Journal of Information Security, **2012, 3, 307-313**
http://dx.doi.org/10.4236/jis.2012.34037 Published Online October 2012 (http://www.SciRP.org/journal/jis)

Identifier Migration for Identity Continuance in Single Sign-On

Yoshio Kakizaki, Kazunari Maeda, Keiichi Iwamura
Tokyo University of Science, Tokyo, Japan
Email: kakizaki@sec.ee.kagu.tus.ac.jp, maeda@sec.ee.kagu.tus.ac.jp, iwamura@sec.ee.kagu.tus.ac.jp

Received June 22, 2012; revised July 26, 2012; accepted August 11, 2012

ABSTRACT

Single sign-on (SSO) is an identity management technique that provides the ability to use multiple Web services with one set of credentials. However, when the authentication server is down or unavailable, users cannot access these Web services, regardless of whether they are operating normally. Therefore, it is important to enable continuous use alongside SSO. In this paper, we present an identity continuance method for SSO. First, we explain four such continuance methods and identify their limitations and problems. Second, we propose a new solution based on an identifier migration approach that meets the requirement for identity continuance. Finally, we discuss these methods from the viewpoint of continuity, security, efficiency, and feasibility.

Keywords: Identity Management; Single Sign-On; Identifier Migration; Identity Continuance

1. Introduction

User authentication is typically required when using personalized online services. We often need to memorize a username (identifier) and password (secret) pair for each service. From a security viewpoint, the use of the same identifier and/or password for multiple service providers is undesirable; however, it is difficult to remember many separate identifier and password pairs. This results in lower usability, with users opting not to register for online services.

Single sign-on (SSO) is an identity management technology that provides multiple applications and supports multiple service providers through a single user authentication. In other words, users enjoy "one-stop authentication" because further authentication is not required. More specifically, with SSO, the user is authenticated only once by the authentication server. The user presents authentication results to other service providers and receives their services. Therefore, the number of username and password pairs that the user must memorize decreases compared to separate authentication for each service provider; as a result, usability dramatically improves.

However, the authentication may be unsuccessful on occasions when the server is unavailable for some reason (e.g., outage, hardware/software failure); in this case, the user cannot receive any of the multiple services provided by the SSO environment. This holds true even when service providers are operating normally. Therefore, a reliable method for receiving continuous service is needed, even when the authentication server is temporarily unavailable.

The key requirements for enabling users to continue using services when the authentication server stops responding are as follows:

1) Continuity: this ensures that users are able to keep using services after the problem occurs.

2) Security: this ensures that a malicious attacker cannot masquerade as a user.

3) Efficiency: this ensures that the user does not experience a slowdown or other such problems during the outage or lack of authentication server availability.

In this paper, we describe four conventional methods for achieving identity continuance in SSO—the Redundant SSO Auth Server method, Alias SSOID method, Multiple SSOID method, and Different SSO combination method. We identify the limitations and problems encountered by these conventional methods, and propose a new solution based on the identifier migration approach. Our method meets the requirement for identity continuance. To evaluate each method, we apply the three requirements introduced above; furthermore, we discuss the range of influence and feasibility of each method.

The remainder of this paper is organized as follows: Section 2 describes the concept of SSO, and Section 3 gives a detailed definition of some useful terms in the SSO model, as well as defining the problem we attempt to solve in this paper. Section 4 presents four conventional solution methods and a discussion of their limitations, and we describe our identity continuance method

in Section 5. Section 6 summarizes our evaluation and discusses the relative merits and limitations of each system, and Section 7 presents our conclusions and ideas for future work.

2. Identity Management and Single Sign-On

Identity management concerns the management of individual identities, their privileges, attributes, and permissions, with the aim of improving security and usability [1,2]. We can divide identity management technologies into three models [3]: isolated, centralized, and federated. Under isolated identity management, each service provider manages the user independently and for a long time. Centralized identity management is implemented in a client-server model, separating the functions of service provider and identity provider. Federated identity management has gained in popularity in recent years [4]; for example, OpenID [5,6], Liberty/ID-WSF [7], Shibboleth [8], and InfoCard/Cardspace all use this method [9,10]. Identity management technology supports the following [3]: 1) End-user requirements; 2) Network operator requirements; 3) Service provider requirements; 4) Administrative requirements; and 5) Legal requirements.

SSO is achieved with centralized and federated identity management. Under SSO, the user does not need to login repeatedly to use multiple services if they have been authenticated once.

SSO techniques can be categorized as agent types or reverse-proxy types [11]. **Figure 1** shows the composition of the agent approach to SSO. An agent, which is part of the service provider, communicates with both the user and the authentication server to exchange authentication results, thus achieving SSO. **Figure 2** shows the composition of the reverse-proxy approach to SSO. The proxy server exists between the user and the service provider. When the user logs in to the proxy server, the service provider receives authentication from the proxy server instead of the user.

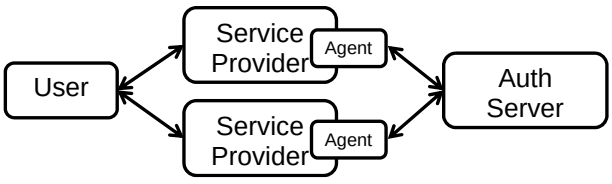

Figure 1. An agent type SSO technique.

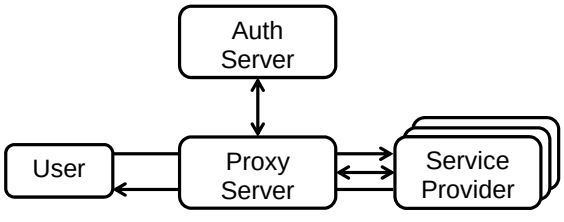

Figure 2. A reverse-proxy type SSO technique.

OpenID [5,6] is a de facto standard user-centric authentication that uses URIs (Uniform Resource Identifiers) and XRIs (eXtensible Resource Identifiers) to authenticate users. OpenID is expressed by a three-party model consisting of an OP (OpenID Provider), RP (Relying Party), and UA (User Agent).

An authenticator requires credentials, such as an identifier (username) and password pair, in order to authenticate a user. It is often assumed that, for their own convenience, users set the same identifier and password pair for different authenticators. In such cases, a malicious authenticator can masquerade as an authenticatee using the credentials it has received. Under OpenID, the RPs do not have credentials: only the OP does. Therefore, an RP requests user authentication from an OP and receives the authentication result from the OP; in other words, the RP cannot authenticate a user itself. In OpenID authenticcation, only one identifier and password pair are required to achieve SSO for multiple RPs.

3. Terms and Problem Statement

In this paper, we adopt and transform the agent type SSO approach illustrated in **Figure 1**. In this section, we introduce relevant terms, present a model for SSO, and summarize the problem we are addressing.

3.1. Terms

SSO Auth Server: An SSO Auth Server is an entity that authenticates users within the SSO environment. First, the SSO Auth Server issues an SSOID to a user. Next, the server attempts to authenticate a user who presents an unauthenticated SSOID as well as authentication credentials. If successful, an authenticated SSOID is issued to the user.

SSO Client: An SSO Client is an entity that provides services to a user within the SSO environment. The SSO Client requests user authentication from an SSO Auth Server, because the SSO Client cannot verify the credentials of a user who claims an unauthenticated SSOID. The SSO Client verifies an authenticated SSOID using the authentication result from the SSO Auth Server and, if successful, provides services to the user. The SSO Client binds an authenticated SSOID to a LocalID to manage the user and their information.

User: A user is an entity who receives service from an SSO Client within the SSO environment. A user must be authenticated by an SSO Auth Server in order to receive services from an SSO Client. Moreover, a user can receive services from different SSO Clients once they have been authenticated by the SSO Auth Server.

SSOID: An SSOID is an identifier that is uniquely assigned to each user within the SSO environment. The

SSO Auth Server and SSO Client identify the user by their SSOID. Each SSOID is a unique and permanent identifier of an individual user. As noted above, we use "unauthenticated SSOID" to indicate the SSOID of a user who has not been authenticated by the SSO Auth Server and "authenticated SSOID" when a user's SSOID has been successfully authenticated by the SSO Auth Server.

LocalID: A LocalID is an identifier used by each SSO Client to manage a user. Each SSO Client binds an authenticated SSOID to a LocalID; therefore, the LocalID is only effective in that SSO Client (*i.e.*, it is unique to each SSO Client).

3.2. Abstraction of Single Sign-On

Figure 3 shows the general model of SSO. In this model, a user obtains services from an SSO Client using an SSOID, which is issued by the SSO Auth Server. There are two fundamental procedures for obtaining services from an SSO Client:

Procedure A: the user presents an unauthenticated SSOID to the SSO client;

Procedure B: the user presents an authenticated SSOID to the SSO client.

In Procedure A, the SSO Client redirects the user to the SSO Auth Server, which authenticates the user with credentials corresponding to the unauthenticated SSOID. If successful, the authenticated SSOID is returned to the user, allowing the user to receive the desired services from the SSO Client by presenting this authenticated SSOID. Procedure B is identical to the latter portion of Procedure A once the user has acquired the authenticated SSOID. In either case, the user can receive services from multiple SSO Clients without further authentication once an authenticated SSOID has been issued by the SSO Auth Server.

The SSO Client manages each user by binding the authenticated SSOID to a LocalID, which is only valid for that particular SSO Client. In **Figure 3**, as an example, SSOID1 and SSOID2 are bound to LIDAA and LIDBB, respectively. Hence the SSO Client maintains that SSOID1 and SSOID2 correspond to different users.

3.3. Problem Statement

Continuing the example above, a user who is issued SSOID1 from SSO Auth Server A receives services from multiple SSO Clients, who each map SSOID1 to a unique LocalID. If we assume that SSO Auth Server A is unavailable for some reason, then users cannot receive service from SSO Clients via SSOID1, because they cannot be authenticated.

Users can receive services by obtaining SSOID2 from, say, SSO Auth Server B; however, each SSO Client views SSOID1 and SSOID2 as belonging to different users, because they are bound to different LocalIDs. Therefore, a user cannot access any information and/or history related to SSOID1 from the SSO Clients.

The key problem we aim to address is how to continuously use the information and history that a user has stored in the past. **Figure 4** shows an overview of the problem.

This problem occurs because the SSO Client, which does not have the credentials of its users, cannot authenticate users. Thus, the SSO Client behaves as if there are different users with different SSOIDs, even if it is the same entity. In this case, one solution is to authenticate a User using a LocalID in each SSO Client. However, the User must give their credentials to each SSO Client, which does not align with the SSO function, so we do not assume this solution.

In conclusion, SSOID continuance is crucial in order to solve this problem and provide users with a less interrupted service.

4. Identity Continuance in Single Sign-On

In this section, we describe four conventional methods for solving the problem of SSOID continuance described above.

4.1. Redundant SSO Auth Server Method

The Redundant SSO Auth Server method involves multiplexing authentication across multiple servers. Server functions can be used continuously as long as at least one server is running in this redundant configuration. **Figure 5** shows a model of the Redundant SSO Auth Server method. In the figure, SSO Auth Server A consists of a two-server redundant configuration, which is transparent to the user. In this case, SSO Auth Server A can be used

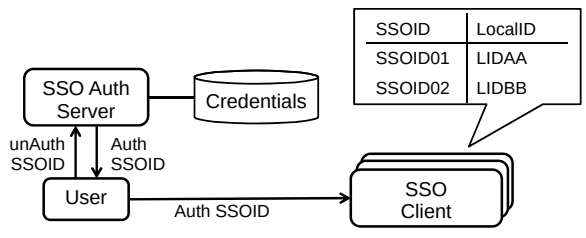

Figure 3. General model of single sign-on.

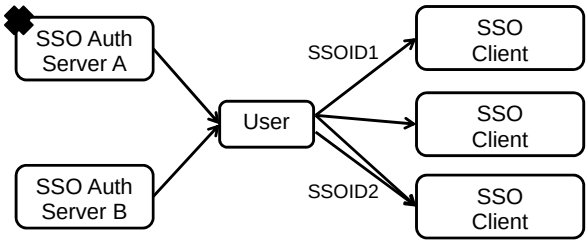

Figure 4. Overview of problem.

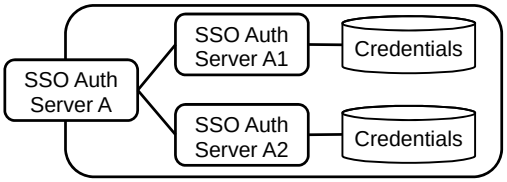

Figure 5. Redundant SSO Auth Server.

continuously as long as one of SSO Auth Server A1 or SSO Auth Server A2 is operational.

To implement the Redundant SSO Auth Server method, the type of redundancy required (either in the same domain or between different domains) must be considered. It is easier to compose a redundant configuration in the same domain; however, functions cannot be continuously used when a problem occurs in a higher-layer network, even if each server is operating normally. Unfortunately, it is more difficult to implement redundancy across different domains, and there is the operational problem of sharing or duplicating user credentials on the different domains.

4.2. Alias SSOID Method

The Alias SSOID method uses a different (or "alias") SSOID as a pseudonym for the SSOID described above.

Figure 6 illustrates the relationship between an alias SSOID and its "canonical" SSOID. In the figure, SSOID0A is an alias SSOID that binds to the canonical SSOID01. SSOID0A is presented to an SSO Client, and the SSO Auth Server that issued SSOID01 authenticates the user. When SSOID01 cannot be used, the user can present SSOID0A to the SSO Client, even though SSOID0A binds to another canonical SSOID.

4.3. Multiple SSOID Method

The Multiple SSOID method binds SSOIDs to LocalIDs. Services can be accessed continuously by submitting other SSOIDs when one particular SSOID cannot be used due to an outage. **Figure 7** illustrates the multiple SSOID method. An SSO Client specifies a user by assigning a LocalID, which binds to the user's SSOID. In the figure, SSOID01 and SSOID11, which are issued from different SSO Auth Servers, both bind to LocalID LIDAA. Therefore, the SSO Client can treat different SSOIDs as the same user. Moreover, this does not require any changes in the SSO Auth Server; it is possible to use any SSOID, issued by any SSO Auth Server.

Users should perform a similar procedure with all SSO Clients. In **Figure 7**, a user with SSOID01 uses SSO Clients 1 and 2. SSOID11 was not added to SSO Client 2, although the user added SSOID11 to SSO Client 1. Therefore, another LocalID, such as LIDEE, is assigned when the user accesses SSO Client 2 with SSOID11, and SSO Client 2 identifies the user as LIDEE. Thus, this method impairs the convenience of SSO.

4.4. Different SSO Combination Method

The Different SSO combination method binds multiple SSO methods to LocalIDs, and is an expansion of the Multiple SSOID method. In this method, users can login via any SSO method that binds to a LocalID.

Protocol translation is another approach. For example, Project Concordia aimed to permit interconnection by translating Security Assertion Markup Language (SAML) and OpenID protocols. In this approach, interconnection can be achieved between an OpenID server and an SAML client, or between an SAML server and an OpenID client. However, as the translation server becomes a single point of weakness, this does not solve the issue of SSO.

5. SSOID Migration Method

We propose the SSOID Migration method to allow an SSOID to be migrated or transferred to another SSOID Auth Server by a pre-arranged mutual agreement between SSO Auth Servers. **Figure 8** illustrates the SSOID Migration method. In the figure, SSO Auth Server A is the migration source; SSO Auth Server B is the migration destination.

Our method uses the Multiple SSO Auth Server and Redundant SSO Auth Server methods. In the Redundant SSO Auth Server method, it is necessary to transfer user credentials, which is problematic. Our method binds both SSOID1 and SSOID2, which are issued by SSO Auth

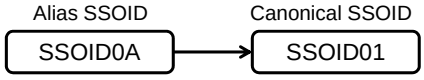

Figure 6. Alias SSOID.

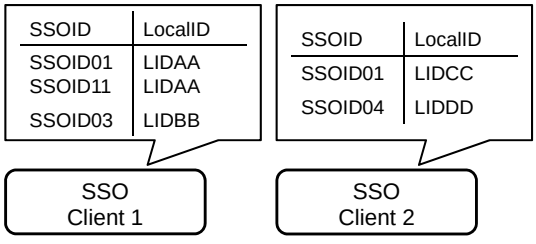

Figure 7. Multiple SSOID.

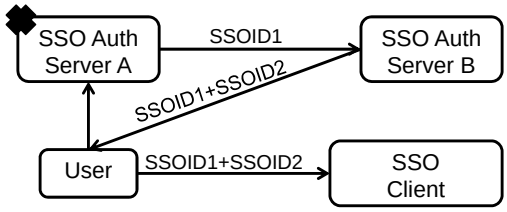

Figure 8. SSOID migration.

Server A and SSO Auth Server B, respectively.

5.1. Migration Phase

We consider the case of binding SSOID1 to SSOID2. SSO Auth Server B, being a different entity than SSO Auth Server A, cannot authenticate a user who presents SSOID1.

At first, the user logs in to SSO Auth Server B. Next, the user logs in to and accesses SSO Auth Server A from SSO Auth Server B as a SSO Client, and prepares to migrate. SSO Auth Server A binds SSOID1, which is issued by itself, to SSOID2, issued by SSO Auth Server B. The additional information transmitted during this binding process includes:

- information showing that both SSOID1 and SSOID2 are bound;
- information showing that SSO Auth Server A permits this binding in agreement with the user;
- information showing that the additional data has not been modified.

SSO Auth Server A redirects the user to SSO Auth Server B with SSOID2 and SSOID1, as well as the additional information specified above. Finally, SSO Auth Server B binds SSOID1 and SSOID2, and stores SSOID1 and the additional information. The migration phase is now complete.

5.2. Continuance Phase

When SSO Auth Server A is unavailable, the user logs in to SSO Auth Server B. SSO Auth Server B redirects the user to SSO Clients with SSOID2, the migrated SSOID1, and the additional information. The SSO Client verifies the additional information and confirms that SSOID1 and SSOID2 are bound. As a result, the SSO Client can assign the same LocalID to both SSOID1 and SSOID2. Henceforth, the user can access services continuously in the SSO Client via SSOID2.

5.3. Summary

Our method uses multiple SSO Auth Servers, and requires users to follow a predetermined procedure for identity continuance. Our method makes it possible to transfer an SSOID to other SSO Auth Servers alongside additional information, which is used to identify the owner of SSOID1 and SSOID2 to SSO Clients. Therefore, our method of SSOID migration can achieve the requirements of SSO in any circumstances.

However, our method does have some shortcomings. SSO Clients cannot verify whether the binding is still valid, even if both SSOID1 and SSOID2 are confirmed as bound by the additional information. To solve this, we can include a validity period as part of the additional information, although we cannot check the revocation time, meaning that additional information is still alive at the end of the validity period.

A second issue concerns the frequency with which the additional information is updated. The user should lie between SSO Auth Servers A and B during the update, because the exchange of additional information requires the user's agreement. As a solution, we propose to use an authorization protocol, such as OAuth [12].

6. Evaluation and Discussion

In this section, we consider the requirements described in Section 1. Furthermore, we discuss the range of influence and feasibility of each of the methods described in Sections 4 and 5. **Table 1** provides a summarized comparison of each method.

6.1. Requirement 1: Continuity

In the Redundant SSO Auth Server method, a user can continuously access a service by changing (transparently) from SSO Auth Server A1 to SSO Auth Server A2, and in the Alias SSOID method, users can continuously access a service by changing their canonical SSOID.

In the Multiple SSOID method, a user is authenticated by the LocalID belonging to each SSO Client, rather than the SSOID. Thus, a user can access a service if each SSO Client is operational, even when the SSO Auth Server has stopped; however, SSO cannot be used because the user is authenticated and now identified using a LocalID.

In the Different SSO combination method, a user can continuously access a service using other SSO methods.

In the SSOID Migration method, it is possible to continuously access a service if the SSO Client accepts and verifies the additional information described previously. Otherwise, the SSO Client cannot verify the relationship between SSOID1 and SSOID2, and the user cannot continuously access the given service.

6.2. Requirement 2: Security

The Redundant SSO Auth Server method faces a security problem when additional information is transmitted between servers. It is necessary to transmit additional information securely, especially if redundancy is achieved across different domains. Moreover, authentication results

Table 1. Comparison of identity continuance methods.

Methods	Req. 1	Req. 2	Req. 3	Range	Feasibility
Redundant	good		good	all	easy to adopt
Alias	good			all	[6]
Multiple		good		edrestrict	[13]
Combination		good		restricted	[14,15]
Ours	good	good	good	all	[16]

may not be reliable when the authentication policy at the destination server is different from that of the source server.

The Alias SSOID method is only secure if a user is revocable. Otherwise, it is possible to masquerade as another user, because the canonical SSOID is easily revocable.

In the Multiple SSOID method, the influence of masquerading as another user is confined to individual SSO Clients. Thus, this method is secure because of its reliance on individual authentication for each SSOID.

In the Different SSO combination method, the overall security level is defined by the lowest of the multiple SSO methods that can be accepted to login. Therefore, security problems will occur when vulnerable SSO methods are accepted, even if other SSO methods have a high level of security.

The SSOID Migration method uses additional information describing the relationship between SSOID1 and SSOID2. If this additional information is modified, a malicious attacker can masquerade as a user; however, the additional information includes modification detection codes. Moreover, this method is secure because of its reliance on individual authentication for each SSOID. Therefore, if a malicious attacker attempted to use SSOID2 fraudulently, the attempt would fail.

6.3. Requirement 3: Efficiency

In the Redundant SSO Auth Server method, it is necessary to select the SSO Auth Server to authenticate SSOIDs. This process is performed on the SSO Auth Server side; users and SSO Clients do not require any changes.

In the Alias SSOID method, the user prepares and binds the alias SSOID to a canonical SSOID. Hence, it is necessary for the entity to resolve the alias SSOID. Furthermore, the user must bind alias SSOIDs to other canonical SSOIDs when the canonical SSOID changes.

In the Multiple SSOID and Different SSO combination methods, the SSO Client binds multiple SSOIDs to a LocalID; therefore, no changes are required in the SSO Auth Servers. Users must perform a similar procedure for all SSO Clients; thus, the user procedure is more complex.

In the proposed SSOID Migration method, SSO Clients and users do not require any changes. The SSO Auth Server must manage the bound SSOIDs and the corresponding additional information, presenting such information on demand; however, the SSOID Migration method is more usable than the Multiple SSOID method, because many SSO Clients exist.

6.4. Range of Influence and Effect

In this section, we discuss the range of influence by applying identity continuance solutions, and this evaluation indicates user experience.

Using the Redundant SSO Auth Server method, the Alias SSOID method, or the SSOID Migration method ensures that the effect reaches all SSO Clients. In the Redundant and Alias methods, the influence extends to all SSO Clients after acquiring an authenticated SSOID. In the SSOID Migration method, the influence does not extend from the SSO Client, which is presented with additional information and an authenticated SSOID.

Conversely, the influence and effect of the Multiple SSOID method and the Different SSO combination method only reaches SSO Clients handled by the user. Of course, influence and effect are not exerted on other SSO Clients at all. Hence, the influence and effect of the Multiple SSOID method has a restricted range, whereas the other methods have a wide range.

6.5. Feasibility

In this section, we discuss the feasibility of each of the four methods, as well as the problems that may occur in their operation. Moreover, we refer to examples of actual use.

The Redundant SSO Auth Server method is easy to adopt. However, it is necessary to select an SSO Auth Server that authenticates any unauthenticated SSOIDs, as described in Section 6.3. Thus, a higher-layer entity (e.g., SSO Auth Server A in **Figure 5**) is needed to implement this method.

An example implementation of the Alias SSOID method is the HTML-based Discovery method of OpenID 2.0 [6]. This method discovers the claimed identifier by showing the OP endpoint URL as a LINK element within the HEAD of an HTML document. HTML-based Discovery is a working example of the Alias SSOID method, if we assume the URL of the HTML document to be an alias and the OP endpoint URL to be canonical.

Similarly, SourceForge [13] is an actual example of the Multiple SSOID method in OpenID. SourceForge supports login with any OpenID that has been registered beforehand.

ATND [14] is a concrete example of the Different SSO combination method. ATND can bind two SSO authentication methods, Twitter OAuth authentication and OpenID authentication. As a result, users can access services from either SSO authentication with one account. Another example is that of Stack Overflow [15], which supports Facebook OAuth authentication and OpenID authentication.

Reference [16] provides an example of the SSOID Migration method in OpenID. This method helps users who utilize other SSO Auth Servers. It is thought that SSO Auth Servers are passive with respect to cooperation in the identity continuance of others, because it is

beneficial for them to issue a lot of identifiers. Hence, an SSO Auth Server will see a chance to acquire new users when another server is unavailable. For this reason, it is difficult to achieve a system that incorporates our proposed method, even though it is suitable from a user aspect.

7. Conclusions

In this paper, we presented four methods for SSO continuance in the event that the authentication server was not available. We then proposed an identity continuance method based on an identifier migration approach. For each method, we discussed the continuity, security, efficiency, range of influence, and feasibility. Our proposed method has advantages over the four conventional methods from the viewpoint of identity continuance requirements. However, our method also has some shortcomings. To address these issues, we propose the use of an authorization protocol, such as OAuth, for achieving updates without user agreements.

In future work, we will study cloud identity management. This will increase in importance with the popularization of cloud technologies, and the SSO concept will spread widely. We expect to develop a trouble-resistant, non-stop SSO system.

REFERENCES

[1] A. Josang and S. Pope, "User Centric Identity Management," *Proceedings of AusCERT Asia Pacific Information Technology Security Conference*: *R&D Stream*, Gold Coast, 22-26 May 2005, pp. 77-89.

[2] J. Goode, "The Importance of Identity Security," *Computer Fraud & Security*, Vol. 2012, No. 1, 2012, pp. 5-7. doi:10.1016/S1361-3723(12)70006-4

[3] Y. Cao and L. Yang, "A Survey of Identity Management Technology," 2010 *IEEE International Conference on Information Theory and Information Security*, Beijing, 17-19 December 2010, pp. 287-293. doi:10.1109/ICITIS.2010.5689468

[4] D. Smith, "The Challenge of Federated Identity Management," *Network Security*, Vol. 2008, No. 4, 2008, pp. 7-9. doi:10.1016/S1353-4858(08)70051-5

[5] D. Recordon and D. Reed, "OpenID 2.0: A Platform for User-Centric Identity Management," *Proceedings of the* 2*nd ACM Workshop on Digital Identity Management* (*DIM*'06), Alexandria, 30 October-3 November 2006, pp. 11-16. doi:10.1145/1179529.1179532

[6] "OpenID Authentication 2.0," 2007. http://openid.net/specs/openid-authentication-2_0.html

[7] "Liberty Alliance Project." http://www.projectliberty.org/

[8] "Shibboleth." http://shibboleth.internet2.edu/

[9] T. Miyata, Y. Koga, P. Madsen, S. Adachi, Y. Tsuchiya, Y. Sakamoto and K. Takahashi, "A Survey on Identity Management Protocols and Standards," *IEICE Transactions on Information and Systems*, Vol. E89-D, No. 1, 2006, pp. 112-123. doi:10.1093/ietisy/e89-d.1.112

[10] T. El Maliki and J.-M. Seigneur, "A Survey of User-Centric Identity Management Technologies," *International Conference on Emerging Security Information, Systems, and Technologies*, Valencia, 14-20 October 2007, pp. 12-17. doi:10.1109/SECUREWARE.2007.4385303

[11] D. Nobayashi, Y. Nakamura, T. Ikenaga and Y. Hori, "Development of Single Sign-On System with Hardware Token and Key Management Server," *IEICE Transactions on Information and Systems*, Vol. E92-D, No. 5, 2009, pp. 826-835. doi:10.1587/transinf.E92.D.826

[12] E. Hammer-Lahav, "The OAuth 1.0 Protocol," *RFC*5849, 2010.

[13] "SourceForge." http://sourceforge.net/

[14] "ATND." http://atnd.org/

[15] "Stack Overflow." http://stackoverflow.com/

[16] K. Maeda, Y. Kakizaki and K. Iwamura, "Identifier Migration in OpenID," *Proceedings of the Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing* (*IMIS*-2011), Seoul, 30 June-2 July 2011, pp. 612-617. doi:10.1109/IMIS.2011.78

[17] Y. Kakizaki, K. Maeda and K. Iwamura, "Identity Continuance in Single Sign-On with Authentication Server Failure," *Proceedings of the* 5*th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing* (*IMIS*-2011), Seoul, 30 June-2 July 2011, pp. 597-602. doi:10.1109/IMIS.2011.37

