A SINGLE SIGN-ON PROTOCOL FOR DISTRIBUTED WEB APPLICATIONS BASED ON STANDARD INTERNET MECHANISMS

Julian Gantner

Andreas Geyer-Schulz

Anke Thede

Information Services and Electronic Markets
Universität Karlsruhe (TH), 76128 Karlsruhe, Germany

Keywords: single sign-on, cookies, Web authentication, Web services, cross-domain

Abstract: Growing e-commerce and personalized Web sites require users to set up many different personal accounts. Personal data has to be entered many times and each user has to memorize different username and password combinations. This reduces system security as users tend to either use passwords that are very easy to guess, or they write them down, or they use the same password for many different accounts. It also increases the cost of the administration of the user accounts.
We propose a protocol for a single sign-on system that allows users to visit multiple internet applications having to login only once. The system is based on standard internet mechanisms. It is composed of different servers that provide authentication and authorization services and is based on cookie technology. The system is designed to be implemented in a heterogenous environment with independent and diverse service providers. The communication between the servers is done via Web services. Additionally, plug-ins are available for other protocols that allow for easy integration of existing authentication and authorization components. A prototype system is operational at the Schroff Stiftungslehrstuhl Information Services and Electronic Markets.

1 INTRODUCTION

As more and more e-businesses and service providers emerge on the internet and especially the World Wide Web (WWW) the number of accounts and passwords a user has to handle and to remember increases rapidly. Many providers require some type of identification and personal data to offer their services. Users either tend to re-use the same account name and password for many different providers or write their credentials down and even carry them with them to have them always available. Both methods increase the risk that a malicious person might get access to personal data and abuse them which might cause an important damage to the user as well as the service provider. In addition, in decentralized organisations multiple user accounts increase IT administration and service costs.

Single sign-on (SSO) systems offer the possibility to use only one account for a multitude of distributed services (Shirey, 2000). The user logs into one of the services and is then able to access resources of other service providers without having to re-authenticate. The set of services for which SSO can be provided is called the SSO domain. SSO systems mainly stem from two sources, namely distributed operating systems and telecommunication infrastructure. Many different systems like Kerberos (Steiner et al., 1988) or Radius (Metz, 1999) have been developed that offer single sign-on for a special kind of environment. SSO solutions for distributed Web applications are special because they have to be based on standard internet mechanisms in order to be deployable. A multitude of user clients (Web browsers) exists on the market and there is no control over the choice of which browser a user decides to use.

Besides the most often stated field of e-commerce, universities are a very promising domain of setting up SSO environments (Murawski, 2000; Anchan and Pegah, 2003). Nowadays extensive user profiles of students, teachers, and researchers are kept digitally and e-learning platforms and administrative systems require digital transmission of data. Universities not offering their students an online time-table system and access to lecture materials are even considered not to be up-to-date. Universities still represent a very heterogeneous system where each institute has already made its proper choice of platform and applications. It is difficult to impose common standards and protocols on these independent entities. Thus, to successfully implement an SSO system in such an environ-

Thede A., Geyer-Schulz A. and Gantner J. (2004).
A SINGLE SIGN-ON PROTOCOL FOR DISTRIBUTED WEB APPLICATIONS BASED ON STANDARD INTERNET MECHANISMS.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 191-198
DOI: 10.5220/0001400201910198

ment it must be able to integrate different types of existing systems and rely on standard mechanisms that can easily be adopted by different kinds of proprietary platforms.

We propose a single sign-on system for cross-domain authentication based on standard internet mechanisms that is designed in order to integrate various types of existing user accounts and user databases of service providers. The strength of our system lies in the clear separation of user credentials and user profiles from role assignment and access permission information. The communication between the different components of our system is based on standardized protocols to which existing applications can easily be adapted. The choice of policies is left to the service providers who can choose their level of trust towards other system components.

In the following sections, we first present our system and describe its functionalities. After that we discuss the characteristics of our system and compare it to other related systems for Web authentication.

2 PRESENTATION OF THE SYSTEM

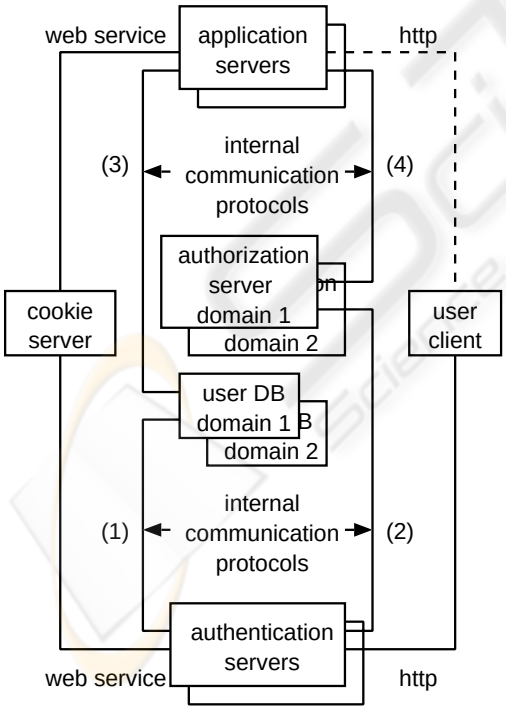

Figure 1: Components of the SSO system and communication links, dashed link: not necessarily secure

The components and their communication protocols are shown in fig. 1. The system consists of one or multiple application servers that contain the Web applications a user wants to access. The application servers can reside in different domains. One or multiple authentication servers handle the authentication of the user. They can access one or more user databases where one database contains the data for one specific user domain. The authentication server is allowed to query only the credentials needed to authenticate the user (typically username and password, link (1) in fig. 1). The user database may contain additional personal data about the user which are only accessible by the application servers in the corresponding domain (link (3)). For each user database domain there is an authorization server that contains the role and access information for each user. Two kinds of roles are available: public roles are valid across domain borders and can be accessed by the authentication server (link (2)). Local roles are restricted to the respective domain and may be queried only by application servers of the corresponding domain (link (4)).

The cookie server is the last element of the system. It contains information about the currently valid cookies, the corresponding users, and their public roles. The authentication servers may insert and delete entries in the cookie server database whereas application servers only perform entry look-ups. The description of the role system is not within the scope of this paper.

The communication between the user client and the application servers depends on the application requirements, it may or may not be secured. The remaining communication channels have to be secured as they carry sensitive data. The communication between the servers is mainly realized by the means of Web services over an SSL (secure socket layer) connection. Web services ease cross-domain access as they only require a standard HTTP connection. The communication with the user databases and the authorization servers may be adapted to the specific type of the underlying system (e.g. kerberos, secured Web service). Different plug-ins in the authentication servers offer customized communication. This allows for integration of Web applications with already existing authentication and authorization schemes.

2.1 Single sign-on procedure

Now how does the system offer single sign-on for multiple cross-domain applications? The procedure is the following (see also fig. 2, the letters refer to the messages in the figure):

1. The user is not logged on to any application. He sends a request for a service to application server *app1* (a).
2. *app1* cannot identify the user as no application server cookie is sent with the request. It there-

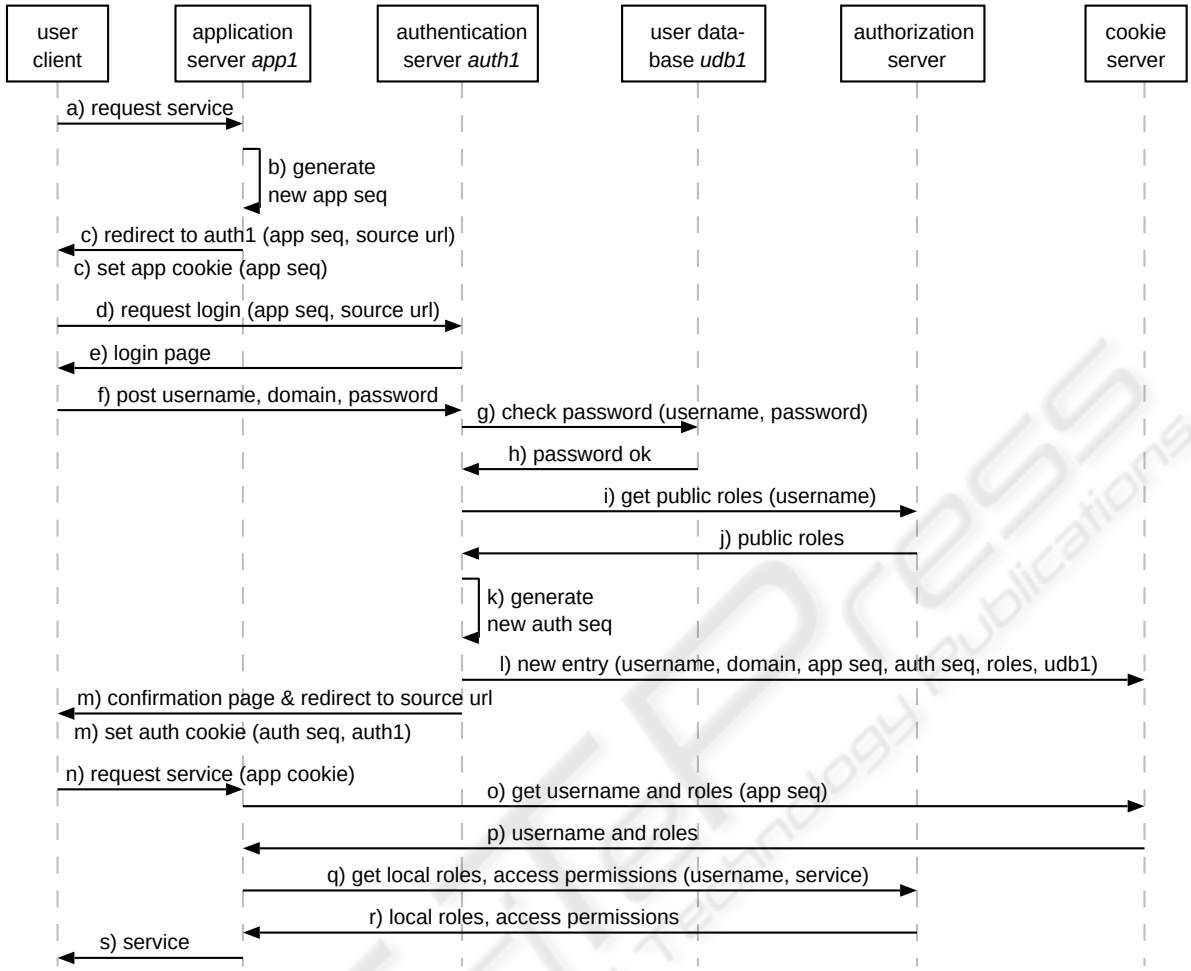

Figure 2: Sequence diagramm of SSO procedure at first login

fore sends a redirection to one of the authentication servers, say authentication server *auth1*. The redirection contains the URL of the originally requested application page and a randomly generated sequence of characters, the application sequence. The application sequence is also included in the application cookie that is sent back to the user along with the redirect request (b, c).

3. *auth1* requests the credentials from the user consisting of a username, a password, and a domain (d, e). The credentials are transmitted over a secure connection (f).
4. *auth1* contacts the user database corresponding to the given domain to verify the correctness of the credentials (g). Once the verification succeeds (h) *auth1* retrieves the public role information for this user from the domain's authorization server. The roles are matched by means of the user name (i, j).
5. *auth1* randomly generates another sequence of characters, the authentication sequence (k). The sequences have to be long enough such that the probability of a malicious user reproducing a valid sequence by random trial is sufficiently small. The two sequences, the user name and domain, the list of public roles, and the name of the user database are transmitted to the cookie server who saves the information in its database (l).
6. An authentication cookie containing the authentication sequence and the identification of *auth1* is created and transmitted back to the user along with a login confirmation page. The page displays for some seconds and then redirects back to the originating application URL (m).
7. The user now has two cookies set: the application cookie and the authentication cookie. Upon the following request to *app1* the application cookie is transmitted (n). *app1* extracts the application sequence from the cookie and requests the user infor-

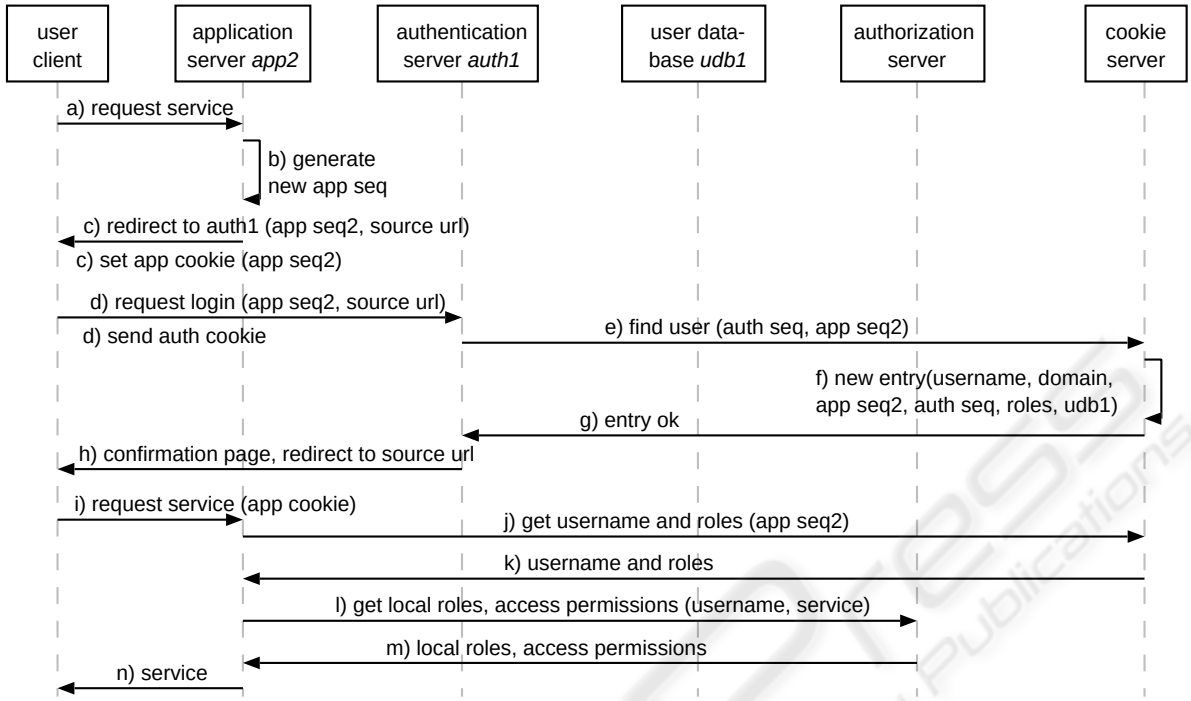

Figure 3: Sequence diagramm of SSO procedure at cross-domain service request

mation from the cookie server (o).

8. The cookie server finds the existing entry containing the application sequence in its database and transmits the user name and his public roles to *app1* (p). *app1* may now look up local roles and access rights in the domain's authorization server and may deliver the requested service (q – s).

The user can now easily access services from all application servers in the domain of *app1* as the corresponding application cookie is always included in the requests. If a user now requests a service from a server residing in a different domain (say *app2*) the application cookie is not sent along with the request and the user is not recognized as logged in. The following steps are then performed (see fig. 3 and corresponding message numbering):

1. The user accesses *app2* without an application cookie (a). The server generates an application sequence (b) and redirects the user to *auth1*, setting the second application cookie (c).
2. *auth1* extracts the authentication sequence from the transmitted cookie (d) and requests the associated user information from the cookie server. The request also includes the new application sequence (e).
3. The cookie server finds the user associated with the authentication sequence and adds another entry in its database differing from the first only in the new application sequence (f, g).
4. *auth1* redirects the user who has now three cookies set back to the originating URL of *app2* (h). *app2* can now identify the user as described in the last two steps of the previous procedure (i – n).

In this scenario *app2* uses the same authentication server (*auth1*) as *app1*. But the system allows for several authentication servers in one SSO domain and each application server may choose which authentication server to use. For different authentication servers to recognize the user as logged in it is necessary that they be able to have a valid authentication cookie transmitted. As cookies (Kristol and Montulli, 1997) may not be set on behalf of other servers the following work-around is deployed. Each authentication server maintains a list of the other known authentication servers. After the first successful login the confirmation page displayed to the user contains for each other authentication server an HTML image tag including the authentication sequence and the name of *auth1*. The requests sent to the other servers allow them to set an own authentication cookie with the same authentication sequence. In reply to each request each authentication server delivers a transparent pixel image. Now the user can be recognized as logged in by all other authentication servers, as well.

However, steps 7 and 8 may increase the network traffic considerably as a request is sent to the cookie server upon each service request to an application server by a user. In order to minimize the network traffic and to increase the performance of the system the application server may locally cache the association of the application sequence to the username and roles. This design decision needs to be carefully considered with respect to the global logout procedure and its implications are discussed in the following section.

2.2 Global logout

When the user logs out of one of the applications with which he is logged in (and of which he consequently possesses an application cookie) the logout has to be propagated to all other applications in the SSO domain, as well. Upon logging out the user is redirected to the authentication server. The authentication server sends a request to the cookie server to delete all entries containing the authentication sequence included in the authentication cookie. The authentication server sends a logout confirmation page to the user and deletes his authentication cookie. To delete the authentication cookies of all other authentication servers with which the user is currently registered the confirmation page contains again transparent images pointing to logout links of the other servers. Upon sending the images the other authentication cookies can be deleted, as well. If the user now accesses an application the application server will fail to find the corresponding entry in the cookie server and consequently can delete the application cookie. This is the standard case as described in the previous section.

If the application server uses caching to minimize network traffic the following two options of cache invalidation are possible:

1. The cache entries are only valid for a limited time and the application server checks the entries with the cookie server on cache expiration (variant 1).
2. The logout message is propagated to the application servers. This possibility is also employed in a similar manner in Microsoft's Passport SSO protocol (Microsoft Corporation, 2004b). It consists in keeping track of all applications the user is logged in with in the current session and to include transparent image requests for cookie deletion in the logout confirmation page, as well. The cookie server could easily keep track of the applications as a separate entry for each application sequence already exists. This would allow the application servers to locally cache user information and avoid subsequent requests to the cookie server while still validating the global logout without delay (variant 2).

2.3 Public roles

Each user can be associated with several public and private roles. Public roles are valid across all applications in the SSO domain whereas private roles are local to an application or domain. A user's public roles are contained in the cookie server entries and can be read by all applications the user accesses. Each application may decide on its own whether to trust the validity of a public role.

Public roles can be used to identify users and user groups and their corresponding access rights across different applications. For example, if a student worker logs in at the site of the department where he works he may be assigned the private role "student worker" and the public role "student at faculty of mathematics". If he subsequently accesses the Web page of the faculty of mathematics he is recognized as a valid student by means of the public role and may be granted access to lecture material without the need to re-authenticate.

Public roles could also be used to identify single users across different domains without the need to exchange user names or to issue globally unique user identifiers. Each user may be assigned a public role corresponding e.g. to his matriculation number. With this information each student can be identified by all university applications whose user databases contain the matriculation numbers of their users.

2.4 Failure of servers

For a single sign-on system it is important to note which of the participating servers constitute single points of failure. Unavailablilty of a server that provides necessary information to perform login functionality means that all services that require login and access control become unusable. It is therefore desirable to have different servers that are able to provide the same functionality. A slow authentication system due to increased server load is still preferable to a completely non-functional system.

In the system described in this paper the authentication server does not constitute a single point of failure. Many different authentication servers may be set up and as long as one of the authentication servers that an application server knows is functional the login can be performed. If an authentication server is temporarily not available this may result in a broken image tag displayed on the login confirmation page after a timeout but the login is still completed.

If a user database or an authorization server breaks down applications in the corresponding domain cannot be accessed but the other domains are not affected. In the current implementation only the cookie server as the only central component constitutes a single point of failure. By replication of the cookie

server's database and network interfaces this risk can be avoided, respectively reduced to a minimum. Mirror cookie servers do not increase the vulnerability of the overall system as the cookie server does not contain extremely sensitive data (like credentials, credit card information etc.). The sequences contained in the cookie server are only valid for a single session. To avoid replay of valid sequences in case that a malicious person gets access to the contents of a cookie server these could be stored encrypted either in the cookies or in the cookie server.

2.5 Scalability

For a single sign-on system to be practically relevant scalability is an important factor. An SSO system must be able to handle a large number of users, accounts, and applications without being considerably slowed down.

Looking at our solution, what does a large number of accounts and users imply?

- Accounts and their authorization information have to be stored in the user databases and authorization servers. As the system is already designed to have separate servers for each domain application specific scalability is not affected by the SSO procedure.
- The bottleneck of the current application is the central cookie server. The size complexity of the database is of the order *number of currently logged in users · number of currently used applications*. The number of concurrent users usually is only a small fraction of the total users, the same is true for the applications so that currently available data base systems should be able to accomodate cookie data both with respect to size and response time. The requests to the cookie server are all of the same structure such that efficient indexing techniques can be deployed. Tests with a single LDAP server containing about 25 million entries showed no performance degradation.
- Another important point is the amount of network traffic to and from the cookie server. For each login to a new application two requests to the cookie server are necessary (see fig. 2 and 3). The global logout requires one request for each authentication server the user logged in with. However, the most important amount of requests is generated at each access to an application service as the application server verifies the validity of the transmitted application cookie. These requests can be reduced or even avoided using the caching schemes as described in sec. 2.2 such that only a manageable amount of network traffic is left. Currently, the variant 1 described in sec. 2.2 is implemented.

2.6 The Role of Cookies for SSO

Building essential infrastructures on cookies has some important drawbacks. Many internet users have privacy concerns and do not want to use cookies, they can simply disable the cookie mechanism locally within their internet browser. There are also some security issues as cookies can on the one hand be modified by the user and on the other hand they may possibly be read or replayed by a third party. For a discussion on cookie security see e.g. (Samar, 1999).

Nevertheless, any SSO solution requires some sort of state management and user identification that cannot be provided by the stateless HTTP. Several possibilities exist to maintain information across multiple HTTP requests. An often used method is to include session information in the URLs, either as part of the query string or as an integral part of the URL itself. In contrast to cookies this does not require any data to be stored in files on the user's computer. This solution works very well for session management in one domain but it poses some problems when applied to cross-domain services. Each web server in an SSO domain would have to take care of including application and authentication sequences only in the URLs of the corresponding server. Otherwise sequences would risk to be exposed to third party websites who would be able to replay this information. Second, in our scenario web servers that issue redirect requests would have to have knowledge of the sequences that correspond to the redirect target server. In fig. 3, consider message (c). *app2* would have to include the authentication sequence of *auth1* in the redirect request. This requires additional transfer of sequence information over the network which means additional exposition of sensitive information to possible attackers. Thus we can see that replacing cookies with URL encoded sequences introduces additional security threats.

Keeping track of session information by using the user's IP address generally poses problems because of the use of proxy servers and network address translations, as well as the use of public terminals by many different users. Identification of the user through HTTP authentication also does not work for cross-domain scenarios. Other solutions require the user to install additional, specialized software and are no longer based on standard Internet mechanisms.

In general, privacy and single sign-on services are contradictory requirements that are difficult to combine. To minimize scepticism it is important to provide the user detailed information about what is stored in the cookies and what they are used for. Furthermore, an SSO solution based on cookies can still offer privacy concerned users alternative login mechanisms without cookies if they are willing to renounce single sign-on and log on to every service separately. In the current version, however, this is not yet implemented.

3 RELATED SYSTEMS

Many different solutions have been proposed in recent years for offering single sign-on in different environments (de Laat et al., 2000; Volchkov, 2001). Many of them are not transferable to a heterogeneous and distributed Web environment as they require a centralized structure of the involved components and common protocols. Other systems work with clients that are required to have additional functionality (Pfitzmann and Waidner, 2003) which we do not consider to be a realizable approach in the World Wide Web environment. In the following, we give an overview over the three solutions known to us regarding single sign-on for Web applications and compare the systems with our solution.

Microsoft's .NET Passport system (Microsoft Corporation, 2004b; Kormann and Rubin, 2000) is the largest functional single sign-on solution with 200 million accounts performing approximately 1350 authentication requests per second (March 13, 2003, (Microsoft Corporation, 2004a)). The system consists of a central Passport server that contains and manages all user accounts and corresponding user information and performs the authentication. Each user has a unique identifier. If a user has logged on to Passport from a service provider's site he can subsequently visit sites of other businesses adhering to Passport and is recognized by his unique identifier. The Passport server as well as the service providers set transient (session-only) or persistent cookies to recognize the user as already logged in. The cookies contain credential information about the user. A global logout from all Passport accounts is realized by including logout requests to all services the user is currently logged in with (and of which he possesses valid cookies) in HTML image tags.

The main difference between Passport and our solution is Passport's central authentication and user database server. This server is a single point of failure, if the service breaks down all businesses using Passport are unavailable to the users. It is not stated which measures are taken to offer scalability and backup servers. Possible dangers of replicated databases are discussed in (Kormann and Rubin, 2000). Our solution works with different, distributed authentication servers and integrates multiple, local user databases without requiring a separate, unique identifier. Identification across multiple domains may be realized at different levels by the means of public roles. No credential information is stored in local user cookies which may potentially be modified by the user himself and is not well protected against access from intruders.

The Liberty Alliance Project (Liberty Alliance Project, 2003) was formed in September 2001 by an association of several major companies in order to specify open standards for federated network identity management. The architecture (Liberty Alliance Project, 2003) allows for each service provider to maintain its own user database and user identities. A user who wants to use single sign-on between service providers with different identities may select to federate these identities between the service providers who are now able to match the foreign identity to their own. Without federation the user has to separately log into each service provider. The decision of whether to trust a user logged in with a different, federated identity remains the service provider's local policy. Identity providers take the role of the authentication server and the cookie server in our scenario, they use cookies to maintain a user's login state.

Like our solution, the Liberty architecture allows for integration of existing user databases and accounts of different service providers. The choice of whether to trust a federated identity can be made locally by each service provider, like it is the case with the public roles introduced in our system. The server architecture is nevertheless different. Liberty does not state whether SSO is possible across multiple identity providers and does not distinguish between authentication server and cookie server. Having the possibility to choose a nearby authentication server reduces the route length over which passwords and other credential information have to be carried to a minimum whereas no such sensitive data has to be transferred to and from the cookie server. Data have to be transmitted between the cookie and the authentication server only upon an initial service provider login. It is therefore useful and improves the security of the system to introduce this flexibility without having an important impact on the overall system performance.

Samar (Samar, 1999) proposes different protocols for cookie-based single sign-on. For cross-domain authentication, he introduces two centralized servers, a cookie server and a login server. The login server contains the authentication information about all users in the SSO domain and recognizes signed-in users by means of a login server cookie. The cookie server contains information about the different Web application cookies. Samar does not give detailed information about the possibility of a global logout in this scenario.

Samar's approach does not offer the integration of decentralized user databases and introduces two single points of failure, namely the login and the cookie server. The approach is similar to Microsoft's Passport except that Passport integrates the two different server types in one central server. Authorization is done locally at the service providers.

4 CONCLUSION

In this paper we propose a single sign-on system architecture based on standard HTTP mechanisms for distributed, cross-domain Web applications. The system allows for integration of distributed, proprietary user databases and authorization servers. User accounts need neither to be centralized nor to be explicitly exposed to other service providers in order to provide SSO services. Public roles offer a flexible mechanism to transport user information across different domains while leaving the final decision of whether to accept public roles to the single service providers. The system works with multiple, cross-domain authentication servers and a centralized cookie server. Different possibilities for implementing a global logout are proposed. We compared the system to other, well-known propositions and solutions for offering cross-domain single sign-on in a world wide Web environment and discussed similarities and differences between the systems as well as advantages and drawbacks.

The prototype system is already functional over different domains at the department *Information Services and Electronic Markets*. To test the system please visit http://demo.em.uni-karlsruhe.de/sso/ and http://sso.itloesungen.com/, log into one of them as indicated on the login screen, and test the system by visiting the other. First experiences with the system revealed no major difficulties concerning usability and the initial user reaction towards the single sign-on service was very positive.

The next step is to integrate other domains of university institutes and departments, affiliated companies and student organisations to test the system at a larger scale and to identify possible improvements especially concerning the adaption of proprietary systems into the SSO environment. Further research will be directed towards the analysis of various attack scenarios and on role contracting models.

REFERENCES

Anchan, D. and Pegah, M. (2003). Regaining single sign-on taming the beast. In *Proceedings of the 31st annual ACM SIGUCCS conference on user services*, pages 166 – 171.

de Laat, C., Gross, G., Gommans, L., Vollbrecht, J., and Spence, D. (2000). RFC 2903: Generic AAA Architecture. Network Working Group.

Kormann, D. P. and Rubin, A. D. (2000). Risks of the passport single signon protocol. *Computer Networks*, 33:51–58.

Kristol, D. and Montulli, L. (1997). HTTP State Management Mechanism. Network Working Group RFC 2109.

Liberty Alliance Project (2003). Liberty Architecture Overview v1.1. Technical report, Liberty Alliance Project. http://www.projectliberty.org.

Metz, C. (1999). AAA protocols: Authentication, authorization and accounting for the internet. *IEEE Internet Computing*, 3(6):75–79.

Microsoft Corporation (2004a). Microsoft .NET Passport for Businesses. http://www.microsoft.com/-net/passport/services/business.asp, accessed Feb 25, 2004.

Microsoft Corporation (2004b). .NET Passport Review Guide. Technical report. http://www.microsoft.com/.

Murawski, R. (2000). Centralized directory services and accounts management project. In *Proceedings of the 28th annual ACM SIGUCCS conference on User services: Building the future*, pages 198 – 201.

Pfitzmann, B. and Waidner, M. (2003). Analysis of liberty single sign-on with enabled clients. *IEEE Internet Computing*, 7(6):38–44.

Samar, V. (1999). Single sign-on using cookies for web applications. In *Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 158–163. IEEE.

Shirey, R. (2000). Internet security glossary. Network Working Group RFC 2828.

Steiner, J. G., Neumann, B. C., and Schiller, J. I. (1988). Kerberos: An authentication service for open network systems. In *Usenix Conference Proceedings*, pages 191 – 202.

Volchkov, A. (2001). Revisiting single sign-on: A pragmatic approach in a new context. *IT Professional*, 3(1):39–45.