FOAF+TLS: RESTful Authentication for Distributed Social Networks*

Henry Story1, Bruno Harbulot2, Ian Jacobi3, and Mike Jones2

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 very simple protocol for RESTful authentication, using widely deployed technologies such as HTTP, TLS and Semantic Web vocabularies. After describing each of these technologies and how they come together in FOAF+TLS4, we show declaratively the reasoning of a server relying on this authentication mechanism to make authorization decisions.

1 Introduction

Many web servers that require authentication rely on centralized systems (for example, backed by an LDAP service) that belong to the same administrative domain as the server. In this model, the user is restrained to this administrative domain and needs to have an account for each organization. This makes creation of links between data related to a given user held within two distinct organizations difficult. In addition, 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 in-depth—for example, in a workplace, where an administrator has to create an account, after verifications out-of-band. Process (a) is lightweight, but will often provide insufficient information, whereas process (b) may be able to give more information about a user, at the expense of a costly initial verification phase during the registration.

Attempts to decentralize this process have been made. Shibboleth5, for example, aims at sharing accounts across administrative boundaries; it does however rely on a rigid federation process between organizations. OpenId, enables authenticating a user against a URI, but requires a separate protocol and the definition of custom attributes for obtaining more information about the user.

* This article will be made available under the "Attribution 3.0 Unported" Creative Commons License, as soon as it is accepted by the SPOT2009 committee. Do not republish until then.

4 This is also known as FOAF+SSL. 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 (REST), thereby making it difficult to gain extra information about a user in the decentralized, hyperlinked ways of the Web.

This paper proposes a novel approach which relies on combining the use of TLS 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— and more complex requirements, where the authorization to a service depends on knowledge of the position in the social network of the authenticating agent, as inferred from documents containing FOAF and other Semantic Web relations. This is made possible by 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+TLS protocol. Section 4 compares this approach to other approaches.

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 close to no central coordination, and that each of the resources thus created be able to refer easily to any of the others, in a seamless manner.

The logical building blocks for this are the following:

1. The specification of universal names, also known as Universal Resource Identifiers (URI) — 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 piece.
3. Canonical methods for manipulating these resource mapped by each URIs, 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 for building 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. Moreover, any user should be able to reach any part of the system in such a way.
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 individual people — e.g. <http://romeo.net/#i> refers
to a human named Romeo;
2. relations between 2 individuals — e.g. the relation of knowing someone which
    <http://xmlns.com/foaf/0.1/knows> refers to;
3. or classes — e.g. the set of people: <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. As 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/XML representation and parse it;
yet other representations could be returned to describe the same information.

The meaning of any semantic web document is 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. As it is tedious to read
and write such URLs, this article uses the N3 @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.net/#> .
@prefix jult: <https://juliet.net/#> .

Therefore, to say that 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 { }. The
relation that maps a URI to its meaning is defined as the log:semantics
relation. Thus, starting with the romeo:i URL one can, after dereferencing it,
state the following, without being obliged to explicitly assert the actual state-
ments within the brackets as true:

romeo:i log:semantics { romeo:i a foaf:Person;
    foaf:name "Romeo";
    foaf:knows jult:me . }

6 Current N3 tutorial at: http://www.w3.org/2000/10/swap/doc/Overview.html
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, 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 WebId – one can describe one’s personal social network by linking oneself to one’s acquaintances by reference. Therefore, if someone had been given the romeo:i URL above by Romeo himself, and had found its log:semantics, there is good reason for him to trust that the information there is correct, and thus to merge it (in a defeasible manner) with his own belief store. As that graph itself contains further URIs such as jult:me’s, the agent could also get its log:semantics to find out more about its referent. The thought then is that, if romeo:i uses a URI, it is because he means to refer to the thing objectively referred to by that URI. By following the canonical way to dereference that URI, one should get a representation of that thing. Meaning is thus objective.

If the advantages gained by keeping information up-to-date is large enough, a peer-to-peer information network arises, where each person specializes in keeping the information they feel responsible for up-to-date, linking to the best sources for objects he or she does not wish to maintain. Current social networking sites such as Facebook and LinkedIn, and older ones such as eBay, have shown how this can work in less distributed settings. 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 so the value of the network grows exponentially, as predicted by Metcalf’s Law, creating a virtuous circle.

This could be combined with trust descriptions so as to create a reputation network, and, in the case of FOAF+TLS, 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. It achieves this 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 generally held only by the person who generated the key-pair. 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 in a reasonable amount of time, because this operation is too computationally
expensive (assumed but not proven to be NP-complete). It also assumes that no two distinct individuals will generate the same key-pair randomly; hence, there exists a need for suitable random number generators. This leads to defining hasPrivateKeyFor as an inverse functional property, in Definition D1

(D1)

\[
\text{hasPrivateKeyFor} \quad \text{a owl:InverseFunctionalProperty;}
\]

\[
\text{rdfs:domain foaf:Agent;}
\]

\[
\text{rdfs:range cert:PublicKey .}
\]

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 into a scrambled message using the public key of a public/private key-pair, so that only it may 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.

A self-signed certificate allows the owner of a key pair to make assertions about himself or herself, whereas a certificate signed by a third party may be used to assert independently a set of attributes associated with the holder of the private key, based on the trustworthiness of the third party.

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 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.

Usually, the successful verification of a new certificate implies the establishment of only a temporary level of trust of the other party for the duration of some operation performed with the certificate. Applications may also, through some external mechanism, add some certificate to their set of trust anchors explicitly.

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. 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 it 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 implicitly trust), but this list can be changed by the user, and other CA certificates can be explicitly trusted. This may happen, for example, for a corporate or institutional PKI.

2.6 Web of Trust

The cryptographic Web of Trust (WoT) is a form of public key infrastructure (although almost never 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 TLS 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\[7\]; its use in HTTP applications is denoted by the https prefix in URLs.

When establishing a TLS connection, as part of the TLS 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 completed, the communication (on top of TLS) 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 much less used variant of the handshake procedure in which the client is requested or required to present a certificate to the server enabling the server to identify the client using exactly the same method as above.

The remainder of this section describes, from a Semantic Web/logical point of view, what follows this handshake. This forms the basis for comparison of how FOAF+TLS differs from this, in Section \[3.2\].

The following describes the reasoning of a server, $S$, that must make an authorization decision regarding some resource, $R$, requested by a client, $\_:\_client$. Server $S$ has a set of trusted CAs. $S$ would state that $issuer$ was a trusted CA with:

(P1) \[
\text{issuerDN a TrustedCA; hasPrivateKeyFor CAKey .}
\]

The CAKey is a cert:PublicKey that is usually identified by a number of inverse functional properties, which form an OWL2 key\[8\]. 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):

(P2) \[
\_\_certDoc log:semantics \_\_certSemantics .
\_\_certSemantics = { <> dc:created issuerDN ; foaf:primaryTopic subjectDN . subjectDN hasPrivateKeyFor pubKey . issuerDN hasPrivateKeyFor CAKey . }
\]

The TLS connection ensured that $S$ knows the client has the private key:

(P3) \[
\_\_client hasPrivateKeyFor pubKey .
\]

The client asserts the content of the certificate (not shown). It also claims it is signed by $issuer$:

\[7\] Unless explicitly noted, this article uses TLS to encompass TLS 1.0 and SSL 3.0.

\[8\] http://www.w3.org/TR/owl2-syntax/#Keys
S can assert after verification, that \( _\text{certDoc} \) has been signed using the private key corresponding to CAKey:

(P5) \( _\text{certDoc} \) signature [ signedWith CAKey ] .

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

(D2) \( \{ \ ?P \ \text{hasPrivateKeyFor} \ ?key . \ ?doc \ \text{signature} \ [ \ \text{signedWith} \ ?key ] \ \} \Rightarrow \{ \ ?P \ \text{claims} \ [ \ \text{is} \ \log:\text{semantics of} \ ?doc \ ] \ \} \).

Then, from the signature verification P5, the certificate contents P2 and the definition D2, S can assert:

(P6) issuer claims \( _\text{certSemantics} \).

To trust someone is to trust their claims. The server S trusts all claims made by TrustedCAs:

(D3) \( \{ \ ?ca \ \text{claims} \ ?s . \ ?ca \ a \ :\text{TrustedCA} \ \} \Rightarrow \ ?s \).

From P1, P6 and D3, S can conclude:

(P7) subjectDN hasPrivateKeyFor pubKey .

Putting P3 gained by the TLS connection and the above P7, together with the definition D2 of hasPrivateKey as a owl:inverseFunctionalProperties, we can deduce:

(P8) \( _\text{client} \) owl:sameAs subjectDN .

TLS as used above functions then by proving the authorship of the signature on the client certificate, and trusting that author.

At this point the server S has a Distinguished Name (DN) as a handle on subject. It can then find out if the DN is authorized access to the resource R. The problem with DN is that, though they can be made to form a URI, the dereferencing mechanism for DN 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. The same is true of the data in LDAP servers; these cannot have fields that point to any resource in any other LDAP server. Such linking is an essential piece required for building a global social network. As a result, a server that follows this procedure needs to have prior agreement with a limited number of servers for the client certificate to be of value. The next sections shows how FOAF+TLS solves this problem.
3 The FOAF+TLS protocol

This section describes the FOAF+TLS protocol. The FOAF+TLS protocol functions on top of TLS; the only difference with PKI uses of TLS is in the way FOAF+TLS verifies and trusts certificates.

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. For each resource then is associated a set of agents that can access it. This set can be specified in a number of ways: by simple enumeration of the members of the set using URIs, or by description — those members satisfying certain properties.

FOAF+TLS 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

Fig. 1. The FOAF+TLS sequence diagram.

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

9 Although the examples we use are based on the Web, FOAF+TLS could in principle also be used for authentication to other TLS-enabled services, such as IMAPS.
1. A client fetches a public HTTP resource which points to a protected resource for example <https://juliet.net/location>.
2. The client romeo:i dereferences this URL.
3. During the TLS handshake (when the connection is initiated), the server requests a client certificate. Because FOAF+TLS 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 TLS handshake has completed successfully, 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.
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 1.2. 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 1.1. Excerpt of a text representation of a FOAF+TLS certificate.

```
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
[...]
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Subject Alternative Name:
URI:http://romeo.net/#i
```

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

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

10 This is allowed by TLS 1.1 in RFC4346 section 7.4.4. It may also be possible to request certificates from well known Open CAs — CAs who, by publishing their private key, would allow anyone to sign with them — enabling FOAF+SSL servers to filter certs to those intended to be used this way.
3.2 Authentication Logic

This section draws a parallel with Section 2.7, again, following the reasoning of the web server $S$ in giving $:\_:\_\text{client}$ access to some resource $R$.

At the end of stage 3 in the FOAF+TLS sequence diagram, the server $S$, 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+TLS certificate are only the URI identifiers to refer to the subject, abandoning thus the limitations of DNs. In addition, since it is asserted by the client, $S$ knows that:

\[
\text{_:client claims _:clientGrph}.
\]

(P9)  \[
\text{_:clientGrph = \{ dc:created romeo:i; foaf:primaryTopic romeo:i. romeo:i hasPrivateKeyFor pubKey. \} }
\]

$S$ may know nothing of $\text{romeo:i}$. However, it knows from the TLS session that:

(P10)  \[
\text{_:client hasPrivateKeyFor pubKey}.
\]

Hence, it can conclude from P9, P10, and D1 that $\text{romeo:i}$ would have to agree with it that it is $\text{_:client}$. This should not be a surprise, as that is indeed what one assumes someone who sends such a certificate intends.

(P11)  \[
(\text{_:clientGrph _:client hasPrivateKeyFor pubKey}) \quad \text{log:conjunction [} \\
\Rightarrow \text{\{ romeo:i = _:client \} ]}
\]

Thus, since $\text{_:client}$ asserts it is $\text{romeo:i}$, it can find no harm in $S$ finding more information about itself by looking at $\text{romeo:i}$. $S$ can do that, as $\text{romeo:i}$ is a dereferenceable URI, whereas $\text{_:client}$ and $\text{pubKey}$ are not.

(P12)  \[
\text{romeo:i log:semantics _:romeoGrph}.
\]

(P13)  \[
\text{_:romeoGrph = \{ romeo:i hasPrivateKeyFor pubKey \} }.
\]

Again, by P12, P10, and D1:

(P13)  \[
(\text{_:romeoGrph _:client hasPrivateKeyFor pubKey}) \quad \text{log:conjunction [} \\
\Rightarrow \text{\{ romeo:i = _:client \} ]}
\]

In other words, both $\text{romeo:i}$ and $\text{_:client}$ would agree, given what $S$ knows, that $S$ can consider $\text{romeo:i}$ $\text{owl:sameAs} \text{_:client}$. In particular $\text{romeo:i}$ cannot repudiate this assertion since $\text{romeo:i}$ itself provided P12 authoritatively. It follows that if $S$ is authorized to serve $R$ to $\text{romeo:i}$, $S$ can serve $R$ to $\text{_:client}$.

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,

\footnote{The signer being the author, following the reasoning from P9, P5, D2 would also end up with this result — for self signed certificates only.}
FOAF+TLS makes only slight changes in the way X.509 certificates are used; it does not require changes in the actual TLS 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+TLS, due in part to OpenID’s reliance on URLs as identifiers, just as FOAF+TLS relies on dereferenceable URLs bearing FOAF data. But it 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+TLS 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+TLS is also RESTful and integrates well with the Semantic Web. Compared with PKI, FOAF+TLS removes the need for hierarchical authorities for asserting identity making it more flexible and less bureaucratic. Thus, this mechanism adapts itself well to the formation and expansion of virtual organisations.

References