

<b>Author:</b>	Brian Gilmore, Keith Farvis, John Maddock
<b>Date:</b>	22 September 2004
<b>Programme:</b>	Common Services
<b>Committee Funding:</b>	JCIIE and JCSR
<b>Structure:</b>	3 pages
<b>Document Reference:</b>	CMSS:Gilmore

## Core Middleware and Shared Services Studies

Single Sign-On Report

---

(c) HEFCE 2004

The copyright for this publication is held by the Higher Education Funding Council for England (HEFCE). The material may be copied or reproduced provided that the source is acknowledged and the material, wholly or in part, is not used for commercial gain. Use of the material for commercial gain requires the prior written permission of the HEFCE.

# Contents

- 1. Project Description .....4
  - 1.1 Project Objectives..... 4
  - 1.2 Project Team ..... 4
  - 1.3 The Candidates ..... 4
  - 1.4 Report Layout ..... 5
  - 1.5 Models ..... 5
  - 1.6 Terminology ..... 6
- 2. Yale Central Authentication Service (CAS) .....7
  - 2.1 General description..... 7
  - 2.2 Implementation ..... 8
  - 2.3 Support ..... 9
  - 2.4 Sign-on/sign-out..... 10
  - 2.5 Reliability ..... 10
  - 2.6 Security ..... 11
  - 2.7 Summary ..... 12
- 3. Pubcookie ..... 14
  - 3.1 General description..... 14
  - 3.2 Implementation ..... 15
  - 3.3 Support ..... 16
  - 3.4 Sign-on/sign-out..... 17
  - 3.5 Reliability ..... 17
  - 3.6 Security ..... 18
  - 3.7 Summary ..... 19
- 4. Stanford WebAuth .....21
  - 4.1 General description..... 21
  - 4.2 Implementation ..... 22
  - 4.3 Support ..... 23
  - 4.4 Sign-on/sign-out..... 23
  - 4.5 Reliability ..... 24
  - 4.6 Security ..... 25
  - 4.7 Summary ..... 26
- 5. Cosign .....27
  - 5.1 General description..... 27
  - 5.2 Implementation ..... 28
  - 5.3 Support ..... 29

5.4 Sign-on/sign-out.....	29
5.5 Reliability .....	30
5.6 Security .....	30
5.7 Summary .....	31
6. KX.509 .....	32
6.1 General description.....	32
6.2 Implementation .....	33
6.3 Support .....	34
6.4 Sign-on/sign-out.....	34
6.5 Reliability .....	35
6.6 Security .....	35
6.7 Summary .....	36
7. A-Select .....	37
7.1 General description.....	37
7.2 Implementation .....	39
7.3 Support .....	39
7.4 Sign-on/sign-out.....	40
7.5 Reliability .....	41
7.6 Security .....	41
7.7 Summary .....	42
8. Other Factors to Consider .....	43
8.1 Simple and Protected GSS-API Negotiation Mechanism (SPNEGO).....	43
8.2 Comparison between cookie based solutions and X.509 certificates .....	43
9. Which is Best? .....	45
9.1 What are the key criteria?.....	45
9.2 A comparison .....	45
10. Summary .....	47

# 1. Project Description

## 1.1 Project Objectives

This is the report on Middleware Study 1 for the Evaluation of Single Sign-on Technologies in response to the JISC ITT for Shared Services and Middleware.

There is considerable interest world-wide in creating single sign-on systems for the web. These can be considered to be in two distinct classes. The first class is for intra-institutional use, and which could scale up to a size which could cover an entire country. The best known of these systems are Shibboleth from Internet2 in the USA and PAPI from Rediris, the academic network provider in Spain and, of course, Athens in the UK. The second class of system is one that is designed to work primarily in a single institution. The scope of this study is the second class.

The interest in these systems can be evidenced from the number of systems that have been implemented world wide. Some of these have only been available at a single site but a number of them have already been implemented by a significant number of sites.

To date there has been no attempt to compare and contrast these various systems. This report describes the University of Edinburgh's project to evaluate these systems.

## 1.2 Project Team

The technical work was carried out by John Maddock, the initial project write-up was by Keith Farvis, and the final editing was by Brian Gilmore, all of the University of Edinburgh Computing Services.

## 1.3 The Candidates

The candidates selected for evaluation were:

- Yale CAS. Originator: Yale University: <http://www.yale.edu/tp/auth/>.
- Washington Pubcookie. Originator: Pubcookie Team originally from the University of Washington: <http://pubcookie.org/>.
- Stanford WebAuth. Originator: Stanford University <http://webauthv3.stanford.edu/>.
- Michigan Cosign (cf. also Michigan KX.509). Originator: University of Michigan: <http://www.umich.edu/~umweb/software/cosign/>.
- X.509 certificates via Kerberos (KX.509). Originator: University of Michigan: <http://www.umich.edu/~x509/>.
- A-Select. Originator: Surfnet (Netherlands): <http://a-select.surfnet.nl>.

---

## 1.4 Report Layout

The report goes through an evaluation of each of the candidates and is followed by a conclusion section which summarises the results.

As a test of the ability of a single sign-on (SSO) candidate to provide authentication to web-based services, the project has investigated three applications:

- The ability of the candidate to provide the single sign-on front end to a common portal product, namely uPortal. Although this doesn't necessarily mean that integration with similar products will be as easy or difficult it does give a good example.
- The IMP email system from Washington State (<http://www.horde.org>), written in the PHP scripting language, is heavily used in Edinburgh for its email system (and hence the internals are understood). It is a good example of a system that presents a web interface but does not use a standard web server and consequently uses its own method for authentication thus presenting special problems in integrating it into a single sign-on environment. These systems are described as requiring 'back-end' authentication in the document.
- An example of a web based application for which a single sign-on solution could front-end the authentication, is the Outlook Web Access (OWA) mechanism for accessing Microsoft's email Exchange services over the Web. The report comments on the ease of integrating OWA into a single sign-on environment.

## 1.5 Models

There are in general two major ways which can be used to implement a single sign-on system. They are by the use of cookies or X.509 certificates.

### 1.5.1 Cookies

In the case of the use of cookies, when a user is authenticated by the single sign-on system, the authenticating agent plants a cookie, which is essentially a small file of information, into the user's browser. When the user attempts to access a resource, the end system can request the cookie. If it is present and is checked as 'correct' then the user can be allowed access. If it is not present or there is something incorrect in the cookie, for example if it is beyond the validity period that the SSO operates, then the user would be asked to re-authenticate.

Standard cookies are inherently insecure as they are not digitally signed but they are protected in as much as they can only be presented to a web application from the same name domain area as the originator of the cookie. Thus a major consideration of any of the SSO schemes is the mechanisms used to protect the integrity of the cookies.

### 1.5.2 X.509 certificates

An X.509 certificate originates from the CCITT X.500 series of standards and consists of a number of pieces of information, such as a 'name' and an 'organisation' which has been digitally signed by a

certificate authority. The signature enables a third party to verify that the certificate is a truly current certificate issued by the particular authority.

## 1.6 Terminology

### 1.6.1 Java

Java code requires an environment in which to run. This environment is commonly known as a 'container' for holding what is known as Java Servlet Technology, hence the description and version number as Servlet 2.3 container. A common container used in the community is Tomcat (<http://jakarta.apache.org>).

Tomcat binary releases are available for Unix and Windows.

### 1.6.2 PAM modules

The Pluggable Authentication Module (PAM) is a framework which provides on Unix systems the facility to include authentication methodologies via runtime modules which can be plugged-in to applications which require authentication (e.g. login, Apache ...) without needing to modify the existing applications. An application needs to be PAM enabled to take advantage of this but this only applies to some Unix applications.

### 1.6.3 Web server integration

There are two major web servers in use in the community, they are Apache running on Unix and Windows and Microsoft's Internet Information Services (IIS) on Windows.

With Apache, the add-on code that forces authentication is described as a 'module' and sometimes as a 'filter' whereas the similar item in IIS is described as a 'filter'. The Apache module is installed via a configuration file whereas the IIS filter is installed using a setup.exe installation script.

### 1.6.4 A 'client' in this context

In this report a 'client' is a system that wishes to use the single sign-on candidate for handling its authentication. The human user of the system is described as the 'end user'.

---

## 2. Yale Central Authentication Service (CAS)

### 2.1 General description

#### 2.1.1 Authentication model

A cookie-based model. Typically an application receiving a request from a user would redirect it to the CAS login URL appending a query string identifying the 'service' - a URL to where the request is redirected on successful authentication. The user is then prompted for a username and password which can then be authenticated by a number of means, see 2.1.5 . Successful authentication results in the generation of a Ticket Granting Cookie (TGC) - a long random number string which maps this TGC to the user. It is used to index into the in-memory credentials cache in the CAS server. If the browser supports cookies then the TGC is sent to the browser as a non-persistent cookie identifying a successfully logged in user thus (if the user accepts the cookie) obviating the need for the authentication step for subsequent requests. CAS then redirects the browser to the service URL appending a Service Ticket (ST) (another random character string which maps to the user and requested service). The application can now extract the ticket and validate using the CAS validation URL passing the ticket and the service as parameters. Successful validation results in the service ticket being destroyed and the username returned to the application. A user is able to obtain a TGC directly by calling the CAS login URL and the TGC can be destroyed by calling the CAS logout URL, otherwise it expires after eight hours.

CAS 2.0 introduced support for proxiable credentials. A proxy is a service which wishes to impersonate a user to another service or target which is willing to accept proxied credentials from one or more known proxies. This is achieved by the proxy requesting a proxy-granting ticket (PGT) from CAS which enables it to produce a Proxy Ticket (PT) which allows access to a target by impersonating a single user. A target may also become a proxy allowing a chain of proxies to be traversed. A proxy ability is particularly important for services such as portals which wish to interrogate further services on a user's behalf without causing the user further authentication checks.

#### References:

<http://www.yale.edu/tp/auth/cas10.html>

<http://www.yale.edu/tp/auth/cas20.html>

#### 2.1.2 Server requirements

The CAS 2.0 server is written entirely in Java and is deployable in any compliant Servlet 2.3 container (e.g. Tomcat 4.x; see 1.5.1 for explanation). It is noted that the Java performance with Tomcat on Windows is poor. Tomcat requires installation of a Java development kit (JDK) version 1.4 or later. As the test environment is using Kerberos then the current JDK version 1.4 makes life easier as it includes the Java Authentication and Authorisation Service (JAAS) package which has kerberos5 support. A Kerberos Java module was obtained from the University of Indiana which requires jakarta-log4j-1.2.8, a logging facility. Building the server requires the Java-based build tool 'ant'. The server was built on a Dell Precision 420 running Red Hat 9 Linux.

### 2.1.3 Client requirements

No special software is required in a user's browser to use CAS for a client system.

The Java client, which handles the original user call and redirects it to the CAS server, requires a compliant Servlet 2.3 container (see 1.5.1). The Apache module, which essentially performs the same activity, requires Apache 2.0.40 or later and OpenSSL 0.9.6b or later.

### 2.1.4 Ease of use for the end user

The use of CAS is straightforward for the user. A request from a user who is unauthenticated is directed automatically to the CAS login page; after being authenticated, the user is presented with a cookie which can be rejected or accepted (possibly automatically if the browser is set up to accept cookies) and thereafter the 'user experience' is dependent on the application.

### 2.1.5 Supplied methods of authenticating a user

The supplied code contains just a dummy authenticator which will authenticate the user if the name is the same as the password. However it was trivial to build a authentication method to use Kerberos (code came from Indiana) but the level of effort to use alternate methods, such as LDAP, is unknown.

## 2.2 Implementation

### 2.2.1 Ease of implementation and support from service provider

The server implementation was reasonably easy. Somewhat surprising given it came with only a dummy authenticator. The module which was required to use Kerberos came from Indiana and not Yale even though it was only a few dozen lines of Java. We were unable to get the Server module implementation using the Apache module `mod_cas` to work, despite putting a lot of work into this and discovering that others on the mailing list were having the same problems. There was no problem with the build, Apache configuration, etc. but any attempted use would result in failure. It should be pointed out though that the recommended server module is the Java client this being mandatory when integrating with uPortal.

Ongoing support would have to be local - the mailing list was disappointing for the filter for Apache (`mod_cas`) and as far as could be seen replies to questions were more likely to come from outwith Yale than from the list maintainer or colleagues. Indeed mailing the advertised CAS maintainer directly failed to elicit a reply.

### 2.2.2 How easy does it seem to be to develop a client (is there example code or a library/API) or are there standard clients or modules available?

The distribution contains various libraries to enable an application to develop a CAS client. They are variously written in Java, Perl, pISQL, Python and C as an Apache module and an Apache PAM module. There is also a contributed ASP script for IIS which uses XML to transfer the ticket and seems to work. Edinburgh University has developed a java class (running in Tomcat) to enable ColdFusion V5 to work with CAS.

### 2.2.3 How easily would it integrate with uPortal?

There is Yale provided example code for getting CAS 2.0 running on a uPortal version 2 installation. uPortal uses the proxy functionality available in CAS 2.0. The Java client is also required (see 2.1.3 for explanation). This has been done by Edinburgh University and the University of Bristol (and others). An implementation is now running in a uPortal service to Alumni with an Apache 1.3.27 front-end and the system runs well.

See: [http://www.bris.ac.uk/is/projects/portal/team/gary/The\\_Build\\_Process.html](http://www.bris.ac.uk/is/projects/portal/team/gary/The_Build_Process.html)

### 2.2.4 How easily would it integrate with systems which require backend authentication (e.g. PHP/IMP, ColdFusion, Oracle, Exchange etc)?

If the required service is running on an Apache (or IIS server) then using a module like mod\_cas would be the standard and optimal way of transparently handling the authentication with the CAS server. Without this module, the backend services would need to implement all this functionality themselves.

Either way, taking the PHP/IMP service as an example, then the login functionality in the package would need to be removed and the CAS service ticket appended to the original URL would have to be validated with the CAS server by the application. On successful validation CAS returns the username. Interestingly in the Indiana Kerberos module the authenticate class had been changed from a Boolean to a String in order to return the username (obviously requiring an equivalent change to CAS) presumably to facilitate a similar strategy. It would be necessary to add a CAS authentication method to the PHP and IMAP c-client library. A PAM module (see 1.5.2) could be used with the IMAP daemon.

Integration with Exchange using Outlook Web Access would be very difficult as Exchange is tightly integrated with Active Directory. It hands off authentication to Active Directory (AD) and there is no obvious way of integrating CAS with AD.

## 2.3 Support

### 2.3.1 Description of the code and the quality of the documentation

The product is written in Java and typically employs Tomcat as the Servlet container.

The documentation with the product is poor and consists of only a README file for server installation and a HISTORY file.

### 2.3.2 How mature is the product?

From the history, CAS 1.0 was deployed as a '1-tier, centralised single sign-on' system in 2001 and CAS 2.0 (n-tier, proxiable single sign-on, plus extra features) in May 2002 with various bugfix releases culminating in the current version, 2.0.11, released in July 2003.

### 2.3.3 How many sites is it used at?

Yale, Indiana, Princeton, Bristol and more.

### **2.3.4 The route to ongoing development and support for the product**

Version 2.1 of the server is being developed at Yale but otherwise it looks as though you do it yourself, which would require intimate knowledge of the code and environment. There is an active mailing list at Yale. The maintainer of the Apache module (mod\_cas) has now left Yale which may explain the lack of response to the mod\_cas problems. The maintainer suggests possible causes for the mod\_cas problems and concedes that the likely scenario could be handled better. This lack of support for mod\_cas is a major concern.

There is no commercial support available.

## **2.4 Sign-on/sign-out**

### **2.4.1 How does the user initially authenticate?**

The user either connects directly to the CAS server login page or is redirected there by a web application and if no cookie is present indicating that authentication has already been successfully achieved, then the user is prompted for their username and password. On successful authentication the user is either informed of this if they called the CAS server directly or is redirected to the page nominated by the application and is offered a cookie.

### **2.4.2 Can the service use the credentials that the user has already acquired by logging in to Active Directory or Kerberos, or do they always have to authenticate through their web client?**

Always through the web client. CAS cookies are random strings which map onto the user or whatever in the credentials cache.

Presumably the web client could be programmed to look for some other pre-existing form of credential and use that. This would require some Java programming.

### **2.4.3 How are credentials cleared down on completion of session?**

The credentials are stored in non-persistent (session) cookies which are deleted by the browser when the browser session is closed.

### **2.4.4 Will server restarts cause the authentication database to be flushed?**

As the credential cache is in the CAS server memory and not written to disk in a retrievable format and because the browser cookies are simply an index into this database, restarting the server will mean that all cookies in use become invalid and so re-authentication of all user sessions will be necessary.

## **2.5 Reliability**

### **2.5.1 How vulnerable is the system to failure of a single component?**

It is vulnerable to network and server failures. Re-authentication or restarting of all sessions would be required because the CAS server holds a credential cache in the server memory.

### **2.5.2 How easy is it to run multiple instances of that component to reduce the risk of failure (i.e. can a resilient infrastructure be built)?**

The question below was sent to the mailing list on 24th December 2003:

CAS servers cluster.

“Is it possible? Anyone current running on such configuration?”

From a previous post it appears to be running in an active-passive configuration using a load balancer. But it would be preferable to have a multi-master setup so that we can have small servers to cope with scalability problem.

No answer was forthcoming, so one presumes it can't be done.

The possibility of a hot standby should be investigated but a server failure would still require all sessions to re-authenticate. There is a significant problem in that both the server and the standby server would need to be sized to cope with the big peak load of everyone re-authenticating at the same time just after the restart of the service.

## 2.6 Security

### **2.6.1 How secure does it seem to be and/or what are its vulnerabilities?**

Bristol have made two changes to the way CAS operates at their site:

- only allowing "authorised" services to request authentication from CAS – so that only named services that appear on the authorised list are permitted to use CAS
- de-coupling the one-to-one relationship of serviceIDs to callbackURIs - so rogue services cannot masquerade as legitimate services by using an authorised service ID and a rogue callbackURI.

“It should be noted that where authorized services act as proxies to target services, the target service validation callbackURI is accessed directly from the proxy and not from CAS. It is not therefore possible for CAS to prevent authorized services requesting a proxy ticket for a valid service and then presenting it to a rogue service's callbackURI. On the assumption that authorized proxies are going to be unwilling for rogue services to dupe them into passing them a valid CAS proxyTicket, CAS can offer a check on the callbackURI for the target service. This check is realised by the provision of both "targetService" and "targetCallbackURI" parameters to the ProxyTicket generating servlet. A filter in front of this servlet performs the checks and interposes a failure message if the targetService is not authorized to use CAS or the targetCallbackURI does not match a valid pattern for the service.”

See: <https://www.bris.ac.uk/is/cas/docs/modifications.jsp>

If end user authentication is being performed using Kerberos then the Kerberos Java module (KJM) from Indiana uses the Java Authentication and Authorization Service (JAAS ) Krb5LoginModule. The default configuration does not use a keytab file to authenticate a principal which is the normally recommended method. To fix this the JAAS configuration file, jaas.conf, should be changed to include 'useKeyTab=true'.

See: <http://java.sun.com/j2se/1.4.1/docs/guide/security/jgss/single-signon.html>

### **2.6.2 What form of encryption is used and where is it used?**

Communications between the client and CAS server use SSL and communications between the CAS server and, in our case, the Kerberos KDC use an encrypted channel. In each case the encryption method used depends upon configuration options and possible negotiation between the two endpoints.

### **2.6.3 How dependent is the protocol on the security of the underlying transport? Will SSL be required for all services?**

SSL is required for transactions with the CAS server, i.e. the login, logout and validate functions. The target application may require a client connection to be over SSL.

### **2.6.4 What steps does it take to ensure that the transport provides sufficient protection?**

The default is not to use SSL on the connection to the CAS server. The README file does not have any mention of SSL, but CAS can be set up to force the use of SSL.

### **2.6.5 How are cross-service attacks prevented?**

See the changes made by Bristol in 2.6.1.

### **2.6.6 How vulnerable is it to man-in-the-middle and impersonation attacks?**

Unprotected application connections are naturally vulnerable. SSL and Kerberos are designed to protect themselves against such attacks but the connections are as vulnerable as the versions of SSL and Kerberos being used. It is essential that security advisories for these products are acted upon immediately.

Note: The apache module, mod\_cas, requires Apache 2 which is not in general use (here). As the current CAS/uPortal integration requires the Java client then this not an issue for uPortal, but would be for other service providers.

## **2.7 Summary**

### **2.7.1 Pros:**

- simple authentication model.

- University of Edinburgh have a working CAS implementation in their pilot uPortal service.
- will work without cookies (the user would be re-challenged by each new service).

**2.7.2 Cons:**

- model inherently subject to single point of failure.
- little or no support from mod\_cas developers, which we and others could not get to work: this has serious implications for services running on Apache.
- little or no documentation.
- requires Apache 2 (a time-limited con, once Apache 2 is in wider use).

## 3. Pubcookie

### 3.1 General description

#### 3.1.1 Authentication model

A cookie based model.

There is an addition of a setup phase between the login server and each application server to negotiate a shared symmetric key for cookie protection.

A user makes a request to a webserver for a resource (a URL) which is Pubcookie protected. The Apache module `mod_pubcookie` (or the ISS filter) intercepts the request and if the session is not already authenticated, and does not carry information from the login server (a "granting cookie"), `mod_pubcookie` generates a "pre-session" cookie scoped to the application and a "granting request" cookie scoped to the login server, and redirects the browser to the login server. The user is then prompted to log in, the authentication method being one of those listed in 3.1.5. Successful authentication results in the generation of a "granting cookie", containing the authenticated username, and a "login cookie" for use in subsequent requests. The browser is now redirected to the original resource, the request now including the granting and pre-session cookies. `mod_pubcookie` again intercepts the request, this time finding the granting cookie which it verifies with the pre-session cookie. If verification succeeds then it passes the username to the application along with the original request, and generates a session cookie for subsequent requests by the user to the application.

See: <http://www.pubcookie.org/docs/how-pubcookie-works.html>.

#### 3.1.2 Server requirements

Pubcookie consists of a login server, an Apache module and an IIS filter.

##### **login server requirements:**

A Unix platform and a web server with CGI support.

##### **application server requirements:**

For the Apache module (`mod_pubcookie`):

Apache version 1.3.x (not Apache 2 yet).

`mod_ssl` or Apache-SSL patches.

OpenSSL

For the IIS Filter:

Microsoft Internet Information Services (5.0 recommended).

Any Windows version that supports IIS.

Intel platform

The servers were built on a Dell Precision 420 running Red Hat 9 Linux.

### 3.1.3 Client requirements

- SSL support.
- HTTP/1.0 or higher.
- Allows HTTP-EQUIV in META tags.
- Cookies supported and enabled.

### 3.1.4 Ease of use for the end user

This is unknown as there were unresolved problems with implementing the Apache module (mod\_pubcookie). Although it could not be made to work in Edinburgh it must be working elsewhere as it is an integral part of Pubcookie.

Cookies need to be enabled, as it will not work without them.

### 3.1.5 Authentication Methods

Supports LDAP, shadow password file and Kerberos for authentication of the user.

## 3.2 Implementation

### 3.2.1 Ease of implementation and support from service provider

Implementation of the login server was reasonably easy but, as mentioned, the test application server still has unresolved problems. The mailing list was contacted on 13/11/03 with a reminder sent on 18/12/03 and a reply was received on 30/12/03 requesting the version of SSL in use. It seems that there have been similar faults caused by Red Hat's OpenSSL library.

There was not time available to be able to track this particular fault any further.

### 3.2.2 How easy does it seem to be to develop a client (is there example code or a library/API) or are there standard clients or modules available?

There is no example code but the source code of mod\_pubcookie could be used as a starting point.

There is a PAM client written for Solaris 2.6 (and later) available from the University of Manitoba.

at: <http://mail.cc.umanitoba.ca/source>

### 3.2.3 How easily would it integrate with uPortal?

Apart from an internet2 conference call suggesting that the University of Hawaii had volunteered to do just this in October 2001, no other implementations were found.

.This project has now been abandoned see: <http://www.hawaii.edu/spit/its2002/>

Proxying capability would need to be added to Pubcookie. Nathan Dors at Washington has written: "There's an undocumented way to proxy Kerberos service tickets via Pubcookie."

There are messages on the Pubcookie email lists about passing the user name (an environment variable called REMOTE\_USER) from Apache using an Apache to Tomcat connector.

### **3.2.4 How easily would it integrate with systems which require backend authentication (e.g. PHP/IMP, ColdFusion, Oracle, Exchange etc)?**

Login functionality would be removed from the application and the authenticated username would be passed on together with a session cookie that is required to be included in the application's response.

It would be necessary to add a Pubcookie authentication method to the PHP and IMP client library.

Integration with Microsoft Exchange using Outlook Web Access would be very difficult.

## **3.3 Support**

### **3.3.1 Description of the code and the quality of the documentation**

All written in C.

There is a small amount of installation and overview documentation. Latterly a current JISC project, IAMSECT, has published a guide to installation of Pubcookie on Red Hat Linux (see <http://iamsect.ncl.ac.uk/inprogress.html>). In particular this covers use of Pubcookie in combination with Windows Active Directory.

### **3.3.2 How mature is the product?**

Originally developed in 1998 for internal use and packaged and released to others in 2001. The first beta of the current version (3.0b1) was in August 2002 and the version tested (3.0.0) was released on 5th June 2003. Since the tests described here were completed an updated version, 3.1.0, was released on 16 May 2004 followed by a further bug-fix release, 3.1.1, on 30 July 2004. Time did not permit these new versions to be tested but the release notes indicate that some useful enhancements have been added.

### **3.3.3 How many sites is it used at?**

Washington, CMU, Wisconsin, UFL, Newcastle and many more.

### **3.3.4 The route to ongoing development and support for the product**

It is now community supported software with active mailing lists for users and developers as well as a bugzilla database. Contribution to these and involvement in the on-going development effort would be sensible.

There is no commercial support available.

## 3.4 Sign-on/sign-out

### 3.4.1 How does the user initially authenticate?

On requesting a protected page, the user is redirected to the login server and prompted for their username and password, if they have not already authenticated. On successful authentication, or if previously authenticated, then the request is served by the application. Cookies are presented to the user and must be accepted for the system to work.

### 3.4.2 Can the service use the credentials that the user has already acquired by logging in to Active Directory or Kerberos, or do they always have to authenticate through their web client?

Always through the web client.

Presumably the web client could be programmed to look for some other pre-existing form of credential and use that. This would require some C programming. AND code on the client.

### 3.4.3 How are credentials cleared down on completion of session?

The credentials are stored in non-persistent (session) cookies which are deleted by the browser when the session is closed.

It is also possible to logout of an individual application which will clear the current session cookie.

### 3.4.4 Will server restarts cause the authentication database to be flushed?

As the "granting cookie" is signed using the private key of the login server, and encrypted using the symmetric key shared by the application server and the login server then a server restart effectively invalidates the authentication cookies.

## 3.5 Reliability

### 3.5.1 How vulnerable is the system to failure of a single component?

It is vulnerable to network and server failures.

### 3.5.2 How easy is it to run multiple instances of that component to reduce the risk of failure (i.e. can a resilient infrastructure be built)?

When testing was being carried out, the documentation contained only a placeholder entitled "Advanced Configuration / Redundant login servers" at <http://www.pubcookie.org/docs/install-login.html>, but no further information on how to configure them. Consequently it appeared that setting up a redundant configuration would be problematic at best.

Since the work described here was completed the following has been added to the documentation:

"Redundant login server configuration"

The login cgi and keyserver can be deployed in a redundant configuration, provided that they share the same cluster name and settings. Use the login\_servers variable to configure the keyserver to push new keys to its peers."

This has been stated as being possible (though not tested in the present work) because the server does not hold any state information regarding the credentials.

## 3.6 Security

### 3.6.1 How secure does it seem to be and/or what are its vulnerabilities?

The following section is a quote from: <http://www.contrib.andrew.cmu.edu/~leg/work/pubcookie-futures.html>

#### **Key management.**

The current key management scheme derives individual application keys from a single master secret; this makes it impossible to revoke keys and creates a cryptographic dependence on what should be unrelated keys.

#### **Use of a widely scoped cookie.**

The design document refers to this cookie as the "granting cookie" and it is used to transfer an authentication assertion between the login server and the target application server. While this cookie is live it can easily be read by any web server in the domain.

#### **Single DES encryption keys.**

Pubcookie using 56-bit keys to secure its authentication assertions, which is no longer adequate for new authentication systems. Further it uses a variety of DES modes which are not explained in the design document.

#### **Inadequate code auditing.**

The current code is hard to read; at CMU several people have noted functions that don't appear to do what their name suggests leading to possible vulnerabilities. (Or when we misread the code and implement some extension, we may also create a vulnerability where one didn't appear before.)'

Attempts have been made to follow this up but without success. Newcastle University have reported that they believe that this has been fixed in release 3.

It seemed to be necessary to make the Kerberos keytab file readable by apache. This had previously been reported to the mailing list by Kevin Bowen at UCSD on 19/02/2003. No response was received. This is deemed to be poor security; for reference see the Kerberos Administration Guide.

### **3.6.2 What form of encryption is used and where is it used?**

Communications between the browser, the application server and the login server use SSL. Communications between the login server and, in the test case, the Kerberos KDC use an encrypted channel. In each case the encryption method used depends upon configuration options and possible negotiation between the two endpoints. See 3.6.1 for possible vulnerabilities.

### **3.6.3 How dependent is the protocol on the security of the underlying transport? Will SSL be required for all services?**

SSL is required for all connections.

### **3.6.4 What steps does it take to ensure that the transport provides sufficient protection?**

Pubcookie must be set up to use SSL between browser, the application server and the login server.

### **3.6.5 How are cross-service attacks prevented?**

Application specific session cookies are included in each application response to the login server enabling detection of rogue requests.

### **3.6.6 How vulnerable is it to man-in-the-middle and impersonation attacks?**

Unprotected application connections are naturally vulnerable. SSL and Kerberos are designed to protect themselves against such attacks but the connections are as vulnerable as the versions of SSL and Kerberos being used. It is essential that security advisories for these products are acted upon immediately.

Pubcookie also guards against these kind of attacks by using a shared symmetric key to encrypt certain messages sent between each application server and the login server.

Note: CMU seem to have a project to integrate Pubcookie with Shibboleth - see: <http://www.contrib.andrew.cmu.edu/~leg/work/library-kiosk.html>.

## **3.7 Summary**

### **3.7.1 Pros:**

- mature product.
- redundant configuration may be possible (it was not possible to test this).
- single point of failure.
- CMU have a project to integrate with Shibboleth.

### **3.7.2 Cons:**

- no work done with uPortal, which is surprising given the maturity of this product.
- dependent on acceptance of cookies to work at all.
- could not get the Apache module: mod\_pubcookie to work.
- there may be code and encryption weaknesses (awaiting a reply to this question).

---

## 4. Stanford WebAuth

This product will be called Stanford WebAuth, because the name WebAuth has been used for a number of similar products from other universities, but this is the most relevant one for this evaluation.

### 4.1 General description

#### 4.1.1 Authentication model

A cookie based model. A user makes a request to an application server (WAS - see 4.1.2) for a resource (a URL) and if the user hasn't been identified yet (as determined by possessing a WAS cookie), the request is redirected to a WebAuth Key distribution Centre (WebKDC – see 4.1.2) which will present a login page to the browser.

On successful authentication the WebKDC will then generate two "tokens". One is placed in a cookie scoped for the WebKDC and used to provide single sign-on in future requests, and one gets sent back to the WAS, which will verify it upon receipt. If the user has already been identified then access to the resource will be granted.

See: [http://webauthv3.stanford.edu/webauthv3\\_spec.html](http://webauthv3.stanford.edu/webauthv3_spec.html)

#### 4.1.2 Server requirements

WebAuth consists of a WebAuth-enabled Application Server (WAS) and a WebAuth Key distribution Centre (WebKDC). Both are written in C. There is also a Perl module that interfaces to the library APIs.

Both require:

- Apache 2.0.43 or later (2.0.46 or later recommended).
- OpenSSL 0.9.7 or later.
- MIT Kerberos v5 1.2.x or later (1.2.8 or later recommended).
- cURL (tested with 7.10.2).

WebKDC additional requirements:

- Perl modules from CPAN (Comprehensive Perl Archive Network – <http://www.cpan.org>).
- HTML::Template.
- CGI.
- CGI.
- FCGLI.
- Crypt::SSLeay.

Optionally the Apache module mod\_fastcgi from [www.fastcgi.com](http://www.fastcgi.com)

#### 4.1.3 Client requirements

SSL support

HTTP/1.0 or higher

Cookies supported and enabled

#### **4.1.4 Ease of use for the end user**

Straightforward. An unauthenticated request is directed to the WebKDC login page and thereafter the 'user experience' is dependent on the application. However cookies need to be enabled, as it will not work without them.

#### **4.1.5 Authentication Methods**

The WebAuth code as supplied by Stanford only supports Kerberos authentication.

## **4.2 Implementation**

### **4.2.1 Ease of implementation and support from service provider**

Implementation was relatively straight-forward. Excellent response to queries on the mailing list although as ever first-line support would have to be local. There is also a test suite included.

### **4.2.2 How easy does it seem to be to develop a client (is there example code or a library/API) or are there standard clients or modules available?**

There is no example code. The source code of the Apache module: mod\_webauth could be used as a starting point, although much more would be required in order to update the Kerberos credentials and pass them to the application. Stanford consider that it should be fairly easy to add PAM/SASL support. It hasn't been done at Stanford because they only use Kerberos 5.

### **4.2.3 How easily would it integrate with uPortal?**

It is being worked on by Ray Miller and his team at Oxford, see:

<http://users.ox.ac.uk/~raym/talks/webauth.pdf>

"We should be able to use Apache 2.0 with mod\_webauth and mod\_jk to proxy requests to the Tomcat instance running uPortal. Will require some changes to uPortal to extract information about the authenticated user from environment variables. This might not be flexible enough for uPortal channels that require secondary tickets. Implement the WebAuth protocol in pure Java for uPortal?"

### **4.2.4 How easily would it integrate with systems which require backend authentication (e.g. PHP/IMP, ColdFusion, Oracle, Exchange etc)?**

As Stanford WebAuth supports Kerberos then there is an option. If the backend system does not support Kerberos then login functionality would be removed from the application and the authenticated username would be passed in a cookie

If the backend system supports Kerberos, then the Kerberos credentials contained within the cookie passed to the web server can be used to authenticate to it. For example, for IMP/IMAP services, PHP with the c-client library can already use Kerberos credentials to authenticate to a backend IMAP server. From Roland Schemers at Stanford:

“It is planned, but not yet done, to be able to use the TGT from the cookie to authenticate to a backend system like a kerberised IMAP.”

## 4.3 Support

### 4.3.1 Description of the code and the quality of the documentation

All written in C.

See 4.1.1. The documentation is excellent and comprehensive. It can be accessed from:

<http://webauthv3.stanford.edu/>

### 4.3.2 How mature is the product?

WebAuth V3 is a complete rewrite of the original Stanford WebAuth system and the initial public release (3.0.0) was on 18/02/03. The current version (3.2.2) was released on 02/03/2004.

See: <http://webauthv3.stanford.edu/project.html>

### 4.3.3 How many sites is it used at?

Stanford, Oxford.

### 4.3.4 The route to ongoing development and support for the product

Support would be on one's own with backup via the mailing list from Stanford.

There is no commercial support available.

## 4.4 Sign-on/sign-out

### 4.4.1 How does the user initially authenticate?

The user requests a protected page and, if not already authenticated, is redirected to the WebKDC and prompted for their username and password. On successful authentication, or if previously authenticated,

then the request is served by the application. Cookies are presented to the user and must be accepted for the system to work.

See "WebAuth Scenarios" at: [http://webauthv3.stanford.edu/webauthv3\\_spec.html#scenarios](http://webauthv3.stanford.edu/webauthv3_spec.html#scenarios).

#### **4.4.2 Can the service use the credentials that the user has already acquired by logging in to Active Directory or Kerberos, or do they always have to authenticate through their web client?**

Always through the web client. WebKDC looks for a cookie that contains cached credentials. There does appear to be the capability in a desktop Unix client to run 'sidentd' to provide this capability. However this is insecure as one would be trusting a potentially insecure remote system to state who the person is.

From Roland Schemers at Stanford:

"There is preliminary support in the WebKDC to allow you to take an existing TGT and turn it into a proxy-token. Support still needs to be added to the weblogin interface, basically a new URL that takes the encrypted TGT as a query parameter, calls the WebKDC function, and returns a new cookie."

#### **4.4.3 How are credentials cleared down on completion of session?**

The recommended method is to close the browser but it is possible to logout of each application visited and then logout from the weblogin page to destroy the cookie.

#### **4.4.4 Will server restarts cause the authentication database to be flushed?**

Restarting the server has no affect on the browser cookies so re-authentication would not be required.

## 4.5 Reliability

#### **4.5.1 How vulnerable is the system to failure of a single component?**

It is vulnerable to network failure but has resilient server capability. Because the cookies are still in the browser and not tied into the server, existing sessions will not need to re-authenticate.

#### **4.5.2 How easy is it to run multiple instances of that component to reduce the risk of failure (i.e. can a resilient infrastructure be built)?**

This information courtesy of Roland Schemers at Stanford:

"Stanford run three WebKDCs sitting behind a load balancer (lbnamed, available from Stanford) so they are all live. The WebKDC was designed to be as stateless as possible (only the keyring which changes once a month needs to be propagated across all three), so a user can get bounced to any one of them at any point and not know the difference.

---

It is more for availability than load sharing, as I don't think the load on any of them ever gets real high. WebAuth is used by lots of critical applications at Stanford, so we wanted to make sure that the login/webkdc was highly available.

Having three also makes it easier to take one or two out of the pool for maintenance and not have to bring down the whole service."

## 4.6 Security

### 4.6.1 How secure does it seem to be and/or what are its vulnerabilities?

A reading of the spec doesn't point to any obvious security problems.

No reported vulnerabilities have been found.

### 4.6.2 What form of encryption is used and where is it used?

Communications between the browser, the WAS and WebKDC use SSL. Communications between WebKDC and, in our case, the Kerberos KDC use an encrypted channel. In each case the encryption method used depends upon configuration options and possible negotiation between the two endpoints.

See: [http://webauthv3.stanford.edu/webauthv3\\_spec.html](http://webauthv3.stanford.edu/webauthv3_spec.html).

### 4.6.3 How dependent is the protocol on the security of the underlying transport? Will SSL be required for all services?

All interactions between the browser and WAS *should* be SSL-protected.

All interactions between the browser and WebKDC *must* be protected via SSL.

### 4.6.4 What steps does it take to ensure that the transport provides sufficient protection?

WebAuth must be set up to use SSL between the browser and WebKDC.

### 4.6.5 How are cross-service attacks prevented?

Tokens are used to authenticate requests and responses between servers, as well as protect data that is stored in URLs and cookie data. Shared symmetric keys (session-keys) are used to encrypt tokens between servers, and private keys are used to encrypt tokens meant for a only a single server to decrypt.

See "Security Model and Key Management" at: [http://webauthv3.stanford.edu/webauthv3\\_spec.html](http://webauthv3.stanford.edu/webauthv3_spec.html).

### 4.6.6 How vulnerable is it to man-in-the-middle and impersonation attacks?

See 4.6.5.

Note: The apache module, mod\_webauth, requires Apache 2 which is not in general use (here).

## 4.7 Summary

### 4.7.1 Pros:

- well documented.
- responsive developers.
- Oxford are integrating it with uPortal.
- Kerberos ticket encapsulated in cookie - good for integration with backend services.
- Multiple WebKDCs provide for resilience.
- re-authentication not required if server fails.

### 4.7.2 Cons:

- not widely used.
- complicated packet formats.
- requires Apache 2 (a time-limited con, once Apache 2 is in wider use).
- dependent on acceptance of cookies to work at all.

## 5. Cosign

### 5.1 General description

#### 5.1.1 Authentication model

A cookie-based model.

There are two scenarios.

In case one: the user makes a connection to the weblogin server which, if there is no login cookie stored in the browser, presents a login page to the browser. If authentication succeeds and the user is accepting cookies the login cookie is associated with the user. Otherwise an error is returned. If the weblogin server has a populated service menu (optional) then it is presented to the browser, a selection is made and the user connected to the required service. A service cookie is returned and the browser passes the login and service cookies to the weblogin server which associates the login cookie with the service cookie. The browser reconnects to the service including the service cookie in the request which is validated out-of-band by the weblogin server and if valid, the service allows access to the protected resource.

In case two: the user makes a connection to the required service which generates a service cookie. As there is no login cookie the weblogin server presents the login page with both cookies and, on successful authentication, the browser goes back to the service and from there on things proceed as in case one. User authentication is via Kerberos in the current release and all connections to this point are via SSL.

see: <http://www.umich.edu/~umweb/software/cosign/overview.html>

#### 5.1.2 Server requirements

Cosign consists of a daemon (cosignd) running on the central cosign server (weblogin server) and an Apache authentication filter (mod\_cosign) running on clients of the weblogin server. There is also an IIS and Java filter.

##### **weblogin server requirements:**

For the Apache module:

- Apache version 1.3.x or later/Apache 2.0 or later.
- OpenSSL 0.9.7a or later.
- MIT Kerberos v5 1.2.7 or later.

For the IIS Filter:

- Microsoft Internet Information Services (5.0 recommended).
- any Windows version that supports IIS.
- Intel platform.

daemon requirements:

- OpenSSL 0.9.7a or later.

### 5.1.3 Client requirements

- SSL support.
- cookies supported and enabled.

### 5.1.4 Ease of use for the end user

Straightforward. An unauthenticated request is directed to the Cosign login page and thereafter the 'user experience' is dependent on the application. However cookies need to be enabled, as it will not work without them.

### 5.1.5 Authentication Methods

Cosign will support a wide range of authentication methods, e.g. LDAP, shadow password file and Kerberos – basically it can use any of the `mod_auth*` modules available for Apache (and similarly for IIS).

## 5.2 Implementation

### 5.2.1 Ease of implementation and support from service provider

Implementation was relatively straight-forward especially given the excellent response to questions by the folks at Michigan.

### 5.2.2 How easy does it seem to be to develop a client (is there example code or a library/API) or are there standard clients or modules available?

There is no example code but the source code of `mod_cosign` could be used as a starting point.

PAM will be supported in release 1.8.

### 5.2.3 How easily would it integrate with uPortal?

Michigan and Edinburgh have done the integration.

n-tier proxiable credentials have been developed and are in the 1.6.0 release. Edinburgh University have an integrated uPortal environment.

### 5.2.4 How easily would it integrate with systems which require backend authentication (e.g. PHP/IMP, ColdFusion, Oracle, Exchange etc)?

For applications which don't support Kerberos, login functionality would need to be replaced in the application and the authenticated username would be passed in a cookie.

For kerberised applications, there is a CosignGetKerberosTickets feature, which tells the Apache module to get the Kerberos credentials from the weblogin server after authentication thus enabling the credentials to be proxied by the web application. Michigan uses it for IMAP, LDAP, AFS and their Kerberos change password application.

## 5.3 Support

### 5.3.1 Description of the code and the quality of the documentation

All written in C.

There is a small amount of installation and overview documentation.

### 5.3.2 How mature is the product?

The product is somewhat immature with the first beta release (V0.7.0) available in October 2002. Future development will include kx.509 integration (date to be announced shortly). Version 1.1.1 was used for this evaluation. The latest version (1.6.0) was released on the 7<sup>th</sup> May 2004.

see: <http://www.umich.edu/~umweb/software/cosign/roadmap.html>

### 5.3.3 How many sites is it used at?

Michigan, Penn State, Auckland and Global-Village.net.

### 5.3.4 The route to ongoing development and support for the product

Roll your own with backup via the mail contact at Michigan and the mailing list. Penn State, who have implemented Cosign for their 80,000 students said:

"We have had very good experience with the Cosign code and the developers."

There is no commercial support available.

## 5.4 Sign-on/sign-out

### 5.4.1 How does the user initially authenticate?

The user requests a protected page and, if not already authenticated, is redirected to the Apache authentication filter (mod\_cosign) and prompted for their username and password. On successful authentication or if previously authenticated then the request is served by the application. Cookies are presented to the user and must be accepted for the system to work.

see: <http://www.umich.edu/~umweb/software/cosign/overview.html>

#### **5.4.2 Can the service use the credentials that the user has already acquired by logging in to Active Directory or Kerberos, or do they always have to authenticate through their web client?**

Always through the web client. However when the KX.509 functionality is available, then an X.509 certificate can be used as pre-authentication to acquire a cookie transparently.

#### **5.4.3 How are credentials cleared down on completion of session?**

The credentials are stored in non-persistent (session) cookies which are deleted by the browser when the session is closed.

#### **5.4.4 Will server restarts cause the authentication database to be flushed?**

Restarting the server has no affect on the browser cookies so re-authentication would not be required.

## 5.5 Reliability

#### **5.5.1 How vulnerable is the system to failure of a single component?**

It is vulnerable to network failure but has resilient server capability. Because the cookies are still in the browser and not tied into the server, existing sessions will not need to re-authenticate.

#### **5.5.2 How easy is it to run multiple instances of that component to reduce the risk of failure (i.e. can a resilient infrastructure be built)?**

Replication between multiple weblogin servers is supported.

## 5.6 Security

#### **5.6.1 How secure does it seem to be and/or what are its vulnerabilities?**

The code needs an in-depth study to answer this.

No reported vulnerabilities have been found.

#### **5.6.2 What form of encryption is used and where is it used?**

Communications between the browser and mod\_cosign and the weblogin server use SSL. Communications between mod\_cosign and, in our case, the Kerberos KDC use an encrypted channel. In each case the encryption method used depends upon configuration options and possible negotiation between the two endpoints.

### 5.6.3 How dependent is the protocol on the security of the underlying transport? Will SSL be required for all services?

All communications between mod\_cosign and the web login server are SSL protected.

### 5.6.4 What steps does it take to ensure that the transport provides sufficient protection?

Communications between the browser and mod\_cosign must be changed in the Apache set up to use SSL between the browser and mod\_cosign.

### 5.6.5 How are cross-service attacks prevented?

All Cosign cookies are host cookies so only the originating host gets them back. This means that a compromise of a service only affects that service. However, if the login server or the user's browser is compromised then any Cosign service could be attacked. In general single-sign-on systems do increase exposure to cross site scripting (XSS) usually in the form of a hyperlink which contains malicious content. Michigan are working to alleviate this.

See: <http://www.cgisecurity.com/articles/xss-faq.shtml>.

### 5.6.6 How vulnerable is it to man-in-the-middle and impersonation attacks?

Unprotected application connections are naturally vulnerable. SSL and Kerberos are designed to protect themselves against such attacks but the connections are as vulnerable as the versions of SSL and Kerberos being used. It is essential that security advisories for these products are acted upon immediately.

## 5.7 Summary

### 5.7.1 Pros:

- Kerberos ticket available – good for integrating with backend servers.
- multiple weblogin servers for resilience.
- re-authentication not required if server fails.
- n-tier proxiable cookies.
- integrated with uPortal.

### 5.7.2 Cons:

- relative immaturity of product.
- dependent on acceptance of cookies to work at all.

## 6. KX.509

KX.509 is an implementation of short life X.509 certificates which builds upon an existing Kerberos infrastructure, like the one we have in the University. As X.509 certificates are understood by all browsers, this technology is of interest to the University in reduced sign-on.

### 6.1 General description

#### 6.1.1 Authentication model

KX.509 enables a user to use their Kerberos Ticket (TGT) to acquire a one-day X.509 certificate. It generates an RSA key-pair and connects to the organization-wide KCA (Kerberised Certificate Authority). It then authenticates as the user to the KCA using standard K4/K5 libraries. Once authenticated, it requests an X.509 certificate for the user with the transmitted public key and gathers information for the X.509 certificate from institutional resources (user's Full Name in particular). The KCA uses this information plus the received public key to generate an X.509 (v3) certificate signed by its CA private key with a validity lifetime of one day.

The KCA returns this resulting certificate to KX509 which, for Unix, stores the key-pair and certificate in the user's ticket file which can be accessed using a plugin for netscape/mozilla or any pkcs#11 compliant browser. For Windows they are stored in the registry in a standard place for IE.

#### References:

<http://www.nsf-middleware.org/documentation/NMI-R1/1/KX509KCA/>

<http://www.umich.edu/~x509/>

#### 6.1.2 Server (KCA) requirements

- OpenSSL version 0.9.6b or later.
- MIT Kerberos version 1.2.1 or later.
- a Kerberos realm already exists for which you are installing a KCA server.

For an environment where credentials are proxied, a Kerberised credential translator (KCT) is required and an Apache module, `mod_kct`, which acquires a Kerberos service ticket on behalf of an SSL authenticated user.

The server is not supported on Windows or MacOS.

#### 6.1.3 Client (kx509) requirements

- OpenSSL version 0.9.6b or later.
- MIT Kerberos version 1.2.1 or later.
- a Kerberos realm already exists and you have a Kerberos login for that domain.

- Kerberos client software is already installed and you can successfully get normal Kerberos tickets.
- a KCA server for the Kerberos realm exists.

#### **6.1.4 Ease of use for the end user**

Using the Netscape plugin on Unix has proved problematic, at least on Solaris, with the browser eventually crashing, although Michigan have it working. Certificates which have lapsed remain in the browser and are not discarded and non-lapsed certificates are a problem on non-personal machines e.g. in an Internet café.

#### **6.1.5 Authentication Methods**

KX.509 only supports digital certificates as a method of authentication.

## **6.2 Implementation**

#### **6.2.1 Ease of implementation and support from service provider**

Requires the installation of the kx509 software on the end system and on Unix the installation of the kpkcs#11 browser module. This would be possible for managed Unix desktop systems in an Institution but would be more difficult to provide elsewhere such as on home systems or an Internet café. It might be possible to enable the web-login page to do the certificate generation as a proxy and then supply it to the browser.

#### **6.2.2 How easy does it seem to be to develop a client (is there example code or a library/API) or are there standard clients or modules available?**

The standard client is kx509 which is available for Unix and Windows and which is the only client required.

#### **6.2.3 How easily would it integrate with uPortal?**

In order to work in a proxied environment there is an application called KCT (see [http://www.citi.umich.edu/projects/kerb\\_pki](http://www.citi.umich.edu/projects/kerb_pki)). mod\_KCT is an apache web server module that acquires a Kerberos service ticket from the KCT on behalf of an SSL authenticated user, which in this case would have been someone authenticated using KX.509. The web server can then act as a Kerberos client on the user's behalf. KCT runs on the same machine that runs the KDC and is therefore replicable just as KDCs are. It accepts user certificates via SSL from mod\_KCT and returns a Kerberos service ticket. The KCT is in service in Informatics. The quality of the KCT code is not very good. The work to integrate with uPortal would have to be done.

#### **6.2.4 How easily would it integrate with systems which require backend authentication (e.g. PHP/IMP, ColdFusion, Oracle, Exchange etc)?**

The apache module `mod_kct` enables the use of the supplied certificate to get Kerberos credentials and using them for onward authentication to a backend service whose authentication is managed by Kerberos.

## 6.3 Support

### 6.3.1 Description of the code and the quality of the documentation

This is a stable and well-used product although KCT and `mod_kct` are still immature. It is in widespread use with the Globus Toolkit and Grid service providers.

There is a small amount of installation and overview documentation.

### 6.3.2 How mature is the product?

Certificate based schemes are very mature, well tested and defined by international standards.

### 6.3.3 How many sites is it used at?

Michigan, CMU, USC, UCSD, Fermilabs, NCSA, Edinburgh and many more.

The many groups using the Globus Toolkit to build grids are able to use KX.509 as Globus uses X.509 authentication.

### 6.3.4 The route to ongoing development and support for the product

Support is from Michigan who are very helpful and responsive.

## 6.4 Sign-on/sign-out

### 6.4.1 How does the user initially authenticate?

If the user doesn't have a current Kerberos Ticket Granting Ticket then `kinit` is used to obtain and cache one from the Key Distribution Centre. This is used by the `kx509` client to get the short-life certificate from the Kerberised Certificate Authority. This needs to be done only once per machine login; after that the user then silently authenticates to any server using X.509 authentication.

### 6.4.2 Can the service use the credentials that the user has already acquired by logging in to Active Directory or Kerberos, or do they always have to authenticate through their web client?

An already acquired, unexpired TGT can be used by `kx509`.

### 6.4.3 How are credentials cleared down on completion of session?

There is also the issue of how you sign-off. The certificate will be valid for the length of the Kerberos credentials, typically about 10 hours.

On Unix systems, the certificate (and key-pair) are stored in a faked-up “Kerberos ticket” in the user's ticket-file/credential-cache and user logout should be set up (via the shell logout script) to delete the credentials.

On Windows they are stored in the registry, which means that KX.509 needs to remain running during a login session so that it can catch the “logout signal” and then delete them.

#### **6.4.4 Will server restarts cause the authentication database to be flushed?**

The KX.509 certificate is held locally to the user and will not be affected by server restarts.

## 6.5 Reliability

### **6.5.1 How vulnerable is the system to failure of a single component?**

It is vulnerable to network failure but has resilient server capability. Because the certificates are still in the browser, existing sessions will not need to re-authenticate.

### **6.5.2 How easy is it to run multiple instances of that component to reduce the risk of failure (i.e. can a resilient infrastructure be built)?**

The KCAs, KCTs and KDCs are all replicable for resilience.

## 6.6 Security

### **6.6.1 How secure does it seem to be and/or what are its vulnerabilities?**

It would appear to be very secure. It is built on tried and tested technology defined by international standards.

There is a problem with unexpired certificates. If the credentials cache isn't destroyed on logout, or if someone can get access to it through the filesystem on a compromised host, then someone can use that certificate to masquerade as the user. The correct way to handle the case of sign-off would be to have users go to a website where the certificate would be revoked and added to a revocation list. However, you then have the problem of how to communicate the revocation list to all the web servers. This is the same problem highlighted by the TIES project. The TIES project describes the issues involved with the deployment of X.509 certificates as an authentication method. See: <http://edina.ac.uk/projects/ties/>.

### **6.6.2 What form of encryption is used and where is it used?**

Communications between kx509 and the KCA use SSL. The encryption method used depends upon configuration options and possible negotiation between the two endpoints.

### **6.6.3 How dependent is the protocol on the security of the underlying transport? Will SSL be required for all services?**

All services use SSL.

### **6.6.4 What steps does it take to ensure that the transport provides sufficient protection?**

KX.509 is always set up to use SSL.

### **6.6.5 How are cross-service attacks prevented?**

Services are not registered and they are completely independent of the authentication system. They just trust the KCA signature. If the KCA was compromised then all services would be compromised.

### **6.6.6 How vulnerable is it to man-in-the-middle and impersonation attacks?**

SSL and Kerberos are designed to protect themselves against such attacks but the connections are as vulnerable as the versions of SSL and Kerberos being used. It is essential that security advisories for these products are acted upon immediately.

## **6.7 Summary**

### **6.7.1 Pros:**

- all browsers can work with X.509 certificates.
- used by Grid services.
- multiple servers for resilience.
- integrates well with web-based services and with Kerberos.
- with KCT, it is also proxiable.

### **6.7.2 Cons:**

- browser handling of X.509 certificates is not user friendly.
- KCT code is poor quality.
- local problems with the kpcs#11 module and Netscape on Solaris.
- clumsiness of revocation list handling.
- browsers accumulate lots of lapsed certificates which are not discarded.

---

## 7. A-Select

### 7.1 General description

#### 7.1.1 Authentication model

A cookie based model. A user makes a request to an A-Select aware application for a resource (a URL) and is directed to the A-Select Server for authentication. If the user has no valid ticket-granting ticket (TGT) - i.e. is not yet authenticated or the TGT has expired - then the user is directed to the appropriate A-Select Authentication Service Provider (ASP) for authentication. If this is successful then a TGT is issued by the A-Select Server. This can then be used by the application (after validation by the A-Select Server) to generate an application ticket (AT). When this is presented by the user's browser to the application and the user / AT combination is valid then access to the resource is granted. If the user already had a valid TGT then the authentication step is omitted.

A-Select uses a simple concept for keeping track of an authenticated user. For this purpose, A-Select uses a ticketing mechanism in which the tickets hold the user's credentials.

Two tickets are used; the A-Select Server issues one ticket after a user is successfully authenticated. We refer to this ticket as the ticket-granting ticket. Users present their ticket-granting ticket to A-Select aware applications in order to get an application ticket. Application tickets are issued by applications. Ticket-granting tickets typically have a long lifetime in order to implement single sign-on functionality.

The A-Select tickets are implemented as non-persistent cookies and stored in the memory of the browser. Notice that non-persistent cookies are only visible to servers that set them. This means that application tickets set by applications running on one host are not visible to applications on other hosts. Similarly, ticket-granting tickets that are set by one A-Select Server are not visible to another.

Proxiable credentials are not apparently supported.

#### References:

<http://a-select.surfnet.nl>

[http://a-select.surfnet.nl/aselect\\_overview.html](http://a-select.surfnet.nl/aselect_overview.html)

[http://a-select.surfnet.nl/functional\\_flows.html](http://a-select.surfnet.nl/functional_flows.html)

#### 7.1.2 Server requirements

A-Select consists of the following components:

- the A-Select Server which controls user authentication.
- the A-Select ASP Server which connects to the required authentication backend (see 7.1.5).
- the A-Select Agent which optionally provides convenient functionality such as session management and generation of application tickets for the A-Select aware applications.

These components are written in Java and run in the Tomcat 4.1 engine which is built in to the A-Select Server and so may be run on both Windows and Unix. They all require Java 2 SE SDK 1.4.0 or higher.

There are also web-server filters provided for:

- Microsoft IIS.
- IIS IP filter.
- Apache 1.3.x.
- Apache 2.0.x.
- Java Servlets.
- Shibboleth / Apache 1.3.

The source code is only provided for the Apache filters.

### **7.1.3 Client requirements**

- SSL support.
- cookies supported and enabled.
- JavaScript must be enabled.

### **7.1.4 Ease of use for the end user**

The use of A-Select is straightforward for the user. A request from a user who is unauthenticated is directed automatically to the A-Select login page. In this respect A-Select is different from the other candidates. If A-Select is being used across multiple sites (authentication domains) then the user is requested to select their home domain from a drop down list. Presumably this can be avoided if the software is being used in a single institution. The user is then authenticated by one of a number of methods listed below. After being authenticated, the user is presented with a cookie which can be rejected or accepted (possibly automatically if the browser is set up to accept cookies) and thereafter the 'user experience' is dependent on the application.

When exiting an A-Select protected application the user can be asked if they wish to logout from the A-Select system as a whole. The act of logging out removes their entry in the A-Select server. Whilst this works well with Netscape and Mozilla, it will easily be circumvented by the IE user who closes the browser down rather than logging off.

### **7.1.5 Supplied methods of authenticating a user**

A user can be authenticated by the following means:

- Radius.
- LDAP.
- Basic PKI (though work still needs to be done).
- One-time password (sent to a mobile phone as an SMS message).

- 
- Face Recognition – via the proprietary Passfaces system (\$9/user/yr).
  - ODBC connection and MySQL.
  - IP Address.
  - Network Login – a long way off.

## 7.2 Implementation

### 7.2.1 Ease of implementation and support from service provider

The University has not implemented A-Select so is not able to comment on this section. However, Surfnet are very keen to see this product implemented outside of the Netherlands and have promised the necessary support to at least (we would surmise) the first implementers of it in the UK. There is an email support list and it is in mainly in English rather than Dutch!

### 7.2.2 How easy does it seem to be to develop a client (is there example code or a library/API) or are there standard clients or modules available?

Sample code. Socket based interface. 2 days for WebCT, average is 5-10 days.

### 7.2.3 How easily would it integrate with uPortal?

uPortal integration is planned.

### 7.2.4 How easily would it integrate with systems which require backend authentication (e.g. PHP/IMP, ColdFusion, Oracle, Exchange etc)?

A-Select already works with a number of products:

- WebCT.
- Oracle Portal.
- Blackboard.
- Citrix.
- Shibboleth.
- Sun-ONE portal.

Being actively considered:

- uPortal.

## 7.3 Support

### 7.3.1 Description of the code and the quality of the documentation

When this work was carried out, it was not possible to inspect the source code for A-Select. The source has however subsequently become available (see 7.3.4).

The quality of the documentation is very good although sometimes it needs to be searched for.

### **7.3.2 How mature is the product?**

The project was started in around 1995 with the first pilot code being produced in around 2002. The version of the code is currently labelled 1.3.

The versions of the code since 2002 have been produced by a contracted external company.

### **7.3.3 How many sites is it used at?**

A-Select is being used in a number of projects in the Netherlands,

see: <http://a-select.surfnet.nl/sites/sites.html>

These include: Hogeschool Saxion ([Niegebach Project](#)), [SURFSPOT.NL](#), Hogeschool voor Economische Studies, Amsterdam (HES) ([Citrix NFuse Portal](#)), Kennisnet.

Also the Netherlands Public Library Association has started an A-Select pilot.

### **7.3.4 The route to ongoing development and support for the product**

When this project began the IPR for the code lay with the commercial partner company Alfa & Ariss. Since then SurfNET have completed negotiations to buy the IPR back so that A-Select is now a pure open source product. A foundation to own the IPR and steer on-going development of A-Select is being set up.

There is commercial support available from Alfa & Ariss.

## **7.4 Sign-on/sign-out**

### **7.4.1 How does the user initially authenticate?**

When the user accesses an A-Select protected application, he/she is redirected to the A-Select server. After authentication, the user is automatically re-directed back to the application.

### **7.4.2 Can the service use the credentials that the user has already acquired by logging in to Active Directory or Kerberos, or do they always have to authenticate through their web client?**

Always through the web client.

### **7.4.3 How are credentials cleared down on completion of session?**

---

There is a two stage logout process, once from the application and then the second time from A-Select if the user wishes to close down entirely. This is achieved by a button on the A-Select bar rather than a change to the application. This process can be circumvented by a user who closes the browser rather than logging out, but that procedure deletes the cookie which makes the server record unusable.

#### **7.4.4 Will server restarts cause the authentication database to be flushed?**

As the information is not written to disk all sessions are lost on a server restart.

## 7.5 Reliability

### **7.5.1 How vulnerable is the system to failure of a single component?**

The version assessed is vulnerable to network and server failures. Re-authentication or restarting of all sessions would be required because the A-Select server holds all information in the server memory.

### **7.5.2 How easy is it to run multiple instances of that component to reduce the risk of failure (i.e. can a resilient infrastructure be built)?**

It would be possible to implement some form of network load balancing but the position is complicated as the server holds information which needs to be matched with the cookie, so that the load balancing system must always send the user back to the same server. The next major release, version 1.4, is stated to include facilities for load balancing and resilience but no details were available at the time of writing.

## 7.6 Security

### **7.6.1 How secure does it seem to be and/or what are its vulnerabilities?**

As the server source code is not supplied it is difficult to answer this.

No reported vulnerabilities have been found.

### **7.6.2 What form of encryption is used and where is it used?**

Communications between A-Select server components use SSL. Communications between the client and the A-Select Filters can use SSL. Where encryption is used the method depends upon configuration options and possible negotiation between the two endpoints.

### **7.6.3 How dependent is the protocol on the security of the underlying transport? Will SSL be required for all services?**

SSL is always used for transactions with the A-Select server, i.e. the login, logout and validate functions. The target application should require a client connection to be over SSL.

#### **7.6.4 What steps does it take to ensure that the transport provides sufficient protection?**

Inter-server communications are SSL protected but between client and an application this is optional. To ensure security, the administrator should require that all communications use SSL.

#### **7.6.5 How are cross-service attacks prevented?**

The tickets are stored in host cookies which are only visible to the servers that set them. Therefore a compromised server only affects that service. If the A-Select server or the user browser is compromised then any service can be attacked.

#### **7.6.6 How vulnerable is it to man-in-the-middle and impersonation attacks?**

Unprotected application connections are naturally vulnerable. SSL is designed to protect against such attacks but the connections are as vulnerable as the version of SSL being used. It is essential that security advisories for SSL are acted upon immediately.

## 7.7 Summary

### **7.7.1 Pros:**

- commercial support available.
- good range of both authentication methods and application interfaces pre-supplied.
- can require different strengths of authentication for different applications.

### **7.7.2 Cons:**

- model inherently subject to single point of failure.
- requires cookies.

## 8. Other Factors to Consider

### 8.1 Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)

In the context of single/reduced sign-on, the emerging technology that needs to be monitored is SPNEGO. Microsoft have developed this standard mechanism, which uses the HTTP protocol to allow a client and server to negotiate a security mechanism for authentication. SPNEGO tokens wrap the Kerberos ticket acquired during login using either an Active Directory or Unix Kerberos desktop client system. This ticket can then be used by the web server to authenticate to other network services.

Under Windows 2000 Server this works with correctly written applications but most applications do not currently work with it. Under Windows 2003, SPNEGO is the default mechanism for authentication modules. IE6 has full support for SPNEGO. There is a reference implementation for Apache at <http://sourceforge.net/projects/modgssapache/>. This is currently only at beta release 0.0.3 but it does work.

It appears likely that this SPNEGO mechanism will be a significant contribution to reduced sign-on in the coming years as a transparent method of moving from a Active Directory or Unix Kerberos authenticated desktop login to web based authentication. This mechanism will only work within the institution domain where the user has a desktop which has logged in to the AD domain, and will not work from outwith the University network.

### 8.2 Comparison between cookie based solutions and X.509 certificates

#### 8.2.1 Cookies

Cookies were not designed as an authentication mechanism but have been used to provide it. From a security point of view, the ability to intercept a cookie and for a third party to use it is probably its biggest weakness and mandates the use of a secure transport layer.

From a usability point of view, cookies are handled well by all browsers and if cookies are accepted by the user, their use is altogether more transparent than the use of X.509 certificates. However some users do not like to accept cookies and if cookies are reused, this will reduce the single sign-on functionality of CAS and will make all the other cookie based solutions unusable. Many browsers nowadays allow the user to specify the sites from which they will accept cookies, and such facilities provide a solution to the problem, but few users are aware that the capability to accept cookies selectively exists and fewer still are prepared to go through the effort of configuring the browser appropriately.

#### 8.2.2 X.509

From a security perspective X.509 technology has been around for a long time and is well tested. Some of the problems associated with using X.509 certificates, namely the need to transport personal keys around on floppies or by other means and the use of revocation lists when a long term certificate needs to be invalidated, are addressed to a certain extent by the short-term certificates of the KX.509 system. However in order to use KX.509, the kx509 client code must be installed on the end-user's system.

Whilst this can be arranged within the institution managed desktops, it is considerably more difficult for user controlled desktops both on and off campus and it is certainly not going to be generally available off-campus, e.g. at a conference or at an Internet café. This is a major drawback to an X.509 solution which will probably rule it out in most circumstances.

From a usability point of view, although certificates are handled by all browsers the user interface often leaves much to be desired, the messages given back to the user being particularly cryptic.

## 9. Which is Best?

### 9.1 What are the key criteria?

Any candidate, in order to be a sound option, must satisfy some basic criteria:

- security – is the protocol deemed to be adequately secure?
- reliability – are there single points of failure?
- credential delegation – is the protocol proxiable?
- cross-platform availability - are there Apache and IIS web server modules?
- on-campus use - is it easy to use by the end user on campus?
- off-campus use - is there a relatively secure way to use it from off campus?
- support - is there an adequate route to on-going support?

It is the authors' understanding that each of the cookie-based candidates fulfil the majority of these criteria, the main issues being:

- CAS in particular scores less well on reliability because of its inherent single point of failure in the server.
- Edinburgh University failed to get the CAS Apache module working.
- Windows IIS support is available but not supported for Stanford WebAuth.
- relatively secure off-campus use would need to be developed for all candidates.
- on-going support is not ideal for any of them except A-Select.

### 9.2 A comparison

The following table attempts to make a comparison of the contenders against a number of criteria. This is very much based on the Edinburgh experience of installing and testing the various contenders and may well change over time, so should only be used as an indicator.

The last-but-one column, 'Availability of connector modules', is included to give the reader an indication of how much effort might be needed at an institution site to link the product up with both the institution's method of authentication and with the various web servers that the authentication is to front-end.

	Usage	Single point of failure	Support	Documentation	Availability of connector modules	Shibboleth enabled
CAS	Moderate	Yes	Poor	Poor	V poor	No
Pubcookie	Widely	[1]	Variable	Small	Variable	Projected

	used			amount		
WebAuth	Not widely used	No	Responsive	V good	Poor	No
Cosign	Relatively new	No	V responsive	Small	Good	Has been demonstrated
A-Select	Moderate inside NL	[2]	Responsive, commercially available	Good	V good	Yes

Notes:

[1] Since this work was completed the documentation has been changed to indicate that a redundant configuration is possible, but this has not been tested.

[2] Since this work was completed we have been informed that the next major release of A-Select, version 1.4, will support redundant configurations.

## 10. Summary

As can be seen from the commentary and the table above, each of the candidates has particular strengths and weaknesses. In addition it is clear that some will require more effort to implement than others with a varying degree of support available.

Of particular interest to our community is whether the solution picked will work together with Shibboleth which is now being developed as an important component of the JISC service strategy for the medium term. Of the candidates, there is a projected project for Pubcookie but only A-Select has actually implemented it at this stage. This capability and degree of integration has not been tested by the project and could usefully be commissioned by the JISC in the near future to add to the overall knowledge of the products.

In summary, any institution will need to consider what aspects are important to them and pick their solution accordingly.