Single Sign-On (SSO)

ProProfs supports Single Sign-On (SSO) which will allow you to authenticate your end users using your authentication service, such as your web application's login. Once verified, your end users can then view your ProProfs site. However, if your end users navigate directly to your ProProfs website without first authenticating, they would be redirected to your login page.

Single Sign-On (SSO) works as a "shared secret" between ProProfs and your local application. When Single Sign-On is enabled, end users are redirected from ProProfs to a login page located on your server. End users log in to your web application just like they usually would. Once successfully logged in, a script will execute and tell ProProfs that the end user is authentic. ProProfs then grants access for the end user to view your ProProfs site.

Single Sign-On (SSO)

How Single Sign-On (SSO) Works

A shared API key between your local authenticating mechanism and ProProfs is the basis of Single Sign-On (SSO). This API key is used to generate a hash used by ProProfs to ensure that end users have permission to visit your ProProfs site. Below is the outline of some used cases.

End users go directly to your ProProfs site

1. Your end user attempts to visit your ProProfs site when Private Site is enabled (either by directly going to the URL, e.g., mysite.helpdocsonline.com or by using a bookmark).

2. ProProfs will check the end user's browser session to determine whether the end user has already been granted access to your ProProfs site.
3. If access was not previously granted, ProProfs redirects the end user to your login page, located on your server (such as http://ourdomain.com/login).
4. The end user logs in to your web application or authentication system in the usual way.
5. When the login is successful, a script runs on your web application. This script passes to ProProfs your site_url, return_page, and md5 API key.
6. ProProfs will match the API key to make sure it is authentic.
7. Then your end user is redirected back to your ProProfs site, and a browser session is established. The user can then browse your ProProfs site. Once the end user logs out or closes the browser window, the end user will be required to reauthenticate as described above.

End-user logs in directly to your web application (skips the routing step)

You can also set up this feature, so your end users aren't redirected back to your login page when they visit your ProProfs site. When your end users log in to your web application, you can authenticate them to establish a session with ProProfs immediately. Then if your end users visit your ProProfs site, they will **not** be redirected and can view your site directly.

1. Your end users log in to your web application.
2. When the login is successful, the SSO script logs out.
3. The end user is authenticated, a session is established in ProProfs (while the end user is still on your site).
4. If the end user clicks a link or goes directly to your ProProfs site, they will be able to browse your ProProfs site without having to authenticate.

Parameters

The remote authorization URL is `http://helpdocsonline.com /access/remote/.` The full redirect URL, with parameters, would look something like this:
`https://www.helpdocsonline.com/access/remote/?api_key=<MD5(api_key)>&site=https://yoursite.helpdocsonline.com&return_page=/pageURL&site_access=45,34`
When you redirect your end users back to ProProfs, you should pass the following parameters:

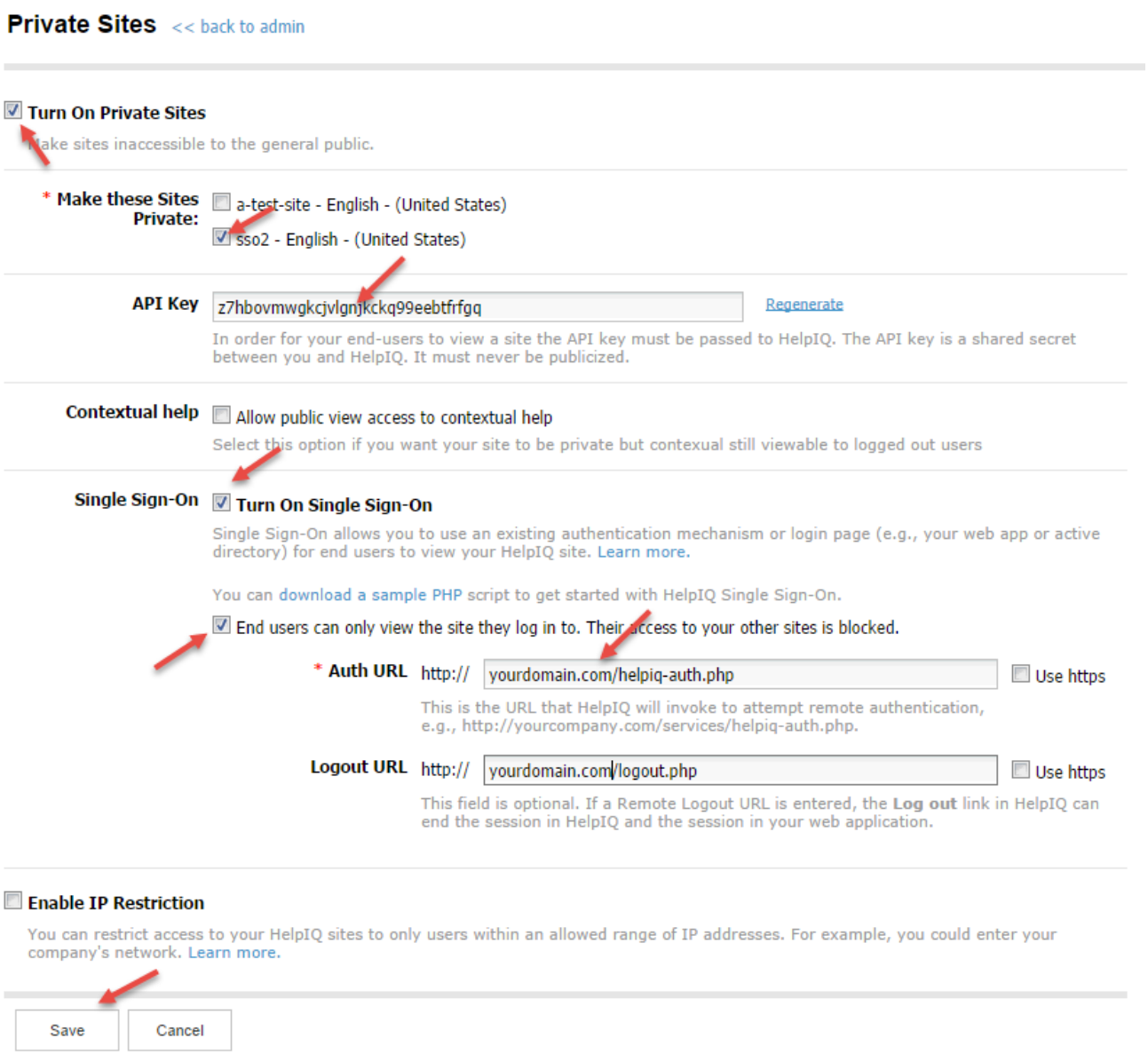
1. `api_key` (required) - The hash-encoded API key is the shared secret between ProProfs and your local website. You can access it from *Settings > Private Sites*.

2. `site` (required) -It is the URL of your ProProfs site, e.g., mysite.helpdocsonline.com
3. `return_page` (optional) - You can also pass the page URL. For example, if an end user visits a specific ProProfs page (such as http://mysite.helpdocsonline.com/pagename) and is redirected to your web application to log in, you can return them to that particular page after they are redirected from your login page to ProProfs.
4. `site_access` (optional) - Specify the ID of the sites that the user is allowed to view. For example, let's say you have ten different websites, but you only want the user to be able to access two of them. You can specify the IDs of those websites, and the end user will just be able to view those sites. Be mindful of separating multiple websites with commas.

Usage of the above parameters is further defined in the sample script.

How to Enable Single Sign-On

1. Log in to your account (as an Administrator) and click **Settings**.
2. Click **Private Sites**.
   The private sites page will appear.



3. Check **Turn On Private Sites**.
4. Select the sites you want to make private.
5. Then check **Turn On Single Sign-On**.

The following settings also appear.

- **API Key** - For SSO to work, the API key must be passed to ProProfs. The API is a shared secret between you and ProProfs. It must never be publicized. Copy this API key to add to your SSO script. You can also change the API key at any time by clicking Regenerate. It will generate a new API key and invalidate the old one.
- **End users can only view the site they log in. They do not have access to your other websites.** Having this checked means if a user logs in from site1 and then went directly to site2, they will be redirected back to site1.

  NOTE: you can also use the site_access parameter to define which sites the end user can

access.

- **Auth URL -** This is the URL that a user is redirected to in case of an attempt to view a private site page, but there's no valid ProProfs viewer session yet. In this event, we always pass the page and website that was being accessed using query parameters *site* and *return_page*. It is handy because the tendency is to redirect the customer back to the previous page after being granted access to your scripts.
- **Logout URL -** This field is optional. If a Logout URL is entered, the **Logout** link in the help site ends the viewer session on ProProfs *and* then redirects the user to this URL.
- **Save -** To save the settings.
- **Cancel -** Click Cancel to return to the Settings page without saving these settings.

Sample SSO Script

We have provided a sample PHP implementation, which you can use as a guide when you incorporate SSO feature into your authentication system. You can download these PHP scripts containing comments at https://github.com/proprofs-account/sso-php. You can also download an ASP.NET version from https://github.com/proprofs-account/sso-php.

How the sample script works

The sample SSO scripts are at work whenever a user without a valid ProProfs viewer-only session visits a page on an SSO enabled private site. The script will do a check to see if the user is already logged into your web app and will allow the user to view your ProProfs site or redirect them to your login page. The redirects are browser-based and do not require ProProfs to access your database, network, or authentication system directly.

In a nutshell, what the system is doing is triggering a remote URL on our end along with your private API key to initiate a valid viewer-only session for your help site. This sample implementation assumes that if a user logs in, you want to give that user access to the help site. Your developer can be creative and alter this criterion based on how you do your authentication.

- First, let's take a look at ***ProProfs-auth*.php**

  This file is merely initiating an object of class **ProProfs_SSO_Support.** You can set your site parameters in this file. It calls the **do_ProProfs_authorization()** method which does several things in this demo app. It's explained in detail in the class file.

```
require_once('lib/ProProfs_SSO_Support.php');
//enter your API key here
$ProProfs_api_key = '9d1e2693fe4fc477cf26bc0df3372985';
```

```
//enter your site URL here
$ProProfs_site_url = 'mysite.helpdocsonline.com';
// parameter site_access: this defines which sites are allowed, separate multiple sites
by comma
// If no value is set and the parameter is not included then we allow access to all
sites.
// If the parameter is set, then we only allow access to whatever sites are included
// If the user tries to go to a site they do not have permission we just take them to a
site they do have permission
$ProProfs_site_access = '';
$ProProfs_sso_support = new ProProfs_SSO_Support($ProProfs_api_key, $ProProfs_site_url);
$ProProfs_sso_support->do_ProProfs_authorization($ProProfs_site_access);
```

- Next, lets look at ***lib/ProProfs_SSO_Support.php***

  This file contains the ProProfs_SSO_Support class. This object is responsible for checking whether a user is logged in to your web application, and controls the redirection to URLs that are needed for different use cases.

  The methods are:

  - **ProProfs_check_local_session()** assumes you identify that a user is logged in to your app if there's a **user_id** key set in your PHP $_SESSION variable.

```
// Upon login of your application or website, a session is established for the user.
// This code will check the users session to determine if they are logged in.
// You can replace 'user_id' with whatever you want such as username, email, etc.
// All the system is doing here is checking to see if there is a value. If there is no
value, user requires to log in.
// If there is a value, pass the site parameters to
http://www.helpdocsonline.com/access/remote/ and establish a session on ProProfs.
public function ProProfs_check_local_session() {
return isset($_SESSION['user_id']) && !empty($_SESSION['user_id']);
}
```

  - **ProProfs_destroy_local_session()** which assumes that unsetting the user_id key in PHP $_SESSION variable terminates your web app user's session.

```
//please destroy your local session data here
public function ProProfs_destroy_local_session() {
unset($_SESSION['user_id']);
}
```

  - **logout_ProProfs()** terminates the web app user session and then logs out the user. Then, it terminates the ProProfs session. After the session is terminated in ProProfs, behind the scene, it redirects to your Auth URL but passes as parameter *action* which is sets to action.

```
//logout the end-user from ProProfs, and it will redirect to your remote auth url
public function logout_ProProfs(){
```

```
$this->ProProfs_destroy_local_session();
$logout_url = $this->ProProfs_remote_url.'logout/?site='.$this->custom_ProProfs_site;
header('location:'.$logout_url);
exit;
}
```

- **do_ProProfs_authorization()** does the heavy work and runs everytime a page is accessed on a private site but no valid ProProfs session is available. This is the case in our sample implementation as it's being called in the Auth URL.

```
public function do_ProProfs_authorization($site_access = '') {
  //If Remote logout URL is entered in ProProfs the 'log-out' link can destroy the
end-users session in ProProfs and the session on your web application.
  $action = isset($_REQUEST['action']) ? (string)$_REQUEST['action'] : 'login';
  $redirect_url = $this->default_login_url;
  //When the Remote logout URL is empty, end-user logged out from ProProfs site,
  //it will pass the logged_out parameter to tell customer's web app don't to give the
end-user access again, just redirect to local login page
  $logged_out = isset($_REQUEST['logged_out']) ? $_REQUEST['logged_out'] : false;
  if ('logout' == $action || 'custom_logout' == $action) {
   $this->ProProfs_destroy_local_session();
   $redirect_url = $this->default_login_url;
  } else {
   //your ProProfs site URL
   $site = (string)$_REQUEST['site'];
   //return_page is passed by ProProfs, it will redirect the end-user to a specific
page ProProfs
   $return_page = (string)$_REQUEST['return_page'];
   // please check your end-user has logged in here
   $url_params = 'site='.$site.'&return_page='.$return_page;
   if (!$logged_out && $this->ProProfs_check_local_session()) {
    // if the end-user has logged in the customer's website/web application, call
ProProfs to estbalish a session
    $redirect_url =
$this->ProProfs_remote_url.'?hash='.md5($this->ProProfs_api_key).'&'.$url_params;
    if(!empty($site_access)) {
     $redirect_url .= '&site_access='.$site_access;
    }
   } else {
    // the end-user does not log in, redirect to error/log in page
    if (isset($_REQUEST['contextual']) && $_REQUEST['contextual']) {
     //if the refer page is a contextual help(lightbox/tooltip), redirect to show
permission limit
     $redirect_url =
$this->ProProfs_remote_url.'permission_limit/?login=false&'.$url_params;
    } else {
     //redirect to your local application login page
     $redirect_url = $this->default_login_url.'?'.$url_params;
    }
   }
  }
  header('location:'.$redirect_url);
}
}
```

- Now let's look at ***login.php*** closely.
  - In this section:

```
 session_start();
$current_url = explode('?', $_SERVER['REQUEST_URI']);
$current_url = explode('/', $current_url[0]);
array_pop($current_url);
$current_url = 'http' . ( !empty($_SERVER['HTTPS']) ? 's' : '') .'://' .
$_SERVER['HTTP_HOST'].implode('/', $current_url);
$login_url = $current_url.'/login.php';
if (isset($_REQUEST['submit'])) {
  $username = trim($_REQUEST['username']);
  $password = trim($_REQUEST['password']);
  if('demo'   == $username && 'demo!'  == $password){
   //establish the local loggedin session
   $_SESSION['user_id'] = 1;
   if (!empty($_REQUEST['site'])) {
    //if site parameters is not empty, redirect to remote log in URL, to establish the
ProProfs session
    header('location:'.$current_url.'/ProProfs-
auth.php?site='.$_REQUEST['site'].'&return_page='.$_REQUEST['return_page']);
    exit;
   } else {
    //redirect to local app
    header('location:'.$current_url.'/test.php');
    exit;
   }
  }
}
```

  We made it so that if the user logs in with username: demo and password: demo!, they are assumed authenticated. We set $_SESSION['user_id'] = 1. Your authentication system will definitely be much more complex than this.

  - Also, in this file, we have the behavior that if the user successfully logs in, they will be redirected to a test.php file in your web app unless there's a URL parameter *site* and an optional *return_page* which will be used to redirect back the user.

- The ***test.php*** file just shows a logout link which points to the logout.php script. It demonstrates how to effectively logout the web app user and then logs out the ProProfs session too.
- The ***logout.php*** script mainly calls the **logout_ProProfs()** method of the SSO class object which does the following:
  - Destroy the session for the web app user.

    Note: We were assuming that you are merely using something like a $_SESSION['user_id'] to indicate whether a user is logged in or not. Your implementation will be much more complicated than this, and your developer needs to update it based on how your authentication system works.

- Then terminate the viewer session in ProProfs.
- Then we redirect back to the **Auth URL,** but this time with query parameter *'action'* being set to log out which is caught by our sample script and just redirects the user to the local login page we have in the SSO class file.

If you need a script in another language, such as Java, you can follow the above examples and create your own. Please feel free to send us any samples.

Login URL

Since you will no longer be able to visit your ProProfs site directly, you can log in at http://www.proprofs.com/knowledgebase/login/.

> Note: Single Sign-On is for end users only. You will still need to log in to ProProfs as you usually do.

Zendesk Authentication

You can set up ProProfs so that only the logged-in Zendesk users can access and view your documentation. If an end user navigates directly to your ProProfs site without first logging into Zendesk, the end user would not be able to view your ProProfs documentation. Zendesk handles the authentication of your end users.

When the end user clicks on the **Documentation** tab in Zendesk, Zendesk will pass the API key to ProProfs and authenticate your end user so the end user can view your ProProfs site. The API code will automatically be included in the Zendesk Documentation-tab code snippet which ProProfs provides.

To use Zendesk authentication, first set your site as private (as defined above). Then follow these steps for adding a documentation tab in Zendesk.

SSO Alternatives

If your company does not have a developer or resources to hook-up SSO using the above method, you can use JavaScript or even pass the API key by URL. However, the above SSO method is recommended over JavaScript or URL because your API key is not exposed.

JavaScript Method:

For this method to work, add this script to any of your web pages. When the script loads, the API key is shared via JavaScript and the user will have access to view your ProProfs site.

```
<script type="text/javascript"
src="http://yoursite.helpdocsonline.com/remote_auth/64ps4xoq2ut33mzzebrb8zfqmj8vzdj6?embed=t
rue"></script> //replace with your ProProfs URL + API key
```

Replace API with your API key.

URL Method:

For this method to work, add the URL+API key to a hyperlink on your referring web page. Then when the user clicks on the hyperlink, the API key will be passed and the user will be granted access to view your ProProfs site.

```
<a
href="http://yoursite.helpdocsonline.com/64ps4xoq2ut33mzzebrb8zfqmj8vzdj6?embed=true">Link
to your help site</a> //replace link your ProProfs URL + API key
```

Related Articles:

Private Sites
IP Restrictions
JWT Single Sign-On