FOAF+SSL: RESTful Authentication by and for Distributed Social Networks

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 – without requiring the user to enter an identifier or password – using existing browsers, and for distributed, open and yet secure social networks. After describing each of these technologies and how they come together in FOAF+SSL,[4] we show declaratively the reasoning of a server relying on this authentication mechanism to make authorization decisions.

1 Introduction

Many services that require authentication rely on centralized systems (such as those backed by an LDAP database). The identity of the user is constrained to that administrative domain, forcing the user 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 users in distinct organizations difficult.

Every time a new user 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 requires a separate protocol and the definition

[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/

of custom attributes to obtain more information about the user. Neither Shibboleth nor OpenID fully comply with web architecture principles (see Section **??**), thereby making it difficult to gain additional information about a user in the decentralized, hyperlinked environment of the Web.

This paper proposes a novel approach to this problem of distributed authentication 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 — as well as more complex requirements, where the authorization to a service depends on knowledge of the position of the user in the social network, inferred from Semantic Web relations. Unlike the aforementioned Shibboleth and OpenID, this proposal makes use of a RESTful architecture, the same that underpins the largest and most successful network of distributed linked information, the Web.

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

2 Background

2.1 The RESTful Web Architecture

Representational State Transfer (REST) [**?**, Chap. 5] is an architectural style for building large-scale distributed information networks, the most famous of these being the World Wide Web [**?**]. 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) — the best known being the subset called Universal Resource Locators (URLs).
2. The mapping of URIs to Resources. This is the reference part of the semantic aspect.
3. Canonical methods for manipulating these 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 accessing the object at a given HTTP URL, information about the resource, known as the *representation* of the resource, can be fetched. The resource can be changed if permitted – including here creation or deletion as limiting cases.

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 interlinked resources. In the Semantic Web, it becomes possible for URLs to refer to anything, be it:

1. concrete things, like individuals — 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 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, clever 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 web page describing the 'knows' relation. A Semantic Web agent could ask for the standard machine-friendly RDF representation; other representations could be returned to describe the same information.

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, and `object` may also be a string literal. 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.` 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 { }.[7] 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` URL, one may state the following, without necessarily asserting the actual statements within the brackets as true:

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

[7] These are similar to the Named Graph curly brackets in SPARQL, except that the N3 notation allow anonymous graphs.

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

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:

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

The `log:` namespace[8] 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 and for what reason merging graphs is reasonable.

2.3 FOAF, reputation networks and the Web of Trust

FOAF,[9] short for Friend-of-a-Friend, is an RDF vocabulary used to describe people, agents, groups and their relations in a practical way. When used on the Semantic Web, this allows each person to describe 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 P**??**) 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 contains further URIs, such as `jult:me`, whose `:semantics` the agent can also GET. If `romeo:i` uses a URI, it is because he means to refer objectively to something further described by the `:semantics` of that URI . 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. There is an incentive to link to existing resources: less work maintaining that information.

As the network grows, the value of the network grows exponentially, as predicted by Metcalf's Law [**?**], creating a virtuous circle. Current social network-

[8] The `log:` namespace is described at http://www.w3.org/DesignIssues/N3Logic.
[9] Defined at http://xmlns.com/foaf/0.1/.

ing sites, such as Facebook and LinkedIn, and older ones, such as eBay with its transaction voting mechanism, 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* [**?**], 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).

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. This leads to the definition of an inverse functional property `:hasPrivateKeyFor`, in Definition D**??**.

(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 public/private key pair, so that it may only be decrypted using the corresponding private key, ensuring that communications may not be decrypted by any other recipient than the one intended.
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 too.

Two different architectures have been developed to make use of third party signing of public key certificates: the hierarchical Public Key Infrastructure (Section **??**) and the cryptographic Web of Trust (Section **??**). 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 the authenticity of the certificate 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* [**?**] (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. 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 (which consists of signing the issued certificate using the private key corresponding to the issuer certificate). 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.

The repository of trust anchors may have different names depending on the platform and application. In practice, 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, in order to add, replace or delete CA certificates. The ability to customize the default list in these ways is widely used in corporate or institutional PKIs.

2.6 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).

Key-signing parties are often used for adding new trust anchors. If a participant, A, checks the identity of participant B, then A may opt to sign B's certificate, thus asserting to A and to any third party who may have A as a trusted introducer that the information in B's certificate is accurate. Furthermore, direct trust may now be established in future communications between A and B through the use of public-key encrypted communications between A and B.

If A also trusts B to make reasonable decisions in what certificates B signs, A can also add B as a trusted introducer, so that A can trust the certificates that B has signed. Furthermore, as more people sign B's certificate, it becomes more likely that a third party will be able to find a certification path from themselves to B via a chain of trusted introducers. To make distribution of keys more practical, these certificates — signed by as many people as possible — may be stored on public key servers.

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) [**?**], itself a successor to the Secure Socket Layer 3.0 (SSLv3) specification;[10] its use in HTTP applications is denoted by the `https` prefix in URLs.

When establishing an SSL connection, as part of the SSL handshake, 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 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 **??**.

The following 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 .
```

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

The CAKey is a cert:PublicKey that is usually identified by a number of inverse functional properties, which form an OWL2 key.[11] 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 are true:

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

Then, from the signature verification P**??**, the certificate contents P**??** and the definition D**??**, S can assert:

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

To trust someone is to trust what they claim. S trusts TrustedCAs, thus:

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

From P**??**, P**??** and D**??**, S can conclude:

(P8)
```
subjectDN :hasPrivateKeyFor pubKey .
```

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

Putting P?? gained by the TLS connection and the above P?? together with the definition D?? of `hasPrivateKey` 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 `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 were designed to be. 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 to any resource 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.[12]

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.

3.1 Protocol sequence

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

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.

[12] 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.

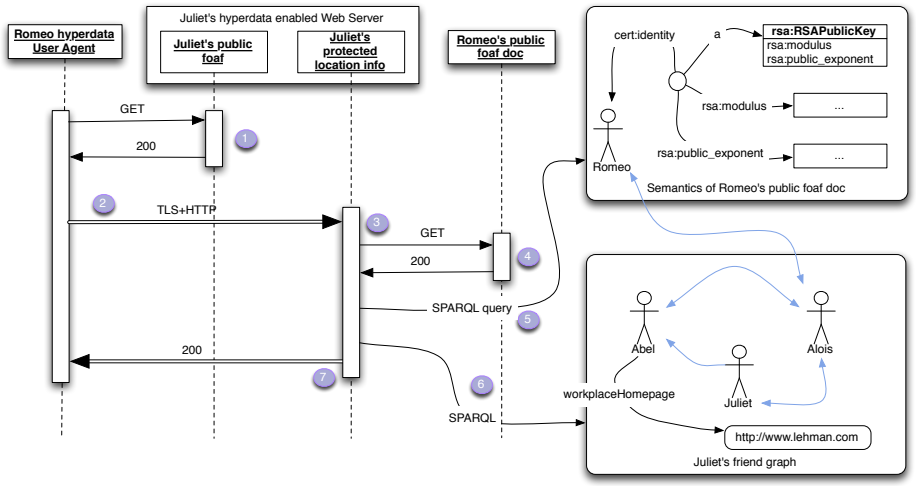

Fig. 1. The FOAF+SSL sequence diagram.

3. During the SSL handshake (when the connection is initiated), 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 **??**) and a *Subject Alternative Name* URI (see "`X509v3 extensions`" in Listing **??**). 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.
4. Juliet's server dereferences the Subject Alternative Name URI found in the certificate and fetches an RDF document.

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
```

5. The document's *log:semantics* is queried for information regarding the public key contained in the X.509 certificate mentioned previously. This can be done in part with a SPARQL query as shown in Listing **??**. If the public key of the certificate is found to be identical to the one published in the FOAF file, this proves that the client is using the URI correctly.
6. Once this fundamental authentication step is complete, Romeo's identity (as represented within the server) may also be augmented with information

regarding its position in a graph of relations (including friendship ones) in order to determine a degree of trust according to some criteria. 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.

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 ] .   }
```

3.2 Authentication Logic

This section draws a parallel with Section **??**, 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. Furthermore, what interests `S` in this FOAF+SSL certificate is only the URI identifiers to refer to the subject, thus abandoning the limitations of DNs. In addition, since it is asserted by the client, `S` knows that:[13]

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

`S` may know nothing of `romeo:i`. However, it knows from the SSL handshake that:

(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. We can write this out as the following rule:

(P12)
```
{  ?client :claims ?clientGraph .
 ( ?clientGrph   secureFactGraph   owlReasoningRules )
    log:conjunction [ log:conclusion ?C ] }
             => { ?client :mustAgree ?C  }
```

Hence, from P**??**, P**??**, and D**??**, `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.

[13] The signer being the author, following the reasoning from P**??**, P**??**, D**??** would also end up with this result — for self signed certificates only.

(P13) `_: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 dereferenceable URI, `S` can obtain more information about what the `_:client` claims to be by dereferencing `romeo:i`. Thus, `S` discovers P**??**. Then, by P**??**, P**??**, D**??**, and P**??**:

(P14) `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 P**??** 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 and how to decide which group of agents may access which resource will be for each application to decide. 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. The decision of what rule to follow to serve up a resource is very much up to its owner, and is a field where a lot still remains to be tried out.

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 [**?**], 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, whereas FOAF offers more flexibility in that respect.

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 spec, loosing thereby the advantages of a networked architecture.

5 Conclusions

FOAF+SSL provides a secure and flexible way to have a global authentication system. Through the use of public key cryptography, it increases security compared with other approaches, such as OpenID. In addition, the use of public key certificates may help verify more properties in the FOAF-based web-of-trust. FOAF+SSL is also RESTful and integrates well with the Semantic Web. Compared with PKI, FOAF+SSL removes the need for hierarchical authorities to assert identity, making it more flexible and less bureaucratic. Thus, this mechanism adapts itself well to the formation and expansion of virtual organisations and distributed social networks.

Acknowledgements

Ian Jacobi acknowledges funding for this project from NSF Cybertrust award #0524481 and IARPA award #FA8750-07-2-0031. Bruno Harbulot acknowledges EPSRC funding under grant EP/E001947/1 (http://www.nanocmos.ac.uk/).