MobileIron Access Cookbook

Access with .NET based SP and iOS application

January 23, 2018

Contents

Overview

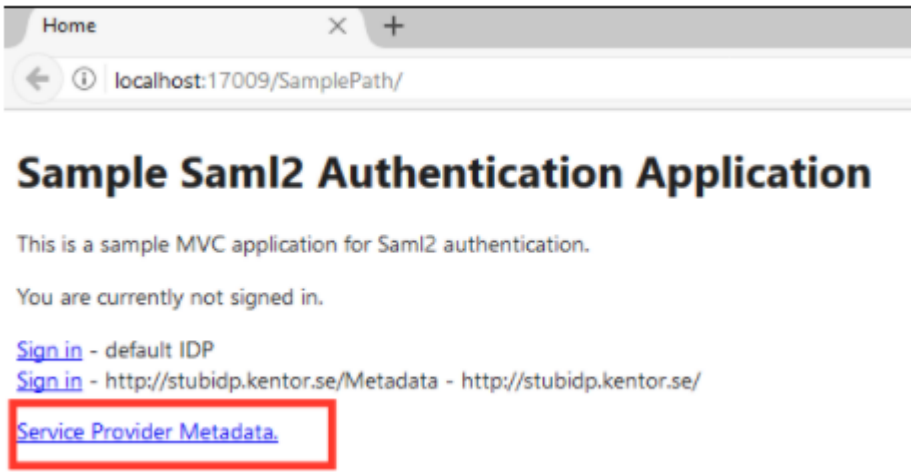
This cookbook lets you configure Access SSO with the custom .NET based service provider. As an example of .NET based service provider, KentorIT's Authservice's sample application is used in this cookbook. Kentor Authservice modules can also be integrated with an existing application to add the SAML capabilities. Kentor Authservice code can be cloned from https://github.com/Sustainsys/Saml2. The SampleApplication is used to explain the Access SSO configuration in the following sections.

Disclaimer:
This cookbook is informational to help with the setup flow and actual screenshots. The steps might vary in your deployment scenario due to changes in versions.

Prerequisites

- **Metadata files for SAML2 Authentication Application**

  Download the service provider metadata from Sample Application home screen.



Integrate Kentor AuthServices with an existing .NET application

Kentor has four modules. Depending on the type of .NET application, the modules must be added.

Prerequisites

- Ensure that you read the following documentation:

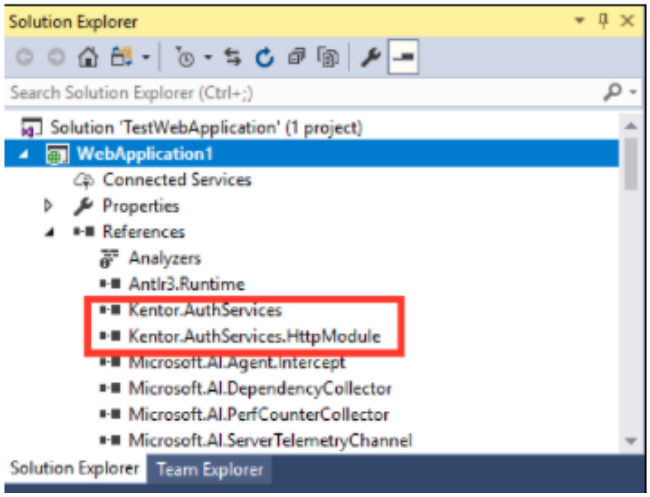
https://github.com/Sustainsys/Saml2#using

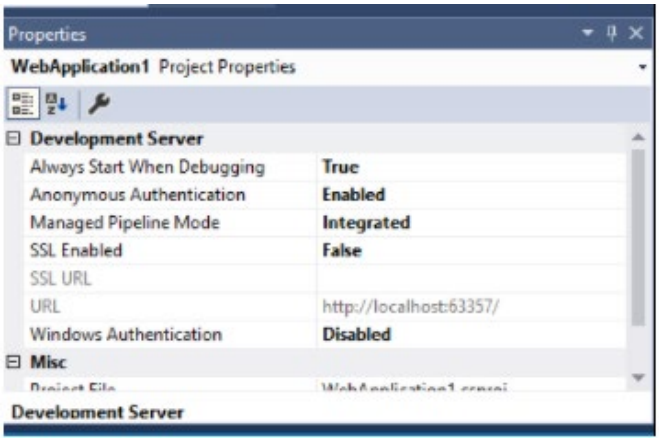
Step: 1

Open **Visual Studio** > **Tools** > **NuGet Package Manager** > **Package Manager Console** and run the following commands:

i) *Install-Package Kentor.AuthServices –Version 0.21.2*
ii) *Install-Package Kentor.AuthServices.HttpModule –Version 0.21.2*

These commands add a reference to the Kentor's core and HTTP module. The newly added modules can be viewed in the application properties.





Step: 2

1. Open the Web.config of the application. In the **Configuration** section add the following sections:

   <configSections>

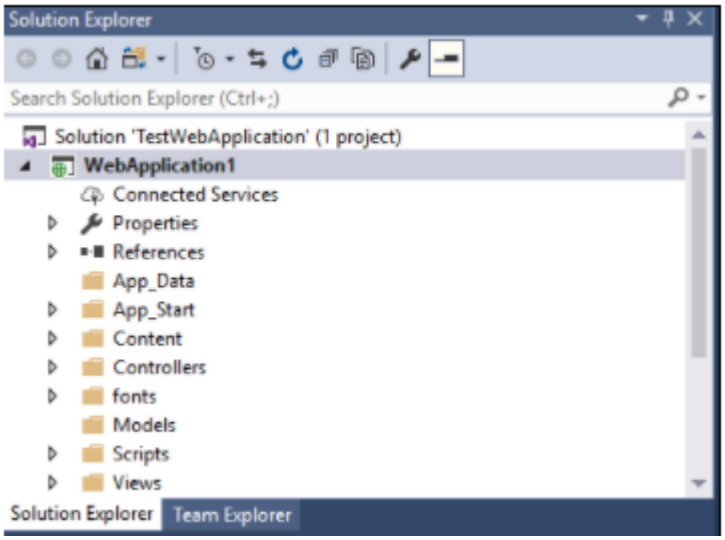
   <section name="system.identityModel"
   type="System.IdentityModel.Configuration.SystemIdentityModelSection,
   System.IdentityModel, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=B77A5C561934E089" />
   <section name="system.identityModel.services"
   type="System.IdentityModel.Services.Configuration.SystemIdentityModelServicesSe
   ction, System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=B77A5C561934E089" />
   <section name="kentor.authServices"
   type="Kentor.AuthServices.Configuration.KentorAuthServicesSection,
   Kentor.AuthServices" />

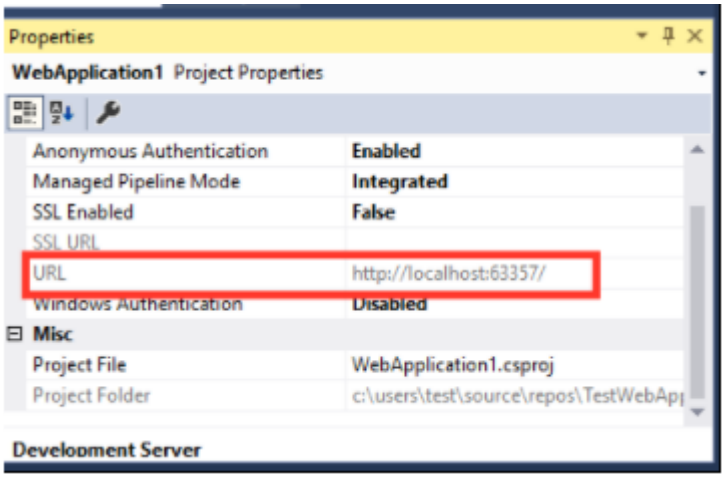
   </**configSections**>

2. In **system.webserver** add the following modules configuration. This includes the necessary session and Saml Authentication modules.

   <system.webServer>
   <validation validateIntegratedModeConfiguration="false" />
   <modules>
   <add name="SessionAuthenticationModule"
   type="System.IdentityModel.Services.SessionAuthenticationModule,
   System.IdentityModel.Services, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089" />
   <add name="Saml2AuthenticationModule"
   type="Kentor.AuthServices.HttpModule.Saml2AuthenticationModule,
   Kentor.AuthServices.HttpModule" />
   </modules>
   </system.webServer>

3. Fetch the URL from the application properties as shown below. This URL is used for **kentor. authservice** section's **entityId** and **returnUrl**.





4. Create a section for **kentor.authservices** and **system.identityModel.services**

```
<kentor.authServices entityId="http://localhost:63357/AuthServices"
returnUrl="http://localhost:63357/"
discoveryServiceUrl="http://localhost:52071/DiscoveryService"
                    authenticateRequestSigningBehavior="Always">
   <nameIdPolicy allowCreate="true" format="Persistent" />
   <requestedAuthnContext classRef="Password" comparison="Minimum" />
   <identityProviders>
     <add entityId="http://stubidp.kentor.se/Metadata"
signOnUrl="http://stubidp.kentor.se/" allowUnsolicitedAuthnResponse="true"
binding="HttpRedirect">
       <signingCertificate fileName="~/App_Data/Kentor.AuthServices.StubIdp.cer"
/>
     </add>
  </identityProviders>
   <federations>
     <add metadataLocation="http://localhost:52071/Federation"
allowUnsolicitedAuthnResponse="true" />
   </federations>
   <serviceCertificates>
```

```
        <add fileName="~/App_Data/Kentor.AuthServices.Tests.pfx" />
      </serviceCertificates>
    </kentor.authServices>
  <system.identityModel.services>
    <federationConfiguration>
      <cookieHandler requireSsl="false" name="WebApplication1Auth"/>
    </federationConfiguration>
  </system.identityModel.services>
```

In entityId section of **Kentor.authServices**, add the URL from the applications properties. Redirect the users after the successful authentication to returnUrl. For more information on each attribute, see https://github.com/Sustainsys/Saml2/blob/master/doc/Configuration.md#attributes

IdentityProviders section is used to point to the Identity Provider. In the following sections, configure the service provider to work with IDP and MobileIron Access.
In the above configuration, **stubidp.Kentor.se** is used. **Stubidp.kentor.se** is a test identity provider by Kentor. For more information on **System.IdentiyModel.Services** section, see: https://github.com/Sustainsys/Saml2/blob/master/doc/Configuration.md#systemidentitymodelservices-section

Step: 3

Ensure that you add the appropriate certificates in **App_Data** folder of your application. If you have used **Stubidp.kentor.se**, download the certificate from https://github.com/Sustainsys/Saml2/tree/master/Samples/SampleHttpModuleApplication/App_Data. The **Federate Kentor AuthService SP to an IDP** and **Configure Access to create a Federated Pair** sections provide details to get the certificate when configuring with IDP or MobileIron Access.

Step: 4

Open login.cshtml / index.cshtml file where you want to provide the sign-in link. Add the **Kentor.AuthServices.Configuration.KentorAuthServiceSection**. This is the class that redirects the user to the IDP for authentication. Below is the sample code that provides a sign in and log out Url.

Sign-in code:

```
 @foreach (var idp in
Kentor.AuthServices.Configuration.KentorAuthServicesSection.Current.IdentityProviders)
    {
      var entityId = idp.EntityId;
      var destinationUrl = idp.SignOnUrl;
      <br />
      <a href="@Url.Content("~/AuthServices/SignIn?idp=" + HttpUtility.UrlEncode(entityId))">Sign
in</a>@: -@entityId - @destinationUrl
    }
```

Logout code:

```
<p>
    You are signed in. <a id="logout" href="@Url.Content("~/AuthServices/Logout?ReturnUrl=" +
Uri.EscapeDataString(Url.Content("~/?Status=LoggedOut")))">LogOut</a>
</p>
```

Note: The complete sample index.cshtml code is available at https://github.com/Sustainsys/Saml2/blob/master/Samples/SampleHttpModuleApplication/Views/Home/Index.cshtml

Configuring Access to create a Federated Pair

You must configure Access to create a federated pair.

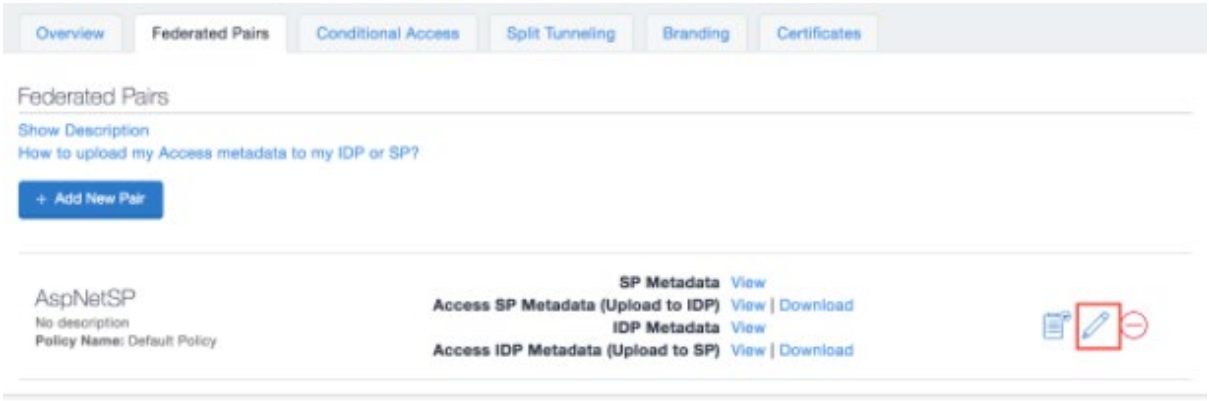
Procedure

1. Log in to **Access**.
2. Click **Profile** > **Get Started**.
3. Enter the Access host information with **-alt** appended, and upload the **ACCESS SSL certificate** in p12 format. All the other fields are set to default. Click **Save**.
4. On the **Federated Pairs** tab, click **Add New Pair** and select **Custom SAML Service Provider** as the service provider.
5. Enter the following details:
    a. Name
    b. Description
    c. Upload the Access Signing Certificate or click **Advanced Options** to create a new certificate.
    d. Add the metadata file details for the service provider. See Prerequisites.
    e. (Optional) Select *Use Tunnel Certificates for SSO* to configure Cert SSO on MobileIron Core. For more information, see Configure Access Cert SSO. Also see *Appendix* in the *MobileIron Access Guide* at https://support.mobileiron.com/docs/current/accs/
6. Click **Next**.
7. Select the Identity Provider of your choice. Click **Next**.
8. Select the Access Signing Certificate or click **Advanced options** to create a new certificate.
9. Upload the IdP metadata file that you downloaded for your IdP. Click **Done**.
10. Download the **ACCESS SP Metadata (Upload to IDP)** and the **ACCESS IDP Metadata (Upload to SP)** files from the federated pair page.
11. Open the web.config of the sample application. Under **Identity Providers** section, replace the **entityId** and **signonUrl** with the entityId from **Access IDP Metadata (Upload to SP)** metadata file downloaded in Step 10.
12. Copy the certificate and save it as a cert file. This is the Signing Certificate in web.config file of .NET application.
13. Compile and **Publish** the .NET Sample Application to IIS.

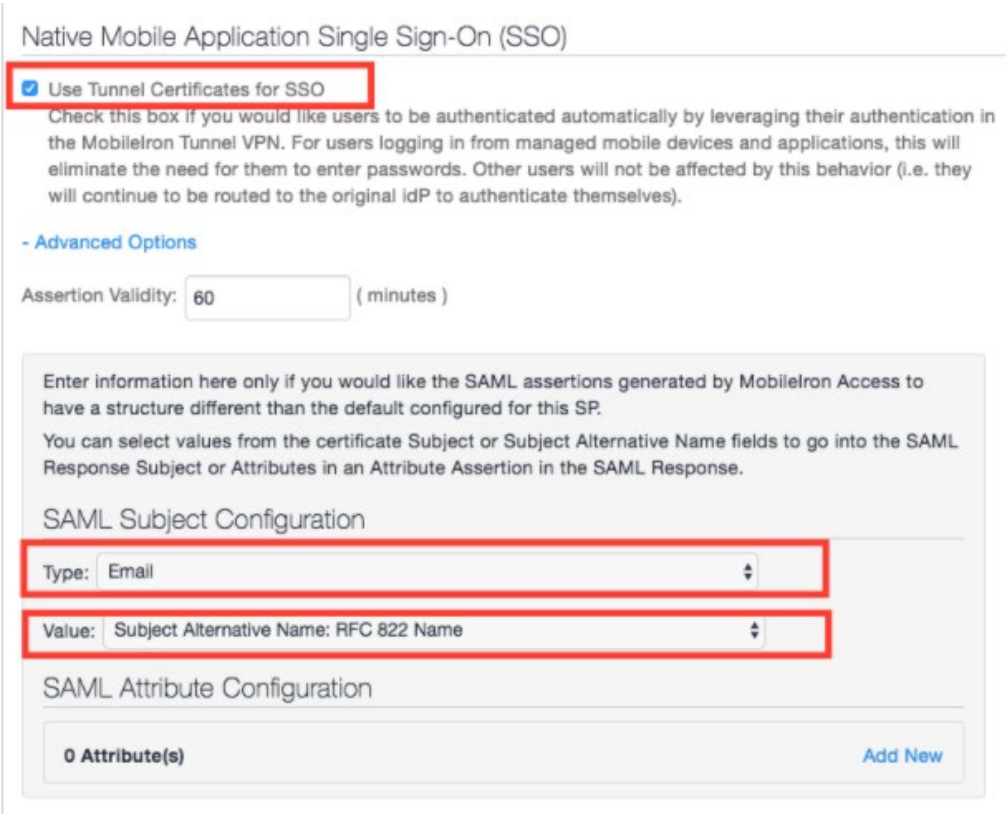
Configure Access Cert SSO

1. Login to MobileIron Access portal.

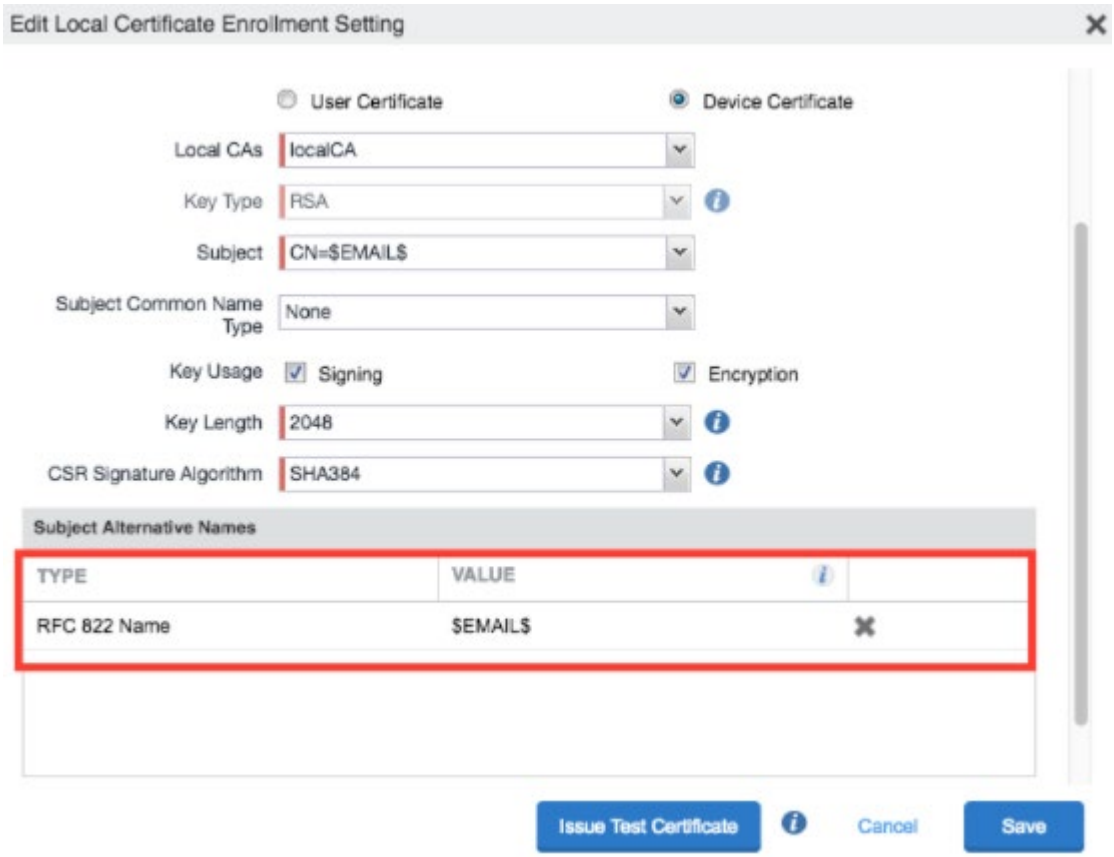
2. On the **Federated Pairs** tab, click **Edit** for the federated pair that you added.



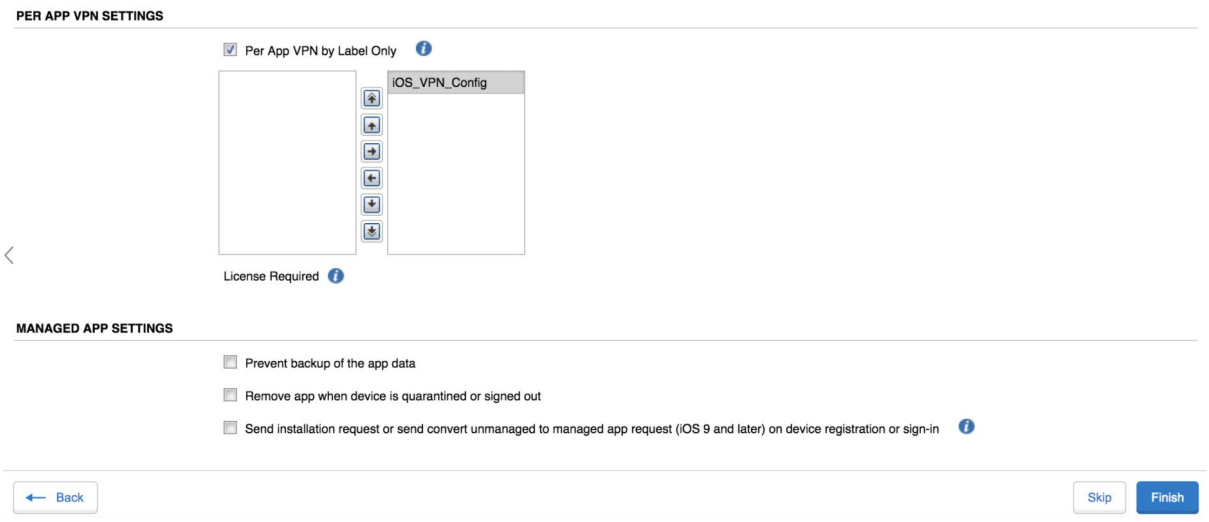
3. On the configure service provider page, enable **Use Tunnel Certificates for SSO**.
4. Click Advanced Option and in SAML subject configuration for Type select **Email** and value as **Subject Alternative Name: RFC 822 Name**.



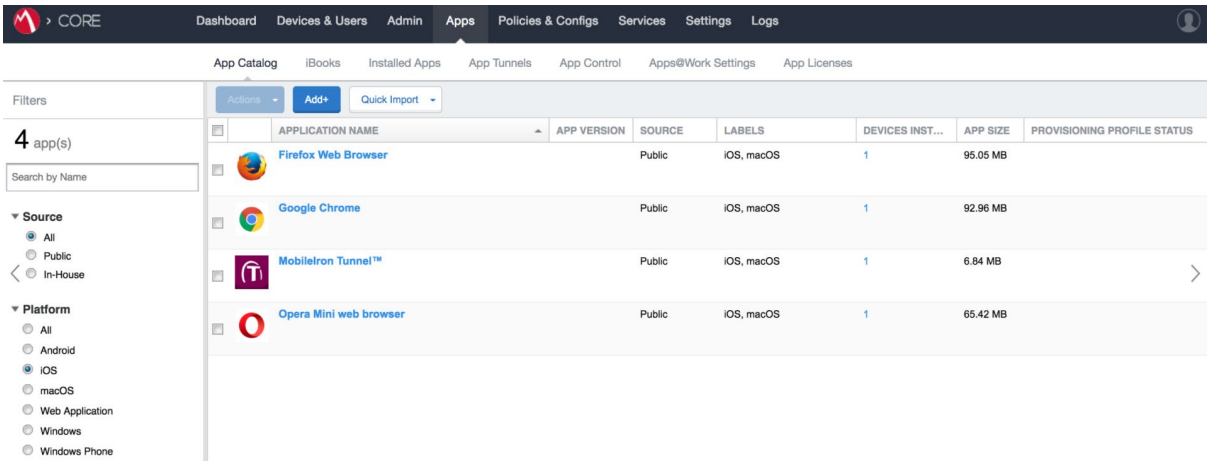
5. On MobileIron Core, click **Policies & Configs** tab.
6. Edit the **Local Certificate Enrollment Settings** and Add Value as **Email** and Type as **RFC 822 Name**.



7. Click **Save**.
8. On MobileIron Core, click **Apps** and add the application.
9. On Per-App settings page, select the VPN profile that is configured.
   Note: For cert SSO, it is required to configure Per App VPN for a managed application. Also ensure that the MobileIron Tunnel is pushed to the device.

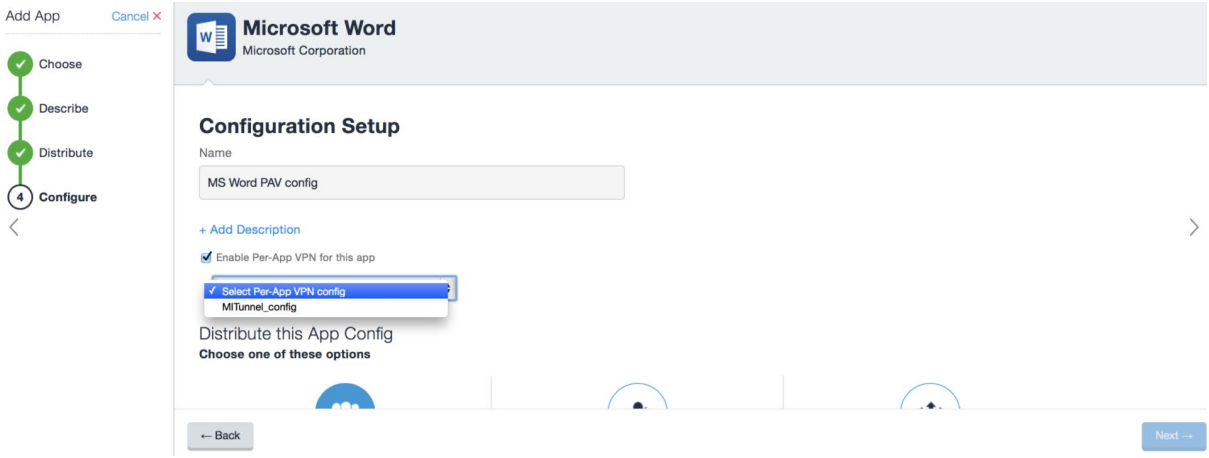


10. On the Apps tab, all the managed applications are displayed.



Cloud configuration

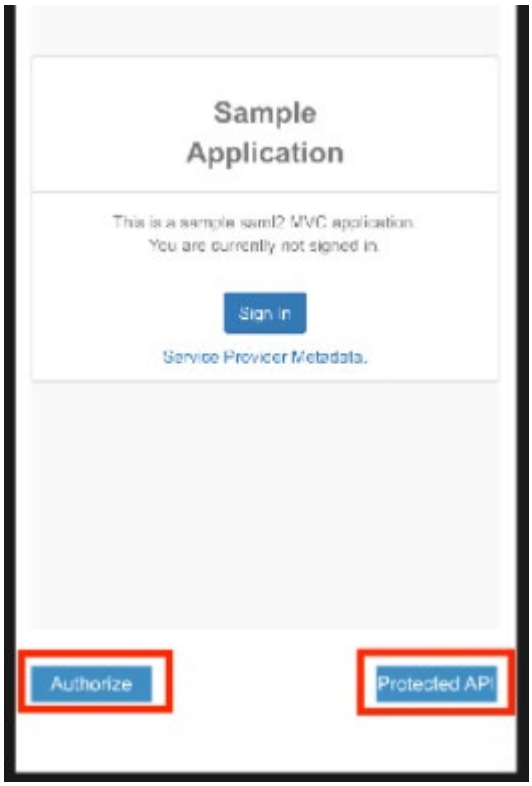
11. On Cloud, follow the same procedure as above.
12. Add the VPN profile and ensure to configure Per App VPN while pushing the apps.



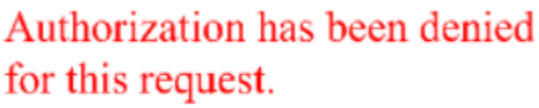
Note (Core and Cloud): Ensure that the profile gets updated on the managed devices. Now, you must be able to use the tunnel certificate SSO feature of Access on managed devices. Login to the application without entering any password from the managed application.

Verification for Sample iOS Application

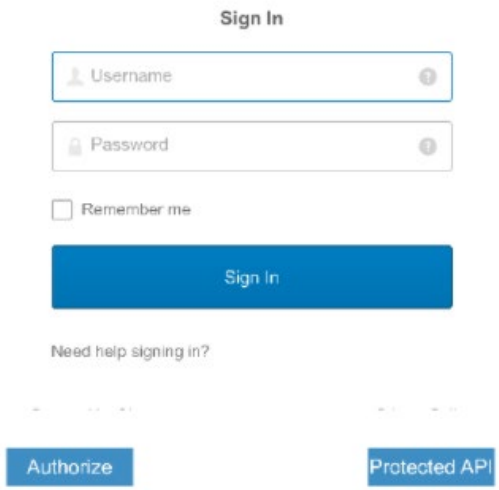
1. The sample IOS application has two buttons **Authorize** and **Protect API**. **Authorize** logs you to the **login**/ **logout** page and **Protect API** calls the protected API. As the API is protected it cannot be called without authorizing the user first.



2. If you click **Protected API**, without the user authorization, **Authorization denied for the request** error displays.



3. Click **Authorize** > **Sign-in**. If Cert SSO is enabled, then Access authorizes you and redirects to Service Providers homepage.
   If Cert SSO is not enabled, then Authorize redirects to the IDP authorization page.



4. Click Protected API and it is successful. In case of sample .NET based service provider, the Protected API returns the status code and the **You are authorized to make this request** message displays.
5. Logout to be redirected to the service providers login page.
   Note: When you logout, the Protected API provides the authorization denied error.

Configure Sample iOS Application

You must configure the Sample iOS app to point to the service provider.

1. On view load, the application loads the homepage of the service provider. Modify the URL to point to the service provider.

```
override func viewDidLoad() {
        super.viewDidLoad()
         // point the URL to service provider's landing page.
       let url = URL(string: "http://10.11.80.251:17009/")
        let request = URLRequest(url:url!)
       wbview.allowsBackForwardNavigationGestures = true
       wbview.load(request)
       // Do any additional setup after loading the view, typically from a nib.
    }
```

2. Authorize API calls the service provider's login page. Change the URL to appropriate URL.

```
@IBAction func authorize(_ sender: Any) {
     // point the URL to service provider's login page.
       let url = URL(string: "http://10.11.80.251:17009/")
       let request = URLRequest(url:url!)
       wbview.allowsBackForwardNavigationGestures = true
       wbview.load(request)
    }
```

3. Protected API is as follows. Modify the URL to point to the Protected API from the service provider.

```
@IBAction func callAPI(_ sender: UIButton) {
        // Call the protected API.
        let url = URL(string: "http://10.11.80.251:17009/api/HelloWorld")
        let request = URLRequest(url:url!)
        wbview.allowsBackForwardNavigationGestures = true
        wbview.load(request)


    }
```