Single Sign-On from Active Directory to a Windows Azure Application

December 16, 2010

Authors: Vittorio Bertocci, David Mowers

Reviewers: Stuart Kwan, Paul Beck

Abstract

This paper contains step-by-step instructions for using Windows® Identity Foundation, Windows Azure, and Active Directory Federation Services (AD FS) 2.0 for achieving SSO across web applications that are deployed both on premises and in the cloud. Previous knowledge of these products is not required for completing the proof of concept (POC) configuration. This document is meant to be an introductory document, and it ties together examples from each component into a single, end-to-end example.

Contents

Introduction

Objectives

The purpose of this white paper is to help developers enable single sign-on (SSO) between Active Directory® Domain Services (AD DS) and cloud applications running on the Microsoft® Windows Azure™ platform.

This paper contains step-by-step instructions for using Windows® Identity Foundation, Windows Azure, and Active Directory Federation Services (AD FS) 2.0 for achieving SSO across web applications that are deployed both on premises and in the cloud. Previous knowledge of these products is not required for completing the proof of concept (POC) configuration. This document is meant to be an introductory document, and it ties together examples from each component into a single, end-to-end example. The "Further Reading and Resources" section lists articles and hands-on labs that offer more detailed coverage of the topics that are presented here.

In This Paper

- Scenario
- Technology Overview
- Creating the On-Premise Infrastructure
- Adding Single Sign-On to the HelloCloudSSO Application
- Additional Considerations for Production Development
- Summary
- Further Reading and Resources

Scenario

When an application developer is given the task of looking at the cloud for application hosting, he or she must look for a solution that provides SSO for both internal users on the organization's intranet and mobile users who are accessing the application over the Internet. The scenario might look like the following illustration.

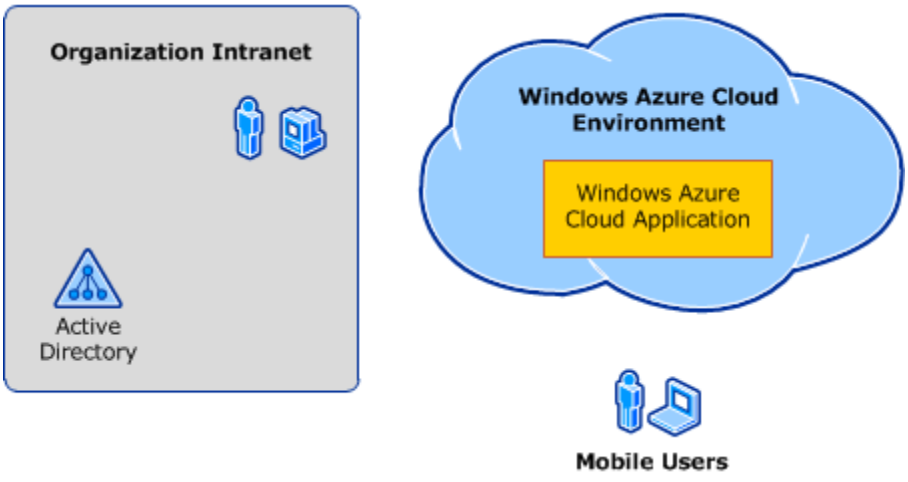

Figure 1 The SSO challenge—cloud version

This scenario assumes the following use cases:

- Corporate users access internal applications while logged in using Active Directory.
- Corporate users access Windows Azure–deployed applications while logged in using Active Directory.
- Mobile corporate users access Windows Azure–deployed applications using Active Directory accounts.

Two common challenges for such a scenario are:

- How to provide the SSO user experience.
- How to provide information about the user so that the application can make authorization decisions.

This paper explains how to address these challenges for the described use cases.

Overview of Web Authentication and Authorization

Before diving into the architecture of cloud application authentication and authorization, it is useful to review developer best practices for internal web applications. Intranet web applications that are hosted on Internet Information Services (IIS) most commonly use Windows Integrated Authentication. Windows Integrated Authentication specifies that Kerberos version 5 (V5) or NTLM authentication is used to validate the application user identity. When the Windows security system performs this type of authentication, it provides the identity of the user and it also provides the set of groups from both the local server and Active Directory that the user is a member of. Active Directory groups are roughly equivalent to the concept of enterprise roles and are often used as such in application authorization logic.

If the internal web application is an ASP.NET application, Active Directory group membership is commonly checked using the ASP.NET **IsInRole()** method:

```
if ( User.IsInRole("domain\\domaingroup") )
   DoRoleProtectedFunction();
```

For the developer who is writing a new application for the cloud—or porting an application to run in the cloud—the good news is that the interfaces for evaluating role membership do not change. Later in this paper, we discuss how AD FS 2.0 configuration affects the format of the Active Directory group (role) claims that AD FS 2.0 delivers.

Beyond Groups and Roles

The claims-based identity model makes it possible for the developer to easily access other information about the user, beyond group memberships. This concept should be familiar to internal application

developers because many applications require additional information about users. This information is commonly stored as attribute information. It might be stored in a Lightweight Directory Access Protocol (LDAP) directory, such as Active Directory, or in a Structured Query Language (SQL) database. Therefore, along with accessing group information using **IsInRole()**, the internal application developer accesses information ,such as the following:

- Employee ID
- Organizational information, such as Department or Group
- Job title or level
- E-mail address
- Building, office, or other geographical location

Other than the Microsoft .NET Framework classes that conveniently abstract access to LDAP directories or SQL databases, there are no real standard mechanisms for accessing such information. Claims-based identity changes the playing field and creates a standard way to make this information available to applications. A "claim" is a simple mechanism that can be used to deliver identity-relevant information about a user, including roles, permissions, rights, and even general information, such as a birth date. Whether the claim comes from an LDAP directory or a SQL database, the claim itself is delivered to the application in the authentication token and is accessed through simple programmatic interfaces. This is a huge benefit for the application developer—potentially eliminating hundreds of lines of code.

Arguably of more importance is the fact that claims-based identity moves the process of accessing attribute information from the application to administrative operations at the infrastructure level. To get a better idea of the advantages of this approach, consider the following scenario:

- An existing application retrieves a certain attribute value from an LDAP directory.
- The organization changes its identity provisioning process and decides to make this attribute available to applications in a SQL database instead.
- Application developers (possibly, for many applications) are then forced to add many lines of SQL code to access the attribute value.

With a claims-based identity infrastructure, such as AD FS 2.0, the application changes are unnecessary. A simple administrator action changes the claim source from LDAP to SQL.

We have just discussed several reasons why claims-based authorization is preferable, even in intranet environments. We now consider cloud-based applications and see that, where in the intranet environment a claims-based infrastructure is better than the alternatives, in the cloud it is required.

Solution Approach—AD FS 2.0, WIF, and Windows Azure

After your organization makes the decision to deploy applications to the cloud using Windows Azure, one obvious question is how to take advantage of the identities that are based in Active Directory when the organization is using the cloud application. Ideally, the solution should provide SSO, as well as consistency with regard to authorization information. Fortunately, this goal is easy to achieve using AD FS 2.0) and WIF.

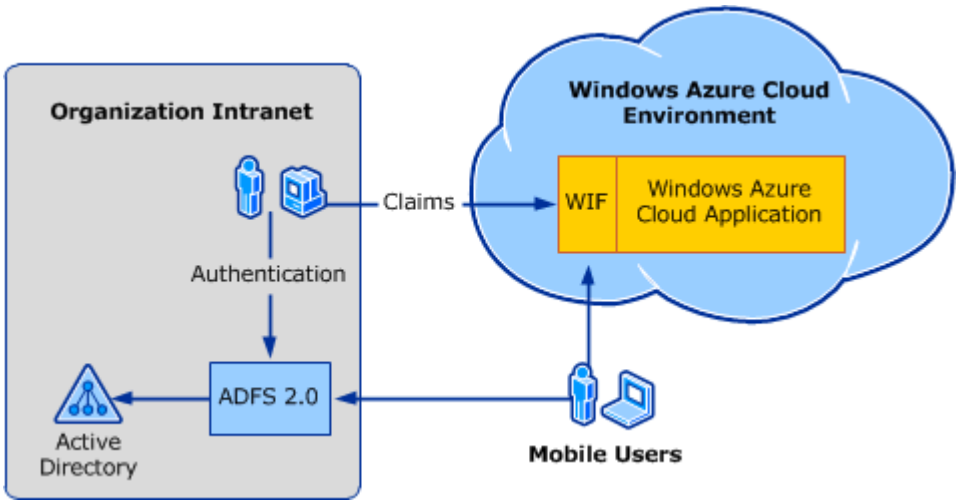

Figure 2 SSO using AD FS 2.0 applied to a solution in which the application runs in the cloud and the users come from the existing directory infrastructure

In this scenario, the application is hosted in the cloud using Windows Azure, but its users are the accounts that are maintained in on-premises Active Directory. If the application tries to authenticate users directly through Kerberos or NTLM, it fails, because of the networking boundaries that separate it from the domain controller. Adding AD FS 2.0 infrastructure and WIF makes it possible for the application to use claims-based identity and makes it possible to instantly reuse existing accounts, even if the application is deployed in the cloud.

A major advantage of the proposed solution is that it also can be extended to other scenarios, including identity federation with identity providers, such as Windows Live ID, and other organizations with federation infrastructure that implements the WS-Federation or Security Assertion Markup Language (SAML) 2.0 industry-standard federation protocols. Developers who are interested in this scenario should review the Access Control Service Samples and Documentation (Labs) (http://go.microsoft.com/fwlink/?LinkId=207806).

Technology Overview

AD FS 2.0

AD FS 2.0 is a key component in cloud application security. In its role as a security token service (STS), it authenticates users and generates authorization information. It does this by implementing web-compatible protocols, including WS-Federation and SAML 2.0. AD FS 2.0 is an infrastructure component. It is typically managed by information technology (IT) staff and not by developers. Developers should understand enough about AD FS 2.0 that they can communicate requirements for claims that are required by the applications that they are developing.

WIF

WIF is a developer framework that enhances the .NET Framework capabilities. WIF integrates seamlessly with the existing mechanisms that the .NET Framework offers for working with identity, the interfaces **IPrincipal** and **IIdentity**. WIF extends those interfaces with **IClaimsPrincipal** and **IClaimsIdentity**, respectively. **IClaimsPrincipal** and **IClaimsIdentity** can still take advantage of existing access control mechanisms, such as **IsInRole()** and the **<authorization>** element. Furthermore, they contain additional members that carry the claim collections that are extracted from the incoming authentication token that AD FS 2.0 generates.

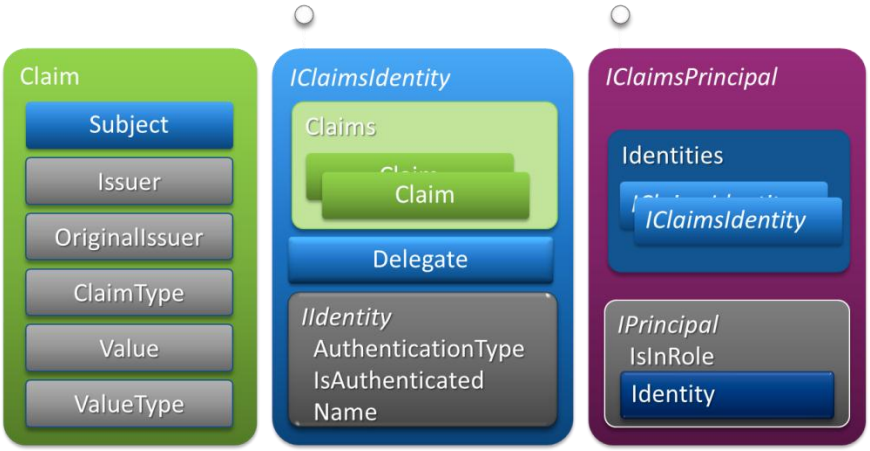

Figure 3 The WIF claims object model

Creating the On-Premise Infrastructure

You can implement this scenario on the Microsoft platform by doing the following:

- Installing AD FS 2.0 on one intranet server
- Using WIF for securing your Windows Azure Web Role application
- Establishing the trust relationship between the application and AD FS 2.0 and configuring the appropriate claims

The rest of this paper provides guidance about how to perform these three activities.

Preparing the Server Hosts

To complete this proof of concept, you use two server instances. One server instance is configured as a domain controller for a stand-alone domain. The environment that is described this paper uses the fictitious namespace of Fabrikam. The distinguished name of the domain controller is Fabrikam-DC.fabrikam.com.

The second server instance serves multiple roles, including the role of host for the Windows Azure application development environment and for AD FS 2.0. This server is named Fabrikam-ADFS.fabrikam.com. It is referred to throughout this paper as Fabrikam-ADFS. In a production deployment, these three roles would be performed on two different servers and a workstation.

Installing Microsoft Visual Studio® 2010

This paper assumes that you are using Visual Studio 2010 as your development environment. Install Visual Studio 2010 Professional on your development computer. This can be a separate instance from the AD FS 2.0 server, but for the proof of concept described in this paper it can just as easily be the same computer.

Creating a Windows Azure Application

Follow steps 1 through 4 at Windows Azure (http://go.microsoft.com/fwlink/?LinkId=207808) to set up the Windows Azure development environment, get an account, and create and test a basic "HelloCloud" web application. You then integrate this application with the AD FS 2.0 infrastructure that is created in the following steps.

Note: This paper refers to both the Visual Studio solution for the Windows Azure application and the Windows Azure service as "HelloCloudSSO". Because all Windows Azure service names must be globally unique, you must choose a different Windows Azure service name. You can choose to use HelloCloudSSO as the Visual Studio solution name or have it match your Windows Azure service name.

Install and Configure AD FS 2.0 and the WIF SDK

The document AD FS 2.0 Federation with a WIF Application Step-by-Step Guide (http://go.microsoft.com/fwlink/?LinkId=179631) walks you through the installation and configuration of a basic AD FS 2.0 environment. Use this guide, and complete all the procedures through step 2 and the portion of step 3 that directs you to install the WIF Software Development Kit (SDK).

Configure the Windows Azure Application to Use SSO

Now that you have the HelloCloudSSO application and AD FS 2.0 set up, you can integrate the two by establishing a trust. This enables SSO to the application using an Active Directory account. After this step is complete, you modify the application to use WIF to use and display the claim information that is sent in the authenticating token.

Before you publish an SSO version of HelloCloudSSO to Windows Azure, you first configure a trust with AD FS 2.0 and then test it locally using the development fabric. The development fabric simply simulates the Windows Azure fabric on the local computer; therefore, it is a useful tool to make sure everything is working right. For more complex web applications, it is a requirement that developers be able to test as they develop using the dev fabric; therefore, this means that they will need to have authentication and authorization working there as well.

Configure HelloCloudSSO to Use SSL

SSO requires that the web application use HTTPS to securely transport security tokens. It is possible to configure everything to not use HTTPS. However, it is not worth going through the extra steps because you will have to switch eventually.

The first step is to create and install a suitable certificate. For the proof-of-concept environment, a self-signed certificate is the best option. While you are logged on as a system administrator, use the

makecert tool (which is available in the Microsoft Windows Software Development Kit (http://go.microsoft.com/fwlink/?linkid=84091)) with the following arguments to create the certificate:

makecert -r -pe -n "CN=hellocloudsso.fabrikam.com" -b 01/01/2010 -e 01/01/2036 -eku 1.3.6.1.5.5.7.3.1 -ss my -sr localMachine -sky exchange -sp "Microsoft RSA SChannel Cryptographic Provider" -sy 12

Note that you can substitute a different Domain Name System (DNS) address for hellocloudsso.fabrikam.com.

In Windows Azure language, the way to enable the application to use Secure Sockets Layer (SSL) is to add an HTTPS endpoint. The process to add an HTTPS endpoint is a three-step process:

1. Configure the endpoint.
2. Upload the certificate to the cloud.
3. Configure the SSL certificate, and then point the endpoint to that certificate.

To make the dev fabric function correctly, you only have to perform step 1 for now.

Configure the Endpoint

To configure the endpoint, in Solution Explorer, right-click **WebRole1**, and then click **Properties**.

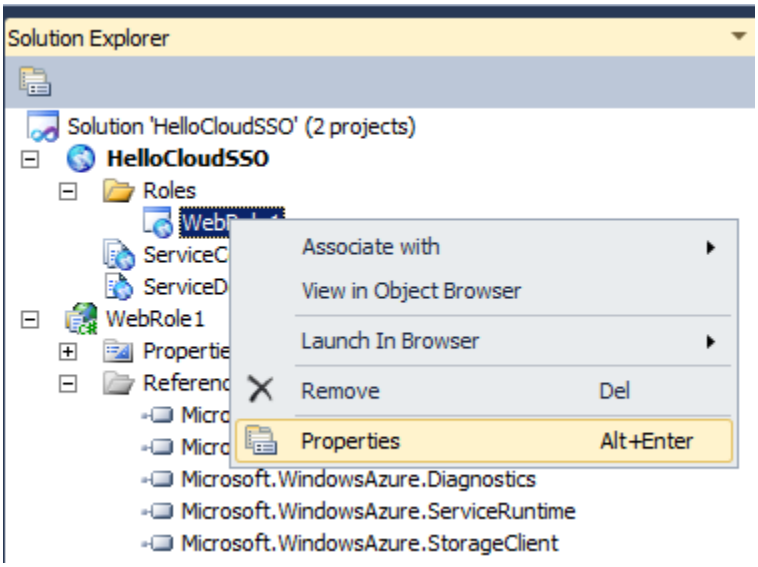

Figure 4 Properties

In the **Properties** window, click **Certificates**, and then click **Add Certificate**. Provide a friendly name for the certificate, and then click the **…** button. Select the certificate that you created in the previous step.

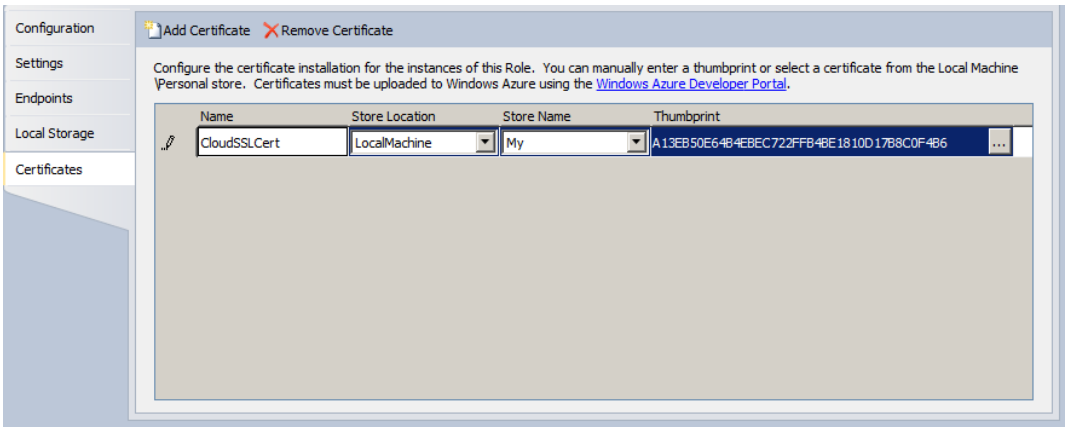

Figure 5 Add an SSL certificate to the HTTPS endpoint

Click **Endpoints**, and then select the **HTTPS** check box. Select the certificate that you just added to the endpoint in the **SSL certificate name** drop-down list.

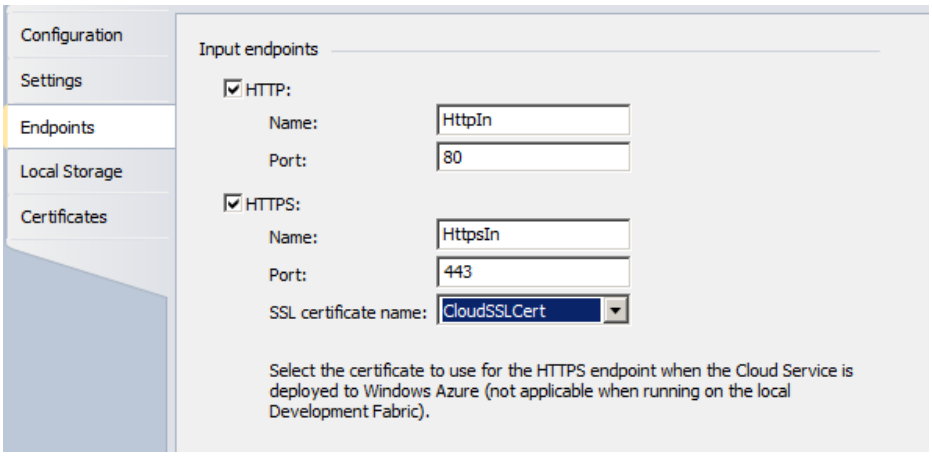

Figure 6 Endpoints configuration

Click **Configuration**, and clear the **Launch browser for: HTTP endpoint** check box. On running or debugging of the application in the dev fabric, the default browser will be launched only for the HTTPS endpoint.

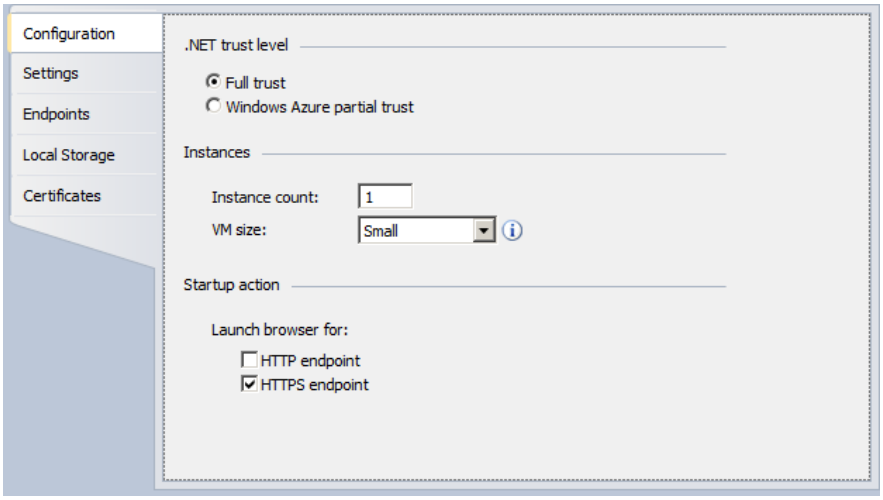

Figure 7 Launch browser for HTTPS endpoint configuration

At this point you can run the application from Visual Studio. The browser opens to the HTTPS endpoint, for example, https://127.0.0.1:449. Note the port number that is used; you will need this number in a subsequent step. If Internet Explorer® is your default browser, it displays a certificate warning because the self-signed certificate that the Windows Azure tool created is not trusted.

Add SSO to the HelloCloudSSO Application

To add SSO to the HelloCloudSSO application, do the following:

- Add a hosts file mapping for the dev fabric application URL
- Add an STS reference to the application
- Add a relying party trust to AD FS 2.0
- Configure claim rules for the application
- Add a request validator
- Configure the application to use WIF and display claims

Add a Hosts File Mapping for the Dev Fabric Environment

To enable consistent access to the HelloCloudSSO application, you create a mapping in the local hosts file to map a friendly application name to the dev fabric localhost IP address.

On the development computer, open the /windows/system32/drivers/etc/hosts file, and add the following line using the localhost IP address and the friendly DNS name of the application:

```
# name mapping for Windows Azure Dev Fabric
127.0.0.1    hellocloudsso.fabrikam.com
```

Save the hosts file.

Add an STS Reference to the Application

In Visual Studio, with the HelloCloudSSO project loaded, right-click the **HelloCloudSSO** project, and then click **Add STS Reference**.

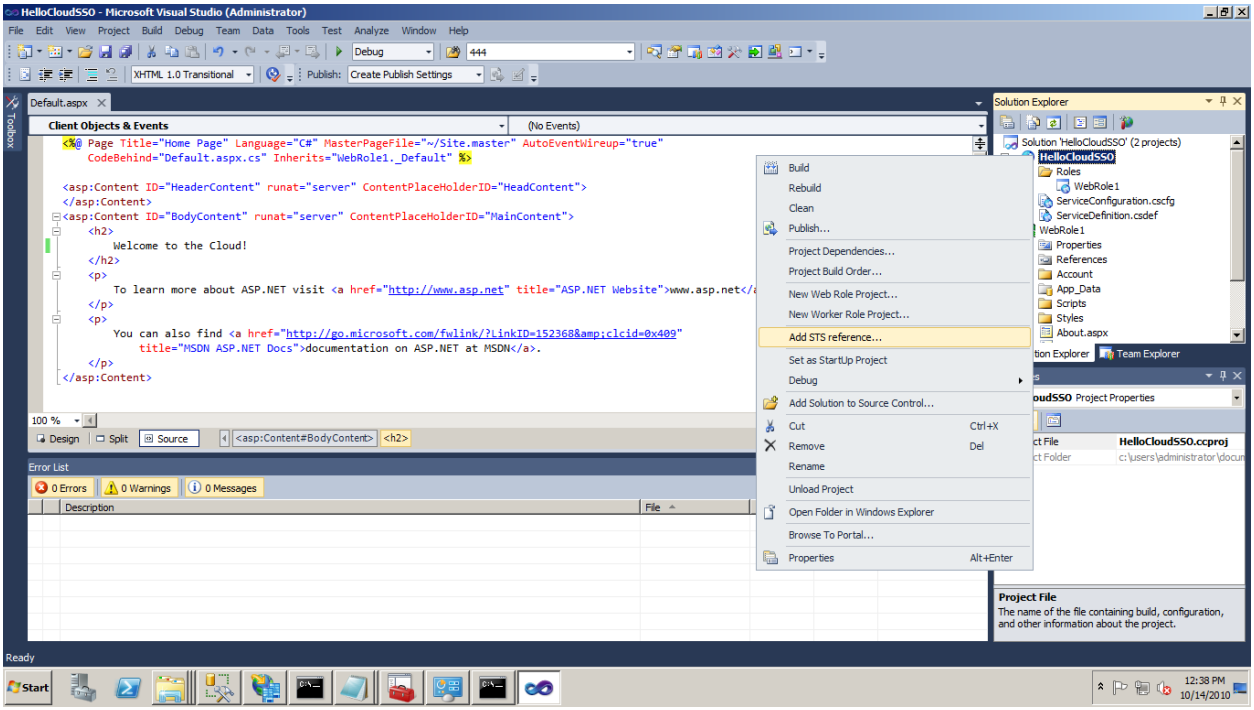

Figure 8 Add an STS reference in Visual Studio 2010

The Federation Utility wizard (FedUtil) starts. Now, you configure a trust relationship between the HelloCloudSSO application and the AD FS 2.0 server. In the Federation Utility Wizard, provide the path to the application's web.config file and the application URI that was composed using the friendly application name and the dev fabric port number.

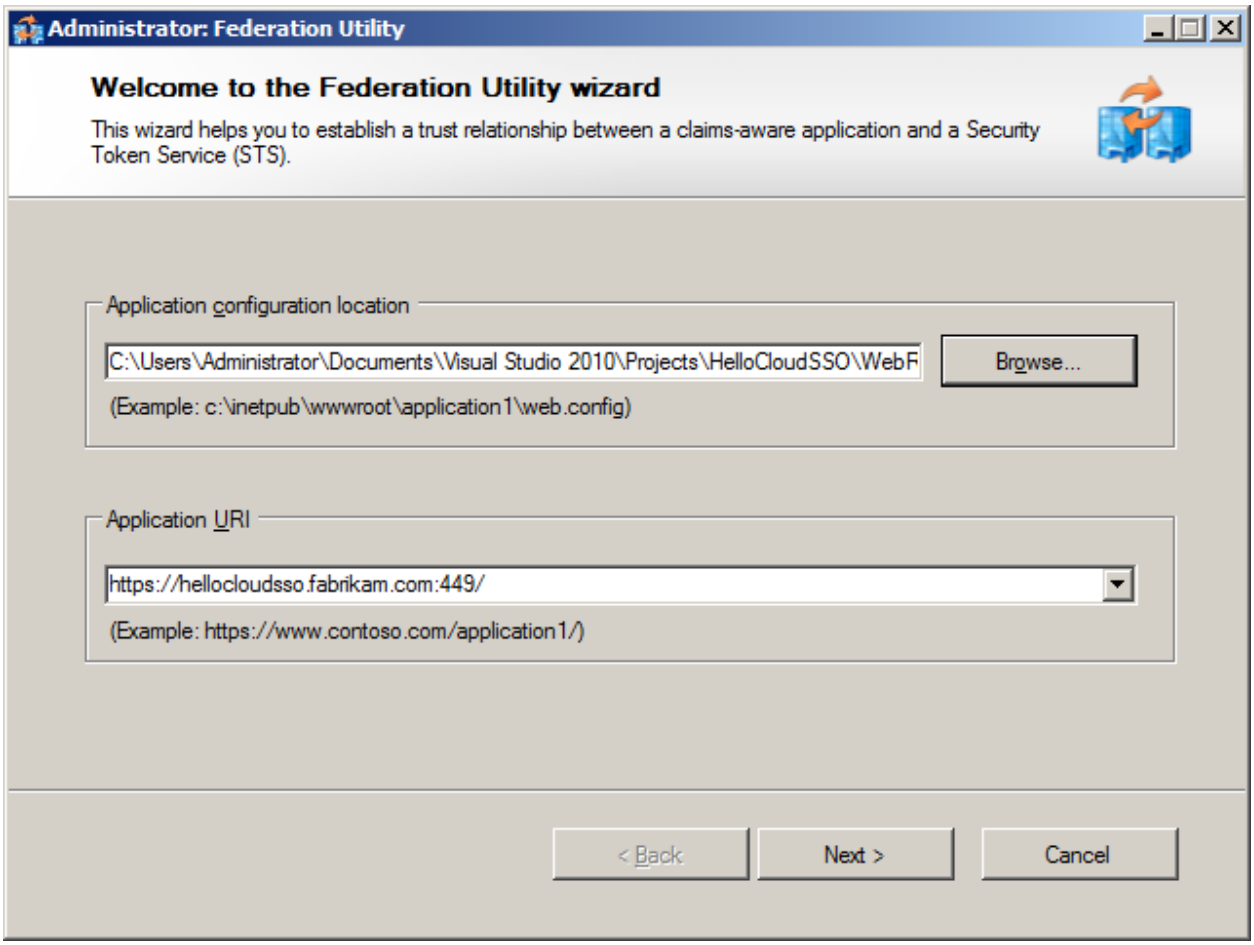

Figure 9 Federation Utility wizard—application configuration

To configure the STS, click **Next**.

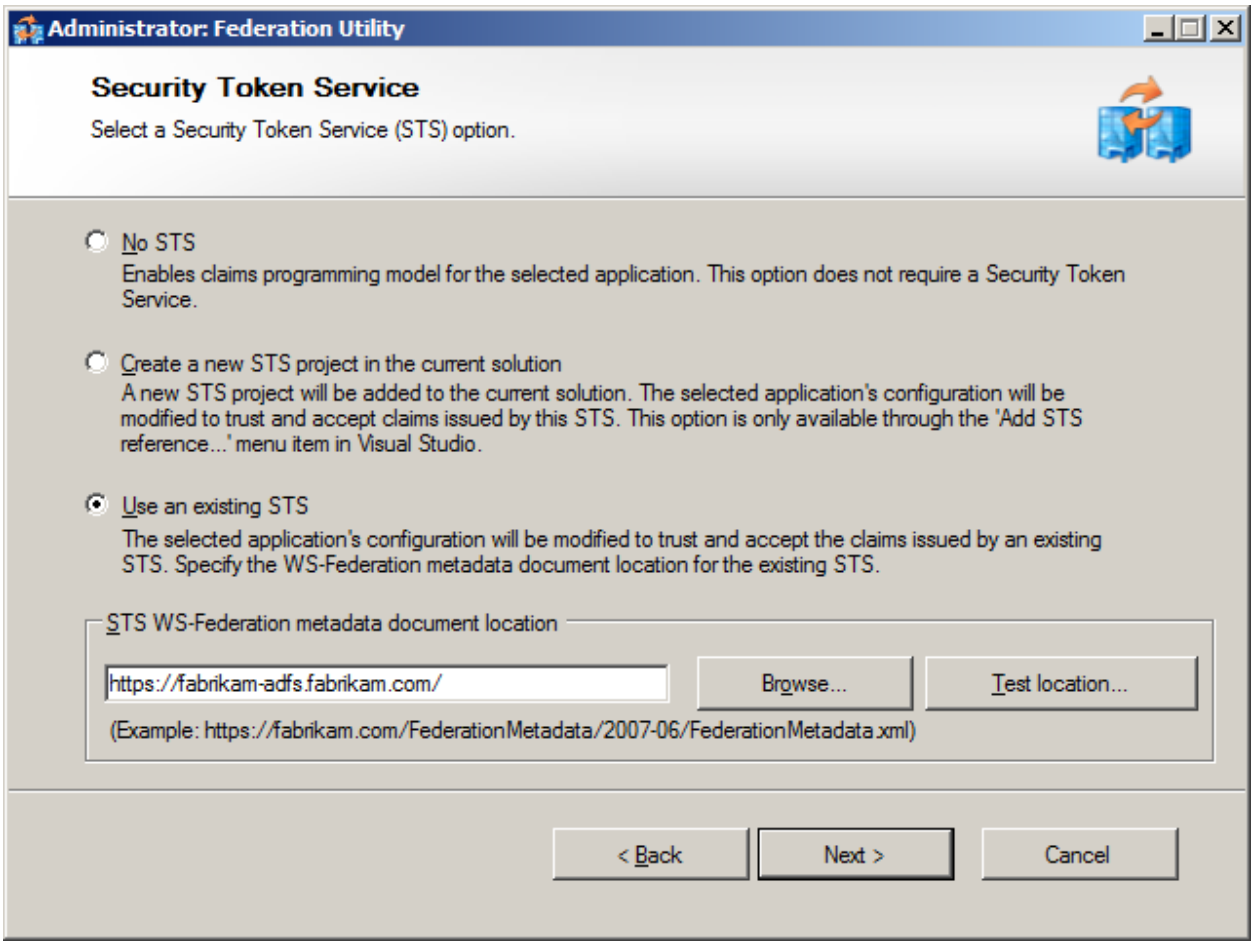

Figure 10 Federation Utility wizard—specify STS

Click **Use an existing STS**, and then provide the URL of the AD FS server in your environment. To locate and validate the URL of the federation metadata document, click **Test location**. To configure certificate chain validation, click **Next**.

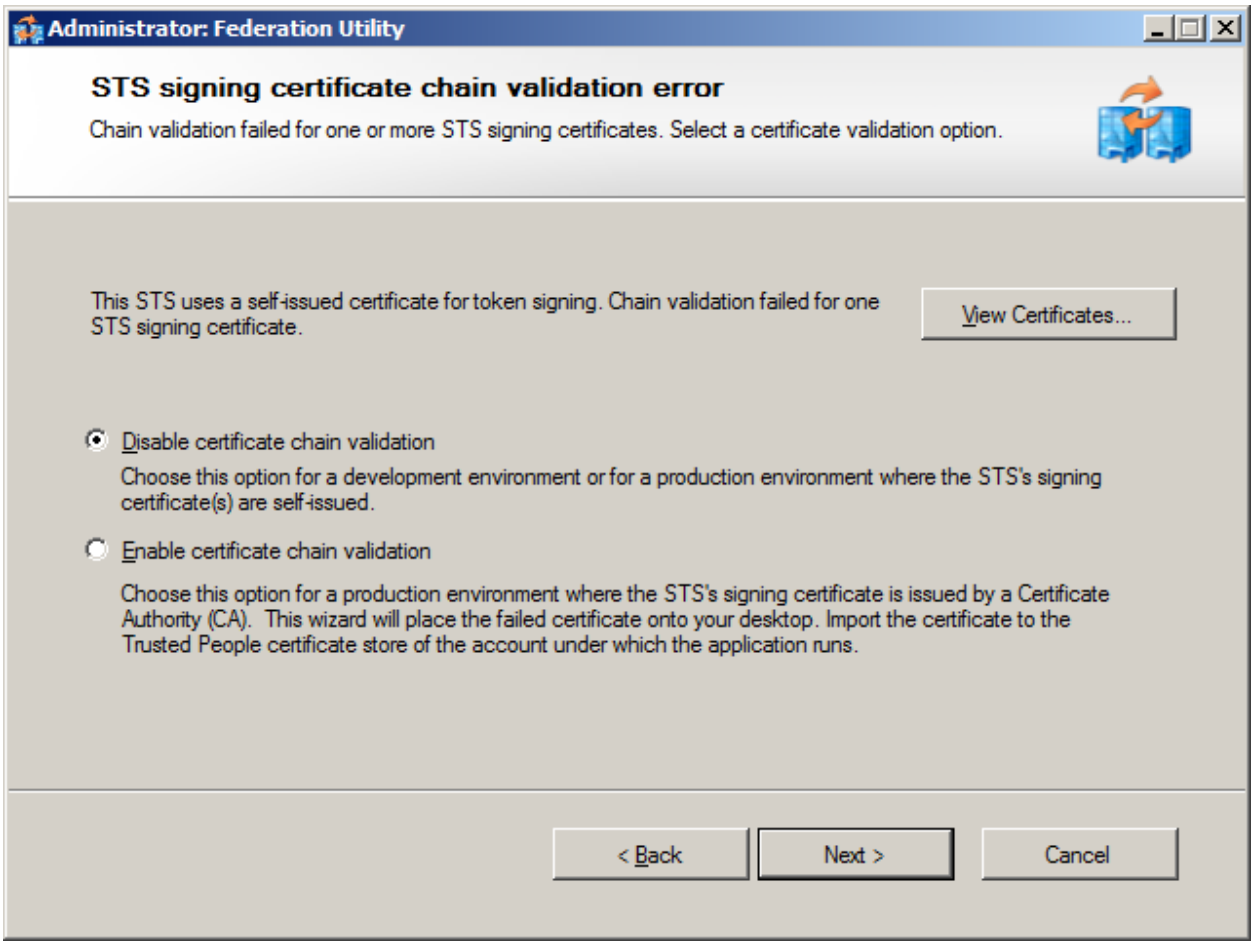

Figure 11 Federation Utility Wizard—certificate chain validation

Click **Disable certificate chain validation,** and then click **Next**.

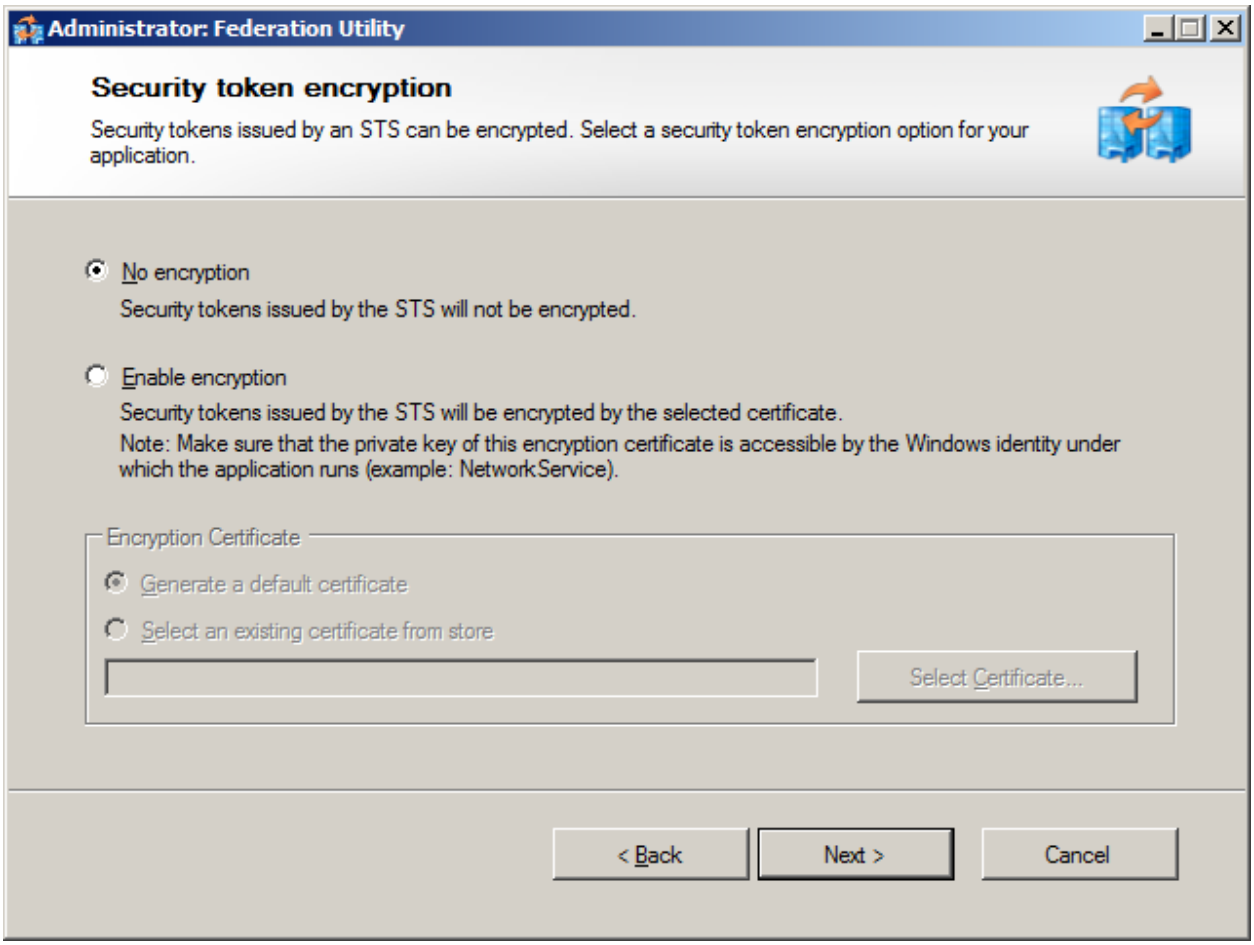

Figure 12 Federation Utility Wizard – Security token encryption

Click **No encryption,** and then click **Next** to complete the Federation Utility Wizard.

Add a Relying Party Trust to AD FS

If you run the HelloCloudSSO application in the dev fabric environment at this point, it will exhibit a change in behavior. When the application opens, the Windows Security dialog box appears and prompts for credentials. If you enter valid credentials, the following message will appear.

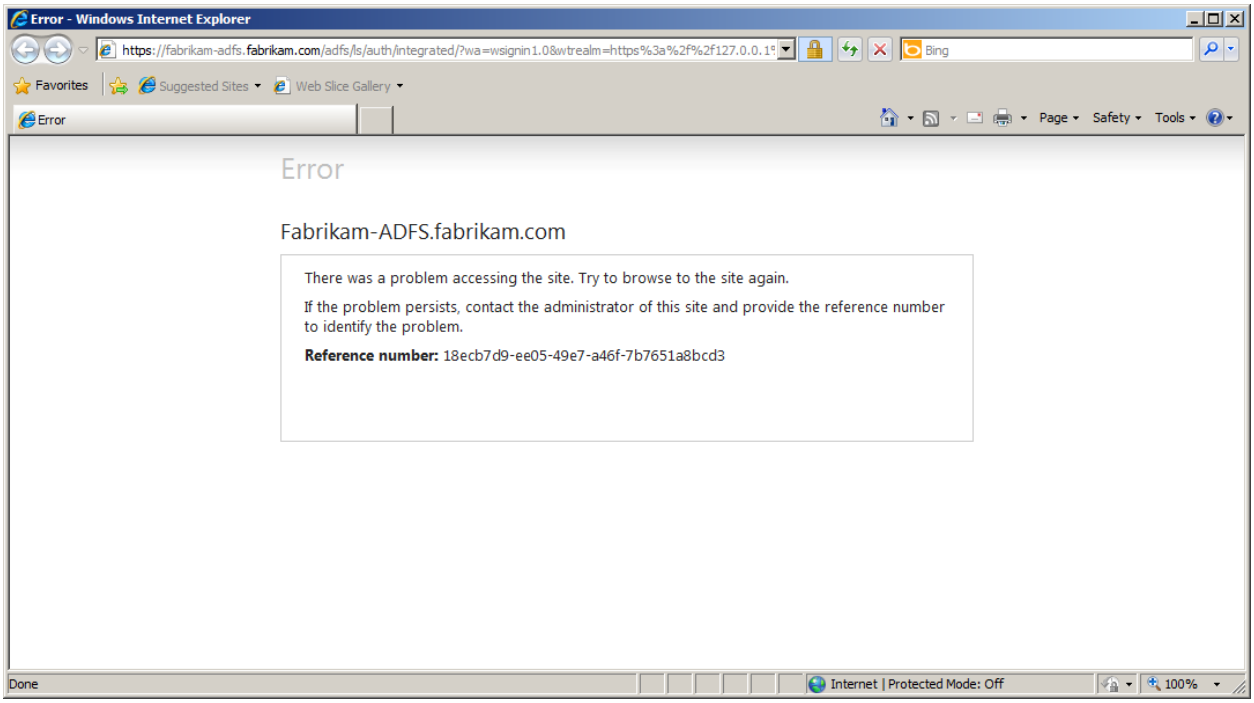

Figure 13 The WIF application error when AD FS RP is not configured

This error is generated because you have not configured AD FS with a relying party that represents the HelloCloudSSO application in the dev fabric. Note that the URL is for the Integrated authorization page on the AD FS server. The HelloCloudSSO application redirects the browser based on changes to the HelloCloudSSO web.config that you made by adding the STS reference to the HelloCloudSSO project.

To add a relying party trust to AD FS

1. In the AD FS 2.0 Management console, click **AD FS 2.0**, expand **Trust Relationships**, and then right-click **Relying Party Trusts.** To start the **Add Relying Party Wizard**, click **Add Relying Party Trust**.

2. On the **Welcome** page, click **Start**.

3. On the **Select Data Source** page, click **Import data about the relying party from a file.**

4. Click **Browse**, navigate to the location of the applications federation metadata document, which is typically found in the application project folder: \WebRole1\FederationMetadata\2007-2006\FederationMetadata.xml. Select the metadata document, and then click **Open**.

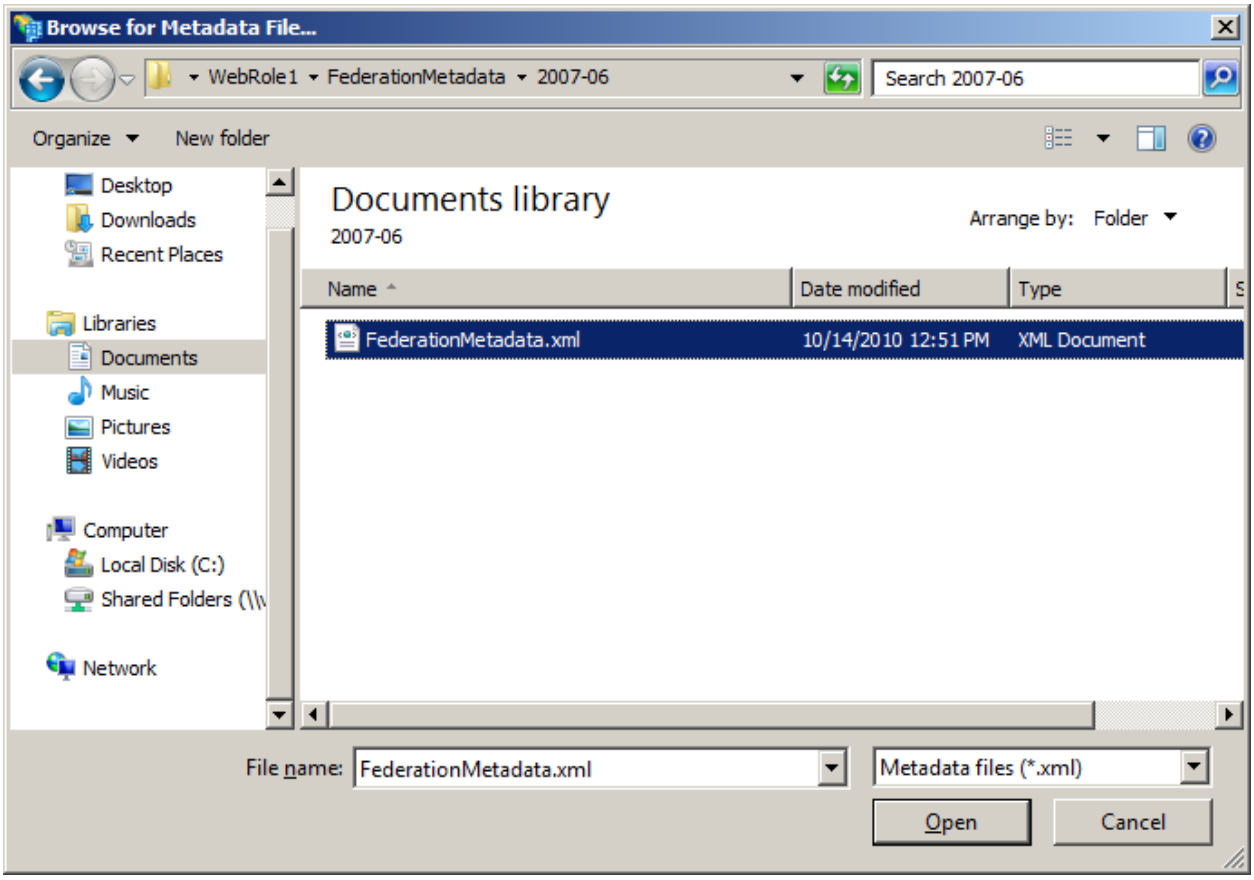

Figure 14 Browse for the metadata file

4. On the **Specify Display Name** page, in **Display name** type the following text, and then click **Next**: **HelloCloudSSO – Dev Fabric**

5. On the **Choose Issuance Authorization Rules** page, click **Permit all users to access this Relying Party**, and then click **Next**.

6. On the **Ready to Add Trust** page, review the relying party trust settings, and then click **Next** to save the configuration.

7. On the **Finish** page, click **Close** to exit the wizard. This also opens the **Edit Claim Rules for HelloCloudSSO – Dev Fabric** properties page. Leave this dialog box open, and then go to the next procedure.

Note: The port number used for dev fabric is not guaranteed to remain the same during different developer sessions. If the port number changes, update the relying party trust information by repeating the previous steps, but use the new port number.

Configure Claim Rules for the Relying Party

Now configure a few claim rules for the HelloCloudSSO application. Claim rules determine what information about the user, or claims, will be sent to the application after the user is authenticated. You

may configure any claim rules that you like, as the application will simply list the claims that are provided. The following claims are a good default set for the application scenario addressed in this document.

To configure claim rules for the relying party

1. On the **Edit Claim Rules for Federated HelloCloudSSO** properties page, on the **Issuance Transform Rules** tab, click **Add Rule** to start the Add Transform Claim Rule Wizard.

2. On the **Select Rule Template** page, under **Claim rule template**, click **Send LDAP Attributes as Claims**, and then click **Next**. This action passes an incoming claim through to the user by means of Windows Integrated Authentication.

3. On the **Configure Rule** page, in **Claim rule name,** type the following text:
LDAP Attribues.
In the **Attribute store** drop-down list, click **Active Directory**. In the **Mapping of LDAP attributes to outgoing claim types** table, click the down arrow in the first column, and then click **Token-Groups – Unqualified Names**. In the second column click the down arrow, click **Role**, and then click **Finish**.

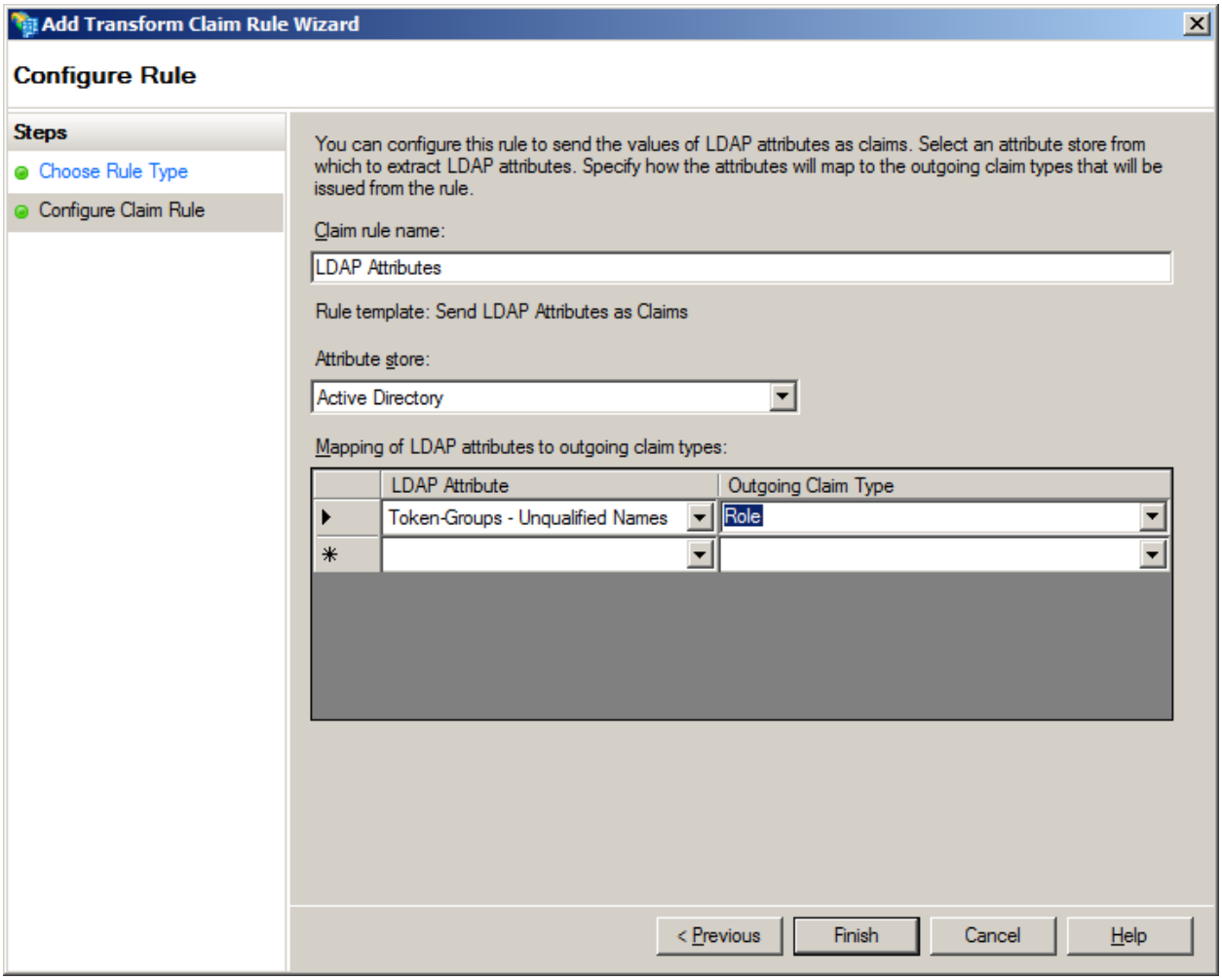

Figure 15 The Add Transform Claim Rule Wizard

4. Click **OK** to save the changes to the relying party trust.

Note: There are several options for the Token-Groups attribute including the following:

- Token-Groups as security identifiers (SIDs)
- Token-Groups that are qualified by domain name
- Token-Groups that are qualified by long domain name
- Token-Groups that are unqualified

The option you choose will depend on the complexity of your environment and whether you have multiple domains. If you have a single domain, then unqualified group names may be sufficient. For the Windows Azure environment you will probably not use SIDs as there is no convenient way for an application in the cloud to look up SID values and generate a text group name.

Add a Request Validator

.NET 4.0 has new behavior that, by default, will cause an error condition on a page request that contains a WS-Federation authentication token. For more information, see the Using the Windows Identity Foundation SDK with Visual Studio 2010 RC blog entry (http://go.microsoft.com/fwlink/?LinkId=207989).

To correct this error for the proof-of-concept environment, you need to add a request validator to the HelloCloudSSO project.

In Visual Studio, right click the **WebRole1** project, click **Add**, and then click **New Item**. In the **Add New Item** dialog box, click the **Web** template, and then click **C# Class**. In the **Name** text box, type the following text, and then click **Add:**
SampleRequestValidator.cs

In the code file, paste the following:

```
//---------------------------------------------------------------------------
//
// THIS CODE AND INFORMATION IS PROVIDED 'AS IS' WITHOUT WARRANTY OF
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
// PARTICULAR PURPOSE.
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//
//
//---------------------------------------------------------------------------

using System;
using System.Web;
using System.Web.Util;

using Microsoft.IdentityModel.Protocols.WSFederation;

/// <summary>
/// This SampleRequestValidator validates the wresult parameter of the
/// WS-Federation passive protocol by checking for a SignInResponse message
/// in the form post. The SignInResponse message contents are verified later by
/// the WSFederationPassiveAuthenticationModule or the WIF signin controls.
/// </summary>

public class SampleRequestValidator : RequestValidator
{
    protected override bool IsValidRequestString( HttpContext context, string value,
RequestValidationSource requestValidationSource, string collectionKey, out int
validationFailureIndex )
    {
        validationFailureIndex = 0;
```

```
        if ( requestValidationSource == RequestValidationSource.Form &&
collectionKey.Equals( WSFederationConstants.Parameters.Result,
StringComparison.Ordinal ) )
        {
            SignInResponseMessage message = WSFederationMessage.CreateFromFormPost(
context.Request ) as SignInResponseMessage;

            if ( message != null )
            {
                return true;
            }
        }

        return base.IsValidRequestString( context, value, requestValidationSource,
collectionKey, out validationFailureIndex );
    }
}
```

Save the file.

Next, add the following to the HelloCloudSSO application's web.config file:

```
<system.web>
    <httpRuntime requestValidationType="SampleRequestValidator" />
```

Save the web.config file.

Use Claims in the Application

The security token provided by AD FS is not only proof that the user successfully authenticated with Active Directory, but it is also a collection of "claims", which are statements about the user made by AD FS 2.0.

To view the claim information provided by AD FS use WIF. Start by adding a reference in the WebRole1 project to the .NET assembly with the component name Microsoft.IdentityModel. After you have added the reference, set the **Copy Local** property of the Microsoft.IdentityModel assembly to **True**.

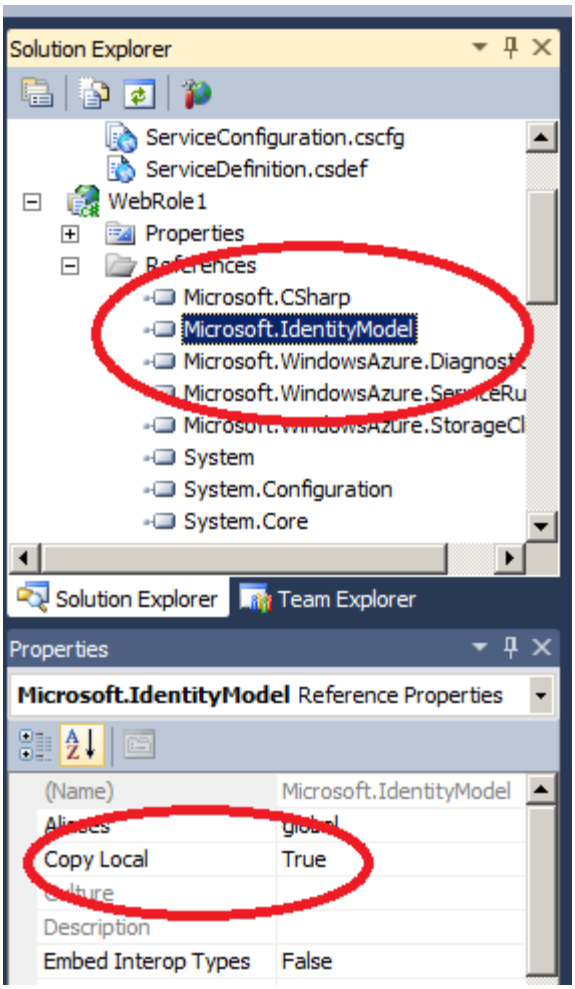

Figure 16 Solution Explorer screen in Visual Studio

This ensures that the Microsoft.IdentityModel assembly is deployed with the package later when you deploy the application to the Windows Azure staging or production environments.

Next, modify the HelloCloudSSO default.aspx.cs file to include a using statement for the Microsoft.IdentityModel.Claims namespace.

```
using Microsoft.IdentityModel.Claims;
```

Next, add code to the sample application that will display all of the incoming claims. Replace the Page_Load method with the following:

```
protected void Page_Load(object sender, EventArgs e)
{
    ContentPlaceHolder cph = (ContentPlaceHolder)
this.Master.FindControl("MainContent");

    IClaimsPrincipal claimsPrincipal = Page.User as IClaimsPrincipal;
    IClaimsIdentity claimsIdentity = (IClaimsIdentity)claimsPrincipal.Identity;
```

```
    Table claimsTable = new Table();
    TableRow headerRow = new TableRow();

    TableCell claimTypeCell = new TableCell();
    claimTypeCell.Text = "Claim Type";
    claimTypeCell.BorderStyle = BorderStyle.Solid;

    TableCell claimValueCell = new TableCell();
    claimValueCell.Text = "Claim Value";
    claimValueCell.BorderStyle = BorderStyle.Solid;

    headerRow.Cells.Add(claimTypeCell);
    headerRow.Cells.Add(claimValueCell);
    claimsTable.Rows.Add(headerRow);

    TableRow newRow;
    TableCell newClaimTypeCell, newClaimValueCell;
    foreach (Claim claim in claimsIdentity.Claims)
    {
        newRow = new TableRow();
        newClaimTypeCell = new TableCell();
        newClaimTypeCell.Text = claim.ClaimType;

        newClaimValueCell = new TableCell();
        newClaimValueCell.Text = claim.Value;

        newRow.Cells.Add(newClaimTypeCell);
        newRow.Cells.Add(newClaimValueCell);

        claimsTable.Rows.Add(newRow);
    }

    cph.Controls.Add(claimsTable);
}
```

Test the Configuration

Launch the HelloCloudSSO application in the dev fabric. Based on the changes that you have made, the browser is redirected to AD FS for authentication. Depending on the configuration of your systems, you will either be presented with the standard Windows Security dialog box or you will be automatically authenticated using Windows Integrated Authentication. If prompted, provide Active Directory credentials. If your authentication is successful, the browser will be redirected to the HelloCloudSSO application. In the application UI, you will now see all of the claims provided by AD FS 2.0.

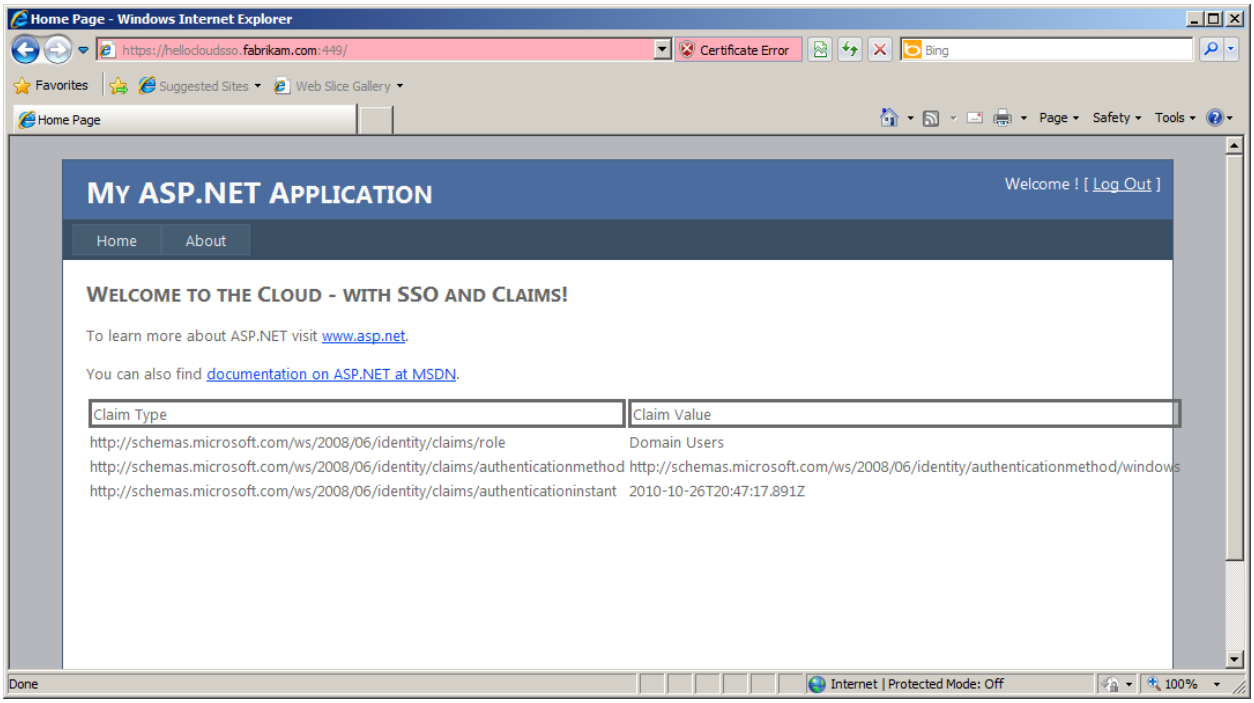

Figure 17 HelloCloudSSO application displaying claims from AD FS 2.0 using the WIF

Note: The most likely reason to be prompted for credentials is because IE does not recognize the URL as being within the intranet authentication zone. For more information, see in the Microsoft Knowledge Base (http://go.microsoft.com/fwlink/?LinkId=207977).

Deploy the HelloCloudSSO to the Cloud

Now that the application is running successfully in dev fabric using SSO and claims the final step is to deploy the application to the Windows Azure cloud environment.

The Cloud Application Development Cycle—Dev Fabric, Staging, and Production

The typical development cycle of a cloud application will likely include iterations of coding and testing on the dev fabric, the Windows Azure staging environment, and the Windows Azure production environment. To better understand the Windows Azure application life cycle, see Application Life Cycle Management for Windows Azure Applications (http://go.microsoft.com/fwlink/?LinkId=207975).

Applications that use AD FS 2.0 have a specific requirement to include configuration of the application URL. In the staging environment, the application URL is dynamically generated when the application is uploaded and cannot be determined beforehand. This creates an issue for AD FS 2.0-integrated applications, because configuration of the application URL must be performed before the application is uploaded.

To test the application in the staging environment, and eventually to move it to production, you use a combination of the local hosts file that resolves a friendly name for the application and a self-signed certificate with the same address.

The required steps to deploy the federated, claims-based application to the Windows Azure environment are:

1. Create and upload the application SSL certificate to Windows Azure
2. Change the application federation settings to redirect to the correct URL
3. Add a new relying party trust for the production and staging environments
4. Upgrade the Windows Azure service
5. Edit the hosts file

Upload the Applications SSL Certificate

To instantiate the certificate that will be used for SSL in the cloud environment, it has to be uploaded using the Windows Azure portal. To prepare for this step, use IIS 7 or the Certificates Microsoft Management Console (MMC) to export the certificate (including the private key) you created earlier using the **makecert** tool. For more information about how to export certificates, see [Import or export certificates and private keys (http://go.microsoft.com/fwlink/?linkid=192401)](http://go.microsoft.com/fwlink/?linkid=192401).

Open the Windows Azure portal, and then click the HelloCloudSSO service. Scroll down until you see the **Certificates** section, and then click **manage**. Complete the wizard to upload the certificate.

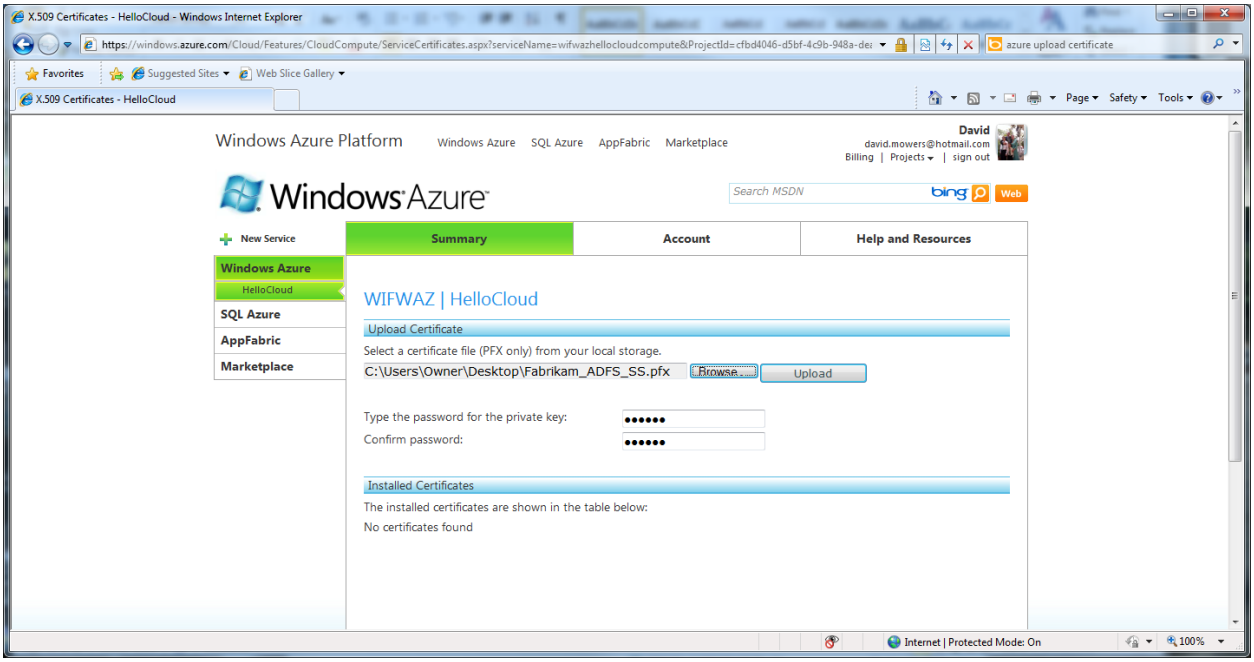

Figure 18 A phase in the upload of a certificate in PFX format to Windows Azure

When the wizard completes, the certificate will be listed for the **Hosted Service**. Find the **Thumbprint** and copy the value to the clipboard, as you will need it for the next step.

Change Federation Settings in the Cloud Application

One of the first steps in this process was to add an STS reference to the HelloCloudSSO application. This action invoked the FedUtil to make certain entries in the application's web.config file that specifies the federation settings. Additionally, this created the federation metadata document for the application that also specifies the application URLs. When the cloud application is moved from running in dev fabric to staging or production, the application's URL changes. Since the application's URL is key to the behavior of the redirects necessary for the WS-Federation Passive authentication profile, it makes sense that the federation configuration in these two files also needs to change when the application URL changes.

To make the necessary changes, find the following section in the application's web.config:

```
<microsoft.identityModel>
    <service>
      <audienceUris>
        <add value="https://hellocloudsso.fabrikam.com:449/" />
      </audienceUris>
      <federatedAuthentication>
        <wsFederation passiveRedirectEnabled="true" issuer="https://fabrikam-
adfs.fabrikam.com/adfs/ls/" realm="https:// hellocloudsso.fabrikam.com:449/"
requireHttps="true" />
        <cookieHandler requireSsl="true" />
      </federatedAuthentication>
```

Examine these entries in your application's configuration file and find the references to the dev fabric application location that contains the port number. To reconfigure the HelloCloudSSO application, comment the references to the dev fabric URL and add new entries with the address of the application without the port number:

```
<microsoft.identityModel>
    <service>
      <audienceUris>

        <!--<add value="https:// hellocloudsso.fabrikam.com:449/" />-->

        <add value="https://hellocloudsso.fabrikam.com/" />

      </audienceUris>
      <federatedAuthentication>

        <!--<wsFederation passiveRedirectEnabled="true" issuer="https://fabrikam-
adfs.fabrikam.com/adfs/ls/" realm="https:// hellocloudsso.fabrikam.com:449/"
requireHttps="true" />-->

        <wsFederation passiveRedirectEnabled="true" issuer="https://fabrikam-
adfs.fabrikam.com/adfs/ls/" realm="https://hellocloudsso.fabrikam.com/"
requireHttps="true" />
```

```
      <cookieHandler requireSsl="true" />
    </federatedAuthentication>
```

Save the application's web.config file.

Next, you will edit the application's federation metadata document.

Locate the file in the Visual Studio project directory (usually WebRole1\FederationMetadata\2007-06\FederationMetadata.xml). Back up the existing file in case you ever need to reestablish a relying party trust with the dev fabric URLs. Open the file for editing and replace the three occurrences of the dev fabric application URL (https://hellocloudsso.fabrikam.com:449/ or similar) with the application's address without the port number (`https://hellocloudsso.fabrikam.com/` or similar). The edited file should be similar to the following:

```
<?xml version="1.0" encoding="utf-8" ?>
- <EntityDescriptor ID="_db7b0a56-88d0-4d60-8975-2ab0ae7e9be9"
entityID="https://hellocloudsso.fabrikam.com/"
xmlns="urn:oasis:names:tc:SAML:2.0:metadata">
- <RoleDescriptor xsi:type="fed:ApplicationServiceType" xmlns:fed="http://docs.oasis-
open.org/wsfed/federation/200706" protocolSupportEnumeration="http://docs.oasis-
open.org/wsfed/federation/200706" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
- <fed:ClaimTypesRequested>
<auth:ClaimType Uri="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
Optional="true" xmlns:auth="http://docs.oasis-open.org/wsfed/authorization/200706" />
<auth:ClaimType Uri="http://schemas.microsoft.com/ws/2008/06/identity/claims/role"
Optional="true" xmlns:auth="http://docs.oasis-open.org/wsfed/authorization/200706" />
</fed:ClaimTypesRequested>
- <fed:TargetScopes>
- <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
<Address>https://hellocloudsso.fabrikam.com/</Address>
</EndpointReference>
</fed:TargetScopes>
- <fed:PassiveRequestorEndpoint>
- <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
<Address>https://hellocloudsso.fabrikam.com/</Address>
</EndpointReference>
</fed:PassiveRequestorEndpoint>
</RoleDescriptor>
</EntityDescriptor>
```

Save the file.

Add a Relying Party Trust for the Cloud Environment

You need to add a new relying party trust in AD FS that refers to the HelloCloudSSO application in the Windows Azure staging and production environments. This is accomplished by repeating the steps

previously performed to create a relying party trust for the dev fabric environment, as detailed in the section “Add a Relying Party Trust to AD FS.“

To recap those steps:

1. Run the **Add Relying Party Wizard** using the modified version of the FederationMetadata.xml file.
2. Create the claim rules.

Upgrade the Federated HelloCloudSSO Application

Next, you need to build and publish the HelloCloudSSO configuration files and upgrade the Windows Azure hosted service package through the Windows Azure developer portal. This document will describe upgrading the staging environment, but the process is the same except for the characteristic of the changing URL.

Note: If the original HelloCloudSSO Windows Azure Hosted Service did not have an HTTPS endpoint, then you will receive an error message when you try to upgrade the service. If this is the case, simply delete the existing service and create a new one based on the current application configuration files that include the HTTPS endpoint.

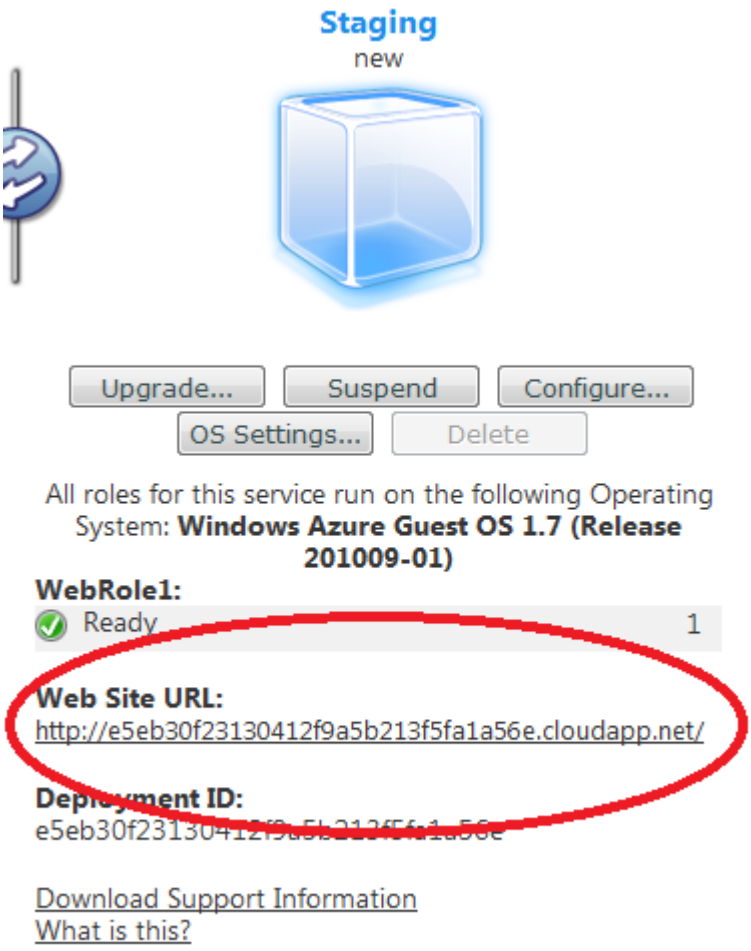

Figure 19 Application uploaded to the staging environment

Make note of the **Web Site URL** for the application in the staging environment. This URL will be used in the following step.

Add a Hosts File Mapping for the Staging Environment

The first step is to find the IP address used for the application in the staging environment, by using the application DNS name portion of the URL from the previous step and the ping.exe tool. For example:

```
ping.exe e5eb30f23130412f9a5b213f5fa1a56e.cloudapp.net
```

The specific DNS address is dynamically generated for your application every time it is uploaded to the staging environment. Substitute **e5eb30f23130412f9a5b213f5fa1a56e** with the address generated for your application

PING will resolve the staging application DNS name to an IP address.

On the development computer, open the /windows/system32/drivers/etc/hosts file and add the following line using the IP address shown by ping.exe and the friendly DNS name of the application:

```
# name mapping for Windows Azure Staging
65.52.195.149    hellocloudsso.fabrikam.com
```

Comment the name mapping for dev fabric as follows:

```
# name mapping for Windows Azure Dev Fabric
# 127.0.0.1    hellocloudsso.fabrikam.com
```

Save the hosts file.

Note: Using PING to resolve the DNS address to a specific IP address should work in practice, but it might create problems if Windows Azure later assigns a different IP address to the application. If you encounter errors accessing the application you should repeat the steps here to find out if the applications IP address has changed and then reconfigure the hosts file if it did.

Domain Names and URLs

It is worth mentioning that URL namespace management is a bit more complex for cloud-based applications then it is for applications on an internal network. For example, this document describes accessing the cloud application using a URL based on the cloudapp.net namespace. But this is almost certainly not how it would be referenced for a real application deployed to the cloud by a business. The article Custom Domain Names in Windows Azure (http://go.microsoft.com/fwlink/?LinkId=207974) describes how an organization can map a URL based on their business will resolve to a cloud application. When this kind of approach is taken, the URLs in both the application configuration and AD FS relying party trust configuration will change to reflect the actual URL used to access the application.

Test the Federated HelloCloudSSO Application in the Staging Environment

Once the application upgrade is completed, open a new browser session and enter the short name for the HelloCloudSSO application that you configured previously. The browser will be redirected to the AD FS 2.0 authentication pages and you will be prompted for credentials to log on using your AD domain account. Once your credentials are entered and verified, the browser will redirect to the cloud application that will once again display the claims issued by AD FS 2.0.

Upgrade and Test in the Production Environment

Perform the following steps to test the HelloCloudSSO application in the Windows Azure production environment:

- Upgrade the Windows Azure web application to the production environment, by uploading the application configuration files through the Windows Azure portal.
- Use ping.exe to resolve the IP address of the HelloCloudSSO application in the Windows Azure production environment.
- Edit the hosts file on the development computer to add a mapping of the application friendly name to the production environment IP address as is shown in the following example:

```
# name mapping for Windows Azure Dev Fabric
# 127.0.0.1    hellocloudsso.fabrikam.com

# name mapping for Windows Azure Staging
# 65.52.195.149    hellocloudsso.fabrikam.com

# name mapping for Windows Azure Production
65.52.198.169    hellocloudsso.fabrikam.com
```

Comment the previous mappings used for the dev fabric and staging environments and save the hosts file.

Launch a new browser session and navigate to https://hellocloudsso.fabrikam.com. You will be presented with the authentication dialog box where you can type your Fabrikam credentials. Once validated, you will be redirected to the application in the staging environment that will once again display the claim set from AD FS 2.0.

Now that there is a mapping in the hosts file for each environment, you can easily move back and forth between the dev fabric, staging, and production environments by commenting and uncommenting the entries in the hosts file. IP addresses assigned by Windows Azure to your application may change, so make sure that you verify the IP address assigned to your application using ping.exe every time that it is upgraded in the staging and production environments.

Although other approaches for managing URLs and trust configurations are possible, we recommend this approach as the Federation trust settings in the application or in AD FS 2.0 do not have to be changed once they are established.

Congratulations! You have now completed a proof of concept for federated, claims-based access to a cloud application hosted in the Windows Azure cloud environment.

Additional Considerations for Production Development

WIF Session Management in Windows Azure

WIF handles sessions transparently, without the need for intervention from the developer. Once the authentication token is received and verified, in the default case a WIF HttpModule saves the claims in a cookie and uses it for the duration of the session. The cookie is protected from reading or tampering through the Data Protection API (DPAPI).

One of the defining features of Windows Azure is the ease with which an application can be run on multiple virtual machine (VM) instances behind a virtual load balancer. Securing a session cookie through DPAPI may not work as expected if the application runs on multiple instances: if the instance B serving a request is different from the instance A that created the session cookie earlier, different computer keys on A and B will cause the cookie to be unreadable from instance B. One solution would be to use sticky sessions, but that would not yield to the best possible utilization of a load-balanced

environment and is not easy to obtain in Windows Azure. Another solution that would be more suitable for the environment is to secure the cookie with the SSL certificate used by the web role.

First, add the following configuration for the service certificate to the Microsoft.IdentityModel section of the application's web.config file. Make sure to use the thumbprint value for the certificate used in the Windows Azure environment:

```
<serviceCertificate>
        <certificateReference x509FindType="FindByThumbprint"
findValue="4EB108C4E2DC9E3170F01047C83C8210188F5351"/>
      </serviceCertificate>
```

Next, add the following to the web role's global.asax.cs file:

```
public class Global : System.Web.HttpApplication
{
  protected void Application_Start(object sender, EventArgs e)k
        {
            FederatedAuthentication.ServiceConfigurationCreated +=
OnServiceConfigurationCreated;
        }

  void OnServiceConfigurationCreated(object sender,
ServiceConfigurationCreatedEventArgs e)
        {
            //
            // Use the <serviceCertificate> to protect the cookies that are
            // sent to the client.
            //
            List<CookieTransform> sessionTransforms =
                new List<CookieTransform>(new CookieTransform[] {
                    new DeflateCookieTransform(),
                    new
RsaEncryptionCookieTransform(e.ServiceConfiguration.ServiceCertificate),
new RsaSignatureCookieTransform(e.ServiceConfiguration.ServiceCertificate)  });
            SessionSecurityTokenHandler sessionHandler = new
SessionSecurityTokenHandler(sessionTransforms.AsReadOnly());

        e.ServiceConfiguration.SecurityTokenHandlers.AddOrReplace(sessionHandler);
        }
...
}
```

At the very beginning of the lifecycle of the web role application instance, the method **OnServiceConfigurationCreated** handles the event that starts when WIF reads its configuration. The first line creates a new set of cookie transforms, based on the web role's certificate as indicated by the WIF

configuration. The rest of the handler replaces the default session management logic with one that employs the custom transformation.

Summary

In this document we have shown how to create a proof of concept that integrates the Windows Azure platform for cloud-based web applications, the WIF for claims-based access control, and AD FS 2.0 that provides identity federation.

In this document, we have not fully explored SSO for users, but this is a key benefit of integrating the platform components that have been described. Once a user has authenticated to AD FS 2.0 within a browser session, they will not be prompted to authenticate again when they access federated applications—whether they are in the local intranet or hosted in the cloud. This is of great benefit to application users; increasing efficiency, improving security, and generally creating a more efficient and more productive workforce.

We have also shown that as you extend the federated, claims-based infrastructure to the cloud, no changes to the basic infrastructure are required. Administrators are, therefore, able to master federation and claim management and be assured that those skills will be applicable to the new application scenarios that internal and external developers will be developing in the future.

Furthermore, application logic does not require any changes as applications move from internal locations to the cloud. This allows developers to focus on functionality and provides for the ability to reuse authorization codes across many different applications that serve various business functions.

Further Reading and Resources

This document describes a basic scenario that can be extended to provide additional functionality. Developers should investigate the following topics to understand the full range of capabilities that are available to federated, claims-based, cloud applications on Windows Azure.:

Programming Windows Identity Foundation (Dev - Pro) Book (http://go.microsoft.com/fwlink/?LinkId=207967)

Windows Identity Foundation home page (http://go.microsoft.com/fwlink/?LinkId=207968)

Developing Applications for the Cloud on the Microsoft Windows Azure™ Platform (http://go.microsoft.com/fwlink/?LinkId=207969)

Moving Applications to the Cloud on the Microsoft Windows Azure Platform (http://go.microsoft.com/fwlink/?LinkId=207970)

A Guide to Claims–based Identity and Access Control (http://go.microsoft.com/fwlink/?LinkId=207971)

Federated Authentication in a Windows Azure Web Role Application (hands-on lab)
(http://go.microsoft.com/fwlink/?LinkId=207972)

Security Best Practices For Developing Windows Azure Applications
(http://go.microsoft.com/fwlink/?LinkId=207973)