PAPER *Special Section on New Technologies and their Applications of the Internet*

Service Independent Access Control Architecture for User Generated Content (UGC) and Its Implementation*

Akira YAMADA†a), **Ayumu KUBOTA**†, **Yutaka MIYAKE**†, ***and*** **Kazuo HASHIMOTO**††, ***Members***

SUMMARY Using Web-based content management systems such as Blog, an end user can easily publish User Generated Content (UGC). Although publishing of UGCs is easy, controlling access to them is a difficult problem for end users. Currently, most of Blog sites offer no access control mechanism, and even when it is available to users, it is not sufficient to control users who do not have an account at the site, not to mention that it cannot control accesses to content hosted by other UGC sites. In this paper, we propose new access control architecture for UGC, in which third party entities can offer access control mechanism to users independently of UGC hosting sites. With this architecture, a user can control accesses to his content that might be spread over many different UGC sites, regardless of whether those sites have access control mechanism or not. The key idea to separate access control mechanism from UGC sites is to apply cryptographic access control and we implemented the idea in such a way that it requires no modification to UGC sites and Web browsers. Our prototype implementation shows that the proposed access control architecture can be easily deployed in the current Web-based communication environment and it works quite well with popular Blog sites.

key words: *Web, User Generated Content (UGC), blog and cryptographic access control*

1. Introduction

In the era of Web 2.0, many users generate Web content as well as just browse through them. Web-based content management systems such as various Blog sites make it easier for users to generate content, which are called User Generated Content (UGC) or Consumer Generated Media (CGM). UGC opens to access to the public and the openness encourages many users to read them, which results in active and personal communications between generator and consumer users. We call a user who creates a content generator user, and a user who reads the content consumer user. However, the openness is not always a desirable feature for generator users. For example, a generator user would want to restrict accesses to a private portion of his content except from certain members, such as his family or friends.

Although some Blog sites offer access control mechanism, their functionality is limited within content hosted by the site and users registered on the site. For example, Vox**, a popular Blog site, allows generator users to classify their content into `friends`, `family`, `neighborhood` and `public`, each of which classes specifies who can access the content. Although they provide the fine granularity of accessing content, the content must be hosted by VOX. In addition, both generator and consumer users must have their accounts on VOX. It means that consumer users cannot read the content unless they have an identifier at the same UGC site as generator users do.

To identify users who do not have an account at the site but at other sites, independent identifiers such as TypeKey [1] and OpenID [2] are proposed. If a user possesses one of those identifiers, UGC sites can identify the user even if the user does not register to the site. However, the access control is limited within content hosted by the UGC site, and is not available for content hosted by other sites even if those content are generated by the same user.

In this paper, we propose new access control architecture for UGC, which enables users to employ a particular access control mechanism separately from the UGC site. The key idea to separate access control mechanism from UGC hosting sites is to use cryptographic access control [3], [4], which is originally proposed for access control of distributed storages. In the cryptographic access control, all content are protected by encryption algorithm so that the only user who has a decryption key can access to the content. Therefore, the system can enforce the access policy of content by controlling delivery of the decryption key instead of controlling accesses to the content.

Our contributions are not only the new architecture but also its implementation. Though the previous works require renovation of access control [5]–[7] or modification of UGC sites [8], [9], our architecture is implemented in such a way that it requires no modification to UGC sites or Web browsers. We evaluate that it can work well with existing UGC sites such as Blogger*** and LiveJournal****. This shows that the proposed architecture can be easily deployed in the current Web based communication environment.

The rest of this paper is organized as follows. We summarize background and problem statements in Sect. 2. In Sect. 3 we explain the proposed architecture, and we describe the implementation in Sect. 4. Section 5 discusses the

Manuscript received February 3, 2009.
Manuscript revised May 21, 2009.

†The authors are with KDDI R&D Laboratories Inc., Fujimino-shi, 356–8502 Japan.
††The author is with the Graduate School of Information Sciences, Tohoku University, Sendai-shi, 980–8579 Japan.

*This paper was originally presented at SAM '08–The 2008 International Conference on Security and Management, and some sections were added to the manuscript for the evaluation of our implementation.

a) E-mail: yamada.akira@kddilabs.jp

DOI: 10.1587/transinf.E92.D.1961

**http://www.vox.com
***http://www.blogger.com
****http://www.livejournal.com

security and related work. Finally, we conclude in Sect. 6.

2. Background

2.1 Problem Statements

Let's take an example of access control on UGC sites. Alice has an identifier at a Blog site, such as Blogger, and has generated many content on the site for a long time. Bob is a reader of Alice's Blog, and frequently comments on the articles. Alice wants to reply to a comment that Bob have written in private. Because the reply comment is relevant to the article and Bob's comment, she wants to write it on her Blog. Of course, there is no such an access control mechanism at Blogger, and Alice does not know the e-mail address of Bob.

Therefore she gets a new identifier on another site, such as MySpace[†], which provides a desirable access control mechanism. Then she transfers the article from Blogger to MySpace, and invites Bob to MySpace. Because Bob has an identifier on another Blog site, such as LiveJournal, and attaches the URL with those comments, Alice can post a comment about an article at Bob's Blog to invite Bob to MySpace. Considering the above example, the problems are summarized as follows.

1. Most UGC sites do not have enough fine-grained access control mechanism for their content.
2. Because the generator users have generated many content on a UGC site, the users want to continue to use the UGC site.
3. Even if a UGC site has access control mechanism, not all consumer users have an identifier at the same UGC site that the generator user does. Because multiple identifiers bother the user, the consumer user does not want to take another identifier.
4. A UGC site cannot control accesses to content hosted by other sites, despite the fact that those content are generated by a same user.

2.2 Conventional Access Control for UGC

Access control is defined as a mechanism to control accesses from subjective users to objective content based on their access policies. A typical UGC site adopts an access control system that consists of an authentication, authorization, and content hosting function (Fig. 1).

At the authentication, the system identifies users by an authentication protocol such as a simple password or cryptographic authentication protocol. The system compares an identifier of a user with the user database, which records a list of registered users. If the identifier is not registered in the database, the UGC site suggests the user to register in the site.

At the authorization, the system entitles the identified users to access the objective content according to how their access policies correspond to the content. Content has an

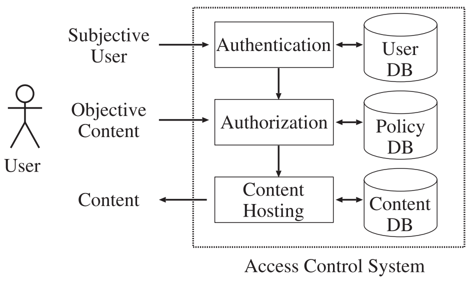


Fig. 1 Conventional access control for UGC.

access policy that lists allowed to or denied users, so the system compares the identifier of the user with the list to determine whether the user is allowed or not to access.

At the content hosting, the system lets the only entitled users to access the objective content. The system also has a function for users to generate content. Because the content database holds unprotected content, usually it is operated with the authorization to prevent from unauthorized accesses to the content.

2.3 Authentication API and OpenID

Currently some UGC sites provide authentication API (Application Program Interface) [1], [10], [11], which enables other UGC site to obtain identifier information of the API's sites. In other words, a UGC site can identify users, who do not have an identifier on the site but have an identifier on the API's site. Using an API, an access control system can delegate authentication function to the API's site. When a user accesses to a UGC site, the access control system redirects a user's session to an API's site. After authentication in API's site, the UGC site receives only the result of authentication.

Alternatively, OpenID [2] authentication protocol, is a similar functionality to authentication APIs. One of the significant features is that OpenID employs a URL (Uniform Resource Locator) based identifier. Because URL primitively indicates the location of resource, the URL of a user's Blog page is used as the identifier URL of the user as it is. Recently, many Blog sites started to provide OpenID to the users, such as LiveJournal, TypePad[††], and so on.

As another advantage of OpenID, the HTML document located on the identifier URL can be a bootstrap to discover the provider of the identifier, including the location of the identifier provider. For example, to use `http://identifier.example.com` as a user's identifier and `http://openid.example.com` as the provider of the identifier, the following tag would be added by the owner of URL to the HEAD section of the HTML document returned when fetching the identifier URL [2].

[†]http://myspace.com
[††]http://www.typepad.com

```
<head>
<link rel="openid.server"
href="http://openid.example.com">
</head>
```

2.4 Cryptographic Access Control

As we describe in Sect. 2.2, the authorization and content hosting functions are usually bonded to each other. This brings that the access control mechanism depends on the content hosting site, which may not have a desirable access control mechanism. To separate those functions, we employ cryptographic access control [3], [4], which is originally proposed to secure distributed databases.

In cryptographic access control, content are protected by encryption algorithm so that the only user who acquires the decryption keys can access to the content. Note that we assume the encryption algorithm is sufficiently robust against any attacks. When a user generates content, the user encrypts the content then puts it on a UGC site. Meanwhile the key used at the encryption is deposited to a Key Manager. Because the content is protected, the UGC site does not have to control accesses to the content. On the other hand, the Key Manager enforces the policy of the content instead of the UGC site. This mechanism makes the authorization free from the location of the content hosting site.

2.5 Related Works

Bauer et al. proposed an access control system for the Web content [8], which can be operated among multiple Web sites. However, the prototype system is implemented with a local Web proxy and server module, so that the scheme requires modification to UGC sites. As a more feasible solution, some Web browsers' extensions [12]–[14] have mechanisms to encrypt messages of Gmail. However, those extensions also require modification of Web browser, and have no function to manage content.

As the protection of XML document, there are some architectures [5]–[7], which control access to a portion of the document hosted by distributed sites. Though the architectures have a potential, they require conventional UGC sites to support the architecture. Alternatively, a URL based Identifier such as OpenID [2] and XRI [15] also have messaging protocols [16], [17], which enables communication between users who have an identifier at different UGC sites. However, those protocols have not been implemented yet for popular Web browsers or UGC sites.

OAuth [18] is an authorization protocol, which has a similar concept to our proposal in terms of content's access control on Web sites. However, there are differences in the assumptions, access control methodologies, and use cases. The proposed architecture assumes that a user cannot trust the Content Hosting Site completely. On the other hand, OAuth protocol assumes that a Content Hosting Site is trustworthy to authorize accesses to its content. In this context, a Content Hosting Site corresponds to a Service Provider in OAuth's specification.

The assumption that our proposal employ is reasonable, because most Content Hosting Sites are started by venture companies, which forces on developing the site's new functions rather than security functions. Furthermore, it is possible that the site's owner sells the site and its hosted content to other company. Concerning such situations, we introduce a Key Manger and User Agents as the trusted third party, which enables to separate authorization from a Content Hosting Site.

Although, our proposal is more secure than OAuth, it has disadvantage in the implementation. We implement User Agent using JavaScript invoked from Bookmark so that a user does not have to modify her Web browser. However, the architecture is troublesome for developers that implement in a limited environment of a programming language JavaScript. As the result, we only target on text-based content in this paper. OAuth does not have such limitation so that a Content Hosting Site can easily control rich content, such as images and videos.

In addition, the proposal and OAuth are different in their use cases. The proposed architecture is convenient to control accesses from individual users, rather than Web applications. On the other hand, OAuth has an advantage in controlling accesses from other Web applications rather than individual users. OAuth protocol requires initialization, which is acceptable for Web applications but bothers individual users. Twitter announces that it provides an interface of OAuth protocol to other Web applications, but not to individual users who possess OpenID. It is an example that OAuth protocol is not good for handling individual users.

3. Proposed Architecture

3.1 Goal

Figure 2 illustrates the goal of our architecture, which enables access control on any Web content based on identifiers issued by any ID providers. The key idea is to apply cryptographic access control of Web content on a Web browser. Currently, there are numerous UGC sites, on which users generate content individually. Our architecture introduces desirable access control to any UGC sites, even if the users

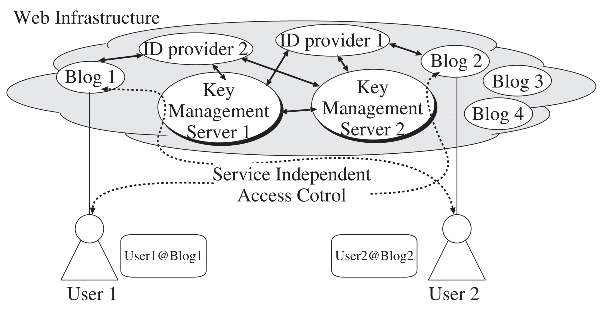


Fig. 2 Overview of proposed architecture.

do not have account at a same UGC site, ID provider, Key Manager, which is newly added a Web-based service in our architecture. For example, User1 can distribute an access-controlled content only for User2, who does not have an account on Blog1 but Blog2.

3.2 Separation of Content Hosting, Authorization, and Authentication

In this section, we compare conventional access controls for UGC and the proposed architecture. As we explain in Sect. 2.2, conventional UGC sites own their access control mechanism consists of an authentication, authorization, content hosting function individually. Obviously, the UGC site cannot distinguish users who do not have an identifier on the site. In addition, the UGC site cannot control content hosted by other sites, even if those content are generated by a same user (Fig. 3 (a)).

Using authentication APIs or OpenID, the UGC site can distinguish users who do not have an identifier on the site but have one on an API's site or an OpenID's ID-provider. Though the authentication function is separated from access control, the authorization function is still tied with the content hosting site. It means that the policies related to content are managed on a UGC site, and accesses to the content are controlled by the UGC site. Therefore, the users cannot control access to their own content that spread over many different UGC sites (Fig. 3 (b)).

The proposed architecture enables to operate an authentication, authorization, and content hosting function independently. Not only users but also policy can be managed by third party entities, which is different from the content hosting site. With this architecture, a user can employ a desirable access control mechanism against the content that might be spread over various UGC sites, regardless of whether the hosting sites have an access control mechanism or not (Fig. 3 (c)).

3.3 Definitions

In the proposed architecture, an access control for UGC is separated into authentication, authorization and content hosting, which are called Content Hosting Site, Key Manager, and User Manager. Each entity manages protected content, decryption keys, and users, using Location ID, Key ID and User ID individually. The definitions of the entities and identifiers are following.

- **Content Hosting Site:** A content hosting site is one of UGC sites where a generator user has an account. The content protected by cryptographic access control, are stored at the Content Hosting Site.
- **Key Manager:** A key manager is a newly introduced entity in our architecture, managing decryption keys corresponding to protected content.
- **User Manager:** We employ OpenID and OpenID's ID provider as User ID and User Manager. The architecture can also support alternative identifiers provided by authentication APIs.
- **Location ID:** Location ID is an identifier of location to discover the location of a protected content. Because protected content are hosted by UGC sites such as Blog, Permalink (permanent link) of an article can be a Location ID. A user can easily retrieve the protected content using a Web browser, given a Permalink as a Location ID.
- **Key ID:** Key ID is an identifier of a decryption key, and is also used to discover the location of the corresponding Key Manager. The Key ID is embedded in the header of a protected content, a user can obtain corresponding Key ID from a protected content. Key ID employs the same protocol to discover the Key Manager as the part of OpenID protocol. It means that Key ID is also an identifier URL, which HTML document includes the location of Key Manager in the HEAD section.
- **User ID:** User ID is an identifier URL of OpenID protocol. In most UGC sites that support OpenID protocol, the top page of each user's Blog can be the identifier URL of the user. OpenID protocol has method to discover the ID provider from identifier URL as the part of the protocol.
- **User Agent:** Because we target only on Web content such as articles on Blog or Wiki, User Agent is a Web Browser. The User Agent can be an agent of a consumer or a generator user of content, accessing to each entity to retrieve and create content.

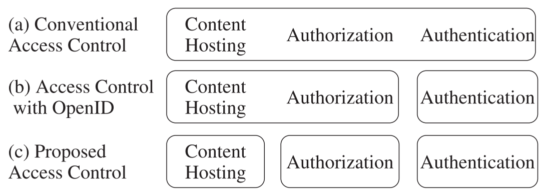


Fig. 3 Separation of content hosting, authorization, and authentication.

3.4 Content Publication

Figure 4 shows a sequence of content publication. (1) At first, a user generates content at a Content Hosting Site. Most Content Hosting Sites, such as Blog sites, have input form to write an article. After generation of content, the user invokes JavaScript to enhance User Agent. (2) The User Agent displays a dialog to be input the User ID, and transfers the User ID to Key Manager. The Key Manager redirects User Agent's session to the User Manager related to the User ID. (3) The User Manager identifies the user using a certain authentication protocol. If the authentication is successful, User Agent's session is redirected again to the Key Manager with a notification. (4) The User Agent displays a dialog to be input Policy, and sends the policy to the Key Manager. (5) The Key Manager issues a pair of Key ID and Key, and sends them to the User Agent. (6) The User

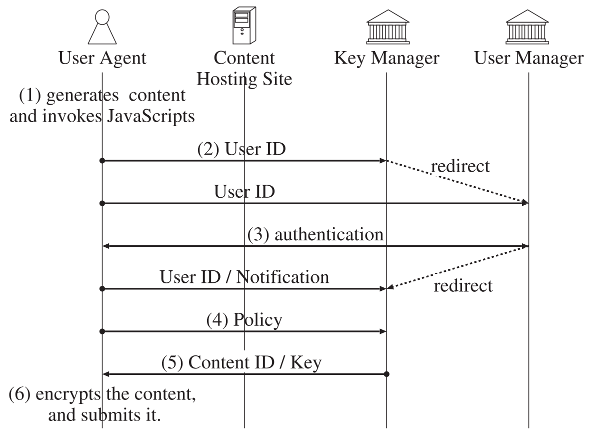


Fig. 4 Content publication.

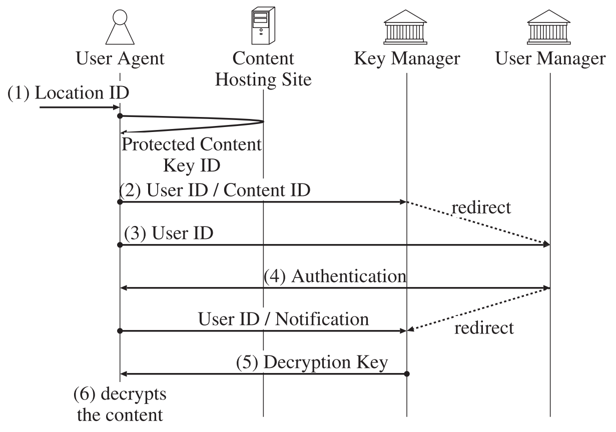


Fig. 5 Content retrieval.

Agent encrypts the content, and replaces the plain content with the protected content and the Key ID. After protection process, the user submits the article to the Content Hosting Site as usual.

3.5 Content Retrieval

Figure 5 shows a sequence of content retrieval. (1) At first, a User Agent gets the Location ID, for example, it is given by a user or is a hyperlink of a HTML document. Because a Location ID indicates the location of the protected content, the User Agent accesses to the Content Hosting Site to get the protected content. (2) From the protected content, the User Agent extracts the Key ID indicating the Key Manager, and accesses to the Key Manager. (3) The User Agent shows the Key ID and the User ID to the Key Manager. Because the User Manager is discovered from the User ID, the Key Manager redirects the User Agent's session to the User Manager. (4) The User Manager identifies the user using a certain authentication protocol. If the authentication is successful, User Agent's session is redirected again to the Key Manager with a notification. (5) The Key Manager determines whether the user is permitted to access or not based on the notification and the access policy related to the Key ID. If the user is permitted to access the content, then the Key Manager sends the decryption key to the User Agent. (6) Finally, the User Agent cancels the protection of the content using the decryption key to obtain the plain content.

3.6 Access Policy

The description of a policy is based on a typical access control list, which is an ordered list of rules with allow or deny. For example, the following policy allows Bob `http://bob.UserManager1` and Carol `http://carol.UserManager2`, and denies all the other users. In this case, Bob's and Carol's identifiers are managed by User Managers `UserManager1` and `UserManager2` individually.

```
deny all;
allow http://bob.UserManager1;
allow http://carol.UserManager2;
```

`all` is a predefined group consists of all users. A user can also define private groups, which include multiple identifiers and groups. The groups are managed by User Managers, where users' IDs are also managed. In order to evaluate a policy composed of group rules, the Key Manager has to queries the User Managers administrating the groups.

We modify a User Manager, OpenID provider, to support group management, because OpenID does not have such protocols. Though the modification is required, it does not affect easy deployment of our architecture. Since User Managers can separate the group management function, the users can employ another User Manager for the purpose of group management.

For example, a user Alice http://alice.UserManager1 defines a group `school` that includes Bob and Carol on `UserManager3`. Alice defines a rule `allow http://school.alice.UserManager3` as a policy instead of the above policy. When a Key Manager asks the members of the group, then `UserManager3` responds the members `http://bob.UserManager1` and `http://carol.UserManager2`.

If Alice want to describe a policy in a form of logical products such as "allow groups `school` **and** `college`," she transforms the form into a logical sum. Using De Morgan's law, she can convert any logical product into a combination of negative and logical sum. For example, the above policy is transformed into the following form:

```
allow all;
deny not http://school.alice.UserManager1;
deny not http://college.alice.UserManager1;
```

By constructing groups includes the other groups, users can define a hierarchical access policy. For example, `neighbors` contains `family` and `friends`, `friends` contains `bob` and `carol`, and so on (Fig. 6). Note that there are some conditions to construct a proper hierarchy. Users have to pay attention to avoid conflicts of some policies.

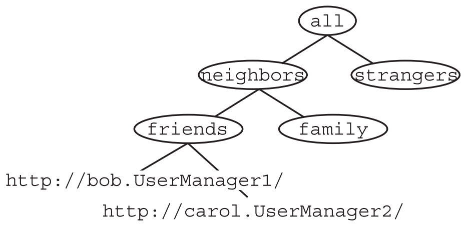


Fig. 6 Hierarchical access policy.

Table 1 Implementation targets.

Entity	Targets
User Manager	OpenID providers, LiveJournal
Key Manager	Apache with CGI on Linux
Content Hosting Site	Blogger, LiveJournal and Gmail
User Agent	JavaScript on Internet Explorer 5, 6 and Firefox 1.5, 2.0

4. Evaluation

4.1 Implementation Targets

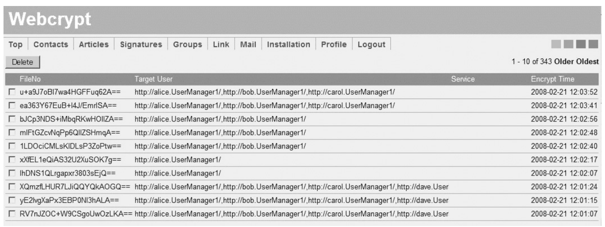
To demonstrate the feasibility of the proposed architecture, we implemented a prototype system. Table 1 shows our implementation targets. The prototype system supports at least three UGC sites: Blogger, LiveJournal and Gmail[†]. Gmail is one of a Web-based e-mail services, the messages of Gmail are also protected by our architecture.

As mentioned previously, our architecture does not require any modification both the Web browser and UGC sites. The cryptographic access control is implemented as a JavaScript program on Web browser, and the only thing a user has to do is to set a line of JavaScript at their bookmark of the Web browser. A JavaScript invoked from Bookmark is called Bookmarklet [19], which requires no modification against typical Web browsers, such as Internet Explorer and Firefox. After the Bookmarklet is invoked, the Web browser is enhanced as a User Agent of the proposed architecture.

4.2 Key Manager

A Key Manager has functions: key delivery, policy management, and OpenID's consumer. Requested by User Agents, the Key Manager distributes a key related to Key ID based on the user's access policy. We are considering Web API to provide the key management service as a Web service, which is available for any other Web service.

The policy management function enables users to manage access polices individually. Figure 7 illustrates the policy management Web site. Each user has an own personalized top page, which manages new articles, policies, and contact-list. When a user has a new article that another user publishes, the Key Manager displays the Key ID on the user's page. Because the Manager displays unused keys only, the user can distinguish whether an article is read or not.


Webcrypt

Top | Contacts | Articles | Signatures | Groups | Link | Mail | Installation | Profile | Logout

Delete — 1 - 10 of 343 **Older Oldest**

FileNo	Target User	Service	Encrypt Time
u+a9J7oBl7wa4HGFFuq62A==	http://alice.UserManager1/,http://bob.UserManager1/,http://carol.UserManager1/		2008-02-21 12:03:52
ea363Y67EuB+l4J/EmrlSA==	http://alice.UserManager1/,http://bob.UserManager1/,http://carol.UserManager1/		2008-02-21 12:03:41
bJCp3NDS+iMbqRKwHOlIZA==	http://alice.UserManager1/,http://bob.UserManager1/		2008-02-21 12:02:56
mlFtGZcvNqPp6QllZSHmqA==	http://alice.UserManager1/,http://bob.UserManager1/		2008-02-21 12:02:48
1LDOciCMLsKlDLsP3ZoPtw==	http://alice.UserManager1/,http://bob.UserManager1/		2008-02-21 12:02:40
xXfEL1eQiAS32U2XuSOK7g==	http://alice.UserManager1/		2008-02-21 12:02:17
lhDNS1QLrgapxr3803sEjQ==	http://alice.UserManager1/		2008-02-21 12:02:07
XQmzfLHUR7LJiQQYQkAOGQ==	http://alice.UserManager1/,http://bob.UserManager1/,http://carol.UserManager1/,http://dave.User		2008-02-21 12:01:24
yE2lvgXaPx3EBP0Nl3hALA==	http://alice.UserManager1/,http://bob.UserManager1/,http://carol.UserManager1/,http://dave.User		2008-02-21 12:01:15
RV7nJZOC+W9CSgoUwOzLKA==	http://alice.UserManager1/,http://bob.UserManager1/,http://carol.UserManager1/,http://dave.User		2008-02-21 12:01:07

Fig. 7 Content is listed at Key Manager Web site. The user can modify the policy at the site.

The user can edit the access control lists on the Key Manager after publication. Of course, the user can delete the keys, if the user wants to stop distribution. In order to edit the access policies, we introduce contact-list, which lists up users' IDs that the user previously used. The Key Manager complements listed users' ID, when the user types prefix of an ID to add a policy.

In order to manage groups seamlessly for users, we develop the User Manager for group management as a part of the Key Manager. After a user login to the Key Manager using an OpenID, the user can manages groups as well as policies. We also implement users' profile page, which provides information about each user. If a user has multiple IDs and want to announce them, the user can put the list of IDs on the profile page.

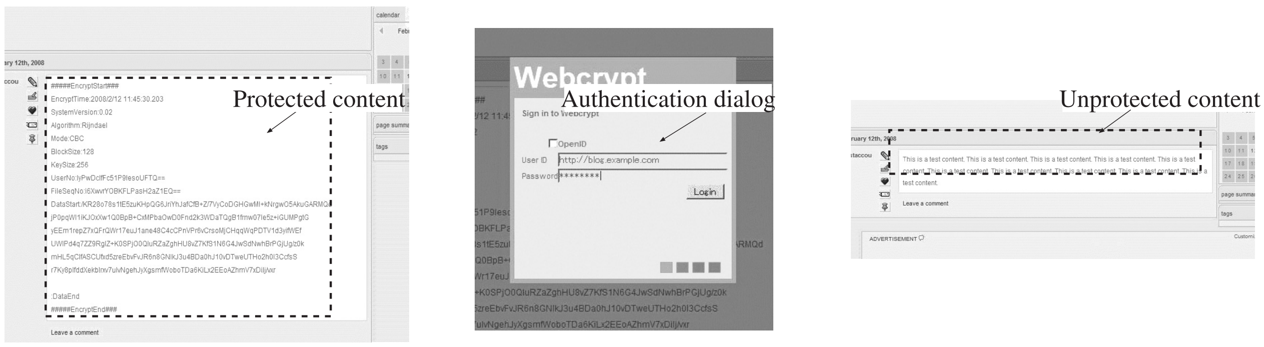
The Key Manage has an interface of OpenID protocol to delegate authentication to User Managers. A Key Manager connects other OpenID providers as an OpenID consumer, accessing the user information of the ID provider. We adapt libopkele 0.0[††], an OpenID library written in C++, for the implementation.

4.3 User Agent

We implement User Agent using JavaScript on two popular Web browsers: Internet Explorer and Firefox. JavaScript is not only embedded in HTML file by a server, but also invoked on displayed HTML file by a client. There are 2 techniques to execute a script at the client side, such as Bookmarklet and Greasemonkey[†††]. We select Bookmarklet since it does not depend on a specific browser, and not require installation except for a line of Bookmark.

Cross-domain access is a hurdle to implement the User Agent. After User Agent invoked on a Content Hosting Site, the Agent has to access objects on the Content Hosting Site, the Key Manager, and the User Manager. However, Web browsers have same origin policy, which restricts accesses between objects hosted on different sites. We employ JSONP (JSON with Padding)[††††] to overcome the restriction.

[†]http://mail.google.com
[††]http://kin.klever.net/libopkele/
[†††]http://www.greasespot.net
[††††]http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/



```
#####EncryptStart###
EncryptTime:2008/2/12 11:45:30.203
SystemVersion:0.02
Algorithm:Rijndael
Mode:CBC
BlockSize:128
KeySize:256
UserNo:IyPwDcIfFc51P9lesoUFTQ==
FileSeqNo:i6XwvrYOBKFLPasH2aZ1EQ==
DataStart:/KR28o78s1tE5zuKHpQG6JriYhJafCfB+Z/7VyCoDGHGwMI+kNrgwO5AkuGARMQd
jP0pqWI1iKJOxXw1Q0BpB+CxMPbaOwD0Fnd2k3WDaTQgB1fmw07Ie5z+iGUMPgtG
yEEm1repZ7xQFrQWr17euJ1ane48C4cCPnVPr6vCrsoMjCHqqWqPDTV1d3yifWEf
UWIPd4q7ZZ9RgIZ+K0SPjO0QIuRZaZghHU8vZ7KfS1N6G4JwSdNwhBrPGjUg/z0k
mHL5qCIfASCUfxd5zreEbvFvJR6n8GNIkJ3u4BDa0hJ10vDTweUTHo2h0I3CcfsS
r7Ky8pIfddXekbIrxv7ulvNgehJyXgsmfWoboTDa6KiLx2EEoAZhmV7xDiIj/vxr

:DataEnd
#####EncryptEnd###
```

User ID: http://blog.example.com
Password: ********

This is a test content. This is a test content. This is a test content. This is a test content. This is a test content. This is a test content. This is a test content. This is a test content. This is a test content. This is a test content.

(a) A protected content is hosted by LiveJournal. The detail of the format is illustlated at Fig. 8.

(b) User Agent is overlaid on LiveJournal using JavaScript invoked by Bookmarklet.

(c) The protected content is replaced with the unprotected content on LiveJournal.

Fig. 9 How it works (on LiveJournal).

Based on the same origin policy, browsers restrict destination of a function `XMLHttpRequest()` (XHR), which enables JavaScript to retrieves XML data on a remote host. In our case, invoked on the domain of the Content Hosting Site, the script cannot access to the Key manager using XHR.

JSON (JavaScript Object Notation) is a text-based, human-readable format to exchange data, and is originally employed by JavaScript. JSONP is JSON data wrapped within parentheses and padded with an arbitrary text. The following JSONP is a response of a Web site[†] supporting JSONP request. The first and last lines are padded text, and the middle is JSON data.

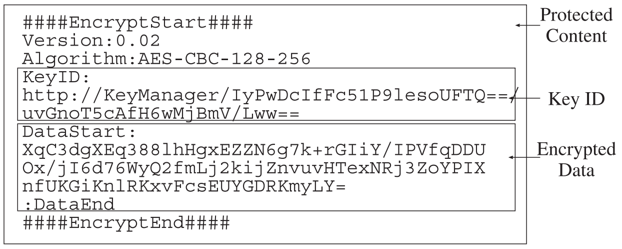
```
mycallback(
[{
  "u":"http:\/\/www.kddi.com\/",
  "d":"KDDI Designing The Future",
  "t":["kddi"],
  "dt":"2008-09-17T02:25:51Z","n":""},
  ...
  ]
)
```

The format of JSONP works as a function in a script even if the function is hosted on the other domain. The same origin policy does not restrict JavaScript files as well as image files or HTML files hosted on the other domain. Therefore, JSONP can substitute XHR in terms of accessing data of the other domains.

4.4 Protected Content

A protected content, which are stored by Content Hosting Site, is text-based format. Usually those sites prohibit users from uploading binary data and tags of HTML/XML as the content, so we cannot employ binary format or XML format for the protected content.

Figure 8 shows a sample of a protected content, which consists of a Key ID and an encrypted content. The plain content is encrypted with the AES (Advances Encryption Standard) algorithm and encoded with BASE64. Because


Protected Content
Key ID
Encrypted Data

```
####EncryptStart####
Version:0.02
Algorithm:AES-CBC-128-256
KeyID:
http://KeyManager/IyPwDcIfFc51P9lesoUFTQ==/
uvGnoT5cAfH6wMjBmV/Lww==
DataStart:
XqC3dgXEq388lhHgxEZZN6g7k+rGIiY/IPVfqDDU
Ox/jI6d76WyQ2fmLj2kijZnvuvHTexNRj3ZoYPIX
nfUKGiKnlRKxvFcsEUYGDRKmyLY=
:DataEnd
####EncryptEnd####
```

Fig. 8 An example of a protected content.

processes of public key cryptography require more calculation power than secret key cryptography, we adapts a secret key block cipher AES. The encrypted content is assigned a Key ID by the Key Manager.

4.5 How it Works

Figure 9 shows a sequence of screenshots that a consumer user retrieves a protected content hosted on a UGC site. At first, the user visits the UGC site pointed by Location ID, that is to say Permalink, and then invokes Bookmarklet to retrieve the protected content (Fig. 9 (a)). We assume that the user set the Bookmarklet in advance. It is also possible that the generator user distributes the line of hyperlink with the Location ID.

After JavaScript is invoked via Bookmarklet, it enhances user's Web browser as a User Agent of our mechanism (Fig. 9 (b)). The User Agent extracts the Key ID pointing the Key Manager, and sends the Key ID and User ID to the Key Manager. The Key Manager redirects the session of the User Agent to User Manager, by which the user is authenticated. After authentication, the Key Manager sends decryption key to the User Agent, according to the access policy.

Finally, the User Agent unprotects the content using the key, and replaces the protected content with the plain content (Fig. 9 (c)). Because the key-retrieval and the content-decryption are processed by the User Agent automatically, all the users have to do is respond the authentication dialog.

[†]http://delicious.com

Table 2 Average time for content publication.

	Content Hosting Site (LiveJournal)		User Agent	User Manager (LiveJournal)		Key Manager	User Agent	Content Hosting Site (LiveJournal)
	/ [sec.]	/update.bml [sec.]	Load [sec.]	Redirect [sec.]	Authentication [sec.]	Key Issue [sec.]	Encrypt [KB/sec.]	Post [sec.]
1	2.33	3.51	0.512	3.95	1.69	0.043	1.75	3.37
2	2.09	3.17	0.505	2.79	2.11	0.042	1.63	2.14
3	2.14	3.13	0.536	2.76	1.46	0.042	1.63	1.71
4	2.06	3.89	0.566	2.64	2.43	0.041	1.75	3.02
5	2.43	3.21	0.559	2.81	1.43	0.043	1.63	2.06
6	2.04	3.14	0.524	3.60	1.84	0.042	1.63	2.14
7	2.22	3.1	0.569	2.88	1.52	0.042	1.63	1.92
8	2.63	3.24	0.578	2.82	1.58	0.049	1.63	2.32
9	2.33	3.03	0.543	3.61	1.51	0.042	1.63	2.41
10	2.11	3.07	0.531	2.77	1.49	0.042	1.63	2.51
Ave.	2.24	3.25	0.542	3.06	1.7	0.043	1.65	2.36

4.6 Performance Evaluation

In order to evaluate performance of our architecture, we measure required time for 10 sequences of content publication. As comparison of a conventional access control on a Web site, we can compare response time of the proposed Key Manager with the Content hosting site or a User Manager, because these sites have an access control mechanism for handling their users. Table 2 shows required time. We measure required time using Web browser Firefox 3.0.1 with an extension FireBug[†] 1.2.1, which enables to monitor loading time for each Web page. The OS and CPU are Microsoft Windows XP Professional SP3 and Intel Pentium 4 3.60 GHz with 2 GB RAM.

We selected LiveJournal as a conventional Content Hosting Site and a User Manager. There is latency about 0.17 [sec.] between Web browser's machine and these sites. LiveJournal's top page "/" and a next page "/update.bml," which is for submission of an article, need about 2–3 sec. for downloading and rendering. As the user executes the Bookmarklet, the invocation of User Agent takes about 0.5 sec. User authentication using OpenID protocol takes about 5 sec., which is dominant in the sequence. The User Agent requests the Key Manager to issue a key, and receives a key in 0.043 sec. User Agent encrypts content with 1.65 KB/sec.; it encrypts an article contains 1,000 ASCII characters within 1 sec. It takes about 2 sec. to post the protected content to the Content Hosting Site. This evaluation shows that the conventional Content Hosting Site and the User Manager takes more time more than the proposed Key Manager does.

We evaluated scalability of the Key Manger. The specifications of the Key Manager are as follows; OS is Linux Fedora 7 and CPU is Intel Core2 1.86 GHz with 3 GB memory. We employ WebLOAD[††] as a load testing tool. Figure 10 shows round time of the Key Manager. X-axis is the number of users and Y-axis is round time. Round time is total time that the Key Manager takes for a set of user authentication, a policy evaluation, a key issue/search, and a key delivery, without waiting for user's input.

The round time increases linearly as the number of users. When 100 users access to the Key Manager at the same time, the Key Manager takes 4.7 and 2.7 second for a content publication and retrieval sequence. Comparing conventional Web sites LiveJournal, the response time of proposed Key Manager is feasible.

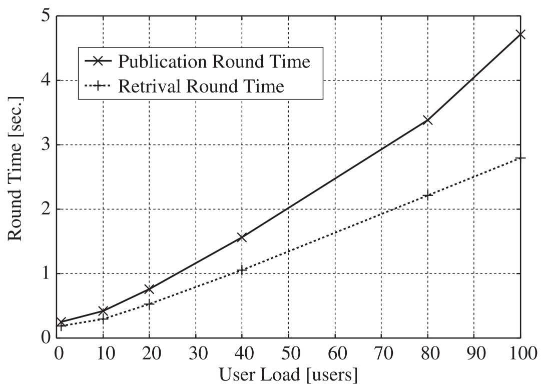


Fig. 10 Round time of Key Manager.

5. Discussion

According to a survey[†††] conducted by Japanese Ministry of Internal Affairs and Communication, 568 million articles on Blog sites occupy 5 Tbyte on a storage. It means that an article contains 9.5 Kbyte of text in average. Another survey[††††] indicates that the number of characters in an article depends on Blog sites, and are about 700 to 2,200. In our implementation, a User Agent can encrypt articles within from 0.41 sec. to 5.7 sec. in average. In a sequence of a content publication process, the proposed architecture increase additional 5.8 sec. to 11.1 sec. in average.

We believe that this additional processing time is acceptable for users who require more security on current Content Hosting Sites. It is true that evaluation of users'

[†]http://getfirebug.com

[††]http://www.webload.org

[†††]http://www.soumu.go.jp/iicp/chousakenkyu/data/research/survey/telecom/2009/2009-02.pdf

[††††]http://glink.jp/files/BlogSustainability060615.pdf

patience is difficult without launching a service of the proposal. However, considering a number of related works and implementations, such as OAuth, we think that the proposed functionalities are potentially desired by users, even if it increases processing time.

The Key Manager plays a strong role in the proposed system, the server handle the every encryption key. However, the server cannot access the plain content directly, because the content is encrypted and decrypted at the Browser of the client. Though the server could also access the encrypted content published on the user's Blog site, the server needs to discover the location of the encrypted content corresponds to the key.

The architecture, which controls all accesses at a single point, has both advantages and disadvantages. As the advantage, the Key Manager can easily enforce the all access policies at a single point. On the other hand, if the Key Manager is compromised by an attacker, then all controlled content become vulnerable. However, there is another barrier; the attackers have to find out the location of the encrypted content, similar as the above Key Managers' case.

Some users may be anxious, even if the above mechanisms guarantee the security. For higher security demands, it is possible to introduce more than one Key Manager. Applying cryptographic secret sharing [20] algorithm, the system can divides the encryption key into several shares. The multiple Key Managers store one of these shares independently. It is cryptographically impossible for any attacker to retrieve the key from shares less than threshold.

Furthermore, a personal Key Manager is another solution to enhance security. Because the format of protected content contains Key ID, which works as a bootstrap to discover the corresponding Key Manager, the user can easily introduce a personal Key Manager. This concept is similar to OpenID protocol, any entity, which includes a user, can establish an ID provider. Because management of a personal Key Manager is costly, the users have to balance between the gain of security and its cost.

6. Conclusion

In this paper, we propose a new access control architecture for UGC, which separates not only authentication but also authorization from Content Hosting Site. This architecture enables users to employ desirable access control mechanism, regardless of whether the UGC site has an access control mechanism or not. The key idea to separate access control mechanism from UGC sites is to apply cryptographic access control, which encrypts all content and controls accesses to decryption keys instead of the content themselves. Using our architecture, the users can publish their content, which can be accessed by certain members, even if the users have an account at the various UGC sites and the content are spread over various sites.

Our contributions are not only architecture but also its implementation. Our architecture can be applied to current Web browser and UGC site without any modifications on both sides. We believe that this architecture stimulates a Web-based communication environment.

References

[1] Six Apart, Ltd., "Typepad - profiles - typekey faq." (Online), ⟨http://www.typepad.com/profiles/typekey-faq.html⟩ (Reference 2009-01-27).

[2] D. Recordon and B. Fitzpatrick, "Openid authentication 1.1," May 2006. (Online), ⟨http://openid.net/specs/openid-authentication-1_1.html⟩ (Reference 2009-01-27).

[3] T. Wilkinson, D. Hearn, and S. Wiseman, "Trustworthy access control with untrustworthy web servers," ACSAC '99: Proc. 15th Annual Computer Security Applications Conference, pp.12–21, Washington, DC, USA, Dec. 1999.

[4] A. Harrington and C. Jensen, "Cryptographic access control in a distributed file system," SACMAT '03: Proc. 8th ACM Symposium on Access Control Models and Technologies, pp.158–165, ACM, Como, Italy, June 2003.

[5] E. Bertino, S. Castano, and E. Ferrari, "Securing xml documents: The author-x project demonstration," SIGMOD 2001: Proc. 2001 ACM SIGMOD International Conference on Management of Data, p.605, Santa Barbara, California, USA, June 2001.

[6] G. Miklau and D. Suciu, "Controlling access to published data using cryptography," VLDB '03: Proc. 29th International Conference on Very Large Data Bases, pp.898–909, Berlin, Germany, Sept. 2003.

[7] L. Bouganim, F.D. Ngoc, and P. Pucheral, "Client-based access control management for xml documents," VLDB '04: Proc. Thirtieth International Conference on Very Large Data Bases, pp.84–95, Toronto, Canada, Sept. 2004.

[8] L. Bauer, M.A. Schneider, and E.W. Felten, "A general and flexible access-control system for the web," Proc. 11th USENIX Security Symposium, pp.93–108, Monterey, Canada, June 2002.

[9] D. Balfanz, "Usable access control for the world wide web," ACSAC '03: Proc. 19th Annual Computer Security Applications Conference, pp.406–415, Las Vegas, Nevada, USA, Dec. 2003.

[10] Yahoo! Inc., "Flickr services." (Online), ⟨http://www.flickr.com/services/api⟩ (Reference 2009-01-27).

[11] Google, Inc., "Authentication for web appplication api — Google code." (Online), ⟨http://code.google.com/apis/accounts/AuthForWebApps.html⟩ (Reference 2009-01-27).

[12] linux solution park, "freenigma gmbh." (Online), ⟨http://www.freenigma.com/index.html⟩ (Reference 2009-01-27).

[13] R. Jones, "Gmail s/mime for firefox." (Online), ⟨http://richard.jones.name/google-hacks/gmail-smime/gmail-smime.html⟩.

[14] M. Langenhoven, "Gmail encryption." (Online), ⟨http://www.langenhoven.com/code/emailencrypt/gmailencrypt.php⟩.

[15] G. Wachob, D. Reed, D. McAlpin, C. Sabnis, P. Davis, and M. Lindelsee, "Extensible resource identifier (xri) resolution v2.0," March 2005. (Online), OASIS Committee Draft Version 01 ⟨http://docs.oasis-open.org/xri/xri/V2.0/⟩ (Reference 2009-01-27).

[16] G. Monroe and C. Howells, "Openid dtp messages 1.0 - draft 03," Dec. 2006. (Online), ⟨http://openid.net/specs/openid-dtp-messages-1_0-03.txt⟩.

[17] D. Reed and G. Strongin, "The dataweb: An introduction to xdi," April 2004. (Online), A. White Paper for the OASIS XDI Technical Committee ⟨http://www.oasis-open.org/committees/download.php/6434/wd-xdi-intro-white-paper-2004-04-12.pdf⟩.

[18] M. Atwood, R.M. Conlan, B. Cook, L. Culver, K. Elliott-McCrea, L. Halff, E. Hammer-Lahav, B. Laurie, C. Messina, J. Panzer, S. Quigley, D. Recordon, E. Sandler, J. Sergent, T. Sieling, B. Slesinsky, and A. Smith, "Oauth core 1.0," Dec. 2007. (Online), ⟨http://oauth.net/core/1.0/⟩.

[19] "Bookmarklets home page - free tools for power surfing." (Online),

⟨http://www.bookmarklets.com/⟩ (Reference 2009-01-27).
[20] A. Shamir, "How to share a secret," Commun. ACM, vol.22, no.11, pp.612–613, Nov. 1979.

Akira Yamada received the Ph.D. degree in Information Science from Tohoku University, Japan and the B.E. and M.E. degrees in Electrical and Electronic Engineering from Kobe University, Japan, in 1999 and 2001, respectively. He joined KDDI in 2001, and has been engaged in research on information security and network security. He is currently a research engineer of Network Security Laboratory in KDDI R&D Laboratories Inc.

Ayumu Kubota received the B.E. and M.E. degrees in Information Science from Kyoto University, Japan, in 1993 and 1995, respectively. He joined KDD (now KDDI) in 1995, and has been engaged in the research on mobile computing and secure communication system. He is currently a research engineer of Network Security Laboratory in KDDI R&D Laboratories Inc.

Yutaka Miyake received the B.E. and M.E. degrees of Electrical Engineering from Keio University, Japan, in 1988 and 1990, respectively. He joined KDD (now KDDI) in 1990, and has been engaged in the research on high-speed communication protocol and secure communication system. He is currently a senior manager of Network Security Laboratory in KDDI R&D Laboratories Inc. He received IPSJ Convention Award in 1995.

Kazuo Hashimoto received the Ph.D. degree in Information science and the M.E. and B.E. degrees in Electronics Engineering from Tohoku University, Japan, in 2001, 1979 and 1977, respectively. He also received his M.S. degree in Computer Science from Brown University, USA, in 1986. He joined KDD (now KDDI) in 1979, was the president and CEO of KDDI Labs. USA. Inc. He won the best paper award in the field of Artificial Intelligence from IEICE in 2001. He is presently the Professor of Tohoku University.