This document looks better online @ http://magnetik.github.com/node-webid-report/.

WebID provider using Node.JS

Julie Garrone
Baptiste Lafontaine

Supervisors:
Olivier Berger
Andrei Sambra

Table Of Contents

1- Introduction

People have been using login and password to register and identify themselves on Internet for many years, but is it the best solution? Large companies, such as Google, are looking for better and more secure ways to identify a user (two steps login for instance).

Aren't you tired of remembering long and complicated passwords (or worse, are you using the same password everywhere)? To enter your name and address every time you want to join a website?

Moreover do you trust multinational companies to hold all your personal data? No? Then...

Then you will love WebID!

The WebID protocol provides a way of identifying a user without the need of the couple username/password. In addition, the user is in control of his own data because he can store it anywhere he wants.

This protocol could be part of the solution of the current issue of the *distributed social networks*. The idea developed here is to enable people to control the access of their own information by having their own server which contains all their personal data. Therefore the social network has no more control over your data.

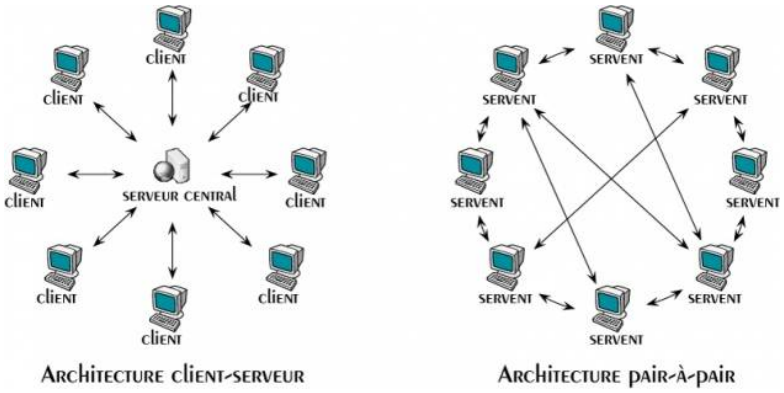

Figure 1: Architectures of a traditionnal social network (as Facebook) versus a distributed social network (like for example Diaspora).

1.1- Topic

The initial topic, available here *[http://www-inf.it-sudparis.eu/COURS/ASR/projets /sujet_OlivierBerger_AndreiSambra.pdf]* (document provided in French), expressed two guidelines to follow:

- Do a WebID identity provider using Node.Js;
- The provider had to be portable

After discussing the subject, we decided doing a state of the art of WebID technology would become part of our work too.

2- Domain analysis

2.1- WebID

Disclaimer: the following describes WebID in its "W3C Editor's Draft 12 December 2011 *[http://www.w3.org/2005/Incubator/webid/spec/drafts/ED-webid-20111212]* " [1] *[#ref1]* version. It may have changed since; the latest version is available on W3C website *[http://www.w3.org /2005/Incubator/webid/spec/]* .

A WebID [2] *[#ref2]* is a way to uniquely identify a person, company, organization, or other agent using a URI. The term "WebID" was coined by Dan Brickley and Tim Berners-Lee in 2000.

The WebID Protocol authenticates a digital identity by allowing an Agent (e.g., a Web Browser) to prove possession of a private key, whose corresponding public key is tightly bound to this WebID. The private key is usually associated with a "certificate" on the user's computer, while the public key and WebID are defined in the FOAF file of the subject and described by a URI in the certificate.

In order to give the full context of a Client interaction with a Server we will illustrate the protocol with the following sequence diagram.

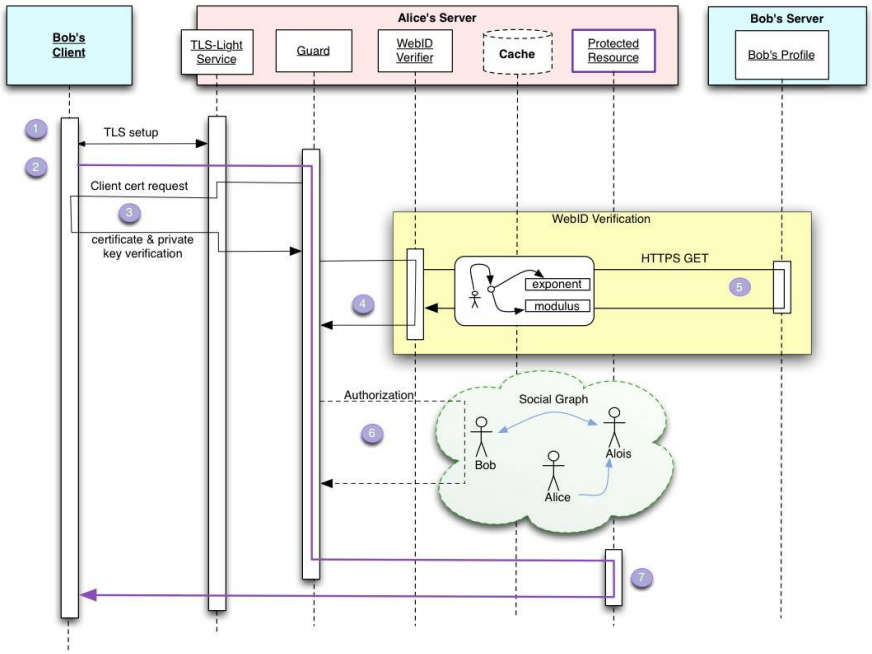

Figure 2: In this figure Bob initiates a connection to Alice's server in order to access a protected resource. Source: WebID spec.

1. Initialization of the TLS Connection
   - This is the first part of the HTTPS process. At this stage, the client identifies the server, but the server does not know the client.
2. Connection at the Application Layer
   - An agent, usually on the Server, called the guard is responsible for looking at a request from the Client and decides if it needs authentication.
3. Request of the Client Certificate
   - The guard requests of the TLS agent that it make a *certificate request* to the client.
   - The client asks Bob to choose a certificate.
   - The TLS agent verifies that the client is indeed in possession of the private key which secures the chosen certificate and its validity. (Part of TLS)
   - Then the TLS agent passes the certificate to the guard.
4. Verification of the WebIDs
   - The Guard asks the verification agent to perform the WebID part of the authentication. All steps before are part of classic TLS double way authentication.
5. Extraction of Bob's public key
   - The verification agent extracts the public key and the URI entries (which are called WebIDs) contained in the *Subject Alternative Name* field of the certificate.
   - The verification agent downloads the file identified by the URI (starts with one entry, and can continue with another is an error occurs).
   - The verification agent compares the public key in the downloaded file with the public key of the certificate. If it's the same, the client is identified !
6. Checking of the access control rules (the exact nature of those Access Control Rules is left for another specification).
7. Access to the protected resource.

It's worth emphasizing that most of the identification is done by TLS. Only steps 5 and 6 are specific to WebID.

SSL/TLS security protocol

We have already seen that WebID uses TLS protocol in order to verify both the server and the client, let's take a deeper look into TLS.

Transport Layer Security (TLS) [3] *[#ref3]* and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communication security over the Internet.

The TLS protocol allows client/server applications to communicate across a network in a way designed to prevent eavesdropping and tampering. The goals of the TLS protocol is to provide: cryptographic security, interoperability, extensibility, relative efficiency. Therefore it provides these different security aspects [4] *[#ref4]* :

- The authentication of the server
- An encrypted session which allows secure data transmission
- The integrity of exchanged data
- **The use of a certificate to authenticate the client**
- The transparency of the connection for the client
- The transparency for the http protocol (allows the same query in https than in http)

In the WebID protocol, the client must have a X509 certificate [5] *[#ref5]* with a subject alternative name URI entry that dereferences to a document which contains the public key for that certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5f:df:d6:be:2c:73:c1:fb:aa:2a:2d:23:a6:91:3b:5c
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: O=FOAF+SSL, OU=The Community of Self Signers, CN=Not a Certification Authority
        Validity
            Not Before: Jun  8 14:16:14 2010 GMT
            Not After : Jun  8 16:16:14 2010 GMT
        Subject: O=FOAF+SSL, OU=The Community Of Self Signers, CN=Bob (Personal)
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:cb:24:ed:85:d6:4d:79:4b:69:c7:01:c1:86:ac:
                    c0:59:50:1e:85:60:00:f6:61:c9:32:04:d8:38:0e:
                    07:19:1c:5c:8b:36:8d:2a:c3:2a:42:8a:cb:97:03:
                    98:66:43:68:dc:2a:86:73:20:22:0f:75:5e:99:ca:
                    2e:ec:da:e6:2e:8d:15:fb:58:e1:b7:6a:e5:9c:b7:
                    ac:e8:83:83:94:d5:9e:72:50:b4:49:17:6e:51:a4:
                    94:95:1a:1c:36:6c:62:17:d8:76:8d:68:2d:de:78:
                    dd:4d:55:e6:13:f8:83:9c:f2:75:d4:c8:40:37:43:
                    e7:86:26:01:f3:c4:9a:63:66:e1:2b:b8:f4:98:26:
                    2c:3c:77:de:19:bc:e4:0b:32:f8:9a:e6:2c:37:80:
                    f5:b6:27:5b:e3:37:e2:b3:15:3a:e2:ba:72:a9:97:
                    5a:e7:1a:b7:24:64:94:97:06:6b:66:0f:cf:77:4b:
                    75:43:d9:80:95:2d:2e:85:86:20:0e:da:41:58:b0:
                    14:e7:54:65:d9:1e:cf:93:ef:c7:ac:17:0c:11:fc:
                    72:46:fc:6d:ed:79:c3:77:80:00:0a:c4:e0:79:f6:
                    71:fd:4f:20:7a:d7:70:80:9e:0e:2d:7b:0e:f5:49:
                    3b:ef:e7:35:44:d8:e1:be:3d:dd:b5:24:55:c6:13:
                    91:a1
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation, Key Encipherment, Key Agreement
            Netscape Cert Type:
                SSL Client, S/MIME
            X509v3 Subject Key Identifier:
                08:8E:A5:5B:AE:5D:C3:8B:00:B7:30:62:65:2A:5A:F5:D2:E9:00:FA
            X509v3 Subject Alternative Name: critical
                URI:https://bob.example/profile#me
    Signature Algorithm: sha1WithRSAEncryption
        cf:8c:f8:7b:b2:af:63:f0:0e:dc:64:22:e5:8a:ba:03:1e:f1:
        ee:6f:2c:f5:f5:10:ad:4c:54:fc:49:2b:e1:0d:cd:be:3d:7c:
        78:66:c8:ae:42:9d:75:9f:2c:29:71:91:5c:29:5b:96:ea:e1:
        e4:ef:0e:5c:f7:07:a0:1e:9c:bf:50:ca:21:e6:6c:c3:df:64:
        29:6b:d3:8a:bd:49:e8:72:39:dd:07:07:94:ac:d5:ec:85:b1:
        a0:5c:c0:08:d3:28:2a:e6:be:ad:88:5e:2a:40:64:59:e7:f2:
        45:0c:b9:48:c0:fd:ac:bc:fb:1b:c9:e0:1c:01:18:5e:44:bb:
        d8:b8
```

Figure 3: Example of X509 certificate used in WebID protocol. We can see the "Subject Alternative Name" field as well as the public key.

For now, one of the main criticism is that TLS on server side is required to use WebID. Although it is a problem, there are more and more people believing that the web will be turned into "HTTPS only" soon [6] *[#ref6]* . For instance, Google or Facebook are now available in "SSL only" mode.

FOAF ontology

FOAF *[http://xmlns.com/foaf/spec/]* is a machine-readable ontology based on the Resource Description Framework *[http://www.w3.org/RDF/]* (RDF) and describing persons, their activities and their relations to other people and objects. It allows groups of people to describe social networks without the need for a centralised database: indeed all the relations between people are described in FOAF files.

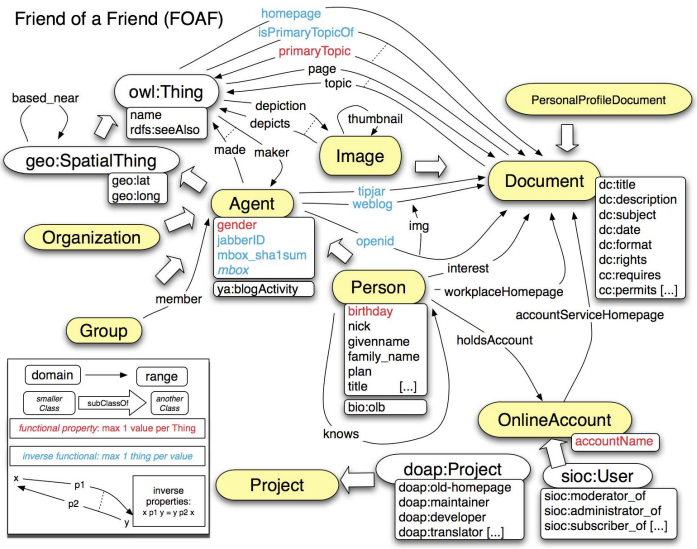

Figure 4: Graph of FOAF ontology. a user can fill in all information or only some of it.

The FOAF Ontology is used in the WebID protocol with accordance to the following graph:

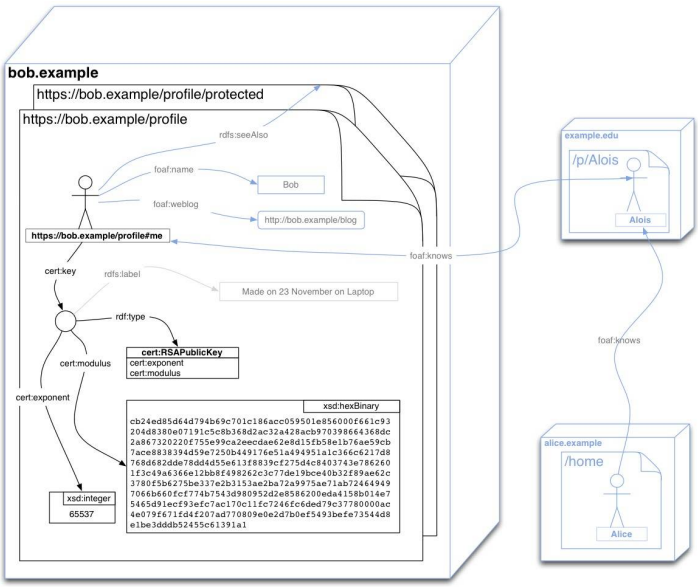

Figure 5: Graph of a FOAF profile in WebID

In this graph the WebID URI is *https://bob.example/profile#me*. The profile is located at the URL *https://bob.example/profile*.

The only mandatory field for a WebID is *cert:key* (composed of *cert:modulus* and *cert:exponent*): these fields must contain the public key(s) of the user. All other fields (name, birth etc.) are completely optionals.

More than authentication

Because WebID uses a RDF file to store the public key of the user on server side, most of WebID users store their key within a FOAF file to describe themselves.

Thanks to the extensibility of RDF, it can store any information.

For now, FOAF file seems to be the best choice because it can contains every data to identify a person, but we can imagine that other application specific data could be included.

Requirements

Despite the fact the client part of WebID is based on the quite old TLS client certificate, this may require some browser improvement in order to be more widely used.

In appendix 1, you can find screen captures of user certificate selection. Firefox is giving the most complicated window with a lot of information that is not very useful for the user, whereas Internet Explorer is showing a pretty nice window. Even if it seemss like a detail, it's important escpecially if it is

to be used by the mass.

Another requirement is a way for the website to interact with the browser TLS stack. There is no clear standard to say what is possible: for instance it's currently possible to logout from WebID with Firefox or Internet Explorer but not with Chrome. Indeed the JavaScript object `window.crypto` (and in particular it's method `window.crypto.logout()`) is note standardized by any organization. This may change because there is an HTML5 related group, the Web Crypto API *[http://www.w3.org/community/webcryptoapi/]* community group which is working on this issue in order to formalize what functions the browser should provide. Results of this group, and quick implementation by most browser manufacturers will be decisive for WebID.

Comparison

There are a few other ways to simplify and improve user login, some are standardized, some aren't.

The closest implementation is **BrowserID**, currently developed through the Mozilla foundation. The main difference is that BrowserID uses Javascript and a JSON certificate in order to work (whereas WebID uses TLS) and it uses the e-mail as the identifier. BrowserID require modification in browser and support from email providers.

By the way, WebID is not incompatible with this login system. For instance, OpenID could be used with WebID: for now, most OpenID providers use classic login/password, but we can imagine that the user login to the OpenID endpoint with WebID, then the classic OpenID process continues. This could be useful in order to prepare the transition to WebID because OpenID is quite widely deployed on Internet (for instance all Google ID are OpenID too).

W3C

Getting the status of a W3C Recommendation *[http://www.w3.org/standards/about.html]* helps to "make the web a better place"[1] by allowing both developers and users to have a better web experience. In January 2011, during one year, WebID was developed through a W3C Incubator Group *[http://www.w3.org/2005/Incubator/webid/]* (XG) which is usually (but not mandatory) the first step in the recommendation redaction process. Members of Incubator Groups are authorized to use W3C infrastructures (such as mailing-list, wiki, bug tracker, ...) and allows to gather experts and every people interested in a technology in one location. It's a process open to anyone, for free; but because the process is based on consensus, it's quite hard to make decision with a large number of people.

Most of discussions are made through the public-xg-webid@w3.org *[http://lists.w3.org/Archives/Public/public-xg-webid/]* mailing list[2]. There are also Teleconference meetings (accessible by phone, SIP[3], IRC...) every month which helps do decide what to focus on during the following weeks.

On January 2012, WebID XG was turned into a Community Group *[http://www.w3.org/community]* (CG). It's not any kind of evolution but the W3C is progressively deleting the concept of XG to CG.

After a chair request, the group can be moved to the "W3C Standards Track"[4]: it will be turned into a Working Group (WG). Because subscription to WG are not open: there are two way to join them:

- Be an employee of a W3C Member organization *[http://www.w3.org/Consortium/Member/List]* and ask to join (Firms have to pay a fee to be members),
- Be an invited expert *[http://www.w3.org/2004/08/invexp.html]* .

When reaching the recommendation track, the work is far from done. The recommendation will have to gp through these steps [7] *[#ref7]* :

1. Working Draft (WD): published as often as possible,
2. Last Call Working Draft (LC): the document is technically complete,
3. Candidate Recommendation (CR): the document is considered as stable,
4. Proposed Recommendation (PR),
5. W3C Recommendation (REC)

For some standards it can take years to complete this procedure. That's why it's important to have working implementation as soon as possible, even if the recommendation is not complete, in order to spread the standard.

At least before publishing the *Candidate recommendation*, a test suite shoud be available allowing people doing the implementation to clearly know if their implementation is correct or not. As of January 2012, there is no complete test suite for WebID which leads to problem such as simple implementation considerate that a WebID profile is correct and some considerate that there is a problem.

2.2- EARL

EARL is an acronym for Evaluation And Report Language. It's a RDF vocabulary defined in a in-development W3C Recommendation (currently in the Last Call process) [8] *[#ref8]* which provides a way to express tests (including assertions and results) in machine-readable format.

EARL + WebID

An EARL vocabulary is currently developed within the WebID group in order to formalize a number of tests that *must* be done in order to verify a WebID identify. The complete ontology *[http://www.w3.org/2005/Incubator/webid/earl/RelyingParty.n3]* currently provides more than twenty tests.

Some examples of the tests [9] *[#ref9]* :

- Does the client certificate contain a subject alternative name?

- Is the profile well formed?
- Does the profile contain only well formed keys for that WebID?
- ...

An example of result, in RDF serialization Notation3 *[http://en.wikipedia.org/wiki/Notation3]* could be:

```
[] a earl:Assertion;
       earl:subject _:certificate;
       earl:test wit:certificateProvidedSAN;
       earl:result [ a earl:TestResult;
               dct:description "";
               earl:outcome earl:passed;
```

We can see an RDF blank node representing an EARL assertion. This assertion has a subject, a test and a result. The subject is the certificate, the test is "is there any Subject Alternative Name (SAN) in this certificate?". The result is another anonymous node with the outcome set to "passed", so we know that this test has been passed successfully.

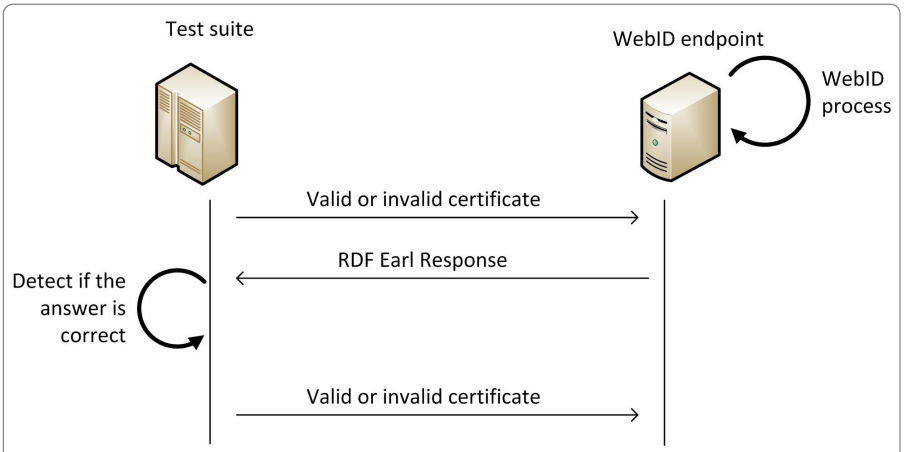

Figure 6: How the test suite works: by sending predefined valid or invalid certificate, the test suite can test specific aspects of the WebID process. The endpoint has to reply an EARL formated answer in order to tell what went wrong during the process.

A Java test-suite is currently developed to help developers test their WebID endpoint by sending corrupted certificate or misconceived RDF. For that purpose, the endpoint must reply with an EARL (RDF) document showing what errors/successes it has found.

It's a very important part of the work of the community group which will be decisive.

2.3- Node.js

Node.js *[http://www.nodejs.org]* (Node) is a software system which aims to produce **highly scalable applications**. Programs are written in JavaScript[5]. Node is built ontop of Google's V8 engine *[http://code.google.com/p/v8/]* which is the open source JavaScript engine used for instance in Google Chrome; it currently offers the best performance.

To be highly scalable, Node uses **event based** programming.

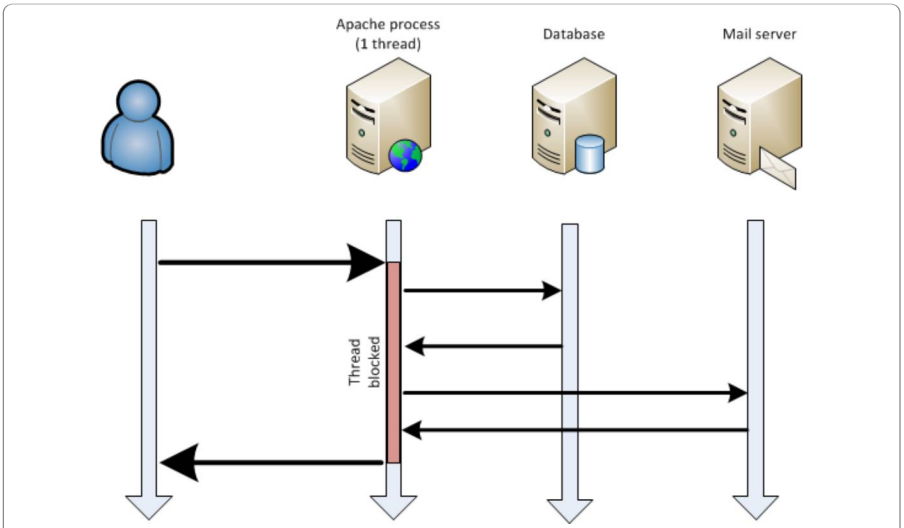

Figure 7: This image shows a single thread: when a client is connected, the thread is blocked (represented in red) while the final result is not ready. Thus, Apache thread is blocked waiting (doing nothing) during long I/O operations such as writing to disk or database.

Traditional servers such as Apache spawns a new thread (or use one of a pools) to handle each request. This thread will be used only for this request as long as the whole process is not finished. If other request arrive, they are put in the queue waiting for the thread to be available.

This approach does not provide good performance because of the memory cost of thread (or process) management. In order to scale well, this kind of server must use lots of threads. But large numbers of threads leads to overloading the machine as well as synchronization problems.

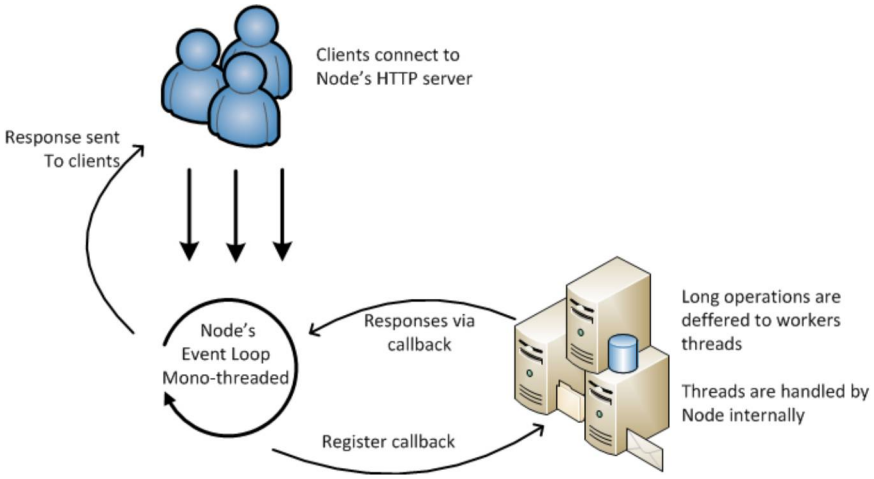

Figure 8: Node runs user's code in a loop called the event loop. Long operations are sent to workers. Inspired from [10] [#ref10]

Node works differently: it mainly uses an event loop that executes your code in a single thread. This loop handles all incoming connections, that's why potentially long operations (such as network or disk I/O, database queries, etc) must be executed asynchronously (with the use of callback) on worker threads.

In order to defer long operations to other threads, Node supports the brand new HTML5 Web Workers *[http://dev.w3.org/html5/workers/]* to delegate work to other threads.

Many features of JavaScript help developers to write effective event-driven programs (with a minimum of lines of code). Moreover, the way that Node is coded allows to do this painlessly: there is no thread management nor synchronization functions...

Node is under heavy development: core APIs may change at any time. It could even be an advantage or a problem. An advantage because if bugs are found or new features requested, development teams can react quickly. But it is also a problem when, for example, the number of arguments of a function changes from a version to another: it may require additional work to follow Node's quick development.

npm *[http://npmjs.org]* is Node's package manager which allows to install packages (with the additional possibility to add your own c++ wrappers if needed), and even to publish your own package on a centralized platform. On January 7th 2012, there were more than six thousand packages available which were covering a wide range of functionalities such as template engines, web pages parsing, certificate management... Anyone can publish a package to npm registry with only a few lines.

2.4- Related technologies

We also studied different aspects of the distributed social networks that will finally not be implemented in our project. Nevertheless these related technologies could be added to the project in the future in order to develop or improve it.

Port of Node.js on Android OS

We studied about how to make the webID protocol as portable as possible. The solution we found was to put the server that contains the FOAF file on a smartphone. Indeed most of us always carry a phone around us with a permanent internet connexion: having all personal information in our phone would be the final stage of "user in control of his data".

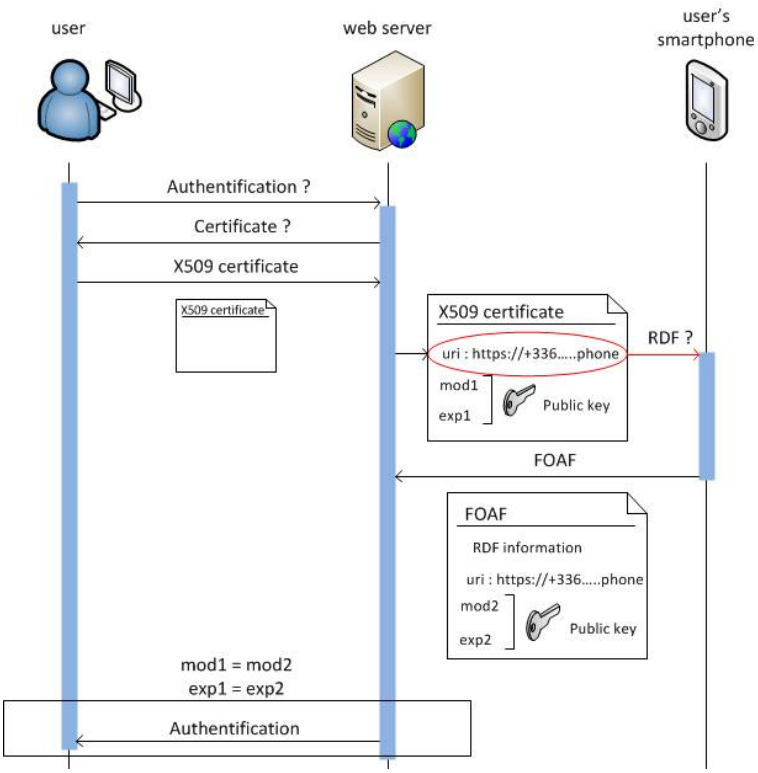

Figure 9: Proposition of a WebID architecture with a server running on a smart-phone.

Nevertheless for the moment Node.js is not compatible on Android OS, there is only an attempt in development on GitHub *[https://github.com/paddybyers/anode]* but it's not quite ready yet. Indeed it can only run on a modified version of Android.

Access control

As seen in the WebID authentication sequence, the access control can be treated during the sixth step. Because users may want to give different information depending on who is asking:

- "I only give my name to this unknown website."
- "I give my name and known contacts to social website."
- "I give all my information to this trusted website."

We thus tried to find an ontology in order to be able to control the access to the data on the FOAF file.

We found quite a few ontologies to do the access control:

- Web Access Control *[http://www.w3.org/wiki/WebAccessControl]* : a decentralized system for allowing different users and groups various forms of access to resources where users and groups are identified by HTTP URIs and uses WebID authentication.
- ACL Schema *[http://www.w3.org/2001/04/ACLS/Schema]* :
- Triple Access Control *[http://ns.bergnet.org/tac/0.1/triple-access-control#]* : an RDF ontology

3- Project

3.1- Organization / Tools

We have chosen to use the platform GitHub *[http://www.github.com]* to store our code because of three main reasons:

- The revision control system Git *[http://git-scm.com/]* is used more and more.
- Our aim is to encourage re-usability of our code: GitHub allows anybody to contribute by proposing addition or correction. Moreover it gives a good visibility on our work.
- GitHub is the official forge of Node.js and almost all npm packages are using GitHub. Being on the same platform allows us to quickly post bug reports and follow related projects.

For intern communication, we used a wiki (powered by dokuwiki *[http://www.dokuwiki.org]* , hosted on a personal server, available here *[http://wiki.magnetik.org/projet_asr]*) to share the minutes of ours meetings, important links about our research, etc.

To generate our modules's documentation, we used the JSDoc toolkit *[http://code.google.com /p/jsdoc-toolkit/]* . It uses in-line annotation (mostly the same as Javadoc *[http://www.oracle.com /technetwork/java/javase/documentation/index-jsp-135444.html]*) within the code (such as @param to describe a function parameter) to create a user-friendly documentation. An example of HTML output can be found on github *[http://magnetik.github.com/node-earl/doc/symbols/EarlDocument.html]* .

3.2- Research on related technologies

Before starting to code, we had to make research on many topics related to WebID that where totally unknown to us (RDF and its serialization RDF/XML and Turtle, TLS, X509 certificate, FOAF, ...).

After having a good understanding of these technologies and noticing that there wasn't much information in our native language (French), we created a few articles on Wikipedia to help the spreading of those technologies:

- WebID *[http://fr.wikipedia.org/wiki/WebID#Protocole_WebID]* ,
- Turtle (syntaxe) *[http://fr.wikipedia.org/wiki/Turtle_(syntaxe)]* ,
- Notation3 *[http://fr.wikipedia.org/wiki/Notation3]* ,
- Evaluation and Report Language *[http://fr.wikipedia.org/wiki/Evaluation_and_Report_Language]* .

By writing small summaries of these technologies and rephrasing already existing articles, we eventually got a clearer view.

3.3- Existent

We searched if other people had already done some work on WebID with Node. A Google search did not give any result, neither did a search on GitHub, but someone mentioned an implementation on the WebID XG list. After contacting the author, Antonio Garrote *[http://antoniogarrote.wordpress.com/]* , he put his code *[https://github.com/antoniogarrote/NodeJS-WebID-demo]* on his GitHub account *[https://github.com/antoniogarrote]* . We have identified one main problem: the lack of RDF library in JavaScript.

JavaScript RDF lib

There are a few RDF libraries (in pure JavaScript or wrappers to other libs) available on the Internet. The following matrix shows RDF support for the main libraries.

	RDFQuery ***[http://code.google.com /p/rdfquery/]***	**RDF parser** ***[http://www.jibbering.com /rdf-parser/]***	**rdfstore-js** ***[http://antoniogarrote.github.com /rdfstore-js/]***	**LinkedData RDFLib** ***[https://github.com /linkeddata /rdflib.js]***	***[ht /r***
Last update	2010	2006	2012	2012	
Parse (and export) RDF/XML	Yes	Yes	No	Yes	
Parse RDFa	Yes	Yes	No	Yes	
Parse (and export) N3 (Turtle)	Yes	Yes	Yes	Yes	
SparQL (1.0) support	Yes	No	Yes	Yes	
Parallel execution	No	No	**Yes**	No	
Requirement	jQuery	/	Node	/	

JavaScript RDF library support

Even if it looks like that there are many libraries, many can be eliminated. RDFQuery is a jQuery *[http://jquery.com/]* plugin, and for now jQuery plugins are not compatible with Node (even if some projects such as jsdom *[https://github.com/tmpvar/jsdom]* aims to implement the DOM, which is

required by jQuery, to node.). RDF parser does not support SparQL which is required to verify the public key.

We have seen in context that Node.js application have to do the least number of operations in the main thread. As a matter of fact the support of WebWorkers of **rdfstore-js** was decisive even if it did not support RDF/XML. Moreover this module is under development and RDF/XML could be implemented in a near future.

Because most profiles on internet are in RDF/XML, we used a web service[6] that translates RDF/XML to Turtle.

3.4- Programming

We started working with Node.Js 0.5.10 which didn't had the *Subject Alternative Name* available in the certificate. After a bug request *[https://github.com/joyent/node/pull/1286]* this has been added to Node for the 0.6.0 beta version.

In order to clearly separate our different modules, we created four different github repositories:

- node-webid *[https://github.com/magnetik/node-webid]* : contains the core WebID implementation.
- node-webid-demo *[https://github.com/magnetik/node-webid-demo]* : contains a demo of our implementation
- passport-webid *[https://github.com/magnetik/passport-webid]* : contains an implementation of WebID login with the passport framework *[http://passportjs.org/]* .
- node-earl *[https://github.com/magnetik/node-earl]* : JavaScript API used to generate EARL document.

node-webid

This is our core implementation. It tries to follow the

Our public API is simple, example of basic usage:

```
var certificate = req.connection.getPeerCertificate();
// Get the CLIENT certificate. In most language, getting this information is as simple
// as a method call.
if (!_.isEmpty(certificate)) {
  // The client can decline the demand of the certificate, so the vaiable certificate is empty
  // in this case, we can't continue the process.
  var verifAgent = new webid.VerificationAgent(certificate);
  // Initialize our WebID framework
  // It basically doesn't do anything
  // We can notice that the only input used to validate a WebID is the certificate
  verifAgent.verify(function (success, result) {
    // The verify method do the whole process
    // - Getting the file located at Subject Alternative Name
    // - Getting the public key in this file (exponent and modulus)
    // - Verifying if this public key equals the key of the certificate
    // Call the callback
    if (success) {
      // In case of success, the 'result' variable contain the complete RDF Profile
      var foaf = new webid.Foaf(result);
      var foafFile = foaf.parse();
      // Another function of our module is to parse the FOAF
      // then a few variable are available, such as:
      // - foafFile.name: contain the name
          // - foafFile.knows: contain known people
    } else {
      // There has been an error during the process
      // OR
      // The profile is not valid!
    }
  });
```

We can see that adding WebID login to a website is as easy as 10 lines of code: this is much easier than OpenID for example. Otherwise it requires the server to use TLS.

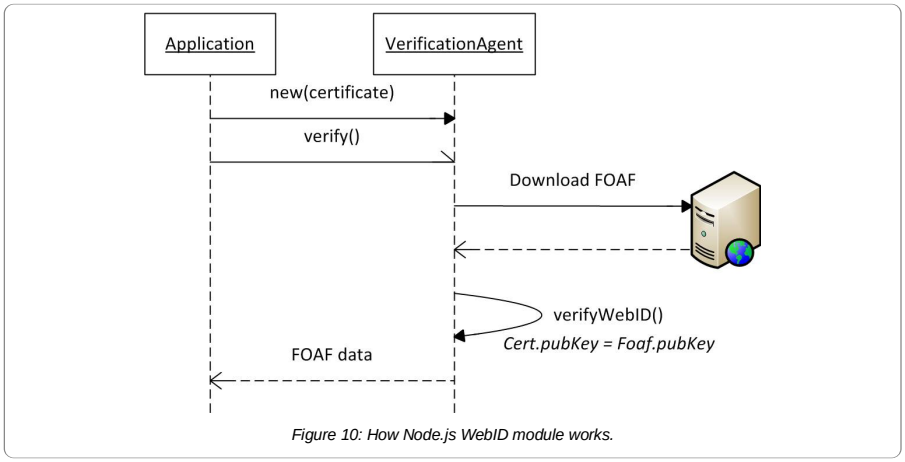

Figure 10: How Node.js WebID module works.

We have published this module on npm (search.npmjs.org/#/webid *[http://search.npmjs.org/#/webid]*): developers who want to use our code, just have to type: `npm install webid` , and the library (and its dependencies) will be installed on their working directory. They can also declare WebID as a dependency of their own package by declaring "webid" into the `package.json` file.

node-webid-demo

This is a simple demo of WebID login using our module.

This demo offers a simple way to test WebID:

Using Node 0.6.x, installation is as simple as:
`git clone git://github.com/magnetik/node-webid-demo.git` to download the code
then:
`node src/server.js` to start the server
Then, using your favorite browser, go to: https://localhost:8081 *[https://localhost:8081]* .

It was initially designed to be an EARL endpoint, but we didn't have enough time to implement all tests. Moreover we weren't able to use WebID java test suite correctly.

passport-webid

Almost all developers using Node.js for web usage use the express *[http://expressjs.com/]* framework, or at least the middleware Connect *[http://www.senchalabs.org/connect/]* . this framework/middleware helps developers program their application without worrying about the underlying considerations such as routes mapping, sessions etc.

Passport is an identification middleware which can be used with connect or express. It's designed to handle many "strategies", from classic login/password, to OpenID, facebook connect, and now... WebID !

Our module is now available as a passport strategy which means that anyone can use it with even less line of code than our "standard" example.

For instance, a local username/password strategy usage using Passport looks like:

```
passport.use(new LocalStrategy(
  function(username, password, done) {
    User.findOne({ username: username, password: password }, function (err, user) {
      done(err, user);
    });
  }
));
```

And the modified code which supports WebID:

```
passport.use(new WebIDStrategy(
  function(webid, done) {
    User.findOne({ webid: webid }, function (err, user) {
      done(err, user);
    });
  }
));
```

Of course, this code does not use the available FOAF file, but with minor adjustments it could be modified.

node-earl

In order to implement EARL answer in our *node-webid module*, we created a module to help generates EARL reports.

Via a simple API that we defined, developers can easily generate an EARL document.

For instance:

```
var earl = require('earl');
// Creating EARL document Object
var earlDocument = new earl.EarlDocument();
// Create an assertion:
// whose subject is the certificate
// and whose name is wit:certificateProvidedSAN
earlDocument.addAssertion(":assert1", ":certificate", "wit:certificateProvidedSAN");
// Tells that this assertion is passed with success
// We could has used .failed() too.
earlDocument.getAssertion(":assert1").passed();
```

Will produce:

```
<assert1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/ns/earl#Assertion> .
<assert1> <http://www.w3.org/ns/earl#subject> <certificate> .
<assert1> <http://www.w3.org/ns/earl#test> <http://purl.org/dc/terms/certificateProvidedSAN> .
<assert1> <http://www.w3.org/ns/earl#result> <assert1Result> .
<assert1Result> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/ns/earl#TestResult> .
<assert1Result> <http://www.w3.org/ns/earl#outcome> <http://purl.org/dc/terms/passed> .
```

Triple access control

We started studying the Triple Access Control ontology to implement a server that could serve a different FOAF file depending on who (or which server) is asking.

With the SparQL query in appendix 2, we can construct a new FOAF file by just changing the name "example.com" in the query: it will only contain the field that are annotated as "readable" by "example.com".

The problem was precisely to identify *WHO* is asking. Indeed during the process, the WebID verifier simply downloads the FOAF file to a distant web server. In order to use our query, the WebID verifier would have to identify itself to the server where the FOAF file is embedded. In a nutshell, for the WebID process, the client identifies himself through WebID to a server, and this server would have to

identify itself to another server (some kind of a "mutual identification").

This problem was out of the scope of our project. After a discussion with *Andrei Sambra*, we decided to stop our investigation at this point.

Git

We used Git submodule [11] *[#ref11]* capabilities. It allows to declare a directory as a checkout of another project. The main advantage was to separate our main module (node-webid) from our implementations (passport-webid and node-webid-demo).

However using git submodules requires to respect a workflow that is not quite obvious. We will give an example of the submodule usage in node-webid-demo (the "super project").

- To **add a submodule** into the */webid/* directory:
  `git submodule add git@github.com:magnetik/node-webid.git /webid/`
- To **update the submodule** when **modified by someone else**: `git submodule update --init`
- **Before** modifying a submodule **within the super project** move to branch master:
  `git branch master` and update `git pull` because git maintains the submodule as detached HEAD.
- To **commit** a change made in the submodule within the super project:
  - `git commit` and `git push` in the **submodule** first.
  - `git commit` and `git push` in the super project

4- Conclusion

Following the beginning of a future standard was exciting and a good opportunity to understand how a standard is written. Even more because this is the only part of the W3C process which is open to anyone.

Learning a new language and perhaps more important a new architecture (event-driven architecture) is interesting. Moreover JavaScript is more and more used on the web in one hand, with application which can be compared to desktop one (Gmail for instance). And on another hand outside the web as script tool in Gnome Shell or Microsoft Vista's gadget. JavaScript is the 10th most used language according to the TIOBE index *[http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html]* , and it's more and more used.

Node.js technology looks promising. Even if we didn't had any benchmark on our own application, it permits to to highly scalable application without having to learn new syntax or operators. Learning Node is easy and the community already very active. For instance we reported a few bugs that have all been fixed within a few days.

WebID is a promising standard but a few barriers are still strong for end-users. For instance the concept of certificate for users, how to handle certificate when not at home? There is no blocking issue, but if an internet major company implement WebID (Google for instance has already implemented OpenID), it would be a great support for the standard. Anyway the recommendation is not quite ready and solutions will be found.

4.1- Issues

We spent a lot of time in research about RDF, TLS and JavaScript because we started with a 0 level on these technologies. We may have coded more impressive demo using an already known language such as Java.

The main issue we had to face was the lack of good Node.js implementations for RDF. Although this technology is quite old, it's not as widespread as XML for instance.

With more than 400 messages per month, following the mailing list daily was really time consuming, that's why we did not participate much in the discussions.

5- References

[1] Henry Story et al., *WebID 1.0*, [Online]. Available: http://www.w3.org/2005/Incubator/webid/spec/

[2] ,*WebIDs and the WebID Protocol* [Online]. Available: http://www.w3.org/wiki/WebID

[3] T. Dierks, E. Rescorla,*The Transport Layer Security (TLS) Protocol, Version 1.2*, August 2008; http://www.w3.org/wiki/WebID

[4] S. Thomas, *Ssl & Tls Essentials: Securing the Web*. New York: Wiley Computer Publishing, 2000.

[5] R. Housley et al., *Internet X.509 Public Key Infrastructure*, IETF RFC 5280, May 2008; http://www.ietf.org/rfc/rfc5280.txt

[6] , [Online]. Available: http://vitobotta.com/why-isnt-ssl-on-by-default-for-all-websites/

[7] Dan Burnett, *W3C Recommendation Track* [Online]. Available: http://www.w3.org/2011/04 /webrtc/wiki/images/5/5c/Webrtc_w3c_rec_track.pdf

[8] Shadi Abou-Zahra, Shawn Lawton Henry et al, *EARL Overview*, [Online]. Available: http://www.w3.org/WAI/intro/earl

[9] Henry Story, *Document describing a vocabulary to allow a RelyingParty to make a report on an attempt at a WebID authentication.* [Online]. Available: http://www.w3.org/2005/Incubator/webid /earl/RelyingParty.n3

[10] Mixu, *Intro to Node.JS for .NET Developers*, [Online]. Available: http://www.aaronstannard.com /post/2011/12/14/Intro-to-NodeJS-for-NET-Developers.aspx

[11] S. Chacon, *Git book*, [Online]. Available: http://book.git-scm.com

6- About

You can download it here *[https://github.com/magnetik/node-webid-report/zipball/gh-pages]*

It's made with HTML5 + CSS3 and JavaScript to generate some content.

 *[http://validator.w3.org/check?uri=referer]* 

7- Appendix

7.1- User certificate selection

These are screen capture of certificate selection under the main mobile and desktop browsers

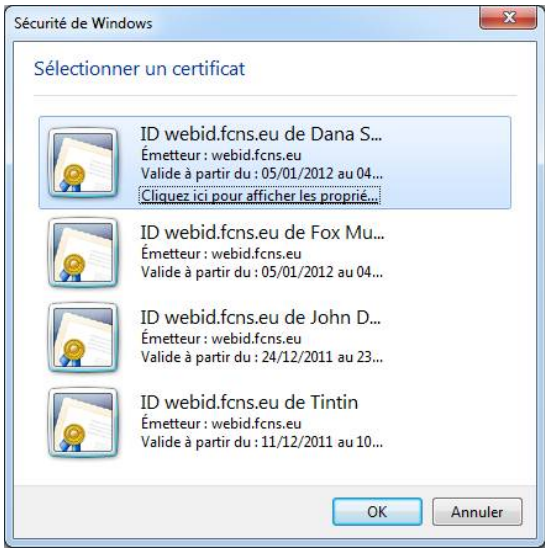

Figure 11: User certificate choice with Internet Explorer 9

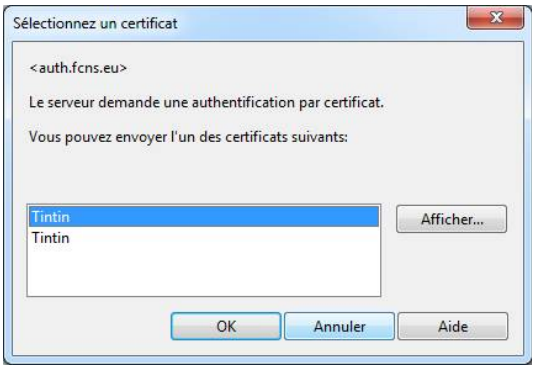

Figure 12: User certificate choice with Opera 11.60

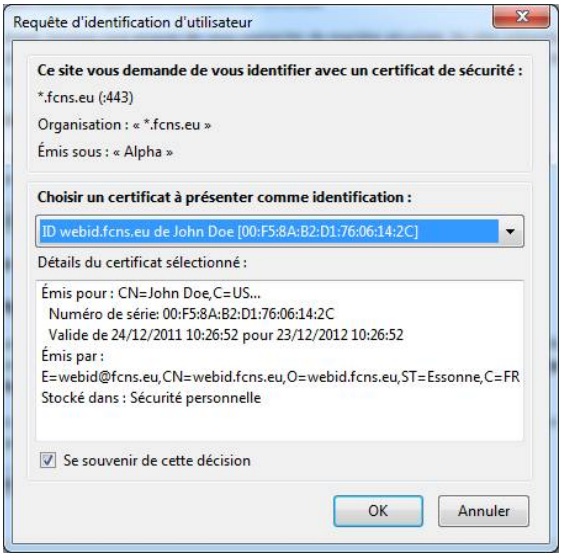

Figure 13: User certificate choice with Firefox 9

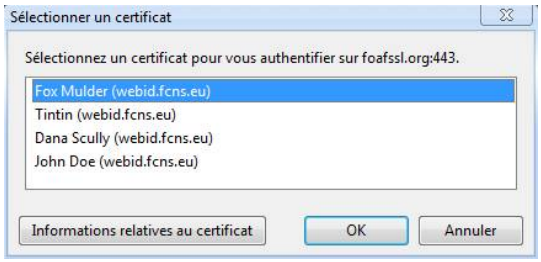

Figure 14: User certificate choice with Chrome 17

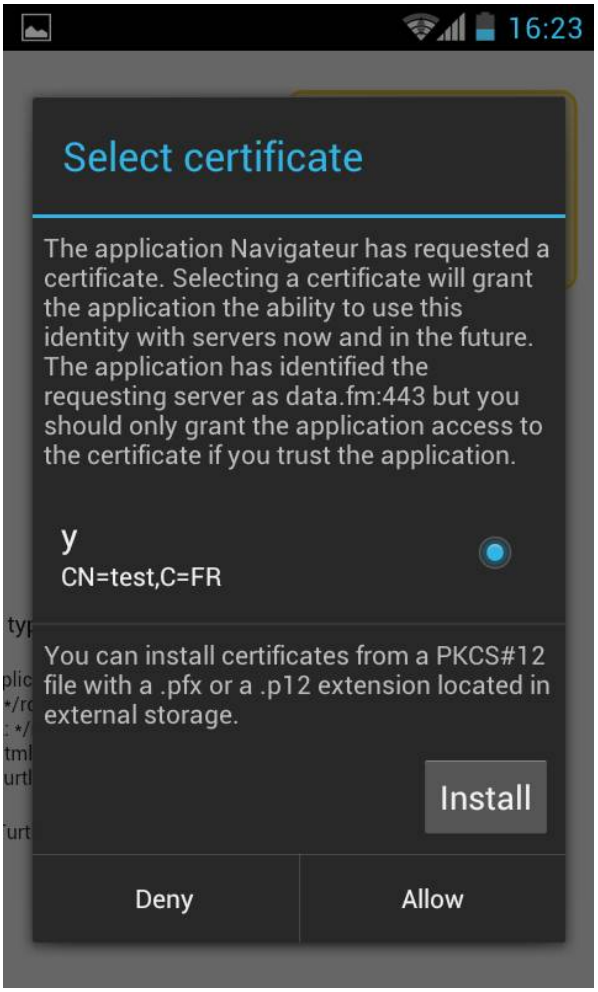

Figure 15: User certificate choice with Android Ice Cream Sandwhich (3.0). Tested on Android 2.3, which wasn't working.

7.2- Triple access control query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX acl: <http://www.w3.org/ns/auth/acl#>
PREFIX tac: <http://ns.bergnet.org/tac/0.1/triple-access-control#>
PREFIX src: <file:/mci/ei0912/garrone/git/tac/src/data/testXML.rdf#>

CONSTRUCT {src:me ?predicate ?data}
  WHERE
    {
          ?group foaf:name "example.com" .
          ?tacAccess acl:agent ?group .
          ?tacAccess tac:accessToTriple ?tripleAccessible .
          ?tripleAccessible tac:mode ?mode .
          ?tripleAccessible tac:filter ?tmpdata .
        { ?tmpdata tac:predicate ?predicate .
          src:me ?predicate ?data .}
        UNION
        { ?tmpdata tac:subject ?subject .
          OPTIONAL { ?tmpdata tac:predicate ?p .}
          FILTER ( !bound(?p) ) .
          ?subject ?predicate ?data . }
}
```

7.3- Example of TAC annotated RDF file

```
@prefix rdf: .
@prefix acl: .
```

```
@prefix tac: .
@prefix foaf: .
@prefix vcard: .
@prefix mywebid: .

# owner has full control
[] a acl:Authorization;
                tac:accessToTriple [ a tac:TripleAuthorization;
                tac:mode acl:Read;
                tac:mode acl:Write;
                tac:filter [ a tac:Filter;
                        tac:subject mywebid:me ]];
        acl:agent mywebid:me.


# friends have read access to
#  the fullname, nickname and all telephone numbers
[] a acl:Authorization;
        tac:accessToTriple [ a tac:TripleAuthorization;
                tac:mode acl:Read;
                tac:filter [ a tac:Filter;
                        tac:subject mywebid:me;
                        tac:predicate vcard:fn ]
        ], [
                tac:mode acl:Read;
                tac:filter [ a tac:Filter;
                        tac:subject mywebid:me;
                        tac:predicate vcard:nickname ]
        ], [
                tac:mode acl:Read;
                tac:filter [ a tac:Filter;
                        tac:subject mywebid:me;
                        tac:predicate vcard:tel ];
                tac:children [  a acl:Authorization;
                        tac:accessToTriple [ a tac:TripleAuthorization;
                                tac:filter [ a tac:Filter;
                                        tac:predicate rdf:type ]
                                ], [
                                tac:filter [ a tac:Filter;
                                        tac:predicate rdf:value ]]]];
        acl:agent _:friends.


# business contacts have read access
#  to the fullname and work telephone numbers
[] a acl:Authorization;
        tac:accessToTriple [ a tac:TripleAuthorization;
                tac:mode acl:Read;
                tac:filter [ a tac:Filter;
                        tac:subject mywebid:me;
                        tac:predicate vcard:fn ]
        ], [
                tac:mode acl:Read;
                tac:filter [ a tac:Filter;
                        tac:subject mywebid:me;
                        tac:predicate vcard:tel ];
                tac:children [  a acl:Authorization;
                        tac:accessToTriple [ a tac:TripleAuthorization;
                                tac:filter [ a tac:Filter;
                                        tac:predicate rdf:type;
                                        tac:object vcard:Work ];
                                tac:required "true"
                                ], [
                                tac:filter [
                                        tac:predicate rdf:type ]
                                ], [
                                tac:filter [
                                        rdf:predicate rdf:value ]]]];
        acl:agent _:businessContacts.


# data
mywebid:me a foaf:Person;
        vcard:fn "John Doe";
        vcard:nickname "Johnny";
        vcard:tel [
                a vcard:Voice;
                a vcard:Home;
                rdf:value "+49 8765 4321"
        ], [
                a vcard:Voice;
                a vcard:Work;
                rdf:value "+49 8765 5555" ].

# groups
_:friends a foaf:Group;
        foaf:name "list of friends".

_:businessContacts a foaf:Group;
        foaf:name "business contacts".
```