Forum CA SSO

CA SSO - OAuth Authentication

17 Mars 2016

ca technologies

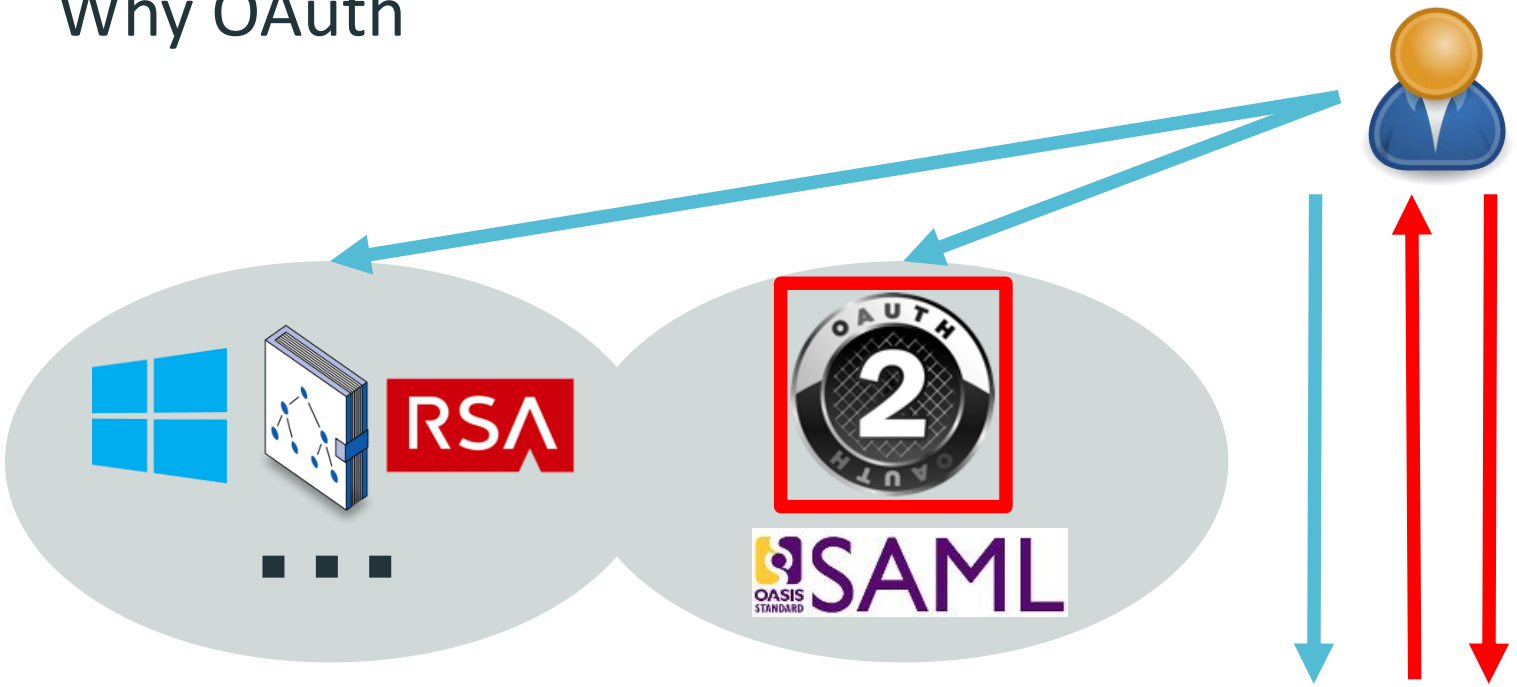
Agenda

- OAuth History
- Why OAuth
- OAuth2 Roles
- General call flow
- Resource Owner Password Credentials Grant
- Authorization Code Grant
- Implicit Grant
- Resource Owner Password Credentials Grant
- OAut token format example
- CA SSO - Supported OAuth grant type
- CA SSO - Authentication Scheme
- CA SSO – OAuth configuration file
- CA SSO – OAuth Authentication sequence
- Demo

OAuth History

- OAuth began in November 2006 for developing the Twitter OpenID implementation.
- The OAuth Core 1.0 final draft was released On December 2007.
- The OAuth 1.0 protocol was published as RFC 5849, an informational Request for Comments, in April 2010.
- The OAuth 2.0 framework was published as RFC 6749, and the Bearer Token Usage as RFC 6750, both standards track Requests for Comments, in October 2012.
- OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol, in April 2015.

Why OAuth



CA SSO

Web based backends

Authentication

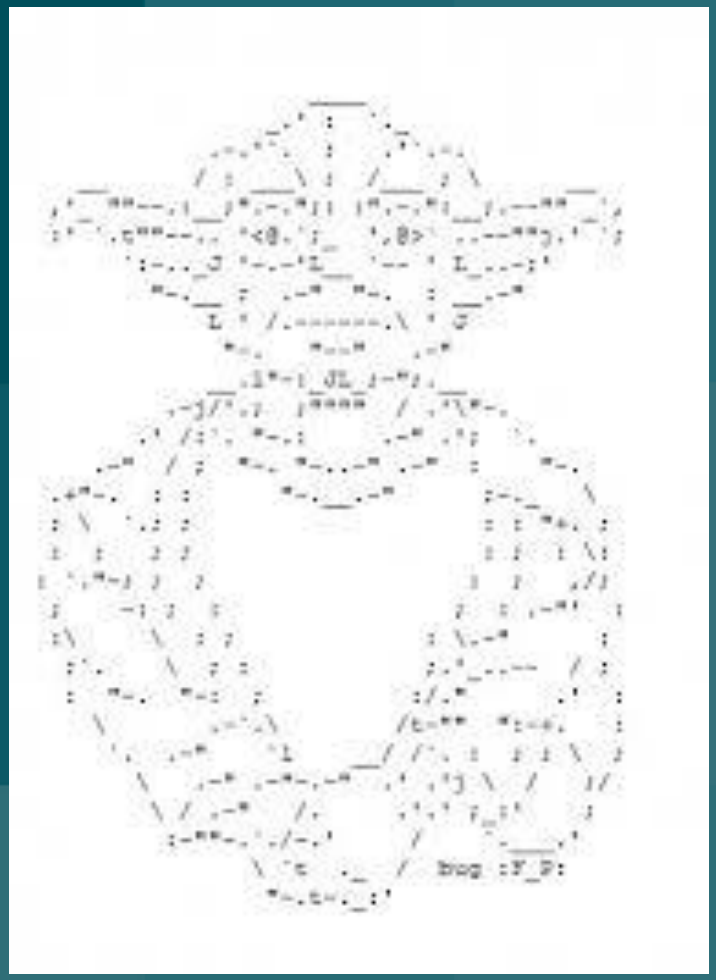
Authorization/

Session Management

ca technologies

RFC 6749 - OAUTH 2.0 Authorization Framework

ASCII art:

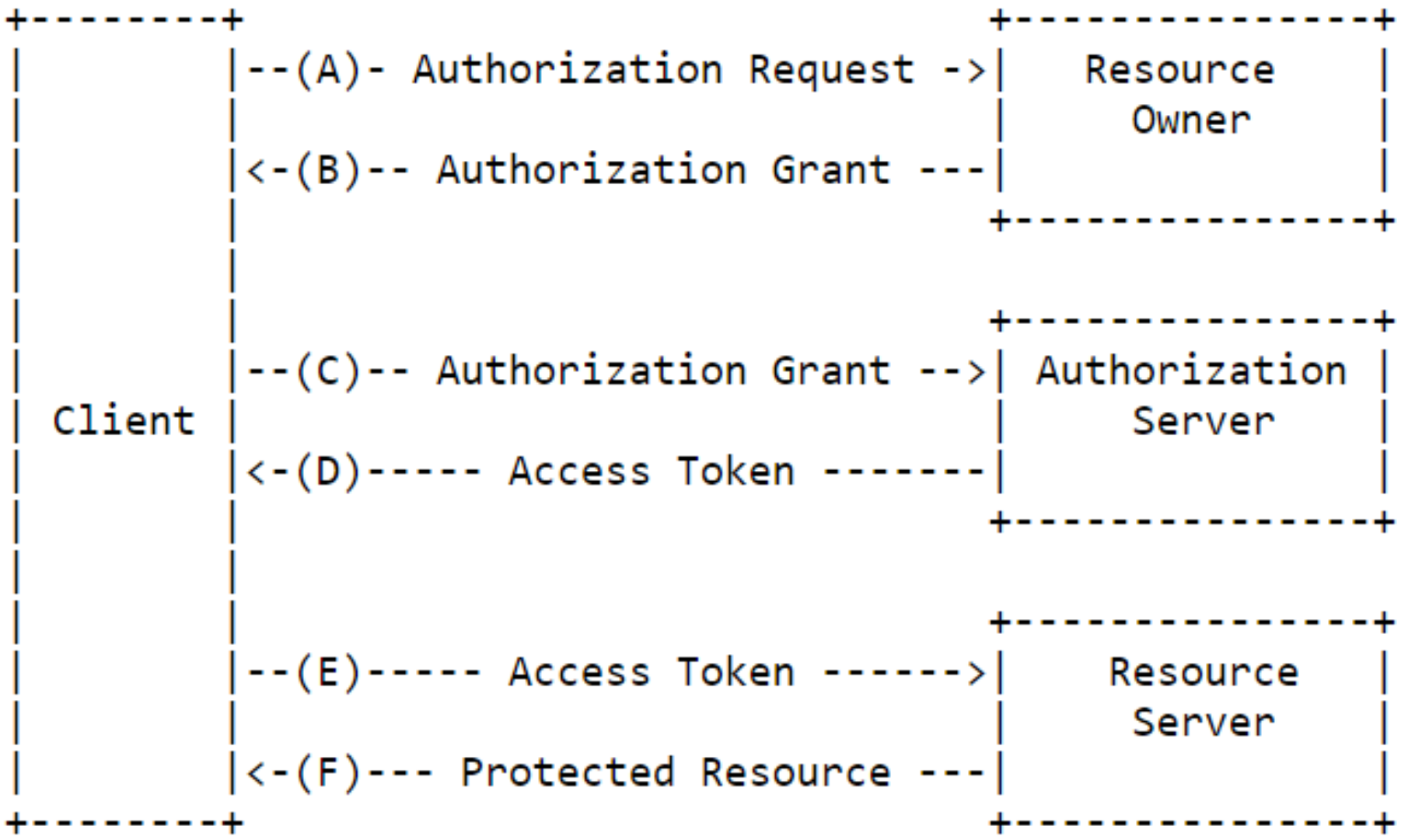


OAuth2 Roles

OAuth defines four roles:

- **Resource owner**
  - An entity capable of granting access to a protected resource.
  - When the resource owner is a person, it is referred to as an end-user.
- **Resource server**
  - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
- **Client**
  - An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).
- **Authorization server**
  - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

General call flow

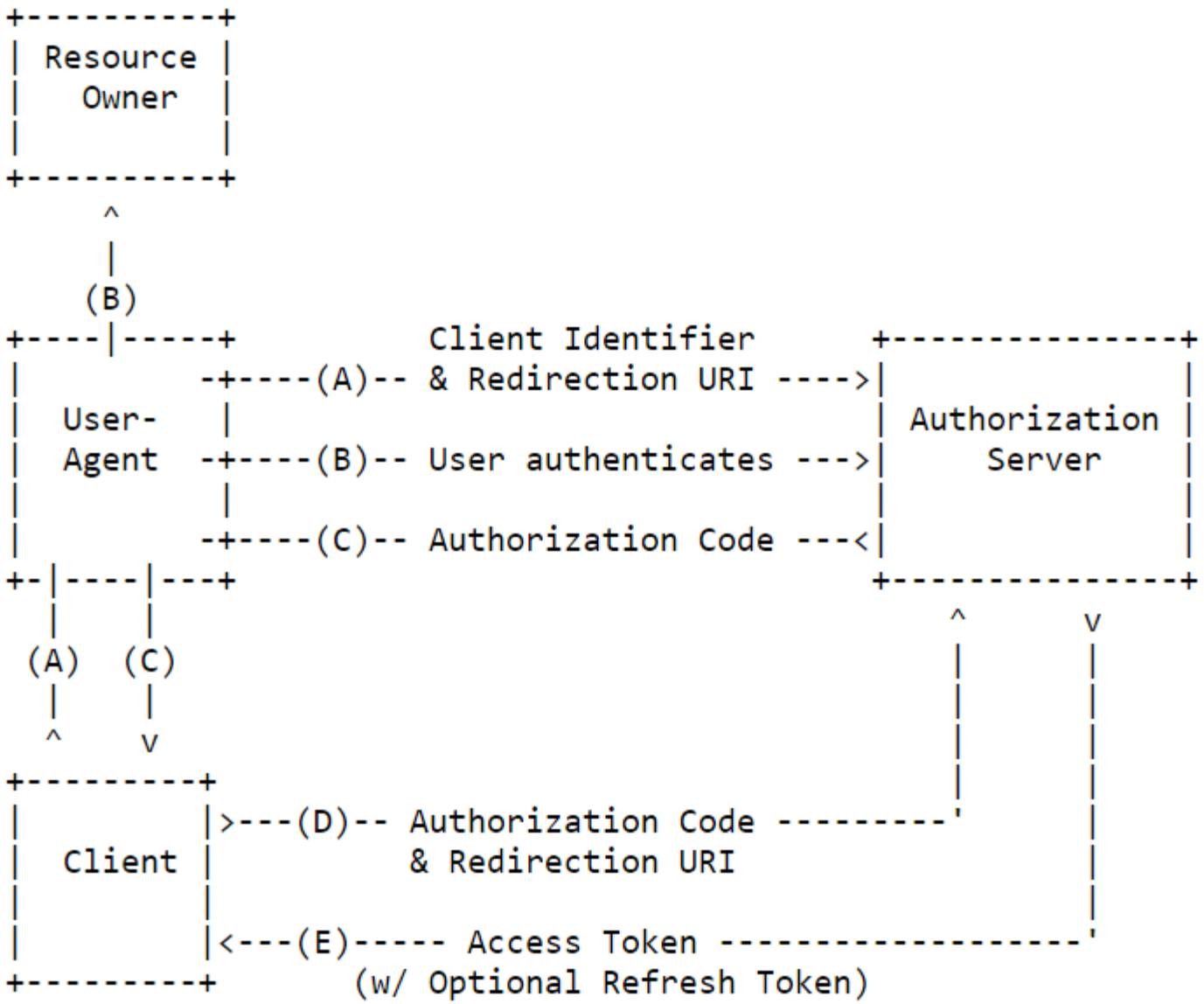


Resource Owner Password Credentials Grant

The flow illustrated in Figure 3 includes the following steps:

A. The client requests an access token by authenticating with the authorization server and presenting an authorization grant.
B. The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token and a refresh token.
C. The client makes a protected resource request to the resource server by presenting the access token.
D. The resource server validates the access token, and if valid, serves the request.
E. Steps (C) and (D) repeat until the access token expires. If the client knows the access token expired, it skips to step (G); otherwise, it makes another protected resource request.
F. Since the access token is invalid, the resource server returns an invalid token error.

Authorization Code Grant



Authorization Code Grant

The flow illustrated in Figure 3 includes the following steps:

A. The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

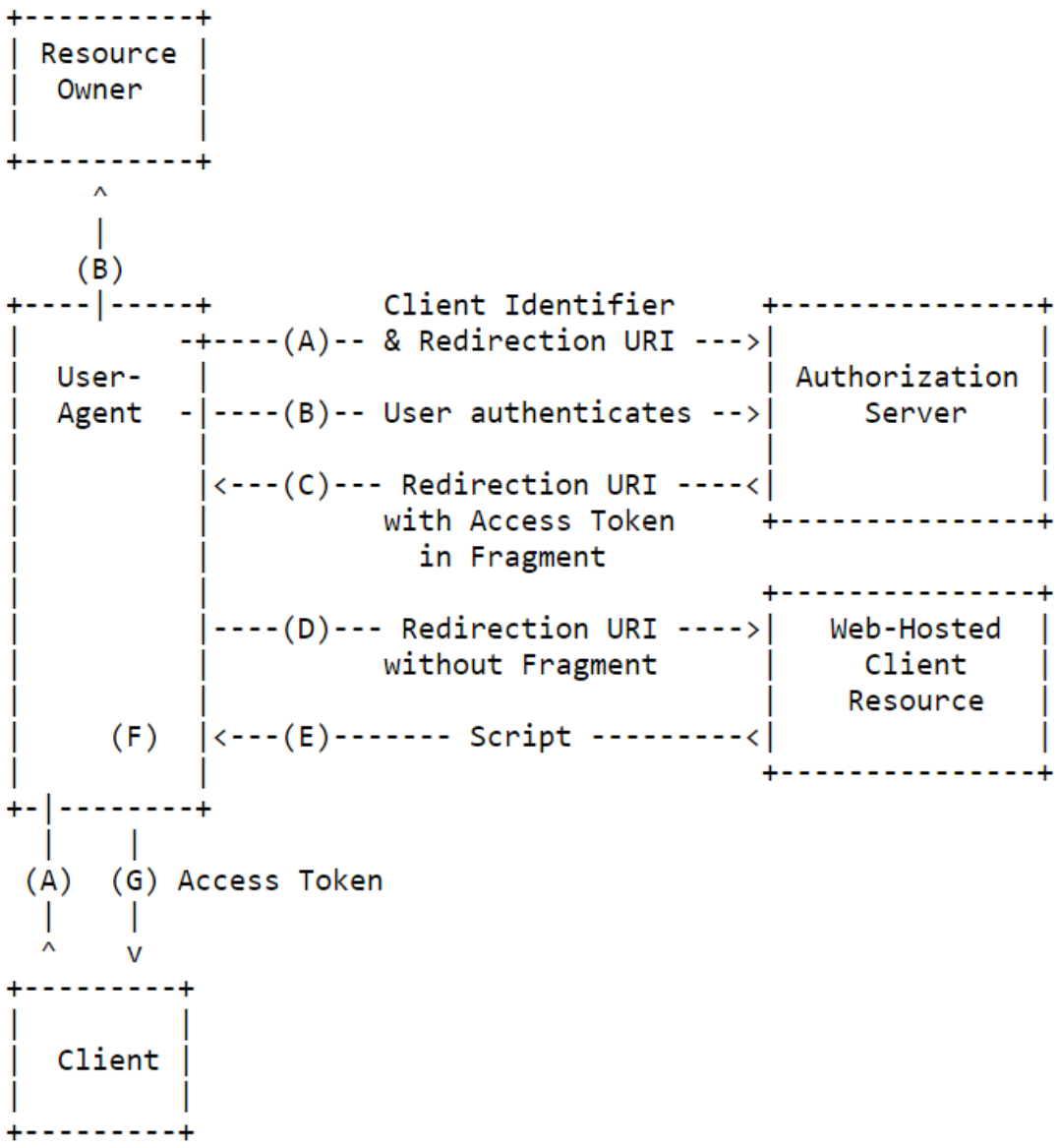
B. The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

C. Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier.

D. The client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. The client includes the redirection URI used to obtain the authorization code for verification.

E. The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client in step (C). If valid, the authorization server responds back with an access token and, optionally, a refresh token.

Implicit Grant



Implicit Code Grant

The flow illustrated in Figure 3 includes the following steps:

A. The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

B. The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

C. Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment.

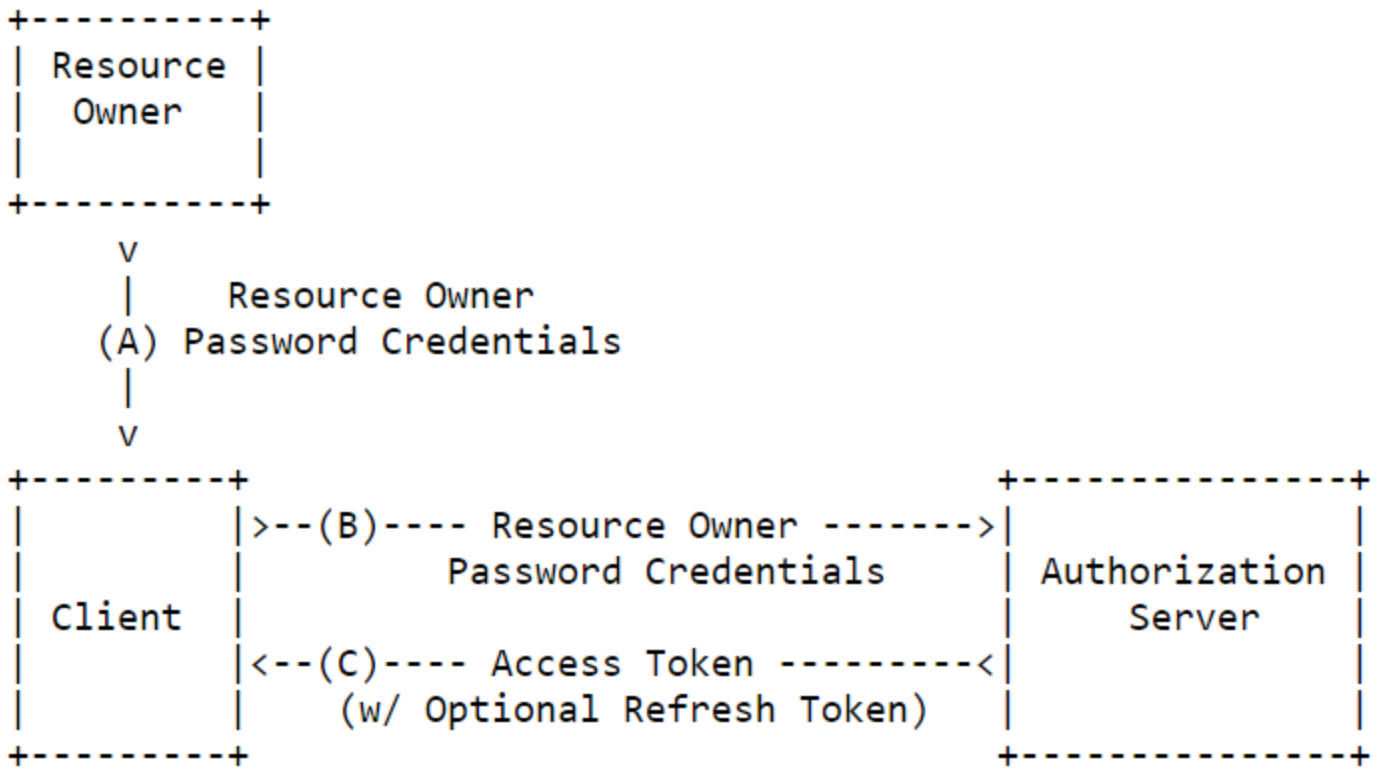
D. The user-agent follows the redirection instructions by making a request to the web-hosted client resource (which does not include the fragment per [RFC2616]). The user-agent retains the fragment information locally.

E. The web-hosted client resource returns a web page (typically an HTML document with an embedded script) capable of accessing thefull redirection URI including the fragment retained by the user-agent, and extracting the access token (and other parameters) contained in the fragment.

F. The user-agent executes the script provided by the web-hosted client resource locally, which extracts the access token.

G. The user-agent passes the access token to the client.

Resource Owner Password Credentials Grant



Resource Owner Password Credentials Grant

The flow illustrated in Figure 3 includes the following steps:

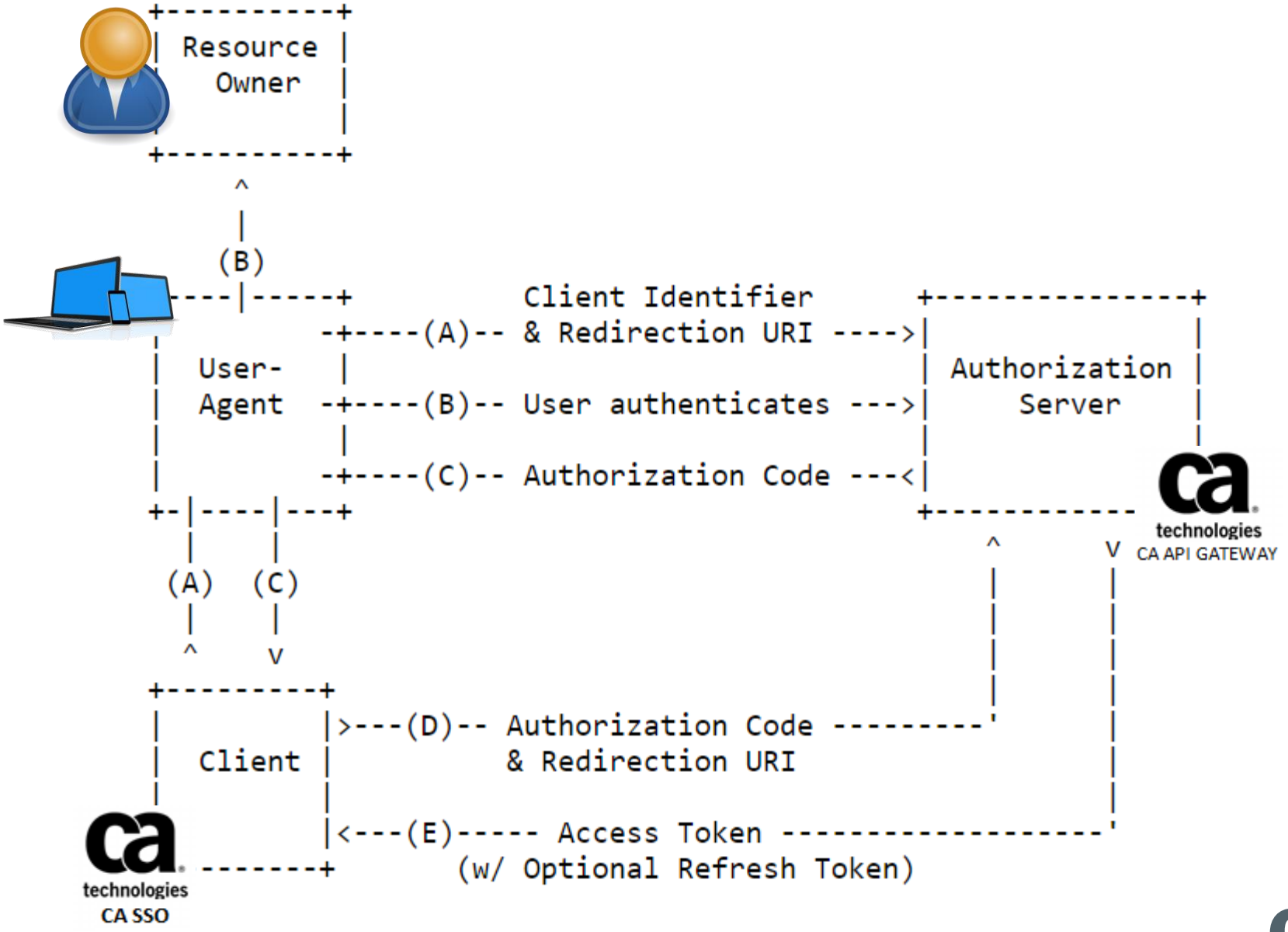
A. The resource owner provides the client with its username and password.
B. The client requests an access token from the authorization server's token endpoint by including the credentials received from the resource owner. When making the request, the client authenticates with the authorization server.
C. The authorization server authenticates the client and validates the resource owner credentials, and if valid, issues an access token.

OAut token format example


```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Content-Length: 201
Date: Mon, 14 Mar 2016 14:37:09 GMT

{
  "access_token" : "ya29.AHES6ZTtm7SuokEB-RGtbBty9IIlNiP9-eNMMQKtXdMP3sfjL1Fc",
  "token_type" : "Bearer",
  "expires_in" : 3600,
  "refresh_token" : "1/HKSmLFXzqP0leUihZp2xUt3-5wkU7Gmu2Os_eBnzw74"
}
```

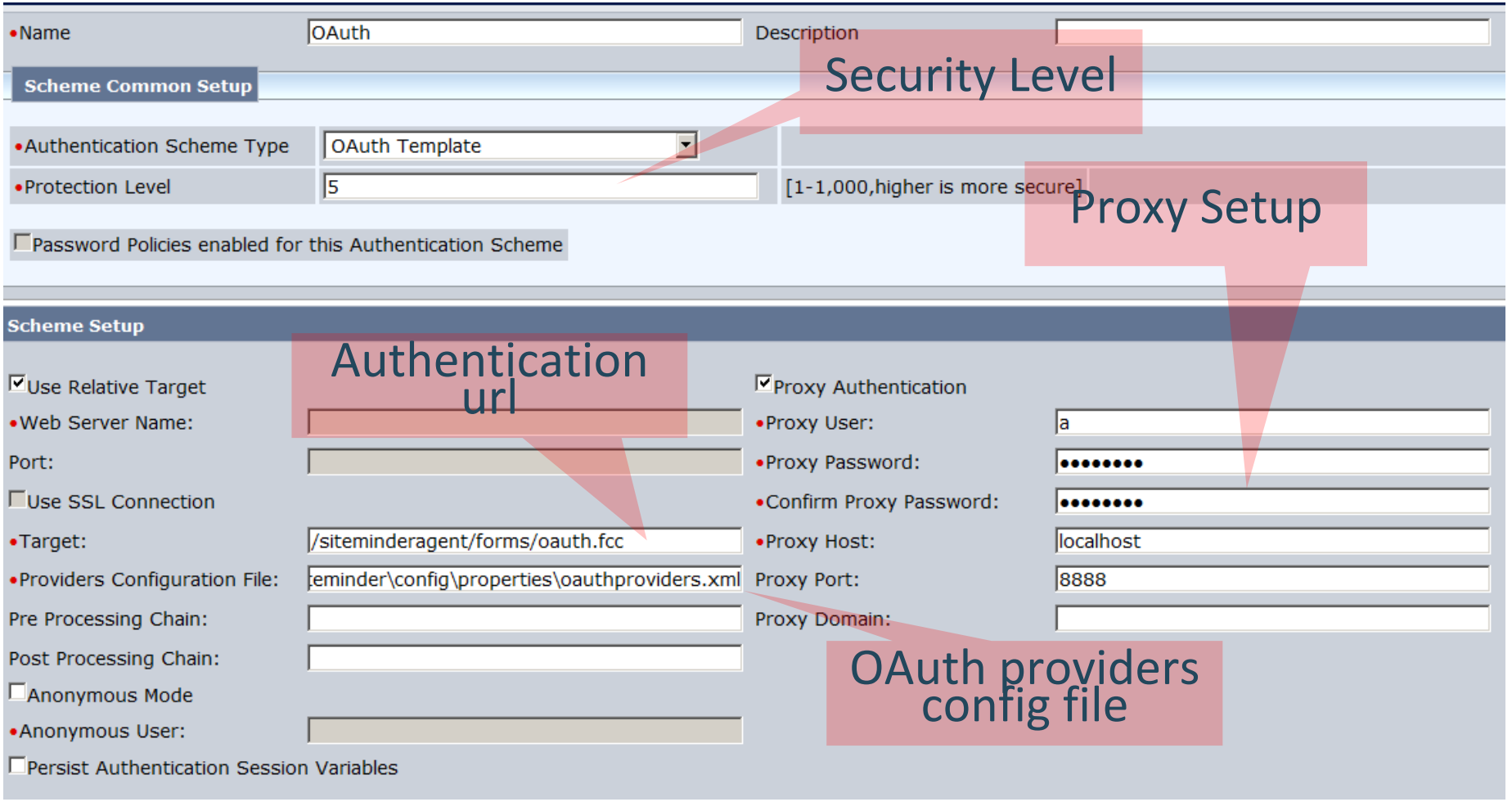
SiteMinder integration

CA SSO - Supported OAuth grant type



technologies

CA SSO - Authentication Scheme



CA SSO – OAuth configuration file

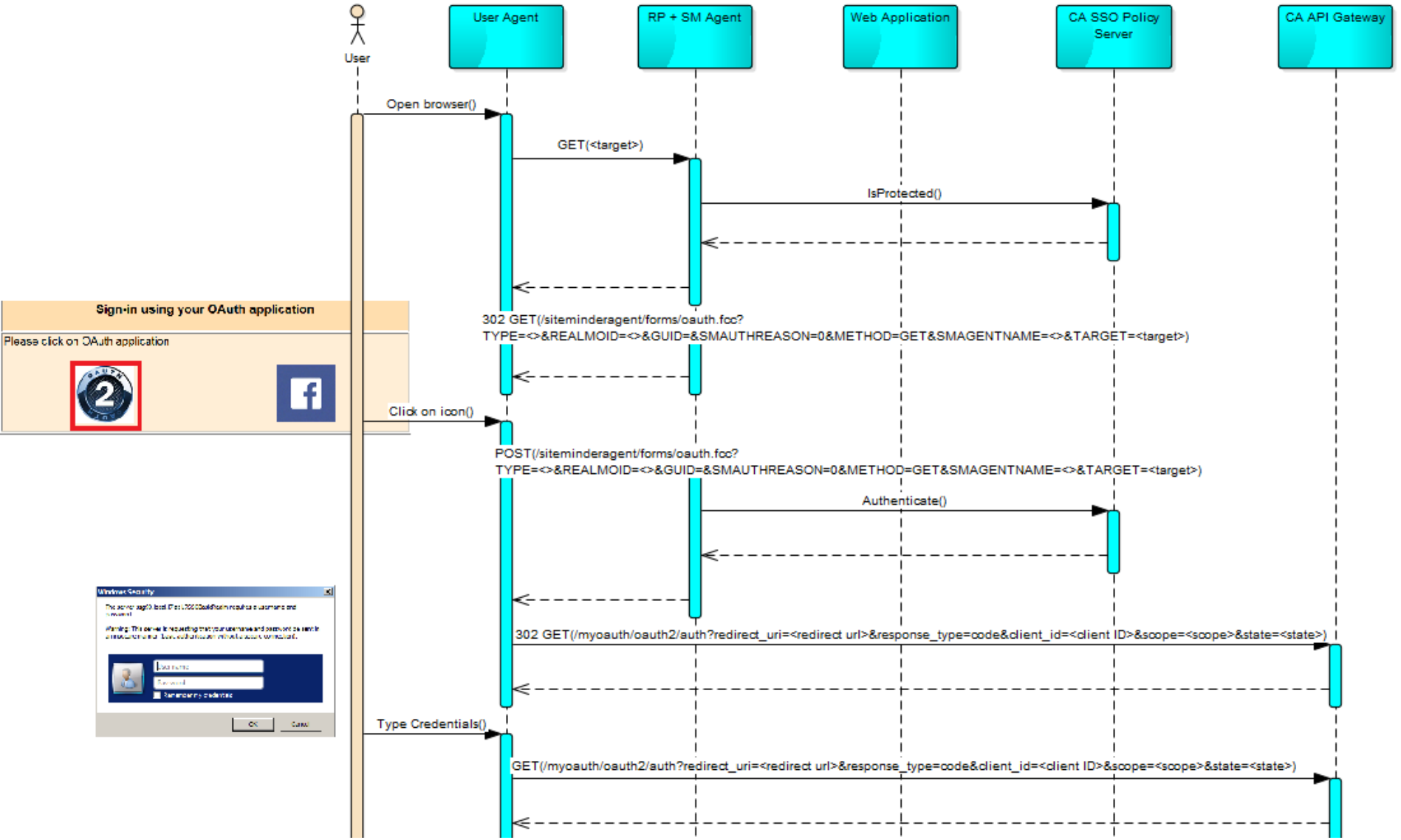
```
<Application appname="MyApp">
    <ApplicationURL>
        <!-- HOSTNAME : WebAgent/SPS host name -->
        http://fed.global.fr:90/siteminderagent/forms/oauthcb.fcc
    </ApplicationURL>
    <ClientID>
        01234567890123456789O
    </ClientID>
    <Secret>
        0123456789
    </Secret>
    <PROVIDERLINK>
        MYOAuth
    </PROVIDERLINK>
    <Scope>
        Read Write
    </Scope>
    <UserInfoURL>
        http://ssg90.local.17:8080/myoauth/oauth2/userinfo
    </UserInfoURL>
    <UserAttribute>
        name
    </UserAttribute>
</Application>
<OAuthProvider providername="MYOAuth">
    <AuthorizationURL>
        http://ssg90.local.17:8080/myoauth/oauth2/auth
    </AuthorizationURL>
    <AccessTokenURL>
        http://ssg90.local.17:8080/myoauth/oauth2/token
    </AccessTokenURL>
</OAuthProvider>
```

oauthproviders.xml

oauth.fcc

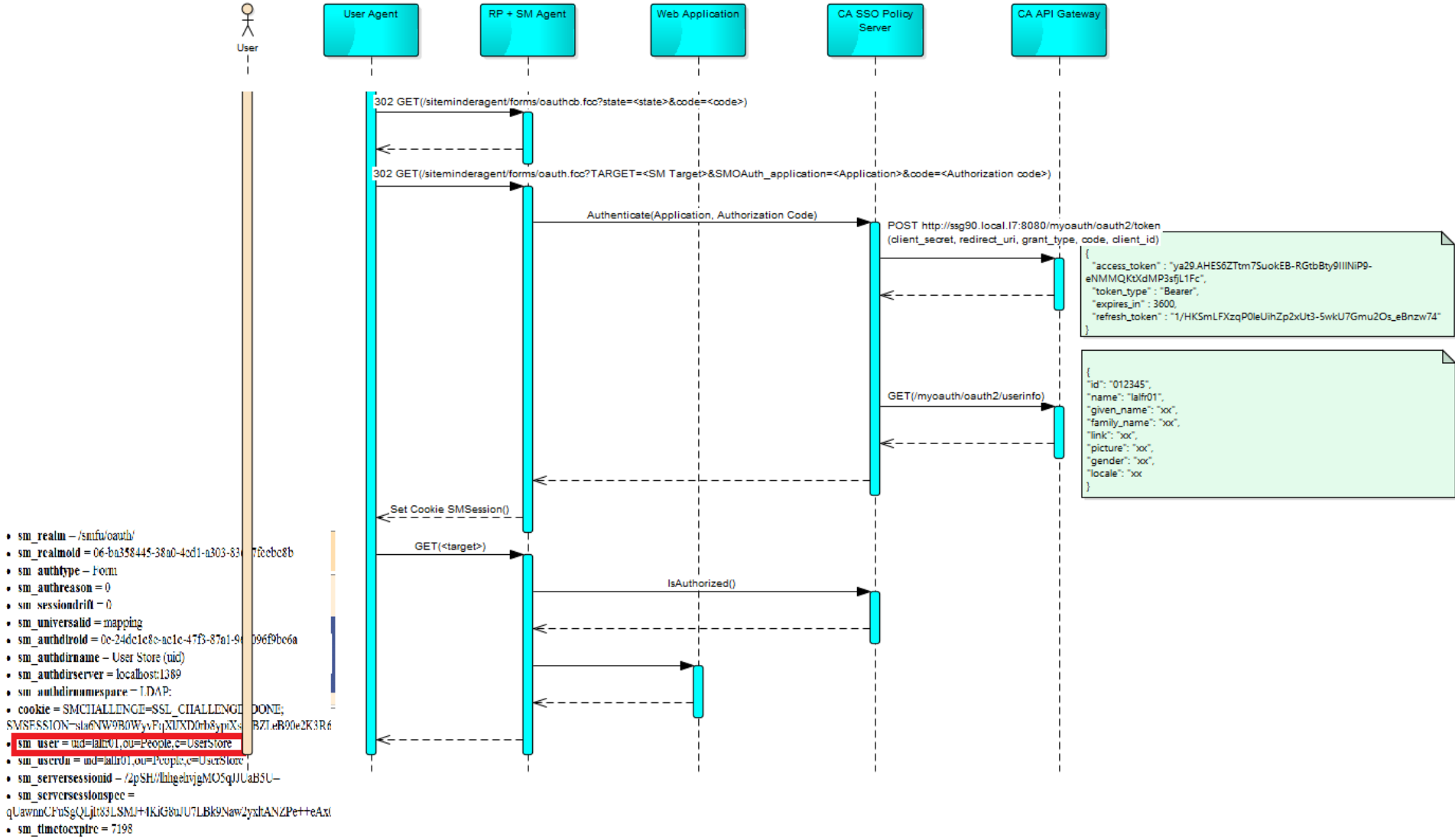
```
var oauth_providers = {

MYOAuth : {
    name : 'MYOAuth',
    image :'',
    apps : 'MyApp'

},
facebook : {
    name : 'Facebook',
    image :'',
    apps : 'testfacebookapp'
}

};
```

CA SSO – OAuth Authentication sequence



ca technologies

CA SSO – OAuth Authentication sequence



ca technologies


Demo