# A New Protocol for Single Sign-on for the Web

Constantin L. Belemuk, Dmitry E. Gouriev

***Abstract*—A new protocol for Single Sign-on for the Web is proposed. The main goal of the introduction of this new protocol is to combine the advantages of widespread protocols OpenID and OAuth and to be free from features which we regard as disadvantages of these protocols. Another problem covered in the article: how a client web site can automatically detect a provider web site where a user already has an account to authenticate against. A special supplementary protocol is proposed for this goal.**

***Keywords*— OAAP, OAuth, OpenID, Single Sign-on.**

## I. Introduction

Single Sign-on methods are intensively developed in the last decade in various areas of applications. On the World Wide Web, a Single Sign-on approach becomes important in connection with wide spreading of blogs, forums and social networks.

The most widely used protocols for Single Sign-on for the Web are OpenID [1] and OAuth [2]. These protocols have slightly different functionality and use cases. OpenID provides a user authentication only, whereas OAuth provides also an access to an API of a provider site. This difference and other ones are discussed in more details below.

An idea of a new protocol is to combine the advantages of both mentioned protocols. At the same time, some features of these protocols we regard as disadvantages, and we want the new protocol to be free of these features.

Below we present the critics of existing protocols, the goals of the new protocol and the requirements for it, as well as describe procedures and features of the new protocol and an experimental implementation of it. Our new protocol gets a preliminary name of Open Authentication and Authorization Protocol (OAAP).

Both OpenID and OAuth have no means to detect provider sites where a user already has an account to authenticate against. Users have to enter site identifiers manually. Alternatively, a client site can allow a user to select from a very limited list. We regard this as an annoying problem which limits a use of Single Sign-on on the Web.

In the OAAP we consider a number of solutions for this problem, which unite under a preliminary name of "Backend". We discuss these possible solutions below. One of the solutions is already implemented within an experimental implementation of OAAP. It has a form of a supplementary protocol, which is preliminary named OAAP Backend Protocol (OAAP BP). Potentially OAAP BP can be used in conjunction with OpenID and OAuth as well as with OAAP.

A work on OAAP, OAAP BP and other Backend solutions should not be regarded as completed. In conclusion of this article we discuss the directions of further works.

## II. Critics of Existing Protocols

The most widely used SSO protocols on the Web are OpenID and OAuth.

With the OpenID protocol a user already registered at one site (server site) can log into another site (client site), without having to register and authenticate at the client site. Authenticity of the user is provided by the server site. OpenID is based on user identifiers called OpenID identifiers.

The OAuth protocol provides the same as OpenID and additionally can establish so called "access token" – a piece of data which allows subsequent calls from the client site to an API of the server site. One important requirement of OAuth is that the client site must be registered with the server site before OAuth transactions occur.

We regard the following features of these protocols as disadvantages:

1) OpenID identifiers are generated by server sites and can have a format which is difficult for users to remember and deal with;
2) It is often difficult to find OpenID identifiers in a server site;
3) An OpenID server site authenticates a user only and does not provide any data (to be fair, an exchange of additional data is allowed, but it is out of the scope of the OpenID specification);
4) The OAuth protocol requires a client site to be registered at a server site; a user cannot pass an authentication using an arbitrary server site, if there is no such preemptive registration;
5) Both OpenID and OAuth cannot detect server sites where a user already has an account to authenticate against.

We want our new protocol (OAAP) to be free of these features.

The 4th feature is introduced in OAuth to limit an access to an API of a server site to trusted client sites only, and, in this way, to make a service more trusted by end users. However, it limits a spread of OAuth. In our new protocol (OAAP) we avoid this measure. Instead, we propose to provide a more granular access control to functions of the

Manuscript received June 3, 2014.

Dmitry E. Gouriev is scientists in Faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University, Moscow, Russia (email gouriev@oit.cmc.msu.ru).

Constantin L. Belemuk is student in Sevastopol Filial Branch of Lomonosov Moscow State University, Sevastopol, Russia (email xxxm1n0rxxx@mail.ru).

API and to allow end users to manage these access rights.

## III. Goals and Requirements

The main idea of a design of the new protocol is to combine the advantages and to be free from the disadvantages of OpenID and OAuth protocols. Following this idea, the new protocol must provide the following functions:

1) Simple and fast authentication of users.
2) Simple and fast authorization of users.
3) An access to an API of a server site (like in OAuth, optional).

The new protocol is proposed to have the following features:

1) No identifiers.
2) No preemptive registration of a client site at a server site.
3) A granular access control to API functions of a server site. This control must be managed by an end user.
4) Automatic detection of server sites where a user already has an account and granting the user an ability to select a site to authenticate against from a list of these sites.

## IV. The OAAP

Here we present the main scenario of the OAAP protocol. A complete specification of the protocol in published on the Internet [3]. See also a sequence diagram of the protocol in Figure 1.

1) The user presents the client site with a URL of a server site which can authenticate the user. In this step it is possible to use one of the functions called a “Backend”, for instance, the OAAP BP protocol described below, to detect a list of server sites where the user already has an account. If the function is used and succeed, then the user can select a server site to authenticate against from the list. If no such function is used or it fails, a user is to enter a URL manually.

2) The client site discovers a URL of the authorization server, using an HTTP GET requests. The authorization server is a part of the server site that handles an authorization process. This discovery can be executed by requesting of an XDRS document [4] from the server site, just like in OpenID. However, the OAAP specification additionally allows several more simple ways. A URL of the authorization server can be extracted from:
   - A special HTTP header,
   - An HTML code (using <meta> element),
   - A file of a special name and location on the server site (just like favicon.ico or robots.txt).

3) The client site and the server site create a shared secret using the Diffie-Hellman algorithm. An HTTP GET request is done by the client site for this purpose. Authenticity of public keys used in the Diffie-Hellman process is not proven in OAAP, so there is a possible hole for a man-in-the-middle attack. However, a use of the TLS transport [5] will prevent this attack. The created shared secret is then used to authenticate subsequent messages.

4) The client site redirects the user’s User-Agent program to the authorization server. Here the user passes authentication, authorization and confirms his intentions. Like in OpenID and OAuth, the authentication step is skipped if the server site detects that the user already have a valid open session with the server site. If an API is provided by the server site, the authorization server allows user to select functions which will be accessible in subsequent API requests. At the protocol level an access to each individual API function is granted or denied, however, in a user interface the authorization server may combine functions in reasonable groups.

5) The authorization server redirects user’s User-Agent program to a URL at the client site to inform it about authorization results.

6) If an API is provided by the server site, functions of the API can be called in subsequent HTTP requests. Only the functions allowed at step 4) can be accessed. The OAAP protocol specifies HTTP message formats to transmit a call and to receive a result of the call.

## V. The Backend

A preliminary name of “Backend” is used in this paper to refer to functions which help to detect, in which server sites a user already have an account to authenticate against.

A Backend could be thought as some kind of a storage or a database, where:

1) Server sites store information that a specific user has an account at a specific server site,
2) A client site can retrieve this information and present a user with a reasonable list of server sites to authenticate against, instead of presenting a fixed list of ‘most popular’ server sites or forcing the user to enter an URL manually.

Possible ways of implementation of the Backend are:

1) A centralized server,
2) A distributed server,
3) A browser plugin or an extension which is installed on an end user’s computer and has access to its local storage.

A centralized server is the simplest solution. However, it has very low level of fault-tolerance. A distributed server is more complex, and it is enough fault-tolerant. Both these solutions can have a problem how to identify a user to associate his data, depending on implementation.

All Backend solutions have a problem of potential violation of user’s privacy. Suppose, an attacker can access information about all accounts of a particular user and

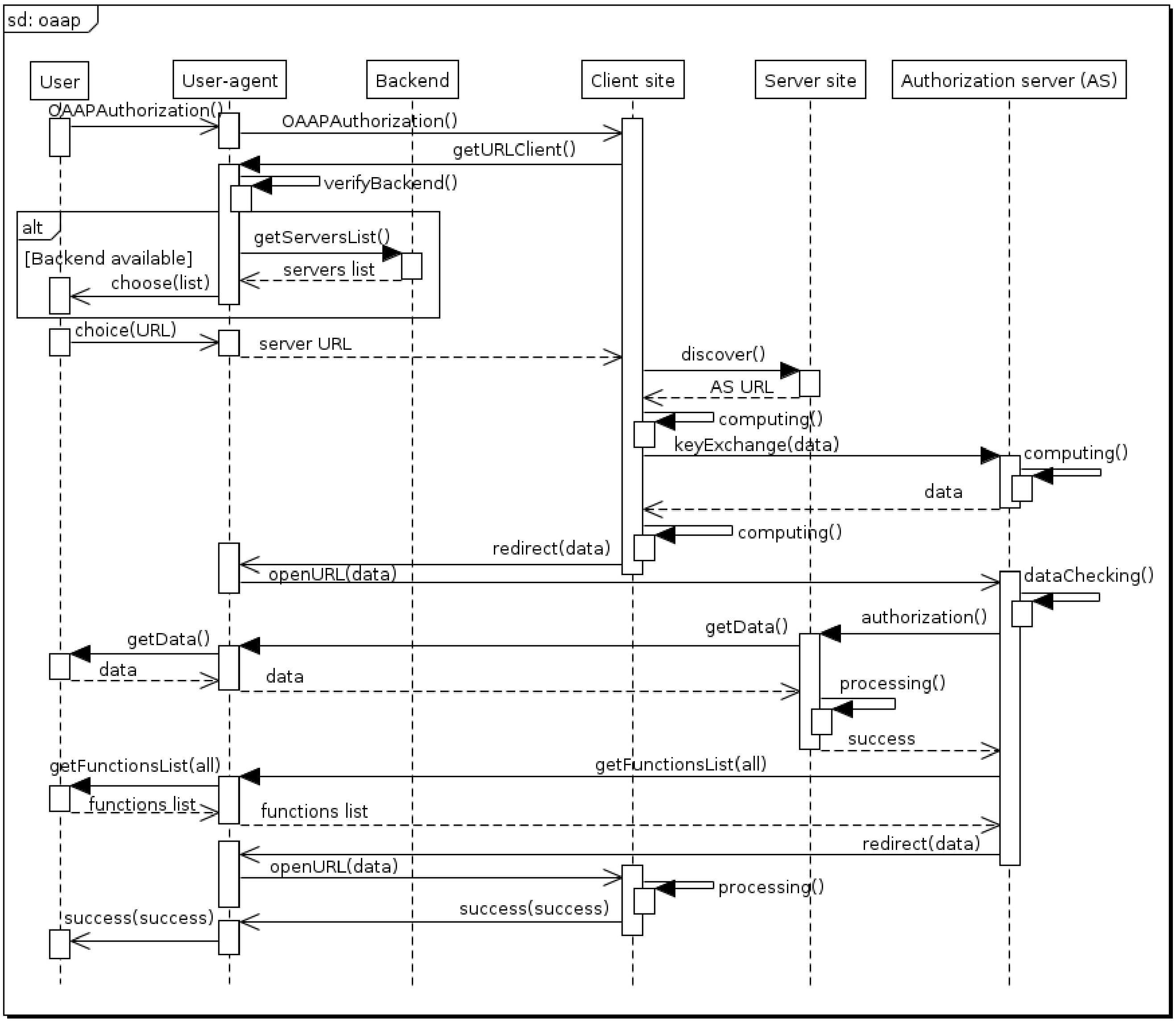

Figure 1. The OAAP sequence diagram.
Processing of errors and exceptions is not shown.

compare it with user's registration information at the attacker's own site.

A solution based on browser plugin can be the safest in regard to this problem. However, it is more difficult to implement, because a special plugin or extension is required for each type of popular web browser.

For now a protocol of a centralized server is specified and implemented (see The OAAP BP below). We suppose that all three approaches will be in use in the future.

## VI. The OAAP BP

The OAAP Backend Protocol (OAAP BP) is a protocol to exchange information with a centralized Backend server. The protocol specifies HTTP messages to store and retrieve information about the existence or an account of a specific user in a specific server site.

Every server site is supposed to store this information at the centralized Backend server at a time when a user passes authorization. Then, a client site can retrieve this information and construct a reasonable list of server sites to allow the user to select from.

The following data items are stored for each account:

1) A URL of the server site,
2) A proposed time of the end of a user's session on the server site,
3) A time when the information was stored by the OAAP BP centralized server.

The OAAP BP requires that a list of server sites must be sorted in such way that the sites with valid open sessions are shown first and the sites with latest authorization time are shown first.

The OAAP BP and its experimental implementation do not use a storage on the centralized server. Instead, all data items are placed in a user's web browser as cookie files [6]. (Requests to the centralized server are nevertheless necessary because this server only can access these cookie files, in accordance with [6]).

A complete specification of OAAP BP is published on the Internet [3].

## VII. An Experimental Implementation

An experimental implementation of the OAAP and OAAP BP protocols has been created and published on the Internet. One can find it here [3]. We invite anyone to evaluate it and to provide a community with a feedback.

## VIII. Conclusion

A new protocol OAAP for Single Sign-on for the Web is proposed, specified and implemented. The protocol is supposed to combine the advantages of OpenID and OAuth protocols and to be free of the disadvantages of ones.

An additional protocol OAAP BP to detect in which server sites a user already has an account is proposed, specified and implemented. Potentially it can be used with existing protocols OpenID and OAuth as well as with the OAAP.

Specifications and implementations of both protocols are in a very experimental stage and will be improved in the future.

Other solutions to detect in which server sites a user already has an account, called Backend solutions, were discussed. These solutions are directions of further works.

We notice a problem of potential violation of user's privacy in connection with Backend solutions. A study of this issue is also a direction of further work.

## References

[1] OpenID Authentication 2.0 – Final.. http://openid.net/specs/openid-authentication-2_0.html Retrieved: Apr 24, 2014

[2] RFC 6749. D. Hardt. The OAuth 2.0 Authorization Framework October 2012.

[3] The OAAP Home Page. http://oaap.oit.cmc.msu.ru/ Retrieved: May 13, 2014

[4] Extensible Resource Identifier (XRI) Resolution Version 2.0. http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.html Retrieved: Apr 24, 2014

[5] RFC 5246. T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. August 2008.

[6] RFC 6265. A. Barth. HTTP State Management Mechanism. April 2011.