Towards a Framework for Iteratively Signing Graph Data

Andreas Kasten
University of Koblenz-Landau
Universitätsstraße 1
56070 Koblenz, Germany
andreas.kasten@uni-koblenz.de

Ansgar Scherp
University of Mannheim
B6, 26
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 support iterative signing of graph data. This paper describes a first step towards a 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 iteratively signing 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 resulting signature statements, even when iteratively signing.

1. INTRODUCTION

In order to track provenance and building trust networks for knowledge-based systems, it is necessary to be able to verify the authenticity and integrity of graph data by signing it. 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.

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 MSGs on their own. Tummarello et al. provide an important early step for signing graph data. However, their approach 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 RDF Statement reification. 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 entire graphs. The approach does not support signing Named Graphs or multiple graphs at the same time. Finally, it does not allow for an iterative signing of graph data as the signature statements become part of the signed 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 paper, we present a framework for signing RDF(S) graphs, Named Graph, and OWL graphs. The framework is built upon canonicalization functions and graph hashing functions such as [2, 3, 4]. A detailed description of the framework including a formalization of the signing process is presented in our TR [5].

2. SIGNING FRAMEWORK

Our 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. We introduce three possible configurations of our graph signing framework and discuss the different characteristics of the configurations.

A) Tummarello et al [1] The first configuration uses a canonicalization function and a hash function of Carroll [2]. Due to their complexity, 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. Additional statements are created for blank nodes sharing the same identifier. 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 signature statements. Signing a graph with r MSGs requires r signatures. This results in a signature overhead of $b_h + 6r$ statements for signing a whole graph.

B) Minimum Signature Overhead Using the canonicalization function and hash function of Fisteus et al. [3] 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 $O(n)$. Thus, the runtime complexity of the singing function σ_N is $O(n \log n)$ and the space com-

K-CAP '13, June 23-26, 2013, Banff, Canada.

plexity is $O(n)$. Since the functions of Fisteus et al. do not create any additional statements, the signature overhead is solely determined by s signature statements storing the created signature. When m graphs are signed at the same time, the m graphs are arranged using RDF bag. This results in a total of $s + 2m$ signature statements.

C) Minimum Runtime Complexity Using the canonicalization function and hash function of Sayers and Karp [4] leads to a minimum runtime complexity. In order to detect already handled blank nodes, the the canonicalization function maintains a list of additional statements created so far. This list contains at most b entries with b being the total number of additional statements. Assuming that each statement of a graph can contain no, one, or two blank nodes and that a blank node is part of at least one statement, the graph can contain at most twice as many blank nodes as statements, i. e., $b \leq 2n$. This results in a space complexity of $O(n)$ of the graph signing function. The signature overhead consists of b statements added by the canonicalization function and s signature statements.

3. EXAMPLE OF SIGNING OF GRAPHS

We describes the use of our graph signing framework. The examples refer to configuration B). Additional examples and a scenario are presented in our TR [5]. In the scenario, the German Telecom receives an already signed Named Graph `bka:bka-sg-1` from the German Federal Criminal Police Office (Bundeskriminalamt, BKA). This graph contains information for regulating access to neo-Nazi material on the Internet but does not describe how to implement this regulation. Thus, the German Telecom adds its own RDF graph `_:gt-data-2` with implementation details such as a proxy server and its IP address. It then signs `_:gt-data-2` together with the received graph `bka:bka-sg-1`. This results in a Named Graph `gt:gt-sg-2` as depicted in Listing 1. The graph contains the created signature statements (lines 2 to 5) as well as the two graphs `_:gt-data-2` (lines 6 to 14) and `bka:bka-sg-1` (lines 15 to 21). The signature statements cover the signature value (line 3) and the ISP's public key certificate `gt:gt-pck-2` (line 4).

```
1  gt:gt-sg-2 {
2    gt:gt-sig-2 a sig:Signature ;
3      sig:hasSignatureValue "YXJlIGJlbG9uZyB0byB1cw==" ;
4      sig:hasVerificationCertificate gt:gt-pck-2 .
5    ...
6    _:gt-data-2 {
7      bka:pr-1 DUL:hasQuality gt:naq-2 .
8      gt:naq-2 a tec:NetworkAddressQuality ;
9        DUL:hasRegion gt:ipr-2 .
10     gt:ipr-2 a tec:IPv4AddressRegion ;
11       tec:hasIPAddress "141.26.83.115" ;
12       tec:hasSubnetMask "255.255.0.0" .
13     ...
14   }
15   bka:bka-sg-1 {
16     bka:bka-gsm-1 a sig:Signature ;
17       sig:hasSignatureValue "TmV2ZXIgR29ubmEgR2l2ZQ==" ;
18       sig:hasVerificationCertificate bka:bka-pck-1 .
19     ...
20     _:bka-pattern-1 { ... } _:bka-rules-1 { ... }
21   }
22 }
```

Listing 1: Example of iteratively signed graphs.

A primary school receives the graph `gt:gt-sg-2` from the German Telecom and another graph `cw:cw-sg-3` from ContentWatch. The graph `cw:cw-sg-3` covers regulation information about adult content such as Internet pornography. The school adds its own regulation details as RDF graph `_:ps-data-4` including a proxy server run by the school. It signs the graph `_:ps-data-4` together with the two graphs `cw:cw-sg-3` and `cw:cw-sg-3`. This results in the Named Graph `ps:ps-sg-4` shown in Listing 2, containing the graph `_:ps-data-4` (lines 6 to 14), the German Telecom's graph `gt:gt-sg-2` (lines 15 to 22), and ContentWatch's graph `cw:cw-sg-3` (line 23). The school's signature statements (lines 2 to 5) cover the created signature value (line 3) and the school's public key certificate `ps:ps-pck-4` (line 4).

```
1  ps:ps-sg-4 {
2    ps:ps-sig-4 a sig:Signature ;
3      sig:hasSignatureValue "QWxsIHlvdXIgYmFzZSBhcmU=" ;
4      sig:hasVerificationCertificate ps:ps-pck-4 .
5    ...
6    _:ps-data-4 {
7      cw:pr-3 DUL:hasQuality ps:naq-4 .
8      ps:naq-4 a tec:NetworkAddressQuality ;
9        DUL:hasRegion ps:ipr-4 .
10     ps:ipr-4 a tec:IPv4AddressRegion ;
11       tec:hasIPAddress "141.26.83.116" ;
12       tec:hasSubnetMask "255.255.0.0" .
13     ...
14   }
15   gt:gt-sg-2 {
16     gt:gt-sig-2 a sig:Signature ;
17       sig:hasSignatureValue "YXJlIGJlbG9uZyB0byB1cw==" ;
18       sig:hasVerificationCertificate gt:gt-pck-2 .
19     ...
20     _:gt-data-2 { ... }
21     bka:bka-sg-1 { ... _:bka-pt-1 { ... } _:bka-rl-1 { ... } }
22   }
23   cw:cw-sg-3 { ... cw:cw-rl-3 { ... } }
24 }
```

Listing 2: Example of multiple signed graphs.

4. CONCLUSION

Our framework allows for signing multiple and distributed RDF(S) graphs, OWL graphs, and Named Graphs. It supports signing together A-box and T-box knowledge of different granularity such as single MSGs, ontology design patterns, and whole graphs. We have discussed three different configurations of the signing framework and have presented an implementation based on TriG [6]. The complete examples as well as a signature ontology are available from: `http://icp.it-risk.iwvi.uni-koblenz.de/wiki/Signing_Graphs`.

5. REFERENCES

[1] Tummarello, G., Morbidoni, C., Puliti, P., Piazza, F.: Signing individual fragments of an RDF graph. In: WWW, ACM (2005) 1020–1021

[2] Carroll, J.J.: Signing RDF graphs. In: ISWC 2003, Springer (2003) 369–384

[3] Fisteus, J.A., García, N.F., Fernández, L.S., Kloos, C.D.: Hashing and canonicalizing Notation 3 graphs. JCSS **76** (2010) 663–685

[4] Sayers, C., Karp, A.H.: Computing the digest of an RDF graph. Technical report, HP Laboratories (2004)

[5] Kasten, A., Scherp, A.: Iterative signing of RDF(S) graphs, named graphs, and OWL graphs: Formalization and application. Technical report, University of Koblenz-Landau (2013) `http://www.uni-koblenz.de/~fb4reports/2013/2013_03_Arbeitsberichte.pdf`.

[6] Bizer, C., Cyganiak, R.: The TriG syntax (2007) `http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/Spec/`.