FOAF+SSL: RESTful Authentication for the Social Web[*]

Henry Story[1], Bruno Harbulot[2], Ian Jacobi[3], and Mike Jones[2]

[1] Sun Microsystems, http://blogs.sun.com/bblfish
[2] The University of Manchester, UK, Bruno.Harbulot@manchester.ac.uk
[3] MIT

Abstract. We describe a simple protocol for RESTful authentication, using widely deployed technologies such as HTTP, SSL/TLS and Semantic Web vocabularies. This protocol can be used for one-click sign-on to web sites using existing browsers — requiring the user to enter neither an identifier nor a password. Upon this, distributed, open yet secure social networks and applications can be built. After summarizing each of these technologies and how they come together in FOAF+SSL,[4] we describe declaratively the reasoning of a server in its authentication decision. Finally, we compare this protocol to others in the same space.

1 Introduction

Many services that require authentication rely on centralized systems. The identity of the user is constrained to that administrative domain, forcing her to have a different account and identifier for each organization she interacts with. This inability to relate identities easily across domains also makes the creation of links between people in distinct organizations difficult.

Every time a person needs authenticated access to a new organization, a new registration needs to be made; this is a burden for both the user and the organization. The process of registration is either (a) minimal — for example, e-mail address confirmation —, or (b) more formal — for example, in a workplace, where an administrator has to create an account after making verifications out-of-band. Process (a) is lightweight, but will often provide insufficient information, whereas process (b) may initially be able to give more information about a user, at the expense of a costly verification phase during the registration.

Attempts to decentralize this process have been made. Shibboleth,[5] for example, aims at sharing accounts across administrative boundaries; however, it relies on a rigid federation process between organizations. OpenID, enables authenticating a user against a URI, but does not build a social web on that.

---

[*] This paper is licensed under the Creative Commons Attribution 3.0 unported license described at http://creativecommons.org/licenses/by/3.0/

[4] Up-to-date information on developments in this protocol are available at http://esw.w3.org/topic/foaf+ssl.

[5] http://shibboleth.internet2.edu/

Neither Shibboleth nor OpenID fully comply with web architecture principles (see Section 2.1 on REST), which in part explains their limitations.

This paper describes a novel approach that relies on combining the use of SSL client certificates and Semantic-Web-based FOAF networks. The result is a secure, open and distributed authentication mechanism, which is able to satisfy simple requirements — such as authenticating a user by URI, like OpenID, but without the user needing to remember this URI — as well as more complex requirements, where the authorization to a service depends on distributed properties of the user, such as his position in a social network. This proposal is built on a RESTful architecture, the same that underpins the largest and most successful network of distributed linked information, the Web.

Section 2 introduces the background technologies of the Semantic Web and FOAF, as well as cryptography and client-certificate authentication. Section 3 presents the FOAF+SSL protocol. Section 4 compares this approach to others.

2 Background

2.1 The RESTful Web Architecture

Representational State Transfer (REST) [1, Chap. 5] is an architectural style for building large-scale distributed information networks, the most famous of these being the World Wide Web [2]. To build such a network requires that each of the parts be able to grow independently of any of the others, with very little central coordination, and that each of the resources thus created be able to refer easily to any of the others. The logical building blocks for this are the following:

1. The specification of universal names, also known as Universal Resource Identifiers (URIs) — such as the familiar Universal Resource Locators (URLs).
2. The mapping of URIs to Resources, also known as the reference relation.
3. Canonical methods for manipulating the resources mapped by each URI, via *representations* of the resource. Such a protocol specifies a canonical dereferencing mechanism, enabling a holder of a URI to find and manipulate the resource referred to by that URI. `http://...` URLs use the HTTP protocol as their dereferencing mechanism, for example. By sending a GET request to the object at a given HTTP URL, a *representation* of the resource is returned. A resource can be created, changed, and deleted using the POST, PUT, and DELETE methods respectively.

REST specifies the architectural style required to build such a protocol with the aim of maximum networkability; that is, any representation should be able to link to any resource from anywhere, using the URI alone to do so.

2.2 The Semantic Web

Whereas URLs in the initial web of hyperlinked documents referred only to documents, the Semantic Web specifies how to extend this to enable a web of resources. In the Semantic Web, it becomes possible for URLs to refer to anything, be it:

1. physical things — for example, `<https://romeo.example/#i>` may refer to a human named Romeo;
2. relations between two individuals — for example, the relation of knowing someone which `<http://xmlns.com/foaf/0.1/knows>` refers to; or
3. classes — for example, people may be described as being instances of `<http://xmlns.com/foaf/0.1/Person>`, the set of persons.

The meaning of these URLs can be found by dereferencing them using their canonical protocol. Thus, doing an HTTP GET on `<http://xmlns.com/foaf/0.1/knows>` should return a representation describing it. Since HTTP is built to allow content negotiation, well configured web servers will return the representation best fitting the client's needs. Entering the above URL in a web browser will return a human readable HTML page describing the 'knows' relation. A Semantic Web agent could ask for the standard machine-friendly RDF representation.

A Semantic Web document is a serialization of a graph of directed relations between objects. Each relation exists as a triple of `<subject> <relation> <object>`, where each of `subject`, `relation` and `object` can be chosen among any of the URIs or string literals. Since it is tedious to read and write such URLs, this article uses the N3[6] `@prefix` notation. The following prefixes will be used throughout this article:

```
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix cert: <http://www.w3.org/ns/auth/cert#> .
@prefix rsa: <http://www.w3.org/ns/auth/rsa#>
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix romeo: <https://romeo.example/#> .  #see RFC2606 on .example domains
@prefix jult: <https://juliet.example/#> .
@prefix : <> . # special vocabulary defined for this paper
```

Thus, to say "Romeo is a person", one can write: `romeo:i a foaf:Person.`. Since each of the resources in that sentence is named by a URL, one can GET further information about each by dereferencing it, and if that representation itself contains relations do the same recursively. This is known as Linked Data[7].

Each representation returned by a resource can be interpreted as a graph of relations, which can be isolated in N3 by placing them within curly brackets { }.[8] The relation that maps a resource to the graph described by the document retrieved using the canonical dereferencing method of its URI is defined as the `:semantics` relation. Thus, after dereferencing `romeo:i`, one may state the following, without asserting the actual statements within the brackets as true:

(P1)
```
romeo:i :semantics { romeo:i a foaf:Person;
                            :hasPrivateKeyFor pubKey;
                            foaf:name "Romeo";
                            foaf:knows  jult:me . }
```

---

[6] Current N3 tutorial at: http://www.w3.org/2000/10/swap/doc/Overview.html.

[7] see Tim Berners-Lee's note http://www.w3.org/DesignIssues/LinkedData.html

[8] Unlike the Named Graph brackets in SPARQL; N3 supports anonymous graphs.

These graphs can also be used to formulate rules, as when we define the above `:semantics` relation in terms of the established `log:semantics` property, which relates a document to its graph, and `:representation` relating a resource to one of its representations (the `?` prefixed variables are universally quantified):

(D1)
```
{ ?resource :representation ?doc . ?doc log:semantics ?graph . }
        => { ?resource :semantics ?graph . }
```

The `log:` namespace[9] tends to make significant use of enclosed graphs, or "formulas". In particular, the `log:includes` property links a subject graph to an object graph by asserting that the latter is a subset of the former graph; the `log:implies` property, also written as `=>`, can serve as the basis for reasoning based on first-order logic (with the introduction of appropriate variables).

Even though the Semantic Web is built in order to make merging of information easy, it is not a requirement to do so. We will be using this notation to help illustrate clearly when merging graphs is reasonable.

2.3 FOAF, reputation networks and the Web of Trust

FOAF,[10] short for Friend-of-a-Friend, is an RDF vocabulary used to describe people, agents, groups and their relations. When used on the Semantic Web, this allows each person to describe himself and his network of friends.

By giving oneself a URI — *aka.* a Web ID —, one can describe one's personal social network by linking oneself to acquaintances by reference. Someone who has been given the `romeo:i` URL by Romeo himself, and then fetched its `:semantics` (ending up with the statements in P1) has good reason to trust that the information there is correct and, thus, to merge it (in a defeasible manner) with his own belief store. This graph itself will contain further URIs, such as `jult:me`, whose `:semantics` the agent can also GET. Similarly, the user can then add `romeo:i` to his FOAF file, to publish `:me foaf:knows romeo:i`.

Thus, a peer-to-peer information network can be built, where each person specializes in keeping up-to-date the information they feel responsible for, linking to the best sources for objects they do not wish to maintain. In return, as the quality of one's information and links increases, others feel more confident linking to it, reducing their own work and responsibilities.

As the network grows, the value of the network grows exponentially, as predicted by Metcalf's Law [3], creating a virtuous circle. Current social networking sites, such as Facebook and LinkedIn, to name a few among many, have shown how this can work in less distributed settings, taking advantage of the same law.

The plain `foaf:knows` relation may be enhanced with trust descriptions so as to create a *reputation network* [4], and, in the case of FOAF+SSL, this trust can be backed by the use of cryptographic keys and signatures, so as to form a secure Web of Trust (as described in the next sections).

---

[9] The `log:` namespace is described at http://www.w3.org/DesignIssues/N3Logic.

[10] Defined at http://xmlns.com/foaf/0.1/.

2.4 Public key cryptography

Public key cryptography allows two peers to communicate securely without requiring them to share a secret, through the use of unique pairs of keys. One key, called the *public key*, may be disseminated widely, and the other, the *private key*, is to be kept only by its owner. This is in contrast to symmetric cryptography, where both participants must share the knowledge of the same *secret* key for both encryption and decryption.

Public key cryptography relies on the conjecture that it is infeasible to obtain any private key that corresponds to a given public key through brute force because this operation is too computationally expensive. It also assumes that no two distinct individuals will generate the same key-pair randomly. We can define in D2 an inverse functional property `:hasPrivateKeyFor`.

(D2)
```
:hasPrivateKeyFor a owl:InverseFunctionalProperty;
    rdfs:domain foaf:Agent;
    rdfs:range cert:PublicKey .
```

Thanks to the dual-nature of the public and private key pair, two distinct actions are made possible:

1. *Encryption* is the obfuscation of a plain text message, generating a scrambled message using the public key of a key pair, so that it may only be decrypted using the corresponding private key.
2. *Signing* is the process of associating a digital signature with a message; this signature is generated using a private key. The authenticity and integrity of the message can then be verified using the corresponding public key.

A *public key certificate* is the signed combination of a public key and some information related to this key. Such a certificate may be self-signed (using the private key that matches the public key it contains) or signed by a third party. The party that signs a certificate (self-signed or not) endorses its contents. Trusting the party that signed a certificate can be a reason for believing its contents.

Two different architectures have been developed to make use of third party signing of public key certificates: the hierarchical Public Key Infrastructure (Section 2.5) and the cryptographic Web of Trust (Section 2.6). In both architectures, an application or hosting environment is initially configured with a trusted set of certificates known as *trust anchors*. When presented with an unknown certificate, an application verifies its authenticity by attempting to build a certification path — or chain — between the certificate and one of the trust anchors. A certificate becomes trusted if and only if it has been signed using a certificate which is already trusted. If necessary, this operation may be repeated to build a path through intermediate certificates, through which the trust relation is transitive.

The hierarchical Public Key Infrastructure model and the cryptographic Web of Trust model mainly differ in the way in which certificates are distributed and intermediates are trusted. In both cases, the initial establishment of trust (i.e. the selection of trust anchors) requires an initial import of certificates which is out of band, but this process is much less onerous than obtaining all public key certificates for all entities likely to take part in secure communications.

2.5   PKI and hierarchical model of trust

The *Internet X.509 Public Key Infrastructure* [5] (PKI) is a hierarchical model for distributing and trusting certificates. In this model, certificates are signed by a *certification authority* (CA). X.509 certificates incorporate a *Subject Distinguished Name* (Subject DN), which identifies the subject of the certificate, and an *Issuer Distinguished Name* (Issuer DN), which identifies the issuer of the certificate — the entity that signs the certificate . An X.509 certificate may only have one Issuer DN, which must be the Subject DN of the certificate that has been used to issue it. This structure builds a hierarchical tree from the root CA certificate, via optional intermediate CA certificates, to the end-entity (i.e. client or server) certificates.

Most web-browsers and operating systems provide a default list of CA certificates which they make their users trust implicitly. This list can usually be changed by the user, a feature often used by institutional PKIs.

2.6   Cryptographic Web of Trust

The cryptographic *Web of Trust* (WoT) is a form of public key infrastructure (although rarely called PKI) where each participant may assert trust in any other participant, without a specific hierarchy.

The Web of Trust model is used by PGP; what is often referred to as a *PGP public key* is, in fact, a form of a public key certificate, since it also contains additional information (such as an e-mail address) and is signed so as to assert its authenticity. Such a certificate is self-signed, but may also contain additional signatures — those by whom the association between the key and this additional information is trusted.

In PGP, the trust anchors are the user's own certificate and the certificates the user trusts, some of which may be from trusted introducers (that is, people through whom trust is transitive). The number a signatures a certificate has reflects the connectivity to other parties. The more signatures a certificate has, the more likely it is that a third party will be able to find a certification path to that certificate via trusted introducers.

2.7   SSL authentication

The most widely deployed protocol for securing communications between a user-agent and a web server is Transport Layer Security (TLS) [6], itself a successor to the Secure Socket Layer 3.0 (SSLv3) specification;[11] its use in HTTP applications is denoted by the `https` prefix in URLs.

During the SSL handshake, at the beginning of the SSL connection, the client obtains an X.509 certificate from the server. At this point, the client relies on its trust anchors to verify it. If this certificate is trusted and verified, the handshake proceeds. Once the handshake has finished, the communication (on

---

[11] Unless explicitly noted, this article uses SSL to encompass TLS 1.x and SSL 3.0.

top of SSL) can proceed in a secure manner; the only other party capable of reading the communication must have the private key corresponding to this server certificate.

There exists a variant of the handshake procedure in which the client is requested or required to present a certificate to the server, enabling the server to authenticate the client using the same verification method as above.

The remainder of this section describes, from a Semantic Web point of view, how trust in a certificate is evaluated. This forms the basis for comparison of how FOAF+SSL differs from this, in Section 3.2. We describes the reasoning of a server, S, for authenticating a client, `:client`, making a request. Server S has a set of trusted CAs. S would state that `issuerDN` was a trusted CA with:

(P2)
```
issuerDN a :TrustedCA;
         :hasPrivateKeyFor CAKey .
```

The `CAKey` is a `cert:PublicKey` that is usually identified by a number of inverse functional properties, which form an OWL2 key.[12] For the sake of brevity, these relations are not shown here. Suffice it to say that CA Keys can be uniquely identified by them.

S requests that `:client` presents a certificate signed by any one of a number of CAs it knows about. S receives `_:certDoc` with semantics such as the following (the subject is also identified via a DN):

(P3)
```
_:certDoc :semantics _:certSemantics .
_:certSemantics = { <> dc:created issuerDN;
                       foaf:primaryTopic subjectDN .
                    subjectDN :hasPrivateKeyFor pubKey .
                    issuerDN :hasPrivateKeyFor CAKey .  }
```

So far, the SSL handshake ensured S that the client has the private key:

(P4)
```
:client :hasPrivateKeyFor pubKey .
```

The client asserts the contents of the certificate (not shown) and that it is signed by `issuer`:

(P5)
```
:client :claims  { _:certDoc :signature _:certSig;
                   _:certSig :signedWith CAKey;
                             :sigString "XYZ SIG" . }
```

S can assert, after verification, that `_:certDoc` has been signed using the private key corresponding to `CAKey`:

(P6) `_:certDoc :signature [ :signedWith CAKey ] .`

Proving that a document is signed by P, is to assert P claims its contents:

(D3)
```
{  ?P :hasPrivateKeyFor ?key .
   ?doc :signature [ :signedWith ?key ]
} => {  ?P :claims [ is :semantics of ?doc ] } .
```

---

[12] http://www.w3.org/TR/owl2-syntax/#Keys

Then, from the signature verification P6, the certificate contents P3 and the definition D3, `S` can assert:

(P7) `issuer :claims _:certSemantics .`

To trust someone is to trust what they claim. `S` trusts `TrustedCA`s, thus:

(D4) `{ ?ca :claims ?s . ?ca a :TrustedCA  } => { ?s a log:Truth } .`

From P2, P7 and D4, `S` can conclude:

(P8) `subjectDN :hasPrivateKeyFor pubKey .`

From P4 gained by the SSL handshake, the above P8 and the definition D2 of `:hasPrivateKeyGot` as a `owl:inverseFunctionalProperty`, we can deduce:

(P9) `:client owl:sameAs subjectDN .`

At this point, the server `S` has authenticated the `:client` as this Distinguished Name (DN). Considering that the `:client`'s request is to access resource `R`, the server can then find out if this DN is authorized access to `R`.

The problem with DNs is that, although they can be made to form a URI, the dereferencing mechanism for DNs is not global in the way `http` URLs are. Therefore, if access to `R` is granted in some rule based way, where more information about `R` needs to be discovered for a decision to be made, then the DN cannot provide a global solution. For very much the same reasons, data in LDAP servers cannot have fields that point resources in any other LDAP server. As a result, current uses of client certificates limit the usage of each to a few domains.

The ability to link globally is an essential piece required for building a global social network. The next sections shows how FOAF+SSL solves this problem.

3   The FOAF+SSL protocol

This section describes the FOAF+SSL protocol. The FOAF+SSL protocol uses SSL but uses a different trust model than PKI to verify certificates.[13]

When protecting a service, it is important to differentiate authentication from authorization. Authentication is the action of verifying the identity of the remote user. Authorization consists of allowing or denying access to or operations on a given resource, based on the identity obtained during authentication.

FOAF+SSL enables a server to authenticate a client given a simple URL. This URL can then be used directly for authorization, or to explore more information in the web of linked data, in order to decide if the referent of the URL satisfies the constraints required for accessing the resource.

---

[13] Although the examples we use are based on the Web, FOAF+SSL could in principle also be used for authentication to other SSL-enabled services, such as IMAPS.
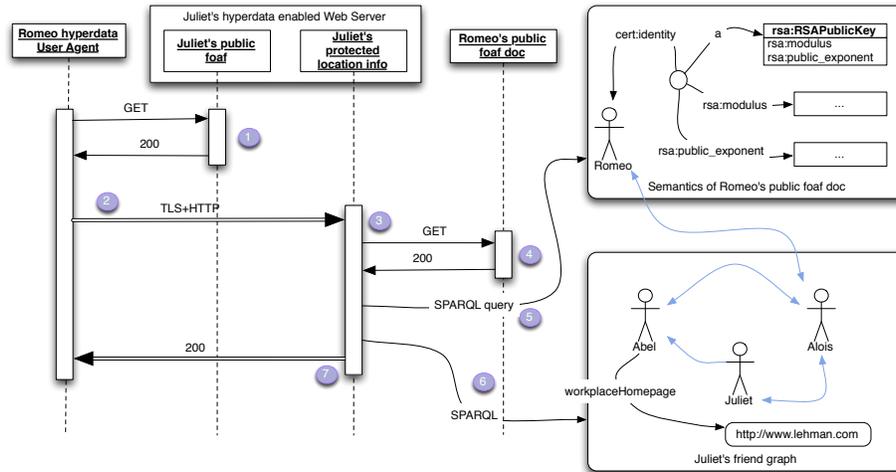
3.1 Protocol sequence

Fig. 1. The FOAF+SSL sequence diagram.

The FOAF+SSL authentication protocol consists of the following steps, as illustrated in Figure 1:

1. A client fetches a public HTTP resource which points to a protected resource, for example `<https://juliet.example/location>`.
2. The client, `romeo:i`, dereferences this URL.
3. During the SSL handshake, the server requests a client certificate. Because FOAF+SSL does not rely on CAs, it can ask for *any* certificate. The client sends Romeo's certificate (which may be self-signed) containing its public key (see "`Subject Public Key Info`" in Listing 1.1) and a *Subject Alternative Name* URI (see "`X509v3 extensions`" in Listing 1.1). Because the SSL handshake has been successful, Juliet's server knows that Romeo's client has the private key corresponding to the public key of the certificate.

Listing 1.1. Excerpt of a text representation of a FOAF+SSL certificate.

```
Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (1024 bit)
            Modulus (1024 bit):
                00:b6:bd:6c:e1:a5:ef:51:aa:a6:97:52:c6:af:2e:
                71:94:8a:b6:da:9e:5a:5f:08:6d:ba:75:48:d8:b8:
                [...]
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Alternative Name:
                    URI:https://romeo.example/#i
```

4. Juliet's server dereferences the Subject Alternative Name URI found in the certificate and ends up with a superset of D1.
5. The document's *log:semantics* is queried for information regarding the public key contained in the X.509 certificate. This can be done in part with a SPARQL query as shown in Listing 1.2. If the public key of the certificate matches the one published in the FOAF file, `romeo:i` is authenticated.

Listing 1.2. SPARQL query to obtain the public key information.

```
SELECT ?modulus ?exp WHERE {
  ?key cert:identity <https://romeo.example/#i>;
       a rsa:RSAPublicKey;
       rsa:modulus [ cert:hex ?modulus; ];
       rsa:public_exponent [ cert:decimal ?exp ] .   }
```

6. Once this authentication step is complete, the position of `romoe:i` in Juliet's social graph can be determined. Juliet's server can get this information by crawling the web starting from her FOAF file, or by other means.
7. Authentication has been done; authorization can now take place.

3.2 Authentication Logic

This section draws a parallel with Section 2.7, again, following the reasoning of the web server `S` trying to authenticate a `:client`.

At the end of stage 3 in the FOAF+SSL sequence diagram, `S` has received the client certificate securely. Being self-signed (or signed by an unknown party), its semantics are somewhat different. `S` is really only interested in the URI identifiers referring to the subject — abandoning thus the limitations of DNs. In addition, since it is asserted by the client, `S` knows that:[14]

(P10)
```
:client :claims { <>  dc:created romeo:i;
                      foaf:primaryTopic romeo:i.
                  romeo:i :hasPrivateKeyFor pubKey .  }
```

`S` may know of `romeo:i` only what it gathered from the SSL handshake:

(P11) `:client :hasPrivateKeyFor pubKey .`

When someone makes a claim, they have to agree with the logical consequences of their claim, including those arising from new facts. Thus, someone who makes a claim `:mustAgree` with the conclusions of the union of what we know securely, what they believe, and established reasoning rules:

(D5)
```
{ ?client :claims ?clientGraph .
 ( ?clientGrph   secureFactGraph   owlReasoningRules )
    log:conjunction [ log:conclusion ?C ] }
           => { ?client :mustAgree ?C  }
```

---

[14] The signer being the author, following the reasoning from P10, P6, D3 would also end up with this result — for self signed certificates only.

Hence, from P10, P11, and D2, `S` may conclude that `:client` would have to agree that it is `romeo:i`. This should not be a surprise, as that is indeed what one assumes someone who sends such a certificate intends.

(P12) `:client :mustAgree [ log:includes { romeo:i = :client } ] .`

Since `:client` asserts it is `romeo:i`, it accepts to be inspected via `romeo:i`. Since `romeo:i` is a URL, `S` can dereference it and thus discover P1. Then, by P1, P11, D2, and D5:

(P13) `romeo:i :mustAgree [ log:includes { romeo:i = :client } ] .`

In other words, both `romeo:i` and `:client` must agree, given what `S` knows, that `romeo:i owl:sameAs :client`. In particular `romeo:i` cannot repudiate this assertion since `romeo:i` itself provided P1 authoritatively. It follows that if `S` is authorized to serve `R` to `romeo:i`, `S` can serve `R` to `:client`.

3.3 Following links

In the previous section, we showed how a server can authenticate a client who claims a Web ID. Authorization policies — how to decide which group of agents may access which resource — will be particular to each application. It could be done by giving a list of authorized Web IDs. It could be done by trusting statements returned by a selected group of Web IDs, for example, when allowing all friends of a given friend access to a resource, as specified by the representation returned by their Web ID. This would make the initial list of trusted Web IDs similar to the trust anchors in PKI and WoT. A lot still remains to be explored.

A topic for further research is to define the various ways in which trust can be transmitted. Let us look at one simple example here. Imagine a user connects to a service and is authenticated as `joe:i`, using the FOAF+SSL method described above. Imagine `joe:i` returns a representation claiming `joe:i owl:sameAs romeo:i`. Since anyone could make that statement, that, in itself, should not be the basis for belief for services that care about security. If, on the other hand, `romeo:i :semantics [ log:includes { romeo:i owl:sameAs joe:i . } ]`, then this could be used as confirmation of the claim, and from there on both IDs could be used interchangeably by `S`.

4 Related work

Unlike the OpenPGP extension to TLS [7], which also aims to rely on a Web-of-Trust by using PGP certificates instead of the usual X.509 certificates, FOAF+SSL makes only slight changes in the way X.509 certificates are used; it does not require changes in the actual SSL stack. With the OpenPGP TLS extension, the problem of key distribution still remains. Public PGP keys can be stored on public key servers, but there is no global dereferencing mechanism for finding a key, as there is in the FOAF+SSL protocol.

OpenID also shares considerable similarities with FOAF+SSL, due in part to OpenID's reliance on URLs as identifiers, just as FOAF+SSL relies on dereferenceable URIs bearing FOAF data. However, OpenID fails to make much use of the information at the OpenID resource, using it only to find the authorization service. As a result, OpenID requires a much higher number of connections to establish identity — 6 as opposed to 2 — and parts ways with RESTful design in the attribute exchange protocol, loosing thereby the advantages of a networked architecture.

5 Conclusions

FOAF+SSL provides a secure and flexible global authentication system. With public key cryptography at its core, it gains all the security advantages of this technology. Because FOAF+SSL is RESTful and integrates well with the Semantic Web, it can discover more information about an entity by walking the Linked Data cloud. Compared with PKI, FOAF+SSL removes the need for hierarchical authorities to assert identity, making it much more flexible. Thus, this mechanism adapts itself well to the formation and expansion of virtual organisations and distributed social networks.

Acknowledgements

References

1. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000) http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
2. Jacobs, I., Walsh, N., eds.: Architecture of the World Wide Web, Volume One. (December 2004) http://www.w3.org/TR/2004/REC-webarch-20041215/.
3. Hendler, J., Golbeck, J.: Metcalfe's law, Web 2.0, and the Semantic Web. Web Semant. **6**(1) (2008) 14–20 http://www.cs.umd.edu/~golbeck/downloads/Web20-SW-JWS-webVersion.pdf.
4. Golbeck, J., Parsia, B., Hendler, J.A.: Trust networks on the semantic web. In: WWW (Posters). (2003) http://mindswap.org/papers/Trust.pdf.
5. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard) (May 2008)
6. Dierks, T., Allen, C.: The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard) (January 1999) Obsoleted by RFC 4346, updated by RFC 3546.
7. Mavrogiannopoulos, N.: Using OpenPGP Keys for Transport Layer Security (TLS) Authentication. RFC 5081 (Experimental) (November 2007)