graph `cw:cw-sg-3` shown in lines 1 to 31. `cw:cw-sg-3` contains the signature statements as its annotation graph and `cw:cw-rules-3` as its content graph.

```
1  cw:cw-sg-3 {

3    cw:cw-sig-3
4      sig:hasGraphSigningMethod cw:cw-gsm-3 ;
5      sig:hasSignatureValue "SXQncyBibHVlIGxpZ2h0" ;
6      sig:hasVerificationCertificate cw:cw-pck-3 ;
7      a sig:Signature .
8    ...

10   cw:cw-rules-3 {

12     cw:naq-3
13       DUL:hasRegion cw:ipr-3 ;
14       a tec:NetworkAddressQuality .

16     cw:uq-3
17       DUL:hasRegion cw:ur-3 ;
18       a tec:URLQuality .
```

```
20     cw:ur-3
21       tec:hasURL "http://www.youporn.com/" ;
22       DUL:hasSetting cw:ri-3 ;
23       a tec:URLRegion .

25     cw:wst-3
26       DUL:hasQuality cw:uq-3 ;
27       DUL:hasSetting cw:ri-3 ;
28       a tec:WebSite .
29     ...
30   }
31 }
```

Listing 2. Example of a signed Named Graph.

7.4 Example 3: Iteratively Signing of Graphs

The German Telecom receives the Named Graph `bka:bka-sg-1`, which is signed by the BKA. `bka:bka-sg-1` contains general regulation information but does not describe how the regulations shall be implemented by the ISP. Thus, the German Telecom adds its own RDF graph `_:gt-data-2` with detailed regulation information including proxy servers and their IP addresses. It then signs the RDF graph `_:gt-data-2` together with the received Named Graph `bka:bka-sg-1` (**REQ-4**). The resulting Named Graph `gt:gt-sg-2` is depicted in Listing 3. It contains the created signature statements (lines 3 to 8), the RDF graph created by the German Telecom `_:gt-data-2` (lines 10 to 25), and the BKA's Named Graph `bka:bka-sg-1` (lines 27 to 38). The signature statements cover the graph signing function `gt:gt-gsm-2` used for signing the graph data (line 4), the resulting signature value (line 5), and the public key certificate `gt:gt-pck-2` of the German Telecom (line 6). The Named Graph `gt:gt-sg-2` contains the signature statements as its annotation graph and the two graphs `_:gt-data-2` and `bka:bka-sg-1` as its content graphs.

```
1  gt:gt-sg-2 {

3    gt:gt-sig-2
4      sig:hasGraphSigningMethod gt:gt-gsm-2 ;
5      sig:hasSignatureValue "YXJlIGJlbG9uZyB0byB1cw==" ;
6      sig:hasVerificationCertificate gt:gt-pck-2 ;
7      a sig:Signature .
8    ...

10   _:gt-data-2 {

12     gt:ipr-2
13       tec:hasIPAddress "141.26.83.115" ;
14       tec:hasSubnetMask "255.255.0.0" ;
15       DUL:hasSetting bka:pi-1, bka:ri-1 ;
16       a tec:IPv4AddressRegion .

18     gt:naq-2
19       DUL:hasRegion gt:ipr-2 ;
20       a tec:NetworkAddressQuality .

22     bka:pr-1
23       DUL:hasQuality gt:naq-2 .
24     ...
25   }
```

```
27   bka:bka-sg-1 {

29     bka:bka-gsm-1
30       sig:hasGraphSigningMethod bka:bka-gsm-1 ;
31       sig:hasSignatureValue "TmV2ZXIgR29ubmEgR2l2ZSBZb3UgVXA=" ;
32       sig:hasVerificationCertificate bka:bka-pck-1 ;
33       a sig:Signature .
34     ...

36     _:bka-pattern-1 { ... }
37     _:bka-rules-1 { ... }
38   }
39 }
```

Listing 3. Example of iteratively signed graphs.

7.5 Example 4: Signing Multiple, Distributed Graphs

The last party in the scenario of Section 2 is the primary school. It retrieves the Named Graph `gt:gt-sg-2` from the German Telecom and the Named Graph `cw:cw-sg-3` from ContentWatch. In order to enrich the generic information encoded in `cw:cw-sg-3` with specific regulation details, the school adds its own regulation data as RDF graph `_:ps-data-4`. This includes proxy servers run by the school. The school then signs the graph `_:ps-data-4` together with the two graphs `cw:cw-sg-3` and `cw:cw-sg-3` (**REQ-5**). This results in the Named Graph `ps:ps-sg-4` which is depicted in Listing 4. `ps:ps-sg-4` contains the graphs `_:ps-data-4`, `gt:gt-sg-2`, and `cw:cw-sg-3` as its content graphs and the school's signature statements as its annotation graph. The graph `_:ps-data-4` is shown in lines 10 to 25, the German Telecom's graph `gt:gt-sg-2` is shown in lines 27 to 50 and ContentWatch's Named Graph `cw:cw-sg-3` is shown in lines 52 to 62. The school's signature graph is shown in lines 3 to 8 and contains the used graph signing function `ps:ps-gsm-4` (line 4), the created signature value (line 5), and the school's public key certificate `ps:ps-pck-4` (line 6).

```
1  ps:ps-sg-4 {

3    ps:ps-sig-4
4      sig:hasGraphSigningMethod ps:ps-gsm-4 ;
5      sig:hasSignatureValue "QWxsIHlvdXIgYmFzZSBhcmU=" ;
6      sig:hasVerificationCertificate ps:ps-pck-4 ;
7      a sig:Signature .
8    ...

10   _:ps-data-4 {

12     ps:ipr-4
13       tec:hasIPAddress "141.26.83.116" ;
14       tec:hasSubnetMask "255.255.0.0" ;
15       DUL:hasSetting cw:pi-3, cw:ri-3 ;
16       a tec:IPv4AddressRegion .

18     ps:naq-4
19       DUL:hasRegion ps:ipr-4 ;
20       a tec:NetworkAddressQuality .

22     cw:pr-3
23       DUL:hasQuality ps:naq-4 .
```

```
24      ...
25    }

27    gt:gt-sg-2 {

29      gt:gt-sig-2
30        sig:hasGraphSigningMethod gt:gt-gsm-2 ;
31        sig:hasSignatureValue "YXJlIGJlbG9uZyB0byB1cw==" ;
32        sig:hasVerificationCertificate gt:gt-pck-2 ;
33        a sig:Signature .
34      ...

36      _:gt-data-2 { ... }

38      bka:bka-sg-1 {

40        bka:bka-sig-1
41          sig:hasGraphSigningMethod bka:bka-gsm-1 ;
42          sig:hasSignatureValue "TmV2ZXIgR29ubmEgR2l2ZSBZb3UgVXA=" ;
43          sig:hasVerificationCertificate bka:bka-pck-1 ;
44          a sig:Signature .
45        ...

47        _:bka-pattern-1 { ... }
48        _:bka-rules-1 { ... }
49      }
50    }

52    cw:cw-sg-3 {

54      cw:cw-sig-3
55        sig:hasGraphSigningMethod cw:cw-gsm-3 ;
56        sig:hasSignatureValue "SXQncyBibHVlIGxpZ2h0" ;
57        sig:hasVerificationCertificate cw:cw-pck-3 ;
58        a sig:Signature .
59      ...

61      cw:cw-rules-3 { ... }
62    }
63  }
```

Listing 4. Example of multiple signed graphs.

7.6 Fulfillment of Security Requirements

The previous sections have demonstrated the fulfillment of the functional requirements given in Section 4 by the use of different examples. The fulfillment of the security requirements regarding authenticity of graph data (**REQ-A**) and integrity of graph data (**REQ-B**) depends on the configuration of the used graph signing function σ_N. A comprehensive security analysis of the graph signing function must consider all of its possible configurations. However, the serialization function ν_N, the basic hash function λ, and the signature function ϵ can generally be used in any configuration. Thus, a security analysis of these functions can be directly transferred to any specific graph signing function and graph signing configuration, respectively. Since the main difference between the configurations presented in Section 6 are the canonicalization function κ_N and the hash function for graphs λ_N, only these functions must be further considered for a security analysis of a specific configuration.

The serialization function ν_N transforms a graph into a set of bit strings. Since this transformation does not require any cryptographic operations, the serialization function does not affect the security of the graph signing function. The cryptographic strength of the basic hash function λ determines the difficulty of modifying the signed graph data without being noticed by the verification mechanism. The more collision-resistant the chosen hash function is, the less likely are unauthorized modifications on the graph data. The National Institute of Standards and Technology (NIST) recommends the use of SHA 2 [16] with an output length of 256 bits [33]. The signature function ϵ determines the difficulty for an attacker masquerading as another party. A cryptographically strong signature function prohibits such attacks. NIST recommends the use of the RSA algorithm [5] with a key length of 2048 bits.

Analysis of Canonicalization Functions for Graphs The canonicalization functions by Carroll [6] and Sayers and Karp [10] as used in configurations A and C do not use any cryptographic operations. These functions are based on sorting the statements of the graph and/or inserting additional statements into the canonicalized graph. These operations only require the graph to be signed as input. They neither require any secret input values such as private signature keys nor must they fulfill any particular security requirements such as collision-resistance. The canonicalization function by Fisteus et al. [8] used in configuration B is based on a hash function for graphs by the same authors. The canonicalization function sorts the hash values computed from the hash function in order to produce a canonicalized graph. As explained above, sorting does not rely on any cryptographic functions. However, the hash function of Fisteus et al. is based on cryptographic operations. Thus, the security of the canonicalization function depends on the security of the hash function for graphs which is further analyzed below.

Analysis of Hash Functions for Graphs The hash function for graphs by Carroll [6] as used in configuration A basically corresponds to sorting the serialized statements of a graph with a basic hash function. Thus, the security of this function solely relies on the security of the used basic hash function. As recommended by NIST [33], SHA-2 [16] with an output length of 256 bits can be used as basic hash function. The hash function for graphs by Sayers and Karp [10] used in configuration C computes the hash values of each statement in the graph separately from each other and combines them into a single value using a combining function. Computing the hash value of a statement is done by using a basic hash function. The overall security of the hash function for graphs thus relies on the used basic hash function as well as on the combining function. For the basic hash function, SHA-2 [16] with an output length of 256 bits can be used as recommended by NIST [33]. For the combining function, Sayers and Karp suggest two different alternatives which are *AdHASH* and *MuHASH* as introduced by Bellare and Micciancio [34]. AdHASH adds all hash values to be combined modulo a large number m and MuHASH multiplies the hash values modulo a large prime p. As identified by Wagner [35], m must be chosen such

that $m \gg 2^{1600}$ in order to ensure 80 bit security. Wagner also states that this would reduce the performance of the combining function. On the other hand, the security of MuHASH is based on the discrete logarithm problem which is proven to be hard to solve [34]. The size of p generally depends on the application in which the combining function is used. For signing graph data, one can choose a prime p with a length of at least 1024 bits.

The hash function for graphs by Fisteus et al. [8] as used in configuration B computes a graph's hash value by combining the hash values of all its statements using a combining function. Thus, the overall security of this function is based on the used combining function as well as on the function for creating the statement's hash values. Fisteus et al. use MuHASH [34] as the combining function. The hash value of each statement is computed in four steps. First, each part of the statement is hashed separately from each other using a basic hash function λ. Second, the hash value of each part is multiplied with a different constant modulo a large prime p. The constant defines the position of the hashed part within the statement. Third, the results of the modulo operation are combined using XOR. Finally, a modulo operation is performed again on the XOR results using the same prime p. This computation can basically be considered a variant of MuHASH. Using $(subj, pred, obj)$ as the statement to be hashed, the hash function h is defined as follows:

$$h((subj, pred, obj)) = (((\lambda(subj) \cdot k_{subj}) \bmod p) \oplus ((\lambda(pred) \cdot k_{pred}) \bmod p) \oplus ((\lambda(obj) \cdot k_{obj}) \bmod p)) \bmod p \quad (13)$$

The pre-defined constants k_{subj}, k_{pred}, and k_{obj} mark the position of a statement's part. Although this computation can be considered secure for a large prime p, it is too complicated to be used for large graphs to be signed. An easier approach for computing the hash value of a statement is presented by Melnik [17]. Melnik computes a statement's hash value by concatenating the hash value of its subject, predicate, and object and hashing the result. The security of this hash value is solely based on the security of the used hash function. As recommended by NIST [33], SHA-2 [16] with an output length of 256 bits can be used as such a hash function.

8 Conclusion

In this paper, we have presented a formally defined framework for iteratively signing different types of graph data such as RDF(S) graphs, OWL graphs, and Named Graphs. The framework allows for signing multiple and distributed graphs and supports signing A-box and T-box knowledge. It also allows for signing different kinds of granularity such as single triples, ontology design patterns, and whole graphs. We have discussed three different possible configurations of the signing process and shown its practical applicability based on an extension of TriG [13].

The complete examples as well as the ontology used for modeling the signature graphs are available online. They can be found at our homepage: `http://icp.it-risk.iwvi.uni-koblenz.de/wiki/Signing_Graphs`.

References

1. Tummarello, G., Morbidoni, C., Puliti, P., Piazza, F.: Signing individual fragments of an RDF graph. In: WWW, ACM (2005) 1020–1021
2. Bundesrepublik Deutschland: §86 StGB (1975) `http://www.gesetze-im-internet.de/stgb/__86.html`.
3. Gangemi, A., Presutti, V.: Ontology design patterns. In: Handbook on Ontologies. Springer (2009) 221–243
4. Schneier, B.: Protocol Building Blocks. [22] 21–46
5. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. CACM **21** (1978) 120–126
6. Carroll, J.J.: Signing RDF graphs. In: ISWC 2003, Springer (2003) 369–384
7. Beckett, D.: N-Triples. W3C (2001) `http://www.w3.org/2001/sw/RDFCore/ntriples/`.
8. Fisteus, J.A., García, N.F., Fernández, L.S., Kloos, C.D.: Hashing and canonicalizing Notation 3 graphs. JCSS **76** (2010) 663–685
9. Berners-Lee, T., Connolly, D.: Notation3 (N3). W3C (2011) `http://www.w3.org/TeamSubmission/n3/`.
10. Sayers, C., Karp, A.H.: Computing the digest of an RDF graph. Technical report, HP Laboratories (2004)
11. Beckett, D.: RDF/XML syntax specification. W3C (2004) `http://www.w3.org/TR/rdf-syntax-grammar/`.
12. Motik, B., Parsia, B., Patel-Schneider, P.F.: OWL 2 web ontology language XML serialization. W3C (2009) `http://www.w3.org/TR/owl2-xml-serialization/`.
13. Bizer, C., Cyganiak, R.: TriG: RDF Dataset Language. Technical report, W3C (2013) `http://www.w3.org/TR/trig/`.
14. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Web Semantics: Science, Services and Agents on the World Wide Web **19** (2013) 22–41
15. Rivest, R.: The MD5 message-digest algorithm. RFC 1321, IETF (1992)
16. NIST: Secure hash standard. FIPS PUB 180-4 (2012) `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`.
17. Melnik, S.: RDF API draft (2001) `http://infolab.stanford.edu/~melnik/rdf/api.html`.
18. NIST: Digital signature standard (DSS). FIPS PUB 186-3 (2009) `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.
19. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE TIT **31** (1985) 469–472
20. Sayers, C., Karp, A.H.: RDF graph digest techniques and potential applications. Technical Report HPL-2004-95, HP Laboratories, Palo Alto (2004)
21. Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E.: XML signature syntax and processing. W3C (2008) `http://www.w3.org/TR/xmldsig-core/`.
22. Schneier, B.: Applied Cryptography. John Wiley & Sons (1996)
23. Freier, A.O., Karlton, P., Kocher, P.C.: The secure sockets layer (SSL) protocol version 3.0. RFC 6101, IETF (2011)

24. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening ontologies with DOLCE. In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, Springer (2002) 223–233
25. Schneier, B.: Security Needs. In: Secrets and Lies: Digital Security in a Networked World. Wiley Publishing (2004) 59–84
26. Arenas, M., Gutierrez, C., Pérez, J.: Foundations of RDF databases. In: Reasoning Web. Semantic Technologies for Information Systems, Springer (2009) 158–204
27. Christodoulou, K., Paton, N.W., Fernandes, A.A.A.: Structure inference for linked data sources using clustering. In: EDBT 2013, ACM (2013) 60–67
28. Patel-Schneider, P.F., Motik, B.: OWL 2 web ontology language mapping to RDF graphs. Technical report, W3C (2012)
29. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs, provenance and trust. In: WWW, ACM (2005) 613–622
30. Beckett, D., Berners-Lee, T.: Turtle. W3C (2011) `http://www.w3.org/TeamSubmission/turtle/`.
31. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, T.: Internet X.509 public key infrastructure. RFC 5280, IETF (2008)
32. Zeilenga, K.D.: Lightweight directory access protocol (LDAP). RFC 4514, IETF (2006)
33. NIST: Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. SP 800-131A (2011) `http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf`.
34. Bellare, M., Micciancio, D.: A new paradigm for collision-free hashing: Incrementality at reduced cost. In: EUROCRYPT 1997, Springer (1997) 163–192
35. Wagner, D.: A generalized birthday problem. In: CRYPTO 2002, Springer (2002) 288–304

Bisher erschienen

Arbeitsberichte aus dem Fachbereich Informatik

(http://www.uni-koblenz-landau.de/koblenz/fb4/publications/Reports/arbeitsberichte)

Andreas Kasten, Ansgar Scherp, Iterative Signing of RDF(S) Graphs, Named Graphs, and OWL Graphs: Formalization and Application, Arbeitsberichte aus dem Fachbereich Informatik 3/2013

Thomas Gottron, Johann Schaible, Stefan Scheglmann, Ansgar Scherp, LOVER: Support for Modeling Data Using Linked Open Vocabularies, Arbeitsberichte aus dem Fachbereich Informatik 2/2013

Markus Bender, E-Hyper Tableaux with Distinct Objects Identifiers, Arbeitsberichte aus dem Fachbereich Informatik 1/2013

Kurt Lautenbach, Kerstin Susewind, Probability Propagation Nets and Duality, Arbeitsberichte aus dem Fachbereich Informatik 11/2012

Kurt Lautenbach, Kerstin Susewind, Applying Probability Propagation Nets, Arbeitsberichte aus dem Fachbereich Informatik 10/2012

Kurt Lautenbach, The Quaternality of Simulation: An Event/Non-Event Approach, Arbeitsberichte aus dem Fachbereich Informatik 9/2012

Horst Kutsch, Matthias Bertram, Harald F.O. von Kortzfleisch, Entwicklung eines Dienstleistungsproduktivitätsmodells (DLPMM) am Beispiel von B2b Software-Customizing, Fachbereich Informatik 8/2012

Rüdiger Grimm, Jean-Noël Colin, Virtual Goods + ODRL 2012, Arbeitsberichte aus dem Fachbereich Informatik 7/2012

Ansgar Scherp, Thomas Gottron, Malte Knauf, Stefan Scheglmann, Explicit and Implicit Schema Information on the Linked Open Data Cloud: Joined Forces or Antagonists? Arbeitsberichte aus dem Fachbereich Informatik 6/2012

Harald von Kortzfleisch, Ilias Mokanis, Dorothée Zerwas, Introducing Entrepreneurial Design Thinking, Arbeitsberichte aus dem Fachbereich Informatik 5/2012

Ansgar Scherp, Daniel Eißing, Carsten Saathoff, Integrating Multimedia Metadata Standarts and Metadata Formats with the Multimedia Metadata Ontology: Method and Examples, Arbeitsberichte aus dem Fachbereich Informatik 4/2012

Martin Surrey,Björn Lilge, Ludwig Paulsen, Marco Wolf, Markus Aldenhövel, Mike Reuthel, Roland Diehl, Integration von CRM-Systemen mit Kollaborations-Systemen am Beispiel von DocHouse und Lotus Quickr, Arbeitsberichte aus dem Fachbereich Informatik 3/2012

Martin Surrey, Roland Diehl, DOCHOUSE: Opportunity Management im Partnerkanal (IBM Lotus Quickr), Arbeitsberichte aus dem Fachbereich Informatik 2/2012

Mark Schneider, Ansgar Scherp, Comparing a Grid-based vs. List-based Approach for Faceted Search of Social Media Data on Mobile Devices, Arbeitsberichte aus dem Fachbereich Informatik 1/2012

Petra Schubert, Femi Adisa, Cloud Computing for Standard ERP Systems: Reference Framework and Research Agenda, Arbeitsberichte aus dem Fachbereich Informatik 16/2011

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, Natalia A. Zenkova, Simulating social objects with an artificial network using a computer cluster, Arbeitsberichte aus dem Fachbereich Informatik 15/2011

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, Simulating medical objects using an artificial network whose structure is based on adaptive resonance theory, Arbeitsberichte aus dem Fachbereich Informatik 14/2011

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, Comparing the efficiency of serial and parallel algorithms for training artificial neural networks using computer clusters, Arbeitsberichte aus dem Fachbereich Informatik, 13/2011

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, A parallel algorithm for selecting activation functions of an artificial network, Arbeitsberichte aus dem Fachbereich Informatik 12/2011

Katharina Bräunlich, Rüdiger Grimm, Andreas Kasten, Sven Vowé, Nico Jahn, Der neue Personalausweis zur Authentifizierung von Wählern bei Onlinewahlen, Arbeitsberichte aus dem Fachbereich Informatik 11/2011

Daniel Eißing, Ansgar Scherp, Steffen Staab, Formal Integration of Individual Knowledge Work and Organizational Knowledge Work with the Core Ontology *strukt*, Arbeitsberichte aus dem Fachbereich Informatik 10/2011

Bernhard Reinert, Martin Schumann, Stefan Müller, Combined Non-Linear Pose Estimation from Points and Lines, Arbeitsberichte aus dem Fachbereich Informatik 9/2011

Tina Walber, Ansgar Scherp, Steffen Staab, Towards the Understanding of Image Semantics by Gaze-based Tag-to-Region Assignments, Arbeitsberichte aus dem Fachbereich Informatik 8/2011

Alexander Kleinen, Ansgar Scherp, Steffen Staab, Mobile Facets – Faceted Search and Exploration of Open Social Media Data on a Touchscreen Mobile Phone, Arbeitsberichte aus dem Fachbereich Informatik 7/2011

Anna Lantsberg, Klaus G. Troitzsch, Towards A Methodology of Developing Models of E-Service Quality Assessment in Healthcare, Arbeitsberichte aus dem Fachbereich Informatik 6/2011

Ansgar Scherp, Carsten Saathoff, Thomas Franz, Steffen Staab, Designing Core Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 5/2011

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch, The prediction of currency exchange rates using artificial neural networks, Arbeitsberichte aus dem Fachbereich Informatik 4/2011

Klaus G. Troitzsch, Anna Lantsberg, Requirements for Health Care Related Websites in Russia: Results from an Analysis of American, British and German Examples, Arbeitsberichte aus dem Fachbereich Informatik 3/2011

Klaus G. Troitzsch, Oleg Kryuchin, Alexander Arzamastsev, A universal simulator based on artificial neural networks for computer clusters, Arbeitsberichte aus dem Fachbereich Informatik 2/2011

Klaus G. Troitzsch, Natalia Zenkova, Alexander Arzamastsev, Development of a technology of designing intelligent information systems for the estimation of social objects, Arbeitsberichte aus dem Fachbereich Informatik 1/2011

Kurt Lautenbach, A Petri Net Approach for Propagating Probabilities and Mass Functions, Arbeitsberichte aus dem Fachbereich Informatik 13/2010

Claudia Schon, Linkless Normal Form for ALC Concepts, Arbeitsberichte aus dem Fachbereich Informatik 12/2010

Alexander Hug, Informatik hautnah erleben, Arbeitsberichte aus dem Fachbereich Informatik 11/2010

Marc Santos, Harald F.O. von Kortzfleisch, Shared Annotation Model – Ein Datenmodell für kollaborative Annotationen, Arbeitsberichte aus dem Fachbereich Informatik 10/2010

Gerd Gröner, Steffen Staab, Categorization and Recognition of Ontology Refactoring Pattern, Arbeitsberichte aus dem Fachbereich Informatik 9/2010

Daniel Eißing, Ansgar Scherp, Carsten Saathoff, Integration of Existing Multimedia Metadata Formats and Metadata Standards in the M3O, Arbeitsberichte aus dem Fachbereich Informatik 8/2010

Stefan Scheglmann, Ansgar Scherp, Steffen Staab, Model-driven Generation of APIs for OWL-based Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 7/2010

Daniel Schmeiß, Ansgar Scherp, Steffen Staab, Integrated Mobile Visualization and Interaction of Events and POIs, Arbeitsberichte aus dem Fachbereich Informatik 6/2010

Rüdiger Grimm, Daniel Pähler, E-Mail-Forensik – IP-Adressen und ihre Zuordnung zu Internet-Teilnehmern und ihren Standorten, Arbeitsberichte aus dem Fachbereich Informatik 5/2010

Christoph Ringelstein, Steffen Staab, PAPEL: Syntax and Semantics for Provenance-Aware Policy Definition, Arbeitsberichte aus dem Fachbereich Informatik 4/2010

Nadine Lindermann, Sylvia Valcárcel, Harald F.O. von Kortzfleisch, Ein Stufenmodell für kollaborative offene Innovationsprozesse in Netzwerken kleiner und mittlerer Unternehmen mit Web 2.0, Arbeitsberichte aus dem Fachbereich Informatik 3/2010

Maria Wimmer, Dagmar Lück-Schneider, Uwe Brinkhoff, Erich Schweighofer, Siegfried Kaiser, Andreas Wieber, Fachtagung Verwaltungsinformatik FTVI Fachtagung Rechtsinformatik FTRI 2010, Arbeitsberichte aus dem Fachbereich Informatik 2/2010

Max Braun, Ansgar Scherp, Steffen Staab, Collaborative Creation of Semantic Points of Interest as Linked Data on the Mobile Phone, Arbeitsberichte aus dem Fachbereich Informatik 1/2010

Marc Santos, Einsatz von „Shared In-situ Problem Solving“ Annotationen in kollaborativen Lern- und Arbeitsszenarien, Arbeitsberichte aus dem Fachbereich Informatik 20/2009

Carsten Saathoff, Ansgar Scherp, Unlocking the Semantics of Multimedia Presentations in the Web with the Multimedia Metadata Ontology, Arbeitsberichte aus dem Fachbereich Informatik 19/2009

Christoph Kahle, Mario Schaarschmidt, Harald F.O. von Kortzfleisch, Open Innovation: Kundenintegration am Beispiel von IPTV, Arbeitsberichte aus dem Fachbereich Informatik 18/2009

Dietrich Paulus, Lutz Priese, Peter Decker, Frank Schmitt, Pose-Tracking Forschungsbericht, Arbeitsberichte aus dem Fachbereich Informatik 17/2009

Andreas Fuhr, Tassilo Horn, Andreas Winter, Model-Driven Software Migration Extending SOMA, Arbeitsberichte aus dem Fachbereich Informatik 16/2009

Eckhard Großmann, Sascha Strauß, Tassilo Horn, Volker Riediger, Abbildung von grUML nach XSD soamig, Arbeitsberichte aus dem Fachbereich Informatik 15/2009

Kerstin Falkowski, Jürgen Ebert, The STOR Component System Interim Report, Arbeitsberichte aus dem Fachbereicht Informatik 14/2009

Sebastian Magnus, Markus Maron, An Empirical Study to Evaluate the Location of Advertisement Panels by Using a Mobile Marketing Tool, Arbeitsberichte aus dem Fachbereich Informatik 13/2009

Sebastian Magnus, Markus Maron, Konzept einer Public Key Infrastruktur in iCity, Arbeitsberichte aus dem Fachbereich Informatik 12/2009

Sebastian Magnus, Markus Maron, A Public Key Infrastructure in Ambient Information and Transaction Systems, Arbeitsberichte aus dem Fachbereich Informatik 11/2009

Ammar Mohammed, Ulrich Furbach, Multi-agent systems: Modeling and Virification using Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 10/2009

Andreas Sprotte, Performance Measurement auf der Basis von Kennzahlen aus betrieblichen Anwendungssystemen: Entwurf eines kennzahlengestützten Informationssystems für einen Logistikdienstleister, Arbeitsberichte aus dem Fachbereich Informatik 9/2009

Gwendolin Garbe, Tobias Hausen, Process Commodities: Entwicklung eines Reifegradmodells als Basis für Outsourcingentscheidungen, Arbeitsberichte aus dem Fachbereich Informatik 8/2009

Petra Schubert et. al., Open-Source-Software für das Enterprise Resource Planning, Arbeitsberichte aus dem Fachbereich Informatik 7/2009

Ammar Mohammed, Frieder Stolzenburg, Using Constraint Logic Programming for Modeling and Verifying Hierarchical Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 6/2009

Tobias Kippert, Anastasia Meletiadou, Rüdiger Grimm, Entwurf eines Common Criteria-Schutzprofils für Router zur Abwehr von Online-Überwachung, Arbeitsberichte aus dem Fachbereich Informatik 5/2009

Hannes Schwarz, Jürgen Ebert, Andreas Winter, Graph-based Traceability – A Comprehensive Approach. Arbeitsberichte aus dem Fachbereich Informatik 4/2009

Anastasia Meletiadou, Simone Müller, Rüdiger Grimm, Anforderungsanalyse für Risk-Management-Informationssysteme (RMIS), Arbeitsberichte aus dem Fachbereich Informatik 3/2009

Ansgar Scherp, Thomas Franz, Carsten Saathoff, Steffen Staab, A Model of Events based on a Foundational Ontology, Arbeitsberichte aus dem Fachbereich Informatik 2/2009

Frank Bohdanovicz, Harald Dickel, Christoph Steigner, Avoidance of Routing Loops, Arbeitsberichte aus dem Fachbereich Informatik 1/2009

Stefan Ameling, Stephan Wirth, Dietrich Paulus, Methods for Polyp Detection in Colonoscopy Videos: A Review, Arbeitsberichte aus dem Fachbereich Informatik 14/2008

Tassilo Horn, Jürgen Ebert, Ein Referenzschema für die Sprachen der IEC 61131-3, Arbeitsberichte aus dem Fachbereich Informatik 13/2008

Thomas Franz, Ansgar Scherp, Steffen Staab, Does a Semantic Web Facilitate Your Daily Tasks?, Arbeitsberichte aus dem Fachbereich Informatik 12/2008

Norbert Frick, Künftige Anfordeungen an ERP-Systeme: Deutsche Anbieter im Fokus, Arbeitsberichte aus dem Fachbereicht Informatik 11/2008

Jürgen Ebert, Rüdiger Grimm, Alexander Hug, Lehramtsbezogene Bachelor- und Masterstudiengänge im Fach Informatik an der Universität Koblenz-Landau, Campus Koblenz, Arbeitsberichte aus dem Fachbereich Informatik 10/2008

Mario Schaarschmidt, Harald von Kortzfleisch, Social Networking Platforms as Creativity Fostering Systems: Research Model and Exploratory Study, Arbeitsberichte aus dem Fachbereich Informatik 9/2008

Bernhard Schueler, Sergej Sizov, Steffen Staab, Querying for Meta Knowledge, Arbeitsberichte aus dem Fachbereich Informatik 8/2008

Stefan Stein, Entwicklung einer Architektur für komplexe kontextbezogene Dienste im mobilen Umfeld, Arbeitsberichte aus dem Fachbereich Informatik 7/2008

Matthias Bohnen, Lina Brühl, Sebastian Bzdak, RoboCup 2008 Mixed Reality League Team Description, Arbeitsberichte aus dem Fachbereich Informatik 6/2008

Bernhard Beckert, Reiner Hähnle, Tests and Proofs: Papers Presented at the Second International Conference, TAP 2008, Prato, Italy, April 2008, Arbeitsberichte aus dem Fachbereich Informatik 5/2008

Klaas Dellschaft, Steffen Staab, Unterstützung und Dokumentation kollaborativer Entwurfs- und Entscheidungsprozesse, Arbeitsberichte aus dem Fachbereich Informatik 4/2008

Rüdiger Grimm: IT-Sicherheitsmodelle, Arbeitsberichte aus dem Fachbereich Informatik 3/2008

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik 2/2008

Markus Maron, Kevin Read, Michael Schulze: CAMPUS NEWS – Artificial Intelligence Methods Combined for an Intelligent Information Network, Arbeitsberichte aus dem Fachbereich Informatik 1/2008

Lutz Priese,Frank Schmitt, Patrick Sturm, Haojun Wang: BMBF-Verbundprojekt 3D-RETISEG Abschlussbericht des Labors Bilderkennen der Universität Koblenz-Landau, Arbeitsberichte aus dem Fachbereich Informatik 26/2007

Stephan Philippi, Alexander Pinl: Proceedings 14. Workshop 20.-21. September 2007 Algorithmen und Werkzeuge für Petrinetze, Arbeitsberichte aus dem Fachbereich Informatik 25/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS – an Intelligent Bluetooth-based Mobile Information Network, Arbeitsberichte aus dem Fachbereich Informatik 24/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS - an Information Network for Pervasive Universities, Arbeitsberichte aus dem Fachbereich Informatik 23/2007

Lutz Priese: Finite Automata on Unranked and Unordered DAGs Extented Version, Arbeitsberichte aus dem Fachbereich Informatik 22/2007

Mario Schaarschmidt, Harald F.O. von Kortzfleisch: Modularität als alternative Technologie- und Innovationsstrategie, Arbeitsberichte aus dem Fachbereich Informatik 21/2007

Kurt Lautenbach, Alexander Pinl: Probability Propagation Nets, Arbeitsberichte aus dem Fachbereich Informatik 20/2007

Rüdiger Grimm, Farid Mehr, Anastasia Meletiadou, Daniel Pähler, Ilka Uerz: SOA-Security, Arbeitsberichte aus dem Fachbereich Informatik 19/2007

Christoph Wernhard: Tableaux Between Proving, Projection and Compilation, Arbeitsberichte aus dem Fachbereich Informatik 18/2007

Ulrich Furbach, Claudia Obermaier: Knowledge Compilation for Description Logics, Arbeitsberichte aus dem Fachbereich Informatik 17/2007

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: TwoUse: Integrating UML Models and OWL Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 16/2007

Rüdiger Grimm, Anastasia Meletiadou: Rollenbasierte Zugriffskontrolle (RBAC) im Gesundheitswesen, Arbeitsberichte aud dem Fachbereich Informatik 15/2007

Ulrich Furbach, Jan Murray, Falk Schmidsberger, Frieder Stolzenburg: Hybrid Multiagent Systems with Timed Synchronization-Specification and Model Checking, Arbeitsberichte aus dem Fachbereich Informatik 14/2007

Björn Pelzer, Christoph Wernhard: System Description:"E-KRHyper", Arbeitsberichte aus dem Fachbereich Informatik, 13/2007

Ulrich Furbach, Peter Baumgartner, Björn Pelzer: Hyper Tableaux with Equality, Arbeitsberichte aus dem Fachbereich Informatik, 12/2007

Ulrich Furbach, Markus Maron, Kevin Read: Location based Informationsystems, Arbeitsberichte aus dem Fachbereich Informatik, 11/2007

Philipp Schaer, Marco Thum: State-of-the-Art: Interaktion in erweiterten Realitäten, Arbeitsberichte aus dem Fachbereich Informatik, 10/2007

Ulrich Furbach, Claudia Obermaier: Applications of Automated Reasoning, Arbeitsberichte aus dem Fachbereich Informatik, 9/2007

Jürgen Ebert, Kerstin Falkowski: A First Proposal for an Overall Structure of an Enhanced Reality Framework, Arbeitsberichte aus dem Fachbereich Informatik, 8/2007

Lutz Priese, Frank Schmitt, Paul Lemke: Automatische See-Through Kalibrierung, Arbeitsberichte aus dem Fachbereich Informatik, 7/2007

Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, Marcel Weinand, Jörg Helbach: Security Requirements for Non-political Internet Voting, Arbeitsberichte aus dem Fachbereich Informatik, 6/2007

Daniel Bildhauer, Volker Riediger, Hannes Schwarz, Sascha Strauß, „grUML – Eine UML-basierte Modellierungssprache für T-Graphen", Arbeitsberichte aus dem Fachbereich Informatik, 5/2007

Richard Arndt, Steffen Staab, Raphaël Troncy, Lynda Hardman: Adding Formal Semantics to MPEG-7: Designing a Well Founded Multimedia Ontology for the Web, Arbeitsberichte aus dem Fachbereich Informatik, 4/2007

Simon Schenk, Steffen Staab: Networked RDF Graphs, Arbeitsberichte aus dem Fachbereich Informatik, 3/2007

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik, 2/2007

Anastasia Meletiadou, J. Felix Hampe: Begriffsbestimmung und erwartete Trends im IT-Risk-Management, Arbeitsberichte aus dem Fachbereich Informatik, 1/2007

„Gelbe Reihe"
(http://www.uni-koblenz.de/fb4/publikationen/gelbereihe)

Lutz Priese: Some Examples of Semi-rational and Non-semi-rational DAG Languages. Extended Version, Fachberichte Informatik 3-2006

Kurt Lautenbach, Stephan Philippi, and Alexander Pinl: Bayesian Networks and Petri Nets, Fachberichte Informatik 2-2006

Rainer Gimnich and Andreas Winter: Workshop Software-Reengineering und Services, Fachberichte Informatik 1-2006

Kurt Lautenbach and Alexander Pinl: Probability Propagation in Petri Nets, Fachberichte Informatik 16-2005

Rainer Gimnich, Uwe Kaiser, and Andreas Winter: 2. Workshop "Reengineering Prozesse" – Software Migration, Fachberichte Informatik 15-2005

Jan Murray, Frieder Stolzenburg, and Toshiaki Arai: Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification, Fachberichte Informatik 14-2005

Reinhold Letz: FTP 2005 – Fifth International Workshop on First-Order Theorem Proving, Fachberichte Informatik 13-2005

Bernhard Beckert: TABLEAUX 2005 – Position Papers and Tutorial Descriptions, Fachberichte Informatik 12-2005

Dietrich Paulus and Detlev Droege: Mixed-reality as a challenge to image understanding and artificial intelligence, Fachberichte Informatik 11-2005

Jürgen Sauer: 19. Workshop Planen, Scheduling und Konfigurieren / Entwerfen, Fachberichte Informatik 10-2005

Pascal Hitzler, Carsten Lutz, and Gerd Stumme: Foundational Aspects of Ontologies, Fachberichte Informatik 9-2005

Joachim Baumeister and Dietmar Seipel: Knowledge Engineering and Software Engineering, Fachberichte Informatik 8-2005

Benno Stein and Sven Meier zu Eißen: Proceedings of the Second International Workshop on Text-Based Information Retrieval, Fachberichte Informatik 7-2005

Andreas Winter and Jürgen Ebert: Metamodel-driven Service Interoperability, Fachberichte Informatik 6-2005

Joschka Boedecker, Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Masaaki Kikuchi, and Minoru Asada: Getting closer: How Simulation and Humanoid League can benefit from each other, Fachberichte Informatik 5-2005

Torsten Gipp and Jürgen Ebert: Web Engineering does profit from a Functional Approach, Fachberichte Informatik 4-2005

Oliver Obst, Anita Maas, and Joschka Boedecker: HTN Planning for Flexible Coordination Of Multiagent Team Behavior, Fachberichte Informatik 3-2005

Andreas von Hessling, Thomas Kleemann, and Alex Sinner: Semantic User Profiles and their Applications in a Mobile Environment, Fachberichte Informatik 2-2005

Heni Ben Amor and Achim Rettinger: Intelligent Exploration for Genetic Algorithms – Using Self-Organizing Maps in Evolutionary Computation, Fachberichte Informatik 1-2005