Sustainsys.Saml2 Documentation

Anders Abel

Sep 19, 2023

Saml2

The Sustainsys.Saml2 library adds SAML2P support to ASP.NET web sites, allowing the web site to act as a SAML2 Service Provider (SP). The library was previously named Kentor.AuthServices. Sustainsys.Saml2 is open sourced and contributions are welcome, please see *contributing guidelines* for info on coding standards etc.

CHAPTER 1

Using Sustainsys.Saml2

Using the Sustainsys.Saml2 library to add SAML2P support into your ASP.NET web applications is a two-step process:

1. Reference the Nuget package
2. Provide the necessary configuration information

The exact nature of these steps depends on the ASP.NET integration you're after. See *Getting Started* for all the details.

CHAPTER 2

Licensing

The library is licensed under the GNU Lesser General Public License (LPGL).

Starting with version 2.0.0, the license has been changed to the MIT license.

2.1 Getting Started

See the sections below which contain information that will help you get started adding SAML2P support into your flavor of ASP.NET.

If you have gotten the appropriate Nuget package installed and then completed the configuration described below and are having any trouble, make sure to check out the *Troubleshooting* for assistance.

A sample SAML identity provider is available to further assist you in getting started if you don't already have a SAML identity provider that you can test with. You can access it directly at https://stubidp.sustainsys.com, or you can download the solution to run it locally yourself (it's a project within the Sustainsys.Saml2 github repository).

2.1.1 ASP.NET Web Forms

The `Saml2AuthenticationModule` provides Saml2 authentication to IIS web sites. In many cases it should just be *configured* in the `web.config` file and work without any code written in the application at all (even though providing an owin ClaimsAuthenticationManager for claims translation is highly recommended).

Nuget Package to use: Sustainsys.Saml2.HttpModule

See *Configuration* for information about how to configure the `web.config` file.

2.1.2 ASP.NET MVC

The `MVC` package contains an MVC controller that will be accessible in your application just by installing the package in the application. For MVC applications a controller is preferred over using the authentication module as it integrates with MVC's error handling.

Nuget Package to use: Sustainsys.Saml2.Mvc

See *Configuration* for information about how to configure the `web.config` file.

2.1.3 Owin Middleware

The `Owin` middleware is modeled after the external authentication modules for social login (such as Google, Facebook, Twitter). This allows easy integration with ASP.NET Identity for keeping application specific user and role information.

Nuget Package to use: Sustainsys.Saml2.Owin

See the *Owin Middleware* page for information on how to set up and use the middleware.

2.1.4 ASP.NET Core 2 Handler

The ASP.NET Core 2 Handler is compatbile with the ASP.NET Core 2.0 authentication model.

Nuget Package to use: Sustainsys.Saml2.AspNetCore2

HOW TO CONFIGURE ASP.NET CORE 2 – owin middleware doc? somewhere else?

2.1.5 IdentityServer[3/4] Integration

If you're using `IdentityServer` (v3 or later), you may want to configure SAML identity providers like Okta or Ping as external identity providers within your IdentityServer implementation.

The `Owin` & `ASP.NET Core2` modules enable SAML identity providers to be integrated within IdentityServer3 and IdentityServer4 packages.

Nuget Package to use for IdentityServer3: Sustainsys.Saml2.Owin Nuget Package for IdentityServer4: Sustainsys.Saml2.AspNetCore2

Review *this document* to see how to configure Saml2 with IdentityServer3 and Okta to add Okta as an identity provider to an IdentityServer3 project. There is also a SampleIdentityServer3 project in the Saml2 repository.

Note: There is also a Sustainsys.Saml2 Nuget package, but this only contains functionality shared across the packages above and is not meant to be referenced directly in other projects.

Note: The protocol handling classes are available as a public API as well, making it possible to reuse some of the internals for writing your own service provider or identity provider.

2.2 Configuration

To use Sustainsys.Saml2 in an application and configure it in `web.config` (which is the default for the `HttpModule` and `MVC` libraries) it must be **enabled** in the application's `web.config`. The sample applications contains complete working web.config examples. For ASP.NET MVC applications see this working web.config example.

Note: Applications using the `Owin` library usually make their configuration in code and in that case no web.config changes are needed. If an Owin library is set up to use web.config (by passing `true` to the `Saml2AuthenticationOptions` constructor) the information here applies.

2.2.1 Config Sections

Three new config sections are required. Add these under `configuration/configSections`. Each of the sections will be a child element of the main `configuration` section and each is described below.

```
<configSections>
    <!-- Add these sections below any existing. -->
    <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.SystemIdentityModelServicesSection, System.IdentityModel.Services,␣
↪Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />
    <section name="sustainsys.saml2" type="Sustainsys.Saml2.Configuration.
↪SustainsysSaml2Section, Sustainsys.Saml2"/>
</configSections>
```

2.2.2 Loading Modules

When using the `HttpModule` and the `MVC` controller, the `SessionAuthenticationModule` needs to be loaded and if using the http module that needs to be loaded as well. The `Owin` package does not need any http modules, please see the separate info on the *Owin Middleware*:.

```
<system.webServer>
    <modules>
        <!-- Add these modules below any existing. The SessionAuthenticatioModule
            must be loaded before the Saml2AuthenticationModule -->
        <add name="SessionAuthenticationModule" type="System.IdentityModel.Services.
↪SessionAuthenticationModule, System.IdentityModel.Services, Version=4.0.0.0,␣
↪Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
        <!-- Only add the Saml2AuthenticationModule if you're using the Sustainsys.
↪Saml2.HttpModule
            library. If you are using Sustainsys.Saml2.Mvc you SHOULD NOT load this␣
↪module.-->
        <add name="Saml2AuthenticationModule" type="Sustainsys.Saml2.HttpModule.
↪Saml2AuthenticationModule, Sustainsys.Saml2.HttpModule"/>
    </modules>
</system.webServer>
```

2.2.3 Sustainsys.Saml2 Section

The `sustainsys.saml2` section contains the configuration of the Sustainsys.Saml2 library. It is required for the http module and the mvc controller. The Owin middleware can read web.config, but can also be configured from code (see *Owin middleware*).

A sample section is shown below. For full details and all avaialble options, see *sustainsys.saml2*.

```
<sustainsys.saml2 entityId="http://localhost:17009"
                  returnUrl="http://localhost:17009/SamplePath/"
                  discoveryServiceUrl="http://localhost:52071/DiscoveryService"
                  authenticateRequestSigningBehavior="Always">
    <nameIdPolicy allowCreate="true" format="Persistent"/>
    <metadata cacheDuration="0:0:42" validDuration="7.12:00:00" wantAssertionsSigned=
↪"true">
        <organization name="Sustainsys IT AB" displayName="Sustainsys" url="http://
↪www.Sustainsys.se" language="sv" />
        <contactPerson type="Other" email="info@Sustainsys.se" />
        <requestedAttributes>
        <add friendlyName ="Some Name" name="urn:someName" nameFormat=
↪"urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true" />
        <add name="Minimal" />
        </requestedAttributes>
    </metadata>
    <identityProviders>
        <add entityId="https://stubidp.sustainsys.com/Metadata"
            signOnUrl="https://stubidp.sustainsys.com"
            allowUnsolicitedAuthnResponse="true"
            binding="HttpRedirect"
            wantAuthnRequestsSigned="true">
        <signingCertificate storeName="AddressBook" storeLocation="CurrentUser"
                            findValue="Sustainsys.Saml2.StubIdp" x509FindType=
↪"FindBySubjectName" />
        </add>
        <add entityId="example-idp"
            metadataLocation="https://idp.example.com/Metadata"
            allowUnsolicitedAuthnResponse="true"
            loadMetadata = "true" />
    </identityProviders>
    <!-- Optional configuration for signed requests. Required for Single Logout. -->
    <serviceCertificates>
        <add fileName="~/App_Data/Sustainsys.Saml2.Tests.pfx" />
    </serviceCertificates>
    <!-- Optional configuration for fetching IDP list from a federation -->
    <federations>
        <add metadataLocation="https://federation.example.com/metadata.xml"␣
↪allowUnsolicitedAuthnResponse = "false" />
    </federations>
</sustainsys.saml2>
```

2.2.4 System.IdentityModel Section

There must be a `<system.identityModel>` section in the config file or there will be a runtime error. The section can be empty (use `<system.identityModel />`).

The reason you might want this to be non-empty is to provide a custom *ClaimsAuthenticationManager* as shown in the sample below (you would obviously provide your own type in place of the Stub shown in the sample).

```
<system.identityModel>
    <identityConfiguration>
        <claimsAuthenticationManager type="Sustainsys.Saml2.Tests.
↪ClaimsAuthenticationManagerStub, Sustainsys.Saml2.Tests"/>
    </identityConfiguration>
</system.identityModel>
```

2.2.5 System.IdentityModel.Services Section

The `<system.identityModel.services>` element configures the built in servies. For testing on non ssl sites, the requirement for ssl for the session authentication cookie must be disabled.

> **Danger:** It is a severe security risk to leave the `requireSsl` setting as false in a production environment.

```
<system.identityModel.services>
    <federationConfiguration>
        <cookieHandler requireSsl ="false"/>
    </federationConfiguration>
</system.identityModel.services>
```

2.3 Owin Middleware

The Sustainsys Saml2 Owin middleware is designed to be used with an Owin authentication pipeline and is compatible with ASP.NET Identity. Sustainsys Saml2 provides external login in the same way as the built-in Google, Facebook and Twitter providers.

To use the Sustainsys Saml2 middleware, it needs to be configured in `Startup.Auth.Cs`.

```
app.UseSaml2Authentication(new Saml2AuthenticationOptions());
```

The `Saml2AuthenticationOptions` class only contains the Owin-specific configuration (such as the name used to identify the login provider). The rest of the configuration is read from the web.config/app.config and *configured in the same way* as when using the http module or the MVC controller.

If you would like to provide the Saml2-related configuration in code, specify `false` for the `loadConfiguration` constructor parameter and then build the options based on your own logic. For example:

```
var mySaml2Options = new Saml2AuthenticationOptions(false)
// more logic to set SPOptions, etc.
app.UseSaml2Authentication(mySaml2Options);
```

You can see a full example of this in the SampleOwinApplication project included in the source code. See the `Startup.Auth.cs` file.

2.3.1 Selecting Idp

An Owin-based application issues an `AuthenticationResponseChallenge` to ask the middleware to begin the authentication procedure. In that challenge, there is a properties dictionary. To use a specified idp, the entity id of the idp should be entered in that dictionary under the key “idp”.

In a typical MVC application that requires some changes to the generated code to enable passing a property to the `AuthenticationProperties` dictionary.

Another, more simple way to pass a value is to put it directly in the Owin environment dictionary under the key “saml2.idp”.

Here’s an example of how to set the Owin environment value through ASP.NET MVC:

```
var context = HttpContext.GetOwinContext();
context.Environment.Add("saml2.idp", new EntityId(YOUR_IDP_ENTITY_ID));
```

2.3.2 Module Path and Metadata

By default the module path is */Saml2* but you can specify a different modulepath in your SPOptions object mentioned above.

The metadata URL is the root of this module path.

2.4 ClaimsAuthenticationManager

When using federated authentication, the identity provider solely decides what claims to use to populate the incoming identity. If using multiple identity providers there is very high probability that they will present the same information in somewhat different ways. That's where the `ClaimsAuthenticationManager` fits in. It works as a translation filter that can modify or replace the incoming identity as soon as it has been constructed from the incoming authentication response.

You can implement a `ClaimsAuthenticationManager` by creating a class derived from the `System.Security.Claims.ClaimsAuthenticationManager` class.

Then register it with a `<claimsAuthenticationManager>` element in the configuration if the configuration is loaded from the config file. If the configuration is done in code (typically for the OWIN middleware) the ClaimsAuthenticationManager should be registered in `Options.SPOptions.SystemIdentityModelIdentityConfiguration.ClaimsAuthenticationManager`.

2.4.1 Single Logout

If you are using Single Logout, you need to make sure that the claims containing the Saml2 logout information are present in the returned identity. The types of the claims are available in `Saml2ClaimTypes.SessionIndex` and `Saml2ClaimTypes.LogoutNameIdentifier`.

2.5 IdentityServer3 with Okta

Sustainsys Saml2 can be used very effectively to extend the functionality of IdentityServer3 by adding support for SAML-based identity providers, such as Okta and OneLogin. This example will show how to add Okta as an identity provider to IdentityServer3 using the Sustainsys Saml2 package.

2.5.1 Step 0: Establish identity server using IdentityServer3

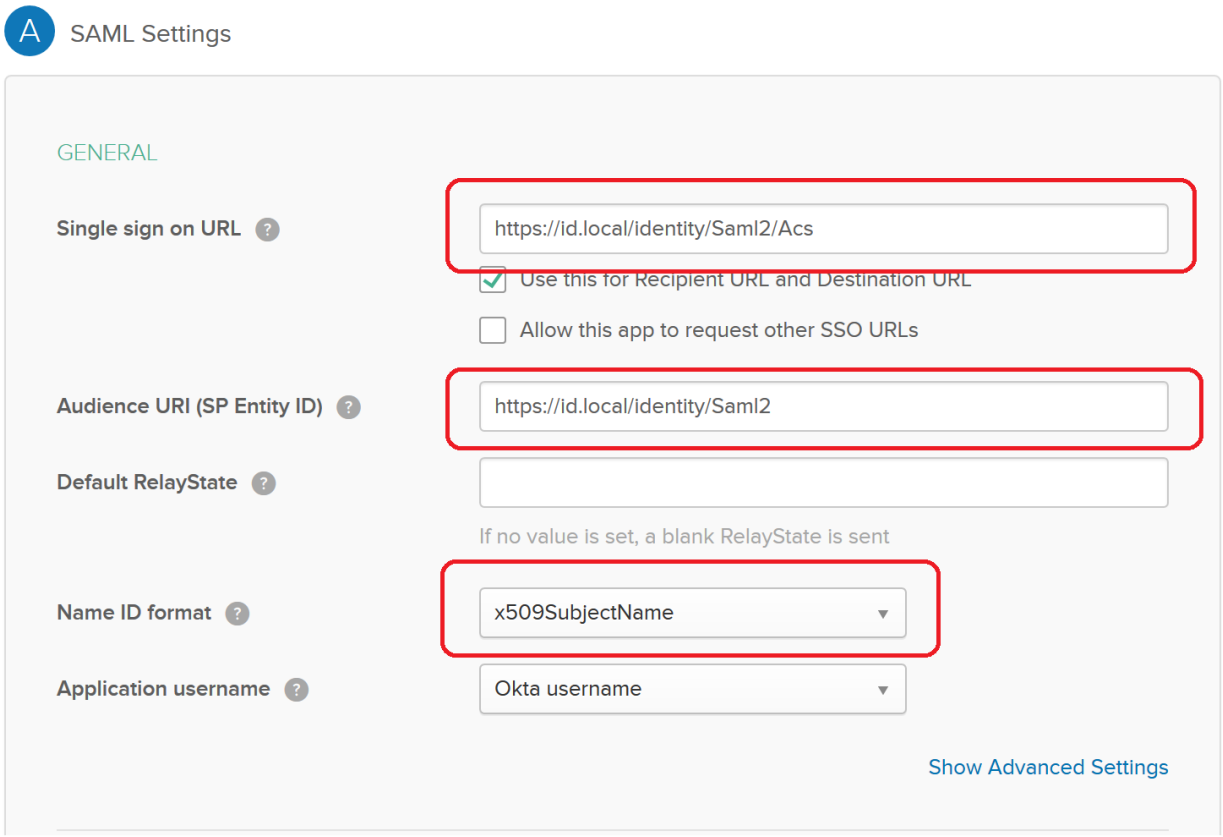
This article assumes that you already have your own identity server project set up and that it uses IdentityServer3. If you haven't, check out the documentation and samples for that project and then come back here when you have a working identity server.

2.5.2 Step 1: Add the NuGet package to your identity server

The package you need is `Sustainsys.Saml2.Owin`. Install that to the project where you have IdentityServer3 established. We'll add the necessary configuration to establish the Okta identity provider in identity server later, after we've set up the application within Okta.

2.5.3 Step 2: Configure an Application within Okta

If you don't already have an instance of Okta (or don't have access to one with admin / configuration privileges), you can create a developer instance. Ultimately, you will want to add an "Application" to this instance. Add one, and give it some kind of name, and you will get to the important part of configuring the application. Below is the SAML settings screen and a description of how to specify these options.



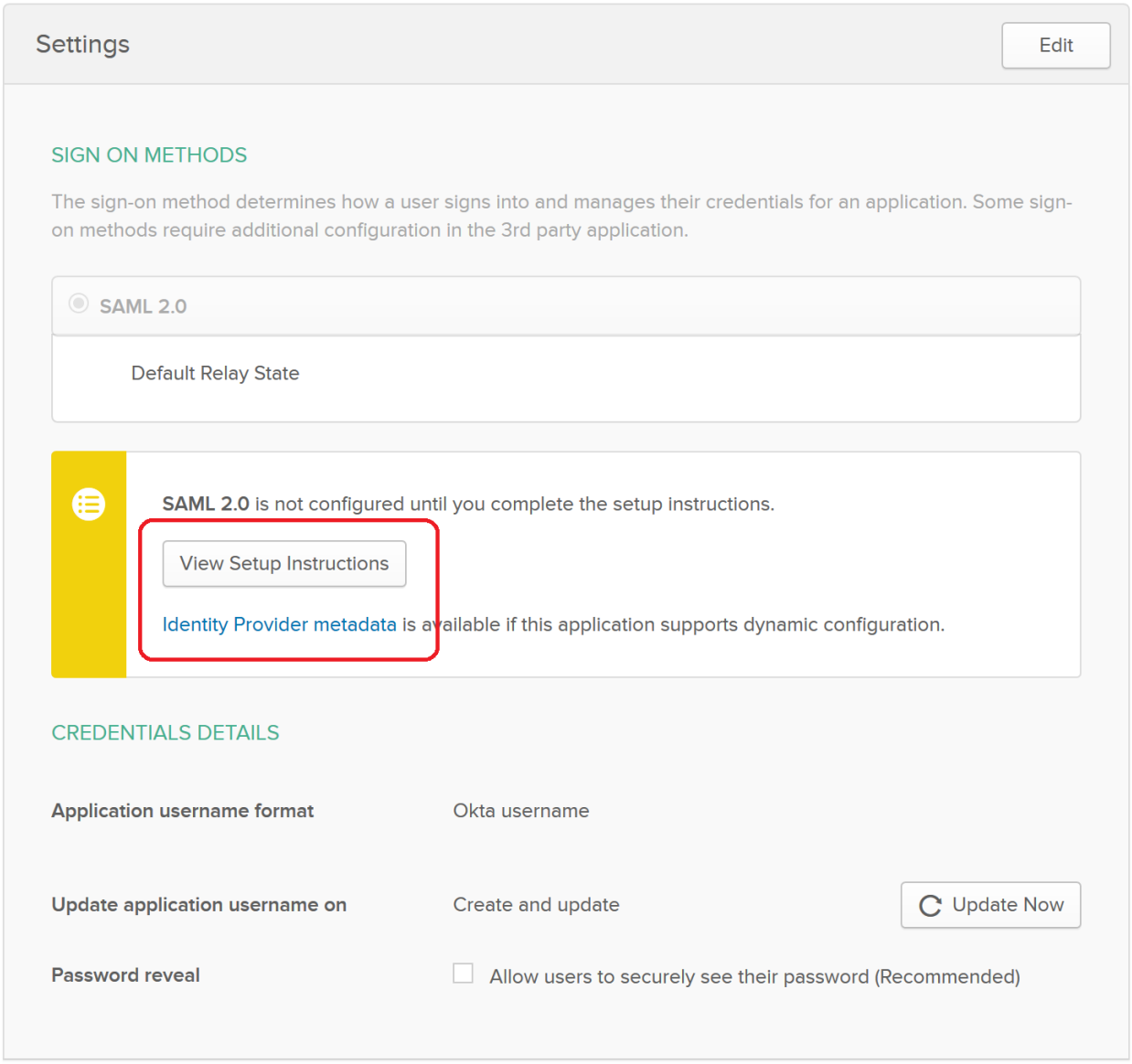
Item	Explanation
Single Sign-On URL	This is the *Assertion Consumer Service (ACS)* endpoint within the application. In our case, this is the core endpoint of the app, plus `/Saml2/Acs`. So if your Identity Server is at `https://id.local/identity`, then your value here would be `https://id.local/identity/Saml2/Acs`.
Audience URI	This should be the metadata URL of the audience (in this case, your identity server's SAML metadata), so use the ACS endpoint minus the ACS. Carrying forward the example, this would be `https://id.local/identity/Saml2`.
Name ID Format	This will be the Okta username for any of your users. I chose the `X509SubjectName` with Okta UserName. I'm assuming that you've probably got some kind of custom user service within Identity Server to get your own claims – or even if you don't, you have the "Users" object defined with some hard coded users or something. What you need to be able to do is determine YOUR username from something that Okta can pass back to you for THEIR username. Choose from the available options, knowing that you may need to translate it a bit to get to your username.

2.5.4 Step 3: Configure your identity server with the new identity provider

Most of what you need to do is pretty easily seen by just looking at the code below along with the comments that simply refer back to the Okta configuration points in Step 2.

Metadata: You will need to provide the "metadata URL" in the code below. To get this, you can look at the "Sign On" tab within the Okta application configuration area, and right-click the "Identity Provider metadata" link and copy the URL.

Entity ID: You can determine this by clicking the View Setup Instructions button and looking for the "Identity Provider Issuer" value.




```
public class Startup
{
    public void Configuration(IAppBuilder app)
    {
        var options = Helpers.GetIdentityServerOptions(); // a helper function where
↪we specify our IdSrv3 options
```

(continues on next page)

(continued from previous page)

```
        options.AuthenticationOptions = new AuthenticationOptions
        {
            IdentityProviders = ConfigureIdentityProviders
        };

        app.UseCookieAuthentication(new CookieAuthenticationOptions());
        app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.
↪AuthenticationType);
        app.Map("/identity", idsrvApp =>
        {
            idsrvApp.UseIdentityServer(options);
        });
    }

    public static void ConfigureIdentityProviders(IAppBuilder app, string␣
↪signInAsType)
    {
        var saml2Options = new Saml2AuthenticationOptions(false)
        {
            SPOptions = new SPOptions
            {
                AuthenticateRequestSigningBehavior = SigningBehavior.Never // or add␣
↪a signing certificate
                EntityId = new EntityId("<okta Audience URI>") // from (B) above
            },
            SignInAsAuthenticationType = signInAsType,
            AuthenticationType = "okta", // the "idp" - identity provider - that you␣
↪can refer to throughout identity server
            Caption = "Okta",  // the caption for the button or option that a user␣
↪might see to prompt them for this login option
        };

        saml2Options.IdentityProviders.Add(new IdentityProvider(
            new EntityId("<OktaIssuerUri>"), saml2Options.SPOptions)  // from (F)␣
↪above
            {
                LoadMetadata = true,
                MetadataLocation = "https://<OktaInstance>/app/<OktaAppId>/sso/saml/
↪metadata" // see Metadata note above
            });

        app.UseSaml2Authentication(saml2Options);
    }
}
```

Note: Regarding the "AuthenticateRequestSigningBehavior" above: Okta sets a value in their metadata that specifies `WantAuthnRequestsSigned="true"`, which means that Saml2 will try to sign outgoing AuthN requests. The code above does work – the "Want" doesn't imply "Require". To actually honor the request, though and enable signing, you need to go a step further:

To enable signing, call `saml2Options.ServiceCertificates.Add(new ServiceCertificate { ... })` to configure the certificate Saml2 should use for signing. That certificate should be something that you have generated on your end, where you have a private key. If you don't have that already, I'd suggest going with the `SigningBehavior.Never` option.

2.5.5 Step 4: Try it out!

Logging in with Okta through your identity server should work at this point. Cheers!

2.5.6 Step 5: Set up IdP-Initiated support

This is an optional but very nice and (I think) important step that will enable Okta users to login to your site by clicking on the "app" icon on their Okta Dashboard. The process involves setting up the Sustainsys options to "allow unsolicited response", setting up a redirect page on your target site (not identity server), and specifying that in the Sustainsys settings. Explanations follow.

The first step is to configure the Saml2/Acs endpoint to allow for unsolicited responses. To do this, modify the code snippet where you are adding IdentityProviders to include the `AllowUnsolictedAuthnResponse = true` line shown below.

```
saml2Options.IdentityProviders.Add(new IdentityProvider(
    new EntityId(oktaEntityId), Ssml2Options.SPOptions)
    {
        LoadMetadata = true,
        MetadataLocation = oktaMetadataUrl,
        AllowUnsolicitedAuthnResponse = true
    });
```

Now you need to set up your target website (not IdentityServer3) to have a page that will simply turn around and redirect to your identity server with an authorize request.

I set up a new page in my webforms site call "IdP_InitiatedRedirect" and required an "idp" query string value in case I want to use other SAML IdP's. Then in the `SPOptions` setup, you add the URL for the `ReturnUrl` property as shown below:

```
SPOptions = new SPOptions
{
    EntityId = new EntityId(serviceProviderEntityId),
    ReturnUrl = new Uri("https://yoursite.com/Idp_InitiatedRedirect.aspx?idp=okta")
},
```

The only thing left is to code the logic on your redirect page to make an authorize request to your identity server. You should already have a reference to the `IdentityModel.Client` package, so then you can write some code that looks like this:

```
protected void Page_Load(object sender, EventArgs e)
{
    var idp = Request.QueryString["idp"];
    if (string.IsNullOrEmpty(idp))
        throw new Exception("No idp included in redirect querystring!!");

    var scopesForAuth = "<the scopes for your application>";
    var state = Guid.NewGuid().ToString("N");
    var nonce = Guid.NewGuid().ToString("N");
    var client = new OAuth2Client(new Uri("https://<youridentityserver>" + "/connect/
↪authorize"));

    var returnUrlForOkta = client.CreateAuthorizeUrl("<youridserverclientid", "id_
↪token token", scopesForAuth,
        "https://yoursite.com/yourreallogintarget"),
        state, nonce, acrValues: string.Format("idp:{0}", idp), responseMode: "form_
↪post");
```

(continues on next page)

(continued from previous page)

```
    Response.Redirect(returnUrlForOkta, false);
}
```

Once you have all of that in place, you should be able to click the app button on the Okta dashboard and successfully log in to your website through IdentityServer3!

2.5.7 Step 6 - Enable multiple Okta instances (multi-tenant / multiple independent Okta IdPs)

If you only have a single Okta instance to enable, you don't need to perform this step. But often, implementations of IdSrv3 find themselves wanting to provide SSO services to multiple Okta instances. Doing this tweaks our approach a little, so read on...

By way of background, when an AuthN request comes in to the IdentityServer3/Sustainsys pipeline, the package middleware needs to determine where to forward the request – this amounts to an EXACT URL – which is not only different for each Okta client, but also different for each application within Okta. And this logic needs to be applied for both standard AuthN requests from your website, AND for the IdP-Initiated Redirect process described above.

Whether the above fully makes sense to you or not, the net effect is that you really should be setting up new instances of the Saml2 middleware for each Okta tenant you have. This approach led me to a little refactoring of the above code – other appproaches are definitely valid, but shown below is some code that works.

There is both Okta configuration that needs to be done for each instance of Okta, and then a follow-up set of config within your IdentityServer that will need to be done. The basic steps in the process for setting up each instance (including editing your single instance) are as follows:

- Identify the idpName and description you will use for the instance in question
- Configure the app within Okta with the single-sign-on URL based on the instance name and the other Okta config options laid out in step 2 above
- Get the entity id and metadata URL that were generated by the Okta configuration step above
- Configure a new instance of Saml2 middleware within IdSrv3 based on all of the above info

Identify the idpName and Description for an Okta App instance

For this, just consider your instance and make up two values that make sense. By way of example, to be "Okta Verified", you need to support their "Okta Application Network" testing instance. To set up an IdP name and description for this, I just chose "okta-oan" for the idp name, and "Okta-OAN" for the description. It could be anything – it just needs to be unique to that instance within your setup. We'll see in both the Okta and middleware configuration how this is important.

Configure the Okta App with Single-Sign-On URL based on Instance Name

All of the Okta configuration options in Step 2 above are still valid - with the one exception being the Single Sign On URL. This is because there is no longer a single "Saml2" endpoint within your identity server – there will be multiple: one for each Okta instance you have.

So this value will use the idpName you came up with in the previous step. The idpname replaces "Saml2" from version 1 of this configruation: `https://id.local/identity/{idpName}/Acs`. If we continue our example of `okta-oan` as an idp name, we would have: `https://id.local/identity/okta-oan/Acs`.

Get the EntityId and Metadata URL from Okta

This will be the same way you got the entity id and metadata URL from Step 2 above, but may involve you requesting it from an Okta administrator of the instance you are trying to set up. The values look something like this:

- entity id: http://www.okta.com/exk4yxtgy7ZzSDp8e0h7
- metadata URL: https://dev-490944.oktapreview.com/app/exk4yxtgy7ZzSDp8e0h7/sso/saml/metadata

Note that even though the entity id does NOT refer to the okta instance you are setting up, the app id inside it (exk...) is unique to the okta instance, so the entity id will indeed be different for each instance.

Configure a new instance of Saml2 middleware within IdSrv3

The 4 basic code components in the approach are (feel free to edit – but at least you can see the approach):

- a `GetOkta{instance}Options` method for each instance (where you place the configuration unique to each instance)
- a single `GetOktaIdentityProvider` method (configures entity id and metadata url based on inputs)
- a single `GetCoreOktaOptions` method (sets up the options common to each instance, and sets the module path based on input param)
- an `app.UseSaml2Authentication()` call for each of your supported Okta instances (make sure you actually add the instance to the pipeline)

The following code I put in a static class called `Helpers-Okta` and shows the first three code components above:

```
internal static Saml2AuthenticationOptions GetOktaOanOptions(string signInAsType)
{
    var saml2Options = GetCoreOktaOptions(signInAsType, "okta-oan", "Okta-OAN");

    const string oktaEntityId = "http://www.okta.com/exk16268xbsV0A213sa23";  // got␣
↪this from an Okta OAN support / admin
    const string oktaMetadataUrl = "https://okta-coe-test.okta.com/app/
↪exk16268xbsV0A213sa23/sso/saml/metadata";  // got this from an Okta OAN support /␣
↪admin

    var oanInstance = GetOktaIdentityProvider(saml2Options.SPOptions, oktaEntityId,␣
↪oktaMetadataUrl);
    saml2Options.IdentityProviders.Add(oanInstance);

    return saml2Options;
}

private static Saml2AuthenticationOptions GetCoreOktaOptions(string signInAsType,␣
↪string idpName, string idpLabel)
{
    string serviceProviderEntityId;
    string oktaRelyingPartyRedirectUrl;

    switch (GetEnvironment())  // determine if you are in dev, test, or production␣
↪here....
    {
        case "PRD":
            serviceProviderEntityId = "https://{productionIdentityServerRoot}/
↪identity/saml";
            oktaRelyingPartyRedirectUrl = string.Format("https://
↪{productionAppWebSiteRoot}/Portal/Pages/IdP_InitiatedRedirect.aspx?idp={0}",␣
↪idpName);
```

(continues on next page)

(continued from previous page)

```
            break;
        case "TEST":
            serviceProviderEntityId = "https://{testIdentityServerRoot}/identity/saml
↪";
            oktaRelyingPartyRedirectUrl = string.Format("https://{testAppWebSiteRoot}/
↪Portal/Pages/IdP_InitiatedRedirect.aspx?idp={0}", idpName);
            break;
        default:
            serviceProviderEntityId = "https://{devIdentityServerRoot}/identity/saml";
            oktaRelyingPartyRedirectUrl = string.Format("https://{devAppWebSiteRoot}/
↪Portal/Pages/IdP_InitiatedRedirect.aspx?idp={0}", idpName);
            break;
    }

    var saml2Options = new Saml2AuthenticationOptions(false)
    {
        SPOptions = new SPOptions
        {
            EntityId = new EntityId(serviceProviderEntityId),
            ReturnUrl = new Uri(oktaRelyingPartyRedirectUrl),
            ModulePath = string.Format("/{0}", idpName)   // this is important -- it␣
↪is what drives the separate instances
        },
        SignInAsAuthenticationType = signInAsType,
        AuthenticationType = idpName,
        Caption = idpLabel,
    };

    return saml2Options;
}

private static IdentityProvider GetOktaIdentityProvider(ISPOptions options, string␣
↪oktaEntityId, string oktaMetadataUrl)
{
    var idp = new IdentityProvider(new EntityId(oktaEntityId), options)
    {
        LoadMetadata = true,
        MetadataLocation = oktaMetadataUrl,
        AllowUnsolicitedAuthnResponse = true
    };
    return idp;
}
```

For the `Startup` code, it becomes pretty simple to add the instances:

```
// Okta Application Network (OAN) instance
app.UseSaml2Authentication(Helpers.GetOktaOanOptions(signInAsType));
```

The code above just calls our new method for the instance we have configured.

Now it should all work and is repeatable for other instances!

Note: The above approach does require code changes for each new instance of Okta you want to add / configure. If you wanted, you could pretty easily put the configuration into a database table and read it that way – the four data points you need for each instance are:

- idp Name

- idp Description
- entity Id
- metadata URL

You could create a table with those entries, then read them and loop through each, creating an instance for them. Note also that making changes to this would likely require at least restarting the application pool for your identity server or something similar to force the running instance to recognize the new configuration.

2.6 Troubleshooting

If you're having trouble - don't give up! :)

The items below may point you in the right direction.

- Check the issues archive.
- Check the SAML2 specification, starting with the core section, or the newer OASIS Saml Wiki.
- Log your actual SAML2 conversation with SAML Chrome Panel or SAML Tracer for Firefox.
- Connect an `ILoggerAdapter` to your `SPOptions.Logger`. If you are using the `OWIN` middleware this is done for you automatically and you can see the output in the OWIN/Katana logging.
- Last but not least, download the Saml2 source and check out what's really happening.

2.7 Contributing

Sustainsys.Saml2 is maintained by and have mostly been developed by Sustainsys in Stockholm, Sweden. The library's source code is hosted on github. When doing work on protocol features, it is recommended to consult the official SAML specifications.

2.7.1 Issue tracking

Github issues are used to keep track of issues and releases. For requests of functionality or to report bugs, please open an issue in the github repo. It is advised to open an issue describing the plans before starting any major coding work. Discussing before writing code significantly reduces the risk of getting a pull request denied.

2.7.2 Versioning

Sustainsys uses semantic versioning as defined on http://semver.org/. Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.

2.7.3 Coding Conventions

The coding conventions follow the classic .NET style of coding, with the following styles:

- Always use `{}` for if statements, even when there is only one line.
- Code analysis is enabled and all code should compile without compiler warnings or code analysis errors. Code analysis warnings that are not relevant are supressed in the source. Rules should only be disabled on a global level if it really is appropriate to disable the rule for the entire code base. Unknown words are added to CustomDictionary.xml instead of suppressing individual warnings.
- Private members in classes are named with camelCasing, no underscores or similar.
- Member variables are not prefixed with `this.` unless required to resolve ambiguity (such as in a constructor having parameters with the same name as the members).
- Any single method is short enough to fit on one screen (on a typical laptop monitor, not a 30-inch development monster-monitor in vertical orientation).
- The code is formatted to (mostly) fit in 80 columns.

2.7.4 Unit Tests

The Sustainsys.Saml2 library has been developed using TDD (Test Driven Development). All functionality is covered by tests, and it will remain that way. Pull requests will only be merged if they contain tests covering the added functionality. Parts of the code that aren't practically possible to test because of tight integration with the web server (see e.g. `CommandResult.ApplyPrincipal`) are excluded from this rule and should be marked with an `[ExcludeFromCodeCoverage]` attribute. The code coverage report is at 100.00% coverage and should remain so.

2.7.5 Continuous Integration / Build Server

All pull requests are built on AppVeyor and code coverage is checked.

2.7.6 Branching

To make a clean pull request, it is important to follow some git best practices. Nancy has an excellent guide that outlines the steps required.

2.7.7 Licensing

The library is licensed under LGPL and by submitting code it is accepted that the submitted code will be released under the same license. Third party code may only be added to the library if the author of the pull request holds the copyright to the code, or the code is previously licensed under a license compatible with LGPL.

2.8 `<sustainsys.saml2>` Element

The `<sustainsys.saml2>` element is a child node of the `<configuration>` element. Its attributes are listed and described below, and its child elements are listed as well and are linked to full explanations of each.

2.8.1 Attributes

`returnUrl` The Url that you want users to be redirected to once the authentication is complete. This is typically the start page of the application, or a special signed in start page.

`entityId` The name that this service provider will use for itself when sending messages. The name will end up in the `Issuer` field in outcoing authnRequests.

The SAML standard requires the entityId to be an absolute URI. Typically it should be the URL where the metadata is presented. E.g. http://sp.example.com/Saml2/.

`discoveryService` (Optional) Specifies an idp discovery service to use if no idp is specified when calling sign in. Without this attribute, the first idp known will be used if none is specified.

`modulePath` (Optional) Indicates the base path of the Saml2 endpoints. Defaults to /Saml2 if not specified. This can usually be left as the default, but if several instances of Saml2 are loaded into the same process they must each get a separate base path.

`authenticateRequestSigningBehavior` (Optional) Sets the signing behavior for generated AuthnRequests. Three values are supported:

- `Never`: Saml2 will never sign any created AuthnRequests.
- `Always`: Saml2 will always sign all AuthnRequests.
- `IfIdpWantAuthnRequestsSigned` (default if the attribute is missing): Saml2 will sign AuthnRequests if the idp is configured for it (through config or listed in idp metadata).

`validateCertificates` (Optional) Normally certificates for the IDPs signing use is communicated through metadata and in case of a breach, the metadata is updated with new data. If you want extra security, you can enable certificate validation (the default value for this attribute is `false`). Please note that the SAML metadata specification explicitly places no requirements on certificate validation, so don't be surprised if an Idp certificate doesn't pass validation.

`publicOrigin` (Optional) Indicates the base url of the Saml2 endpoints. It should be the root path of the application. E.g. The SignIn url is built up as `PublicOrigin + / + modulePath + /SignIn`. Defaults to Url of the current http request if not specified. This can usually be left as the default, but if your internal address of the application is different than the external address the generated URLs (such as `AssertionConsumerServiceURL` in the `saml2p:AuthnRequest`) then this will be incorrect. The use case for this is typically with load balancers or reverse proxies. It can also be used if the application can be accessed by several external URLs to make sure that the registered in metadata is used in communication with the Idp.

If you need to set this value on a per-request basis, provide a `GetPublicOrigin` Notification function instead.

`outboundSignAlgorithm` (Optional) By default Saml2 uses SHA256 signatures if running on .NET 4.6.2 or later or when you have called `GlobalEnableSha256XmlSignatures()`. Otherwise, it uses SHA1 signatures. Use this attribute to set the default signing algorithm for any messages (including metadata) that Saml2 generates. Possible values:

- `SHA1` (or http://www.w3.org/2000/09/xmldsig#rsa-sha1)
- `SHA256`
- `SHA384`
- `SHA512`

The full url identifying the algorithm can also be provided. The algorithm can be overridden for each IdentityProvider too.

`minIncomingSigningAlgorithm` (Optional) The minimum strength required on signatures on incoming messages. Messages with a too weak signing algorithm will be rejected. By default

Saml2 requires SHA256 signatures if running on .NET 4.6.2 or later or when you have called `GlobalEnableSha256XmlSignatures()`. Otherwise, it uses SHA1 signatures.

Possible values:

- `SHA1` (or http://www.w3.org/2000/09/xmldsig#rsa-sha1)
- `SHA256`
- `SHA384`
- `SHA512`

The full url identifying the algorithm can also be provided.

2.8.2 Elements

The following are the possible children elements of the `<sustainsys.saml2>` element. Each are provided as a link below with full explanations of each.

- *nameIdPolicy*
- *requestedAuthnContext*
- *metadata*
- *identityProviders*
- *federations*
- *serviceCertificates*
- *compatibility*

2.9 `<nameIdPolicy>` Element

This is an **optional** child element of the *sustainsys.saml2* element.

This element controls the generation of `NameIDPolicy` element in AuthnRequests. The element is only created if either `allowCreate` or `format` are set to a non-default value.

2.9.1 Attributes

`allowCreate` (Optional) Default value is empty, which means that the attribute is not included in generated AuthnRequests. Supported values are `true` or `false`.

`format` (Optional) Sets the requested format of `NameIDPolicy` for generated authnRequests.

Supported values (see section 8.3 in the SAML2 Core specification for explanations of the values).

- `Unspecified`
- `EmailAddress`
- `X509SubjectName`
- `WindowsDomainQualifiedName`
- `KerberosPrincipalName`
- `EntityIdentifier`

- `Persistent`
- `Transient`

If no value is specified, no format is specified in the generated AuthnRequests. If `Transient` is specified, it is not permitted to specify `allowCreate` (see 3.4.1.1 in the SAML2 Core spec).

2.10 `<requestedAuthnContext>` Element

This is an **optional** child element of the *sustainsys.saml2* element.

2.10.1 Attributes

`classRef` (Optional) Class reference for authentication context. Either specify a full URI to identify an authentication context class, or a single word if using one of the predefined classes in the SAML2 Authentication context specification:

- `InternetProtocol`
- `InternetProtocolPassword`
- `Kerberos`
- `MobileOneFactorUnregistered`
- `MobileTwoFactorUnregistered`
- `MobileOneFactorContract`
- `MobileTwoFactorContract`
- `Password`
- `PasswordProtectedTransport`
- `PreviousSession`
- `X509`
- `PGP`
- `SPKI`
- `XMLDSig`
- `Smartcard`
- `SmartcardPKI`
- `SoftwarePKI`
- `Telephony`
- `NomadTelephony`
- `PersonalTelephony`
- `AuthenticatedTelephony`
- `SecureRemotePassword`
- `TLSClient`
- `TimeSyncToken`

- `unspecified`

`comparison` (Optional) Comparison method for authentication context as signalled in AuthnRequests. Valid values are:

- `Exact` (default)
- `Minimum`
- `Maximum`
- `Better`

`Minimum` is an inclusive comparison, meaning the specified `classRef` or anything better is accepted. `Better` is exclusive, meaning that the specified `classRef` is not accepted.

2.11 `<metadata>` Element

This is an **optional** child element of the *sustainsys.saml2* element.

The metadata part of the configuration can be used to tweak the generated metadata. These configuration options only affect how the metadata is generated, no other behavior of the code is changed.

2.11.1 Attributes

`cacheDuration` (Optional) Describes for how long in anyone should cache the metadata presented by the service provider before trying to fetch a new copy. Defaults to one hour.

Examples of valid format strings:

- 1 day, 2 hours: `1.2:00:00`
- 42 seconds: `0:00:42`

`validDuration` (Optional) Sets the maximum time that anyone may cache the generated metadata. If `cacheDuration` is specified, the remote party should try to reload metadata after that time. If that refresh fails, `validDuration` determines for how long the old metadata may be used before it must be discarded.

In the metadata, the time is exposed as an absolute `validUntil` date and time. That absolute time is calculated on metadata generation by adding the configured `validDuration` to the current time.

Examples of valid format strings:

- 1 day, 2 hours: `1.2:00:00`
- 42 seconds: `0:00:42`

`wantAssertionSigned` (Optional) Signal to IDPs that we want the Assertions themselves signed and not only the SAML response. Saml2 supports both, so for normal usage this shouldn't matter. If set to `false` the entire `wantAssertionsSigned` attribute is dropped from the metadata as the default values is `false`.

2.11.2 Elements

The following are the possible children elements of the `<metadata>` element. Each are provided as a link below with full explanations of each.

- *organization*
- *contactPerson*

- *requestedAttributes*

2.12 <organization> Element

Optional child element of the *<metadata> Element* element.

Provides information about the organization supplying the SAML2 entity (in plain English that means the organization that supplies the application that Saml2 is used in).

2.12.1 Attributes

name The name of the organization.

displayName The display name of the organization.

url URL to the organization's web site.

language In the generated metadata, the `name`, `displayName` and `url` attributes have a language specification. If none is specified, the `xml:lang` attribute will be generated with an empty value.

2.13 <contactPerson> Element

Optional child element of the *<metadata> Element* element.

2.13.1 Attributes

type The type attribute indicates the type of the contact and is picked from the `ContactType` enum. Valid values are:

- `Administrative`
- `Billing`
- `Other`
- `Support`
- `Technical`

company (Optional) Name of the person's company.

givenName (Optional) Given name of the contact person.

surname (Optional) Surname of the contact person.

phoneNumber (Optional) Phone number of the contact person. The SAML standard allows multiple phone number to be specified. Saml2 supports that, but not through the configuration file.

email (Optional) Email address of the person. The SAML standard allows multiple email addresses to be specified. Saml2 supports that, but not through the configuration file.

2.14 <requestedAttributes> Element

Optional child element of the *<metadata> Element* element.

List of attributes that the SP requests to be included in the assertions generated by an identity provider. Each attribute is added to the list with an <add> element.

The element should look something like this:

```
<requestedAttributes>
    <add name="" friendlyName="" nameFormat="" isRequired=""/>
    <add name="" friendlyName="" nameFormat="" isRequired=""/>
    ...
</requestedAttributes>
```

2.14.1 Attributes

name The name of the attribute. This is usually in the form of an urn/oid, e.g. `urn:oid:1.2.3`. The format of the name should be specified in the `nameFormat` attribute.

friendlyName (Optional) An optional friendly (i.e. human readable) friendly name of the attribute that will be included in the metadata. Please note that the SAML2 standard specifically forbids the friendlyName to be used for anything other than information to a human. All matching of attributes must use the `name`.

nameFormat (Optional) Format of the name attribute. Valid values are:

- `urn:oasis:names:tc:SAML:2.0:attrname-format:uri`
- `urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified`
- `urn:oasis:names:tc:SAML:2.0:attrname-format:basic`

isRequired (Optional) `true` or `false` value indicating whether the attribute is required by the service provider or just a request that it would be nice if the Idp includes it.

2.15 <identityProviders> Element

This is an **optional** child element of the *sustainsys.saml2* element.

It indicates a list of identity providers known to the service provider.

Each identity provider is added as an <add> element to the <identityProviders> element and the element will end up looking something like what is shown below. Note the possible child element of the <signingCertifcate> which is shown in the second added identity provider element below.

```
<identityProviders>
    <add entityId="" signOnUrl="" logoutUrl="" allowUnsolicitedAuthnResponse="" binding=""
        wantAuthnRequestsSigned="" loadMetadata="" metadataLocation="" disableOutboundLogoutRequests="" outboundSigningAlgorithm=""/>
    <add entityId="" signOnUrl="" logoutUrl="" allowUnsolicitedAuthnResponse="" binding=""
        wantAuthnRequestsSigned="" loadMetadata="" metadataLocation="" disableOutboundLogoutRequests="" outboundSigningAlgorithm="">
        <signingCertificate storeName="" storeLocation="" findValue="" x509FindType="" />
```

(continues on next page)

(continued from previous page)

```
    </add>
    ...
</identityProviders>
```

2.15.1 Attributes

entityId The issuer name that the idp will be using when sending responses. When `<loadMetadata>` is enabled, the entityId is treated as a URL to for downloading the metadata.

signOnUrl (Optional) The url where the identity provider listens for incoming sign on requests. The url has to be written in a way that the client understands, since it is the client web browser that will be redirected to the url. Specifically, this means that using a host name only url or a host name that only resolves on the network of the server won't work.

logoutUrl (Optional) The url where the identity provider listens for incoming logout requests and responses. To enable single logout behaviour there must also be a service certificate configured in Saml2 as all logout messages must be signed.

allowUnsolicitedAuthnResponse Allow unsolicited responses. That is, Idp initiated sign on where there was no prior AuthnRequest. If true `InResponseTo` is not required and the IDP can initiate the authentication process. If false `InResponseTo` is required and the authentication process must be initiated by an AuthnRequest from this SP. Note that if the authentication was SP-intiatied, `RelayState` and `InResponseTo` must be present and valid.

binding (Optional) The binding that the services provider should use when sending requests to the identity provider. One of the supported values of the `Saml2BindingType` enum.

- `HttpRedirect`
- `HttpPost`
- `Artifact`

wantAuthnRequestsSigned (Optional) Specifies whether the Identity provider wants the AuthnRequests signed. Defaults to `false`.

loadMetadata (Optional) Load metadata from the idp and use that information instead of the configuration. It is possible to use a specific certificate even though the metadata is loaded, in that case the configured certificate will take precedence over any contents in the metadata.

metadataLocation (Optional) The SAML2 metadata standard strongly suggests that the `Entity Id` of a SAML2 entity is a URL where the metadata of the entity can be found. When loading metadata for an idp, Saml2 normally interprets the EntityId as a url to the metadata. If the metadata is located somewhere else it can be specified with this configuration parameter. The location can be a URL, an absolute path to a local file or an app relative path (e.g. `~/App_Data/IdpMetadata.xml`)

disableOutboundLogoutRequests (Optional) Disable outbound logout requests to this idp, even though Saml2 is configured for single logout and the idp supports it. This setting might be usable when adding SLO to an existing setup, to ensure that everyone is ready for SLO before activating.

outboundSigningAlgorithm (Optional) By default Saml2 uses SHA256 signatures if running on .NET 4.6.2 or later and otherwise SHA1 signatures. Set this to set the signing algorithm for any outbound messages for this identity provider. Possible values:

- `SHA1`
- `SHA256`
- `SHA384`

- SHA512

2.15.2 Elements

The following are the possible children elements of the <identityProviders> element. Each are provided as a link below with full explanations of each.

- *signingCertificate*

2.16 <signingCertificate> Element

Optional element of the *identityProvider* element.

The certificate that the identity provider uses to sign its messages. The certificate can either be loaded from file if the fileName attribute is specified or from a certificate store if the other attributes are specified. If a fileName is specified that will take precedence and the other attributes will be ignored.

Warning: File-based certificates are only recommended for testing and during development. In production environments it is better to use the certificate store.

2.16.1 Attributes

fileName A file name to load the certificate from. The path is relative to the execution path of the application. Make sure to heed the warning above – *best to use store-based certificates for non-development environments.*

storeName Name of the certificate store to search for the certificate. It is recommended to keep the certificate of the identity provider in the "Other People" store which is specified by the AddressBook enum value. Valid values are those from the System.Security.Cryptography.X509Certificates.StoreName enumeration.

storeLocation The location of the store to search for the certificate. On production services it is recommended to use the LocalMachine value, while it makes more sense to use CurrentUser in development setups. Valid values are those from the System.Security.Cryptography.X509Certificates.StoreLocation enumeration.

findValue A search term to use to find the certificate. The value will be searched for in the field specified by the x509FindType attribute.

x509FindType The field that will be seach for a match to the value in findValue. For security, it is recommended to use FindBySerialNumber.

Valid values are those from the System.Security.Cryptography.X509Certificates.X509FindType enumeration.

Warning: There is a nasty bug when copying a serial number from the certificate info displayed by certificate manager and the browser. There is a hidden character before the first hex digit that will mess up the matching. Once pasted into the config, use the arrow keys to make sure that there is not an additional invisible character at the start of the serial number string.

2.17 <federations> Element

This is an **optional** child element of the *sustainsys.saml2* element.

This element contains a list of federations that the service provider knows and trusts.

As with some other elements, individual items are added via an <add> element inside this element, so you'll end up with XML that looks like the following:

```
<federations>
    <add metadataLocation="" allowUnsolicitedAuthnResponse="" />
    <add metadataLocation="" allowUnsolicitedAuthnResponse="" />
    ...
</federations>
```

2.17.1 Attributes

metadataLocation URL to the full metadata of the federation. Saml2 will download the metadata and add all identity providers found to the list of known and trusted identity providers. The location can be a URL, an absolute path to a local file or an app relative path (e.g. `~/App_Data/IdpMetadata.xml`)

allowUnsolicitedAuthnResponse `true` or `false` value indicating whether unsolicited authn responses should be allowed from the identity providers in the federation.

2.18 <serviceCertificates> Element

This is an **optional** child element of the *sustainsys.saml2* element.

Specifies the certificate(s) that the service provider uses for encrypted assertions (and for signed requests, once that feature is added). If neither of those features are used, this element can be ommitted.

The public key(s) will be exposed in the metadata and the private key(s) will be used during decryption/signing.

Individual certificates are added via an <add> element, so the resulting XML will be similar to the following:

```
<serviceCertificates>
    <add use="" status="" metadataPublishOverride="" />
    <add use="" status="" metadataPublishOverride="" />
    ...
</serviceCertificates>
```

2.18.1 Attributes

use Indicates how the certificate will be used. Options are:

- `Signing`
- `Encryption`
- `Both` (default)

status Indicates whether the certificate is a current or future certificate – used in key rollover scenarios. Options are:

- `Current` (default)

- `Future`

`metadataPublishOverride` By default the certificate will be used and published by the rules shown in the table below. To override this behavior choose one of the following options for this attribute:

- `None` (Default) - published according to the rules in the table below.
- `PublishUnspecified`
- `PublishEncryption`
- `PublishSigning`
- `DoNotPublish`

Use	Status	Published in Metadata	Used by Saml2
Both	Current	Unspecified *unless Future key exists*, then Signing	Yes
Both	Future	Unspecified	For decryption only
Signing	Current	Signing	Yes
Signing	Future	Signing	No
Encryption	Current	Encryption *unless Future key exists* then not published	Yes
Encryption	Future	Encryption	Yes

2.19 `<compatibility>` Element

This is an **optional** child element of the *sustainsys.saml2* element.

Enables overrides of default behaviour to increase compatibility with identity providers.

2.19.1 Attributes

`unpackEntitiesDescriptorInIdentityProviderMetadata` (Optional) If an `EntitiesDescriptor` element is found when loading metadata for an IdentityProvider, automatically check inside it if there is a single `EntityDescriptor` and in that case use it.

`IgnoreAuthenticationContextInResponse` Do not read the `AuthnContext` element in Saml2Response. If you do not need these values to be present as claims in the generated identity, using this option can prevent XML format errors (`System.Xml.XmlException: ID0013: The value must be an absolute URI`), when value cannot parse as absolute URI.