Verifiability and Traceability in a Linked Data Based Messaging System

Florian Kleedorfer, Yana Panchenko, Christina Maria Busch
Studio Smart Agent Technologies
Research Studios Austria FG
Thurngasse 8/2/16, 1090 Vienna
fkleedorfer,ypanchenko,cbusch @researchstudio.at

Christian Huemer
Institute for Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstrasse 9-11, 1040 Vienna
huemer@big.tuwien.ac.at

ABSTRACT

When linked data applications communicate, they commonly use messaging technologies in which the message exchange itself is not represented as linked data, since it takes place on a different architectural level. When a message cannot be verified and traced on the linked data level, trust in data is moved from message originators to service providers. However, there are use cases in which the actual message exchange and its verifiability are of importance. In such situations, the separation between application data and communication data is not desirable. To address this, we propose messaging based on linked data, where communicating entities and their messages are represented as interconnected Web resources, and we show how conversations can be made verifiable using digital signatures.

CCS Concepts

•Information systems → World Wide Web; •Networks → Web protocol security; •Human-centered computing → Collaborative and social computing systems and tools;

Keywords

linked data, messaging, RDF signatures, provenance

1. INTRODUCTION

Linked open data allows applications on the Web to make their data publicly available in a standardized form. This approach effectively transforms the Web into a large, distributed resource description framework (RDF) database that can be shared by all applications. While managing data is an important aspect of any program, so is communication between entities within the application or across system boundaries. This, however, typically requires the use of subsystems that do not operate on the linked data layer. If any information about the communication is to be made accessible as linked data, it has to be converted to that format, thereby duplicating much, if not all, of the data in two different architectural layers. Making communications available as linked data is beneficial in situations that, for example, require provenance tracking, or in which the conversation between two entities constitutes a contract: third parties with the responsibility of auditing or settling conflicts may access and analyze the whole conversation in an automated manner and understand the factual situation quickly. Parties wanting to evaluate whether to trust a given Web entity may analyze its past conversations if given access. In the context of business process automation, it may be beneficial to have access to the state of all processes, including their message history. It has been stated as one of the challenges facing the semantic Web stack that linked data applications interoperate only on the data level, but questions of provenance are largely unsolved; so is semantic user management and cross-linking semantic applications [5]. To that effect, it might be beneficial for the interoperability and modularization of lightweight, interconnected Web applications to build upon a unified interface to data and communication.

We propose to use linked data for both communication and application data. The initial motivation for our design is derived from our effort to build a demand-driven marketplace infrastructure we call *Web of Needs* [11]. Our approach aims to prevent modification of messages by using cryptographic signatures and the modification of conversations by chaining signatures of subsequent messages.

To the best of our knowledge, no messaging system that uses linked data as message format and persistence layer has been proposed to date. Closely related systems extend existing communication frameworks to allow for the transmission of RDF datasets [16], focus on message composition but not persistence or publishing [1, 7], or are designed more generally for expressing provenance but do not specifically address messaging [3, 12]. Other work has focused on providing a model that supports a wide range of messaging systems for the sake of seamless integration in applications, but did not aim for a standalone semantic Web based messaging system [14]. Another work integrates messaging with semantic Web agents [4].

2. OUR APPROACH

Although our messaging approach is applied with a specific application in mind [11], we strove to develop a generic linked data based messaging platform. In our approach, both messages and communicating entities are part of semantic web data: each communicating entity and each message, including its payload and metadata, is an RDF resource identified by a URI and accessible as linked data. The communicating entities use publishing services of their choice as communication relays. Both the publishing services and the communicating entities use WebID [15] to make their public keys available. These key pairs are used to create the cryp-

SEMANTiCS 2016, September 12 - 15, 2016, Leipzig, Germany

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4752-5/16/09. . . $15.00

DOI: http://dx.doi.org/10.1145/2993318.2993342

tographic signatures referred to in the following.

The purpose of sending a message is the transfer of a number of RDF graphs (called *content graphs*) from the sender to the recipient. In order to achieve this, the sender first constructs a dataset containing the content graphs. This dataset is then *signed* by the sender: for each named graph in the dataset, a *signature graph* is added, containing a cryptographic signature of the content graph. An additional *envelope graph* is added to the dataset containing sender URI, recipient URI, and a reference to the signatures of the content graphs. Then, the envelope graph is signed by the sender.

As we want the message to be available as linked data, it must be possible to de-reference its URI. It therefore must be transferred[1] to an online service (the publishing service) that will provide the data upon request. In order to defend against unauthorized copying of the message, its URI is minted by the original sender within the URI space controlled by the publishing service and is used within the message itself.[2] The sender's publishing service creates an additional envelope graph in the message dataset, adds a reference to the previously outermost signature and a triple pointing to another newly minted URI in the URI space controlled by the recipient's publishing service. The envelope graph is then signed by the publishing service. This dataset is made available at the URI that was minted by the original sender. With the addition of another envelope and signature graph, reflecting the change of URI from the sender's to the recipient's URI space, the message is then transferred to the recipient's publishing service under the new URI. Upon receiving the message, that publishing service adds another envelope, a reference to the previously outermost signature, and signs the envelope by adding another signature graph. The result is transferred to the recipient and made available at the new URI.

Communicating entities and publishing services send and receive messages using their URIs for addressing, messages themselves reference other messages or related linked data resources. Messages are signed using the entity's or publishing service's private keys and verified with the respective public keys. Authentication and access to participation in the conversation is based on digital signatures of the messages. Verifiability and traceability of the communication during or after the conversation is achieved through referencing signatures of previous messages or of the descriptions of communicating entities in the innermost envelope graph.

2.1 Message composition

```
 1 @prefix msg: <http://purl.org/webofneeds/message#> .
 2 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
 3 @prefix rdfg: <http://www.w3.org/2004/03/trix/rdfg-1/> .
 4 @prefix sig: <http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/
       signature.owl#> .
 5 <URI2/S/m2#c1> { # the content graph
 6   <URI2/S/m2>
 7     msg:hasTextMessage  "Hey there!"^^xsd:string . }
 8 <URI2/S/m2#c1s> { # signature of the content graph
 9   <URI2/S/m2#c1s>
10     a  sig:Signature ;
11     # some signature triples left out indicating algorithm and parameters
12     sig:hasSignatureValue  "MGQCMDRsm4n77svH+hDK2CBkXJeLDRky..." ;
13     sig:hasVerificationCertificate <URI2/S/> ;
14     msg:hasSignedGraph <URI2/S/m2#c1> . }
15 <URI2/S/m2#e1> { # the inner envelope (added by the sender S)
16   <URI2/S/m2#e1>
17     a msg:EnvelopeGraph ;
18     rdfg:subGraphOf <URI2/S/m2> .
19   <URI2/S/m2>
20     a msg:FromOwner ;
21     msg:hasContent <URI2/S/m2#c1> ;
22     msg:hasMessageType msg:ConnectionMessage ;
23     msg:hasRecipient <URI1/R/> ;
24     msg:hasRecipientService <URI1/P/> ;
25     msg:hasSender <URI2/S/> ;
26     msg:hasSenderService <URI2/P/> ;
27     msg:hasSentTimestamp "1431964656854"^^xsd:long ;
28     msg:hasPreviousMessage [ msg:hasSignatureGraph <URI2/S/m1#e2s> ;
29       sig:hasSignatureValue "MGQCMG5Kw6kGjGTwJ1zjmLCJeICqCg23..." ;
30       msg:hasSignedGraph <URI2/S/m1#e2> ];
31     msg:referencesSignature [ msg:hasSignatureGraph  <URI2/S/m2#c1s> ;
32     sig:hasSignatureValue "MGQCMDRsm4n77svH+hDK2CBkXJeLDRky..." ;
33     msg:hasSignedGraph  <URI2/S/m2#c1> ] .  }
34 <URI2/S/m2#e1s> { # signature of the inner envelope
35   <URI2/S/m2#e1s> a sig:Signature ; # some signature triples left out
36   sig:hasSignatureValue "MGQCMHFU3s0jbDNtRBYCXEJEehx3SAaO2..." ;
37   sig:hasVerificationCertificate <URI2/S/> ;
38   msg:hasSignedGraph <URI2/S/m2#e1> . }
39 <URI1/R/m2#e2> { # the outer envelope (added by the sender's publishing service)
40   <URI1/R/m2#e2>
41     a msg:EnvelopeGraph ;
42     msg:containsEnvelope <URI2/S/m2#e1> ;
43     rdfg:subGraphOf <URI1/R/m2> .
44   <URI1/R/m2>
45     a  msg:FromExternal ;
46     msg:hasRemoteMessage <URI2/S/m2> ;
47     msg:hasSentTimestamp "1431964657038"^^xsd:long ;
48     msg:referencesSignature [ msg:hasSignatureGraph  <URI2/S/m2#e1s> ;
49       sig:hasSignatureValue "MGQCMHFU3s0jbDNtRBYCXEJEehx3SAaO2..." ;
50       msg:hasSignedGraph <URI2/S/m2#e1> ] .  }
51 <URI1/R/m2#e2s> { # signature of the outer envelope
52   <URI1/R/m2#e2s>  a   sig:Signature ; # some signature triples left out
53     sig:hasSignatureValue "MGUCMQCGPBxuqFHo83rD6v7EF9tZIsaNo..." ;
54     sig:hasVerificationCertificate <URI1/P/> ;
55     msg:hasSignedGraph <URI1/R/m2#e2> . }
```

Listing 1: The message URI1/R/m2, as it is received by the recipient's publishing service, serialized in TriG.

In the following, we describe main aspects of message composition, illustrating them with Listing 1, an example message sent by sender *S* to recipient *R* as it is received by *R*'s publishing service.

Message addressing. Each of the communicating entities and the publishing services are addressed by their URIs. In our example message *R* and *S* have the URIs URI1/R/ and URI2/S/, using publishing services URI1/P/ and URI2/P/ (see lines 23–26).

Message URI. A message between communicating entities is represented by two URIs: namely the URI it has on the sender side and the URI it has on the recipient's side. The message in the example in Listing 1 is identified by URI1/R/m2 (line 44). It is the recipient's message URI and it is linked to its remote counterpart URI2/S/m2 via msg:hasRemoteMessage relation (line 46). Each message is also linked either to at least one previous message via msg:hasPreviousMessage, or, if it is the first message in the conversation, to both communicating entities via the msg:hasOrigin relation. In our example, the message specifies (lines 28–30) that its previous message is a message with outer envelope URI2/S/m1#e2.

Message Graphs. The message is an RDF dataset identified by the message URI. Content graphs, envelope graphs and signature graphs are named graphs whose names are composed of the message URI and a fragment identifier. In order for messages to remain verifiable after having been imported into a triple store, it is required that there be no default graph in any dataset representing a message. Moreover, all envelopes are explicitly marked as subgraphs of the message resource so that they can be collected in an RDF store. In the example in Listing 1, the content part of the message expressing the text message from *R* to *S* is represented as a named graph with URI URI2/S/m2#c1 (lines 5–7), the envelope parts are represented as named graphs with URIs URI2/S/m2#e1 and URI1/R/m2#e2 (lines 15–33 and 39–50), and the signatures

[1]The concrete way of transferring is not prescribed by our approach, be it an HTTP Post Request (or Get for the reverse direction), as a Websocket message or as the payload in another communication system like e-mail or message-oriented middleware.

[2]The handling of URI clashes that may happen is beyond the scope of this paper.

as named graphs with URIs URI2/S/m2#c1s , URI2/S/m2#e1s and URI1/R/m2#e2s (lines 8–14, 34–38, 51–55).

Envelope Chaining. A message received by a publishing service may need to be passed on, possibly after adding additional data. When adding data, the message is composed in a chained manner: previous messages and data are referenced from the newly added data. This is implemented by creating a new envelope graph each time new data is added, and by referencing the previous message envelope and content graphs and their signatures from that new envelope. The outermost envelope graph is the one that must be processed first by the recipient to determine how to process the message, working inward from there. Therefore, a message envelope includes references to a content graph and/or to a preceding envelope. In the example in Listing 1, the envelope URI2/S/m2#e1 references the message content URI2/S/m2#c1 by msg:hasContent relation (line 21). When *S*'s publishing service prepares the message for sending to *R*'s publishing service, it adds a new envelope URI1/R/m2#e2. This new envelope references the envelope URI2/S/m2#e1 by msg:containsEnvelope relation (line 42), building a chain. The signatures are referenced by the relation msg: referencesSignature (lines 31–33, 48–50).

Message Signatures. Our approach is based on solutions for cryptographic signing of RDF graphs [3, 10] and on approaches to securing provenance information in distributed networks [8, 17].

We define a message to be signed when each envelope graph and each content graph of the message is signed. The triples defining the signatures are placed in separate named graphs that are part of the message. The example in Listing 1 demonstrates signature graphs URI1/R/m3#c1s, URI1/R/m3#e1s, and URI2/S/m3#e2s (lines 8–14, 34–38, 51–55). The signature is defined by the signature name (URI of the signature graph), by the URI of the graph signed by this signature, by the resolvable signer URI (WebID) by which the signer's public key can be obtained, and by the signature value. Additionally, the signature contains the triples describing the signing algorithm (not shown in the example for the sake of brevity). These triples are necessary in order to calculate the hash for the signature verification in exactly the same way as it was calculated during signing, and apply the same cryptographic algorithm for verification as was used for signing.

We use the Signingframework library [9] for signing or verifying RDF graphs, applying the algorithm developed by Fisteus et al. [6] due to its minimum signature overhead in comparison to other algorithms, as suggested by Kasten et al. [10]. Elliptic Curve Digital Signature Algorithm (ECDSA) with 384-Bit curves is used as the signature generation and verification algorithm.

Assigning a URI to a signature makes it possible to refer to signatures. Such references allow for iterative signing by chaining the signatures just like content and envelopes are chained, forming a merkle tree [13]. Whenever a content graph or an envelope graph is referenced from another envelope graph, a reference to the respective signature graph is added. The signature reference includes the signature graph name, the signed graph name, and the signature value (lines 31–33, 48–50). For example, in Listing 1 the signature URI1/R/m3#e1s is referenced from the envelope URI2/S/m3#e2. Additionally, references to external document's signatures serve as a means to derive and verify message history. In the example in Listing 1, the envelope URI2/S/m3#e2 references the signature of the previous message, creating a verifiable sequence (lines 28–30).

2.2 Signing and Verification

Here, well-formedness, signing, and verification are defined.

Well-formedness. A message dataset D is said to be well-formed if it satisfies the following conditions: 1. The default graph is empty. 2. It contains at least one envelope graph. 3. All graphs are either signature graphs or a signed graphs. 4. The reference structure of envelopes is a chain. 5. All values in signature references must be identical to the values in the respective signatures. 6. There must be exactly one envelope graph that is not referenced from within any other envelope graph. 7. There must be exactly one signature graph that is not referenced from within any other envelope graph. 8. Any signature graph references exactly one envelope graph or one content graph. 9. Each envelope must be signed according to sender/recipient information. 10. Any graph name must be a hash URI based on its message URI. 11. Any message URI used must conform to a pattern specified by the respective publishing service.[3]

Signing. This is defined as the process of adding a signature graph for each unsigned named graph in the input dataset and collecting the new signature information in an envelope graph that eventually is also signed. Let K be the private key of the signer, u_K the URL for downloading that key; $D = M \cup E \cup C$ denote the dataset constituting a message ready for signing, where E is the newly added envelope graph, C denotes a set of newly added content graphs, and M is an RDF dataset that is either empty or well-formed. It follows that if D is non-empty, it includes top-level signature graph S_t and top-level envelope graph E_t that are both not referenced from within the dataset. Then, Algorithm 1 defines how messages are signed.

Algorithm 1: Signing a message dataset

```
   input: $K$, $u_K$, $D$
   Result: RDF Dataset $D$ contains the signed message
1  if $D \neq \emptyset$ :
2      add reference to $E_t$ to $E$
3      add reference to $S_t$ and its signature value to $E$
4  foreach $c \in C$ do
5      compute signature graph $S_c$ for $c$ using $K$ and $u_K$
6      add $S_c$ to message dataset
7      add reference to $c$ to envelope graph $E$
8      add reference to $S_c$ and its signature value to $E$
9  compute signature graph $S_E$ for $E$ using $K$ and $u_K$
10 add $S_E$ to $D$
```

Verification. The verification algorithm checks the message by ensuring there are no unsigned graphs, verifying all signed graphs against their signature and the signer's public key, and makes sure that the signature value of the signature graph references are correct. Let D denote a well-formed message dataset, the boolean-valued function `isSignatureGraph`(G) test if a given named graph is a signature graph, and the boolean valued function `verify`(g, s, K) determines if the private key of K was used to create signature graph s for graph g. Then, Algorithm 2 defines how messages are verified.

3. EVALUATION

We evaluate the applicability of the proposed approach by measuring performance during messaging. The time of a messaging action is evaluated empirically by simulating a conversation in the Web of Needs prototype [2]. Here, publishing services react to messages with additional SUCCESS or FAILURE responses, constructed according to the same principles as the messages. Time measurements were taken at different stages in this process: SENT

[3]The mechanism for obtaining and checking this pattern is covered by a different part of our protocols and is not discussed further in this work.

Algorithm 2: Verifying a message dataset

```
input  : $D$
output: $b$:boolean indicating success
1 $V \longleftarrow \emptyset$ /* set of verified graphs */
2 foreach $s \in D$ do
3     if isSignatureGraph (s) :
4         $K \longleftarrow$ public key referenced by $s$
5         $g \longleftarrow$ graph in $D$ that is signed by $s$
6         if not verify (g,s,K) :
7             return false
8         $V \longleftarrow V \cup \{g,s\}$
9 return $|V| = |D|$
```

Stage	*Without signatures*	*With signatures*
SENT	11	21
SENDINGCONFIRMED	98	268
RECEIVED	106	341
RECEPTIONCONFIRMED	183	520

Table 1: Mean duration until stage reached (milliseconds)

– the sender has sent the message to its publishing service; SENDINGCONFIRMED – the sender has received the success response from her own publishing service; RECEIVED – the recipient has received the message; RECEPTIONCONFIRMED – the sender has received the success response of the recipient's publishing service. The average duration over 500 chat messages is stated per stage in Table 1. Each message contains a short chat text (about 30 characters long) as its payload. The time measured depends on the implementation as well as network and hardware characteristics. For comparison, we provide time required for the same actions, but excluding processing related to message signing, verification and wellformedness checks.

Memory requirements are presented in Table 2. Here, we consider the data generated at stage RECEPTIONCONFIRMED. The numbers in brackets are sizes without signatures, documenting the additional space requirements for traceability and publishing of messaging metadata.

4. CONCLUSION

In this work we present a messaging framework based on linked data and public-key cryptography. The cost of this approach is measured as additional execution time and storage space required for the use of cryptographic signatures in our reference implementation. Future work will tackle end-to-end encryption of message content and parts of the metadata, integration with existing key management systems, and an approach to key revocation.

5. ACKNOWLEDGMENTS

This work was supported by the Austrian Research Promotion Agency (FFG) in the COIN project *USS WON – Usability, Scalability and Security on the Web of Needs*.

originator	*datasets*	*graphs*	*quads*	*size as TriG*
user	2	20 (10)	200 (58)	26.5 kB (8.7 kB)
system	3	18 (9)	192 (69)	25.8 kB (10.4 kB)
total	5	38 (19)	392 (127)	52.4.kB (19.1 kB)

Table 2: Space per message at stage RECEPTIONCONFIRMED

6. REFERENCES

[1] Linked data signatures 1.0. *Draft Community Group Specification*, 11 May 2015.

[2] Web of needs on github. https://github.com/researchstudio-sat/webofneeds/, 2016.

[3] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 613–622, New York, NY, USA, 2005. ACM.

[4] Sebastian Dietzold, Jörg Unbehauen, and Sören Auer. xoperator - interconnecting the semantic web and instant messaging networks. In *Proceedings of 5th European Semantic Web Conference (ESWC 2008), 1-5 June, 2008, Tenerife, Spain.*, pages 19–33, 2008.

[5] Dominic DiFranzo and James Hendler. The semantic web and the next generation of human computation. In Pietro Michelucci, editor, *Handbook of Human Computation*, pages 523–530. Springer New York, 2013.

[6] Jesus Arias Fisteus, Norberto Fernández García, Luis Sánchez Fernández, and Carlos Delgado Kloos. Hashing and canonicalizing notation 3 graphs. *Journal of Computer and System Sciences*, 76(7):663 – 685, 2010.

[7] Jonas Halvorsen and Bjørn Jervell Hansen. Integrating military systems using semantic web technologies and lightweight agents. Technical Report 01851, Norwegian Defence Research Establishment (FFI), October 2011.

[8] Ragib Hasan, Radu Sion, and Marianne Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, pages 1–14, Berkeley, CA, USA, 2009. USENIX Association.

[9] Andreas Kasten. A software framework for iterative signing of graph data. https://github.com/akasten/signingframework, 2015.

[10] Andreas Kasten, Ansgar Scherp, and Peter Schauß. A framework for iterative signing of graph data on the web. In *The Semantic Web: Trends and Challenges*, pages 146–160. Springer, 2014.

[11] Florian Kleedorfer, Christina Maria Busch, Christian Pichler, and Christian Huemer. The case for the web of needs. In *16th IEEE Conference on Business Informatics (CBI)*, volume 1, pages 94–101. IEEE, 2014.

[12] Paolo Missier Luc Moreau. Prov-overview. W3C Working Group Note, April 2013.

[13] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO'87*, pages 369–378. Springer, 1987.

[14] Dennis Quan, Karun Bakshi, and David R Karger. A unified abstraction for messaging on the semantic web. In *WWW (Posters)*, 2003.

[15] Andrei Sambra, Henry Story, and Tim Berners-Lee. Webid 1.0 - web identity and discovery, December 2013.

[16] Joshua Shrinavier. Rdfagents. Technical report, Rensselaer Polytechnic Institute, 2011.

[17] Xinlei Wang, Kai Zeng, K. Govindan, and P. Mohapatra. Chaining for securing data provenance in distributed information networks. In *Military Communications Conference(MILCOM) 2012*, Oct 2012.