

Implementing Smart Card Authentication with ASP.NET

In my previous article last October on [Smart Card Authentication with ASP.NET](#), I introduced the topic of using Smart Cards to handle Authentication and Authorization with ASP.NET for non-Active Directory users. This is a more complete follow-up article now that I have done a bit more research into the topic and now have a full working implementation that I'm happy with.

In this implementation, the IIS Web Server is handling the basics of the Smart Card authentication, much like Windows authentication works.

The goals of this project are to provide the following:

1. A tight interface to strongly typed objects that are Smart Card aware
2. Allow these typed objects to be available to the entire web application on every page request
3. Provide Role based authorization via the smart card through groups. This will require a mechanism to associate a User's smart card with an associated group

Some Caveats

If you do not need to apply role authorization through Principals then this methodology may be more than you need. If specific requirements stipulate you need information on the Smart Card/Client Certificate to display on the screen or track the Certificate Subject, simply using IIS for Authentication and the `HttpContext.ClientCertificate` property of the Context object (`Context.Request.ClientCertificate`) will be sufficient.

If you plan on doing Client Certificate Mapping to User Accounts on the OS or Domain, this methodology is not necessary. If the reason is not immediately obvious to you, the short answer is that when using Certificate to AD Account mapping, IIS is taking the smart card and mapping it to a Windows Account – so in the end, IIS is using Windows Authentication. Once you have mapped an account from a smart card to a windows account, you can use Windows Authentication in the Web.Config and the `WindowsPrincipal` object for Authorization based on Windows Groups.

Background

In my original implementation of Smart Card authentication and authorization with ASP.NET, I used Forms Authentication in combination with the `Request.ClientCertificate` to automatically authenticate the user. Instead of having the user enter their credentials on the Forms Login Page, this methodology still redirected them to the forms login page and the code behind automatically resolved the `Request.ClientCertificate` and authenticated the Client Certificate by resolving their account in the database. Next the code created the `FormsAuthenticationTicket` (which included the roles) as a cookie, and then redirected them back to the original page they were requesting. Finally, the Principal was attached to the `Context.User` object in `Application_AuthenticateRequest` event of the `Global.asax`. For implementation details, see: [How To: Create GenericPrincipal Objects with Forms Authentication](#) and [How To: Use Forms Authentication with SQL Server 2000](#) – I combined and modified these methodologies for use with the Smart Card.

After having used Forms Authentication in the original version, here are some “lessons learned” I will share. The Forms Authentication approach ended up not being a very good approach for several reasons:

1. Forms Authentication relies on page redirects to handle authentication via a Login page. This adds unnecessary overhead since there is no reason for the browser to jump around via redirects to retrieve the `Request.ClientCertificate`. The certificate is immediately available once the user is authenticated via IIS.
2. Forms Authentication relies on encrypted cookies to store the users' authentication data in the `FormsAuthenticationTicket.UserData` property (which was the users' roles are typically

stored). Since we already have the *Request.ClientCertificate*, using cookies for authentication adds additional complexity to the code.

3. Encrypting and decrypting the *FormsAuthenticationTicket* adds extra processing overhead.
4. It does not make sense to push data down to the client in a cookie since we've received all authentication information we need already. From a threats/countermeasures perspective, since the cookie comes from the client, we must treat it suspiciously and perform extra validation on the User Data to make sure it was not tampered with. Yes, even though its 3DES encrypted [MAC](#) is enabled –the threat must be considered where one is able to crack the 3DES key, however unlikely, and decrypt the *FormsAuthenticationTicket*. If we put nothing on the client, this threat will never be realized.
5. Forms Authentication adds unnecessary authentication complexity to the code in Global.asax as well as code in the Forms Login page.
6. The Forms authentication mechanism caches the login for duration and has 'logout' functionality via cookie expiration. Smart Cards in ASP.NET do not ever 'log out', they timeout. Since IIS handles the re-authentication transparently to ASP.NET, the .NET authentication code need not handle timeout and re-authentication.

A more appropriate way of handling the Smart Card authentication and authorization is using a feature of ASP.NET called HTTP Modules.

HTTP Pipelines

ASP.NET has built in a nice way to insert code into the HTTP Pipeline. This is an ideal place to insert code to handle new types of Authentication (as well as many other things). This will not be any shock to those already familiar with the ASP.NET HTTP pipeline – the other authentication models in ASP.NET (Forms, Windows, and Passport) are implemented in the HTTP pipeline using HTTP Modules. Session management and URL Authorization are also handled with HTTP Modules. If you are not familiar with HTTP Modules and the HTTP Pipeline in ASP.NET see Appendix A for more reading on the topic.

The concept behind the HTTP Pipeline is simple. A web request comes in from IIS. If the page is mapped to run through the ASP.NET engine, IIS passes the request off to ASP.NET and then ASP.NET moves the request through all the HTTP Modules installed in the *machine.config* as well as the *web.config*. Here is a picture demonstrating the pipeline:

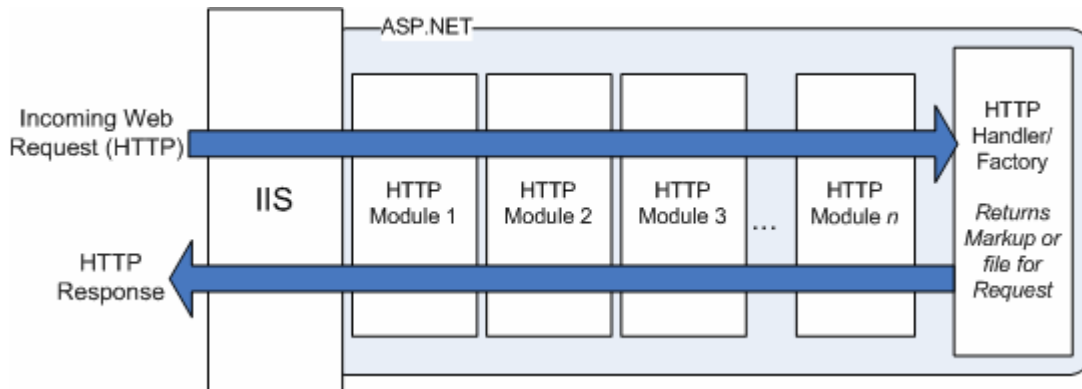


Figure 1 – HTTP Pipeline in ASP.NET

To resolve programmatically the HTTP Modules loaded currently in ASP.NET, drop the following code in the *_Load* event of any ASP.NET page:

Implementing Smart Card Authentication with ASP.NET

```
C#  
private void Page_Load(object sender, System.EventArgs e)  
{  
    Response.Write("<b>Loaded HTTP Modules</b>");  
    Response.Write("<br>");  
  
    foreach (string httpModule in this.Context.ApplicationInstance.Modules)  
    {  
        Response.Write(httpModule);  
        Response.Write("<br>");  
    }  
}
```

```
VB.Net  
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
  
    Response.Write ("<b>Loaded HTTP Modules</b>")  
    Response.Write ("<br>")  
  
    For Each httpModule As String In Me.Context.ApplicationInstance.Modules  
        Response.Write(httpModule)  
        Response.Write("<br>")  
    Next  
End Sub
```

The output will look something like this:

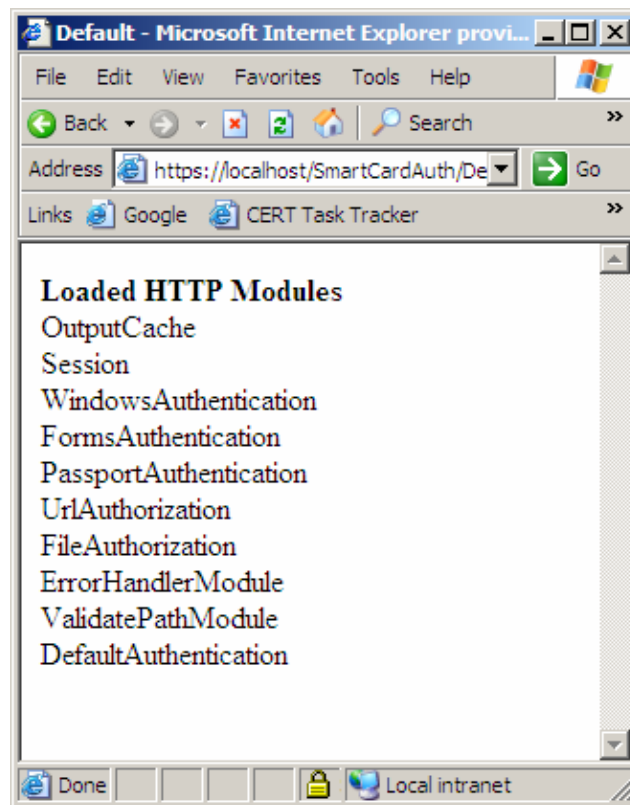


Figure 2 – ASPX page lists the currently running ASP.NET modules.

Notice that all of the HTTP Modules that come with ASP.NET are loaded even when a particular feature is not being used (i.e. Session Management, Forms Authentication). While loaded, they do not do any work on the Request unless the web.config file is used to enable it.

Here is an outline of the steps this article will cover:

1. How to Build a Smart Card HTTP Module
2. Configure an ASP.NET application to use the Smart Card HttpModule in the web.config file.
3. Configure IIS to support Smart Cards
4. Create the Smart Card Principal and Smart Card Identity objects and add the logic

Building and Installing the Smart Card HttpModule

The IHttpModule interface we need to implement is very simple. Here is the interface, as defined by Microsoft in the .NET Framework. *IHttpModule* is in the *System.Web* namespace:

```
C#  
interface IHttpModule  
{  
    // called to attach module to app events  
    void Init(HttpApplication app);  
    // called to clean up  
    void Dispose();  
}
```

```
VB.Net  
Interface IHttpModule  
' called to attach module to app events  
Sub Init(ByVal app As HttpApplication);  
' called to clean up  
Sub Dispose()  
End Interface
```

To get a basic HTTP Module up and functioning is incredibly trivial. There are really only three steps involved:

1. Create a class that Implements IHttpModule

```
C#  
public class SmartCardAuthenticationModule : IHttpModule  
{  
    public void Init(HttpApplication context)  
    {  
    }  
    public void Dispose()  
    {  
    }  
}
```

```
VB.Net  
Public Class SmartCardAuthenticationModule  
    Implements System.Web.IHttpModule  
    Public Sub Init(ByVal context As System.Web.HttpApplication) _  
        Implements System.Web.IHttpModule.Init  
    End Sub  
    Public Sub Dispose() Implements System.Web.IHttpModule.Dispose  
    End Sub  
End Class
```

2. Next wire up the events to handle in the *Init()* method of the class – compile it in an assembly that you reference in your web project (or include it in your web project directly).

Implementing Smart Card Authentication with ASP.NET

```
C#
public void Init(HttpApplication context)
{
    context.AuthenticateRequest += new EventHandler(Me.OnAuthenticateRequest);
}

private void OnAuthenticateRequest(object sender, EventArgs e)
{
    // Here's where the work of authentication takes place.
}
```

```
VB.Net
Public Sub Init(ByVal context As System.Web.HttpApplication) _
    Implements System.Web.IHttpModule.Init

    AddHandler context.AuthenticateRequest, _
        New EventHandler(AddressOf Me.OnAuthenticateRequest)
End Sub

Private Sub OnAuthenticateRequest(ByVal source As Object, ByVal eventArgs _
    As EventArgs)
    ' Here's where the work of authentication takes place.
End Sub
```

3. Install the Smart Card HttpModule into your ASP.NET application using the Web.Config and deny all anonymous users in the authorization section.

```
<configuration>
  <system.web>
    <httpModules>
      <add name="SmartCardAuthentication"
          type="SmartCardAuthentication.SmartCardAuthenticationModule,
              SmartCardAuthentication" />
    </httpModules>
    <authorization>
      <!-- Deny all Anonymous Users -->
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

Once added to the web.config, re-run the code that displays installed HTTP Modules. The SmartCardAuthentication module should show up in the pipeline:

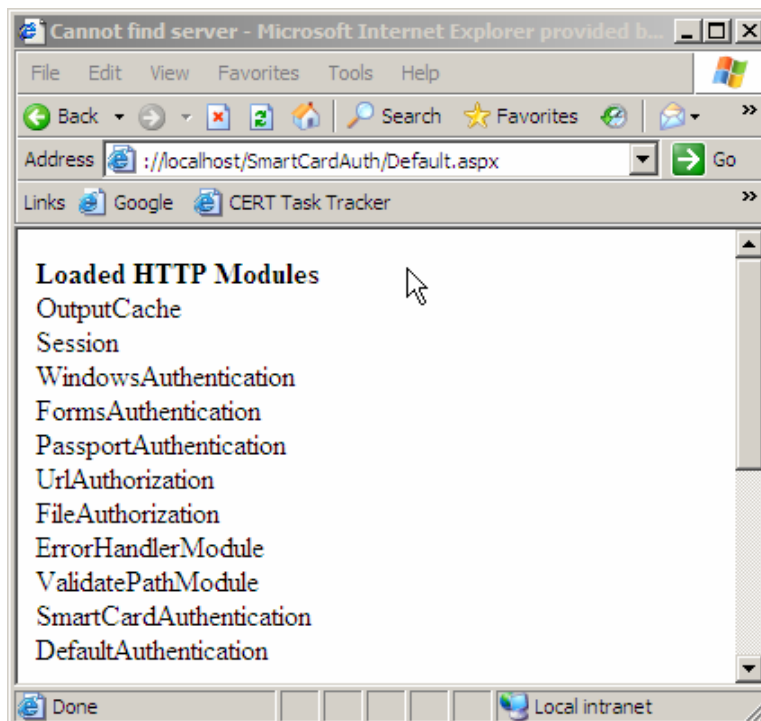


Figure 3 – ASPX page shows that the Smart Card module is installed.

In Figure 2, the addition of SmartCardAuthentication in the list. This is how you can tell if your module is installed and running correctly.

Above is the most basic skeleton of code I'll be working from, but before getting into the details of the code, IIS must be configured to support Smart Card Authentication.

Two Important Points about IIS Configuration as it relates to Smart Cards/Client Certificates:

- If IIS is not configured to actually accept and present the Client/Smart Card Certificate (by way of the HttpCertificate object) to ASP.NET, it is critical that the SmartCardAuthenticationModule code deny access to anyone accessing the site – using the principal of [failing securely](#).
- On the flip side, if IIS is not configured to limit what Certificates are acceptable through the Certificate Trust Lists (CTL), the web server will inappropriately grant permissions to more users than expected. We can do some extra checks in code as well to [fail securely](#) in this case as well.

IIS Configuration

If you have not worked with Client Certificates/Smart Cards with IIS before, then this will probably be new information. This setup will work with IIS 5.0 (Windows Server 2000), IIS 5.1 (Windows XP) and IIS 6.0 (Windows Server 2003). I've not had the opportunity to try this on Vista yet, but I suspect the configuration options are similar, if not the same, – though care will need to be taken in choosing the right components to install to add Client Certificate, and or Active Directory account mapping support in IIS 7 since the installation options are incredibly modular now. If you do not see the options for Client Certificates in IIS on Vista, you probably have not installed the proper components.

The Web Server will need to be aware of and fully integrated into your Enterprise PKI Solution:

- The Trusted CA's will need to be installed in the Trusted Roots Certificate Stores
- Certificate Revocation will need to be configured to work with IIS ([CRL/OCSP](#), etc.)

Implementing Smart Card Authentication with ASP.NET

- If a users' Smart Card certificate is revoked before it expires, you need to be able to prevent the user from accessing the web site.
- Third party [CRL/OCSP](#) solutions DO support IIS integration; check your vendor's documentation. It is typically as easy as selecting a checkbox in the software's configuration properties.
- Etc. – there are potential complexities and specific implementation details based on the PKI deployment that are outside the scope of this article. You do need to know how your specific implementation of PKI works, and that all the pieces are in place to provide proper authentication to your web site.

Here are the specific steps to setup Smart Card Authentication in IIS once the Web Server has been PKI enabled.

1. Generate an SSL Certificate Request for the Root Web Site you will PKI enable.
2. Get the Request signed by a Certificate Authority (CA), most likely you will use your internal Root Certificate Authority (CA) or Intermediate CA.
 - a. If your users are strictly Int**R**ANet users, you can use your internal CA that you use for PKI to Sign the Certificate
 - b. If your users are Int**E**Rnet Users, you will need a way to deploy your Root CA's to the client's computer Certificate Store for your Root CA servers to be trusted on a client's computer.
3. In IIS, on the folder or Web Site you want to enable for smart card authentication/authorization, open the properties and Click on the Directory Security Tab.
4. In the Secure Communications section, click on the "Edit" button
5. For each folder or web site requires Smart Card authentication, check the "Require Secure Channel (SSL)".
6. Under "Client Certificates", the **default** option is to "Ignore Client Certificates" – for all the sites or folders you want to accept Smart Cards, choose "**Require Client Certificates**"

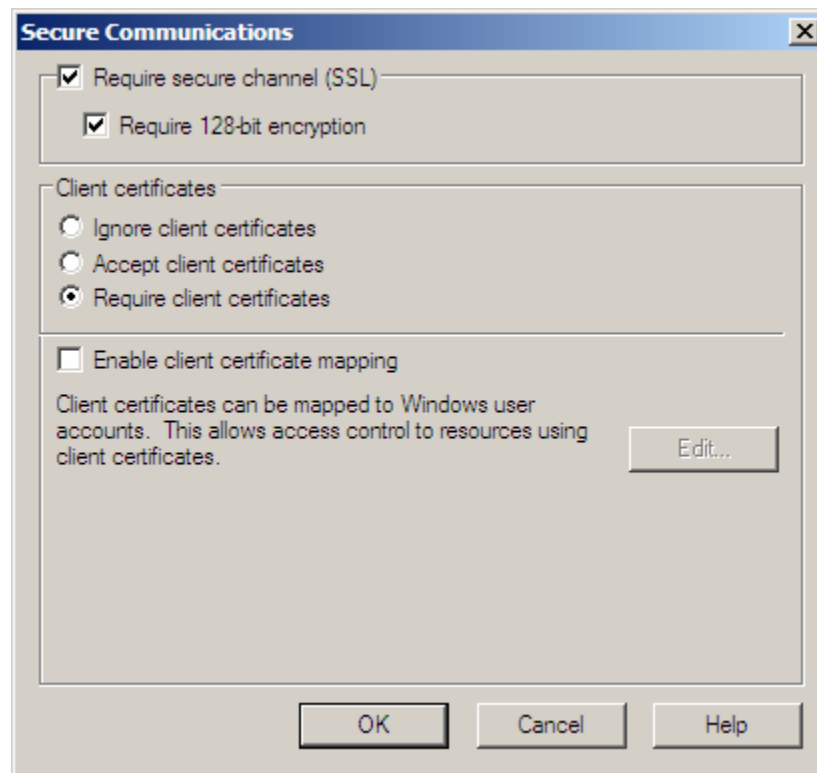


Figure 4 – SSL/Client Certificate Settings for Production Systems

NOTE: If you are doing this on a development system and you want to be able to use Visual Studio.NET, you **cannot** “Require secure channel (SSL)” or “Require Client Certificates” or VS.net will not be able to interact properly with the web site. Also, you must LEAVE “Integrated Security” on as well, or you will have a similar problem.

When launching your ASP.NET web application for Debugging from Visual Studio, manually change your URL in IE when debugging from HTTP:// to HTTPS://. As long as “Accept Client Certificates” is checked, it will prompt you to use a client certificate providing HTTPS:// is in the URL. Alternatively, change the Web Project settings of the site to launch the web page with the HTTPS:// prefix in the URL.

7. There are a couple ways to map Smart Cards to Active Directory Users. Check the “Enable client certificate mapping” option and then click Edit. On this screen you will setup the 1-to-1 mapping. To map all Smart Cards to one Active Directory Users, setup the Many-to-1 mappings.

NOTE: I don't recommend the 1-to-Mapping if you have a large number of users accessing the web site

The difficulty with the 1-to-1 mapping is you must have a copy of the client certificate (the public portion of X509 certificate) for each user you want to map. You also need to know the Domain Password of each user as well. If you have a large enterprise and all the users need access, you are in for a management nightmare. If someone has other suggestions for easily managing this, please let me know.

A Many-to-1 Mapping has more promise since you can map a portion of Smart Cards based on Wildcard Rules using fields in the X509 Certificate Distinguished Name to one AD User Account. This allows different sets of users to be mapped to related AD Accounts based on role, essentially allowing Role based authorization with Windows Principals.

If you haven't realized this yet, if you are able to do 1-to-Many Mapping or 1-to-1 Mapping, there is no need for this HttpModule as you can move over to the built in Windows Authentication model in ASP.NET and apply Principal Permissions based on Groups in Active Directory.

8. Check the “Enable certificate trust list” and create a new IIS CTL. Trust only the CA's that have Signed the Smart Card certificates that users will be using to authentication with.

NOTE: *This option is a Web Site setting. You will not see “Enable certificate trust list” option setting for each Virtual Folder.*

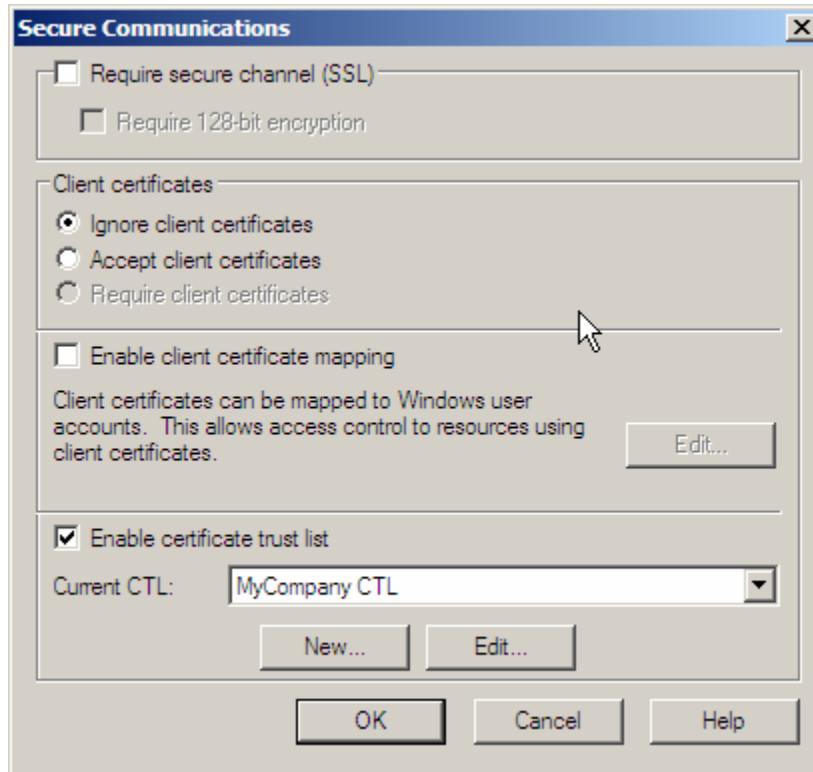


Figure 5 – Example of setting Certificate Trust Lists for Web Site.

In the above screen, “Require SSL and Require Client Certificates” are not selected because we set it on a folder by folder basis. You could set them here to apply it to the entire Web Site depending on your requirements.

IMPORTANT: If you include any other Certificate Authorities other than the ones who signed your Smart Card Certificates (like Verisign’s CA’s, or others) any user could purchase a client Certificate from Verisign and present it as a Client Certificate to your web site and you would let them in!!

9. Finally, make sure you have enabled Anonymous Access to this web site in Directory Security. Otherwise users will be presented with a system or domain logon prompt as well as the Client Certificate dialog box.

IHttpModule Implementation

Once IIS is configured properly and you have the skeleton of the IHttpModule built, development of the custom IHttpModule can commence.

Here’s an overview of the logic used to authenticate users:

1. When a user browses the web site, the Smart Card HttpModule will authenticate the Smart Cards against a database of users.
 - a. If users have a smart card validated by IIS and:
 - i. They **ARE IN** the Database; authenticate and authorize them with their roles in the Database (i.e. a small set of users that are Administrators, Moderators, Content Managers, etc.)
 - ii. They **ARE NOT IN** the Database; we authenticate them in the default ‘Users’ role (i.e. the mass of users who typically have read only access or have limited privileges)
 - b. If a smart card isn’t authenticated by IIS, give the user a 401.1 access denied page.

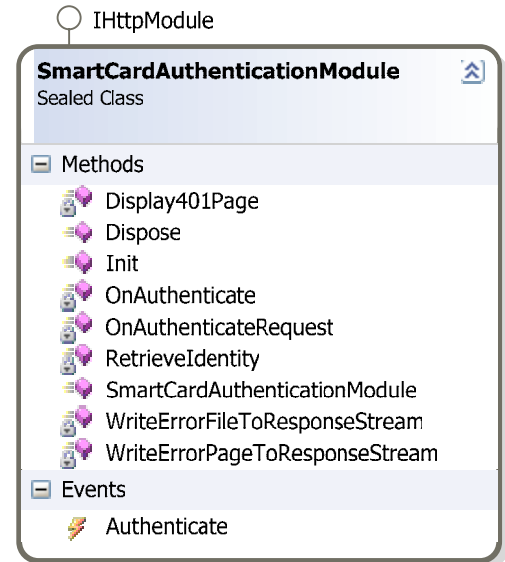
2. Create a strongly typed Smart Card Identity object based on the smart card.
3. Create a strongly typed Smart Card Principal object and assign it the Identity object.
4. Attach the Smart Card Principal to the HttpContext, by assigning it to the HttpContext.User property that is available application wide.

SmartCardAuthenticationModule Implementation

SmartCardAuthenticationModule – Inherits from *System.Web.IHttpModule*

The *SmartCardAuthenticationModule* class is the *HttpModule* that handles the authentication. Its *raison de'être* is to create an Identity and Principal and attach it to the *HttpContext* classes User property. Once a principal is attached to the *HttpContext* object, authorization using that principal will be performed, otherwise if the *HttpContext.User* property is null, the user is considered unauthenticated and is thus denied by the following authorization setting in the web.config:

```
<authorization>
  <deny users="?" />
</authorization>
```



An additional feature of the *SmartCardAuthenticationModule* is the Authenticate Event. This event is automatically wired up by ASP.NET to a method in the *Global.asax* file called *SmartCardAuthentication_Authenticate* like so:

```
Public Class Global
  Inherits System.Web.HttpApplication
  ' OPTIONAL CODE TO OVERRIDE DEFAULT SMARTCARD HTTP MODULE BEHAVIOR
  Sub SmartCardAuthentication_Authenticate(ByVal sender As Object, _
    ByVal e As SmartCardAuthenticationEventArgs)

    ' Obviously you'd need to load an actual certificate on the next line,
    ' not a byte array with {0,0,0} in it, but an actual X509 certificate
    Dim customC509Certificate As New X509Certificate(New Byte() {0, 0, 0})

    Dim smartCardIdentity As New SmartCardIdentity(customC509Certificate, True)
    Dim smartPrincipal As New SmartCardPrincipal(smartCardIdentity)

    ' Now set either User property of the HttpContext or the
    ' EventArgs User property
    e.Context.User = smartPrincipal
    ' OR
    e.User = smartPrincipal
  End Sub
End Class
```

In the *Global.asax*, custom code can override the default behavior of how the Principal and Identity is resolved and assigned to the *HttpContext.User* object. You can create and add your own SmartCard Principal/Identity that is resolved perhaps through an already defined database, or another technology. To override the default Principal and Identity objects, code within this method must set either *the e.Context.User* or the *e.User* property of the *SmartCardAuthenticationEventArgs* class.

Next follows the *SmartCardAuthenticationModule* implementation:

```

C#
using System;
using System.Security.Cryptography.X509Certificates;
using System.Security.Permissions;
using System.Security.Principal;
using System.Web;

namespace SmartCardAuthentication
{
    public delegate void SmartCardAuthenticationEventHandler(object sender,
        SmartCardAuthenticationEventArgs e);

    public sealed class SmartCardAuthenticationModule :
        System.Web.IHttpModule
    {
        public event SmartCardAuthenticationEventHandler Authenticate;

        public SmartCardAuthenticationModule() { }

        public void Dispose() { }

        public void Init(System.Web.HttpApplication context)
        {
            context.AuthenticateRequest +=
                new EventHandler(new EventHandler(this.OnAuthenticateRequest));
        }

        private void OnAuthenticateRequest(object source, EventArgs eventArgs)
        {
            HttpApplication httpApp = (HttpApplication)source;
            HttpContext context = httpApp.Context;
            SmartCardIdentity smartCardIdentity = this.RetrieveIdentity(context);

            this.OnAuthenticate(
                new SmartCardAuthenticationEventArgs(smartCardIdentity, context)
            );
        }

        private void OnAuthenticate(SmartCardAuthenticationEventArgs e)
        {
            if (this.Authenticate != null)
            {
                // Fire any subscribers to the Authenticate event,
                // typically from Global.asax Smart
                this.Authenticate(this, e);
            }

            // Has Context.User already been assigned
            // from the Authenticate() event above?
            if (e.Context.User == null)
            {
                // Context.User not populated

                // Did e.User get assigned from
                // from the Authenticate() event above?
                if (e.User != null)
                {
                    // Yes, assign the user and exit.
                    e.Context.User = e.User;
                }
                else if (e.Identity != null)
                {
                    // No, but Identity isn't null from DB Call.
                    // User is authenticated, attach principal to HttpContext
                    e.Context.User = new SmartCardPrincipal(e.Identity);
                }
                else
                {
                    // User isn't authenticated
                    this.Display401Page(e);
                    e.Context.User = null;
                }
            }
        }
    }
}

```



```

VB.Net
Imports System.Security.Cryptography.X509Certificates
Imports System.Security.Permissions
Imports System.Security.Principal
Imports System.Web

Public NotInheritable Class SmartCardAuthenticationModule
    Implements System.Web.IHttpModule

    ' Events
    Public Event Authenticate(ByVal sender As Object, _
        ByVal e As SmartCardAuthenticationEventArgs)

    Public Sub New()
    End Sub

    Public Sub Dispose() Implements System.Web.IHttpModule.Dispose

    End Sub

    Public Sub Init(ByVal context As System.Web.HttpApplication) _
        Implements System.Web.IHttpModule.Init

        AddHandler context.AuthenticateRequest, _
            New EventHandler(AddressOf Me.OnAuthenticateRequest)
    End Sub

    Private Sub OnAuthenticateRequest(ByVal source As Object, ByVal eventArgs As EventArgs)
        Dim httpApp As HttpApplication = DirectCast(source, HttpApplication)
        Dim context As HttpContext = httpApp.Context

        Dim smartCardIdentity As SmartCardIdentity = Me.RetrieveIdentity(context)
        Me.OnAuthenticate(New SmartCardAuthenticationEventArgs(smartCardIdentity, context))
    End Sub

    Private Sub OnAuthenticate(ByVal e As SmartCardAuthenticationEventArgs)
        ' Fire any subscribers to the Authenticate event,
        ' typically from Global.asax Smart
        RaiseEvent Authenticate(Me, e)

        ' Has Context.User already been assigned
        ' from the Authenticate() event above?
        If (e.Context.User Is Nothing) Then
            ' Context.User not populated

            ' Did e.User get assigned from
            ' from the Authenticate() event above?
            If (Not e.User Is Nothing) Then
                ' Yes, assign the user and exit.
                e.Context.User = e.User
            ElseIf Not e.Identity Is Nothing Then
                ' No, but Identity isn't null from Database Call.
                ' User is authenticated
                e.Context.User = New SmartCardPrincipal(e.Identity)
            Else
                ' User isn't authenticated
                Me.Display401Page(e)
                e.Context.User = Nothing
            End If
        End If
    End Sub

    Private Function RetrieveIdentity(ByVal context As HttpContext) As SmartCardIdentity
        Dim identity As SmartCardIdentity = Nothing

        ' Validate X509 Certificate
        If CryptoUtility.IsHttpCertificateValid(context.Request.ClientCertificate) Then
            ' Valid SmartCard, Create the SmartCardIdentity
            identity = New SmartCardIdentity(context.Request.ClientCertificate)
        Else
            ' No Smart Card/Invalid SmartCard
        End If
    End Function
End Class

```

```

        ' Set the identity to null so they aren't authenticated
        identity = Nothing
    End If

    Return identity
End Function

Private Sub Display401Page(ByVal e As SmartCardAuthenticationEventArgs)
    Dim pageName As String = Configuration.UnauthorizedPage
    ' Now dump the Unauthorized Page
    If pageName.Equals(String.Empty) Then
        Me.WriteErrorPageToResponseStream(e.Context.Response, _
            "<html><body><h1>Unauthorized!</h1></body></html>")
    Else
        Me.WriteErrorFileToResponseStream(e.Context.Response, _
            Configuration.UnauthorizedPage)
    End If

    ' finally, bypass all further modules in the HTTP pipeline chain of execution
    ' and directly execute the EndRequest event
    e.Context.ApplicationInstance.CompleteRequest()
End Sub

Private Sub WriteErrorPageToResponseStream(ByVal response As HttpResponse, _
    ByVal pageContent As String)
    response.StatusCode = 401
    response.StatusDescription = "You are not authorized to access this site."
    response.ContentType = "text/html"
    response.Write(pageContent)
    response.Flush()
    response.End()
End Sub

Private Sub WriteErrorFileToResponseStream(ByVal response As HttpResponse, _
    ByVal pageName As String)
    response.StatusCode = 401
    response.StatusDescription = "You are not authorized to access this site."
    response.ContentType = "text/html"
    response.WriteFile(pageName)
    response.Flush()
    response.End()
End Sub
End Class

```

SmartCardAuthenticationEventArgs Implementation

SmartCardAuthenticationEventArgs Class - Inherits from *System.EventArgs*

The *SmartCardAuthenticationEventArgs* is a class used to pass references around of the *HttpContext*, User (Principal), and Identity objects to events that are subscribed to the Authenticate event of the SmartCardAuthenticationModule.

```

C#
using System;
using System.Web;
using System.Security.Principal;
using System.Security.Permissions;
using System.Security.Cryptography.X509Certificates;

namespace SmartCardAuthentication
{
    public sealed class SmartCardAuthenticationEventArgs : EventArgs
    {
        private HttpContext _context;
        private SmartCardIdentity _identity;
        private IPrincipal _user;
    }
}

```

```

public SmartCardAuthenticationEventArgs(SmartCardIdentity identity,
    HttpContext context)
{
    this._identity = identity;
    this._context = context;
}

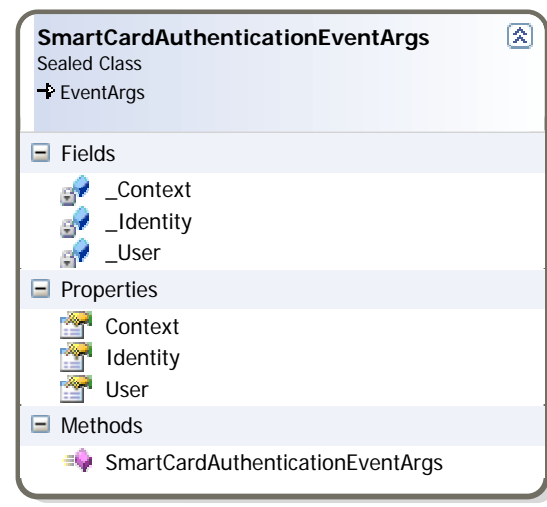
public HttpContext Context
{
    get { return this._context; }
}

public SmartCardIdentity Identity
{
    get { return this._identity; }
}

public IPrincipal User
{
    get { return this._user; }
    set
    {
        // Make sure up-stack callers have permission to change the principal.
        SecurityPermission controlPrincipalPermission =
            new SecurityPermission(SecurityPermissionFlag.ControlPrincipal);
        controlPrincipalPermission.Demand();

        this._user = value;
    }
}
}
}

```



VB.Net

```

Imports System.Web
Imports System.Security.Principal
Imports System.Security.Permissions
Imports System.Security.Cryptography.X509Certificates

Public NotInheritable Class SmartCardAuthenticationEventArgs
    Inherits EventArgs

    Private _context As HttpContext
    Private _identity As SmartCardIdentity
    Private _user As IPrincipal

    Public ReadOnly Property Context() As HttpContext
        Get
            Return Me._context
        End Get
    End Property

    Public ReadOnly Property Identity() As SmartCardIdentity
        Get
            Return Me._identity
        End Get
    End Property

    Public Property User() As IPrincipal
        Get
            Return Me._user
        End Get
        Set(ByVal Value As IPrincipal)
            ' Make sure upstream callers have permission to change the principal.
            Dim controlPrincipalPermission As _
                New SecurityPermission(SecurityPermissionFlag.ControlPrincipal)
            controlPrincipalPermission.Demand()
        End Set
    End Property
End Class

```

```

        Me._user = Value
    End Set
End Property

Public Sub New(ByVal identity As SmartCardIdentity, ByVal context As HttpContext)
    Me._identity = identity
    Me._context = context
End Sub
End Class

```

SmartCardIdentity Implementation

SmartCardIdentity Class – Inherits from *System.Security.Principal.IIdentity*

The *IIdentity* Interface defines the basic functionality of an identity and is used to encapsulate information about the user or entity being validated (*MSDN Documentation*).

This object will hold the state of the current User's Identity using information retrieved from the Smart Card.

```

C#
using System;
using System.Web;
using System.Security.Cryptography.X509Certificates;
using System.Security.Principal;

namespace SmartCardAuthentication
{
    public class SmartCardIdentity : IIdentity
    {
        private string _subject;
        private string _email;
        private string _publicKeyHash;
        private X509Certificate _certificate;

        public string AuthenticationType
        {
            get { return "SmartCard"; }
        }

        public bool IsAuthenticated
        {
            get { return true; }
        }

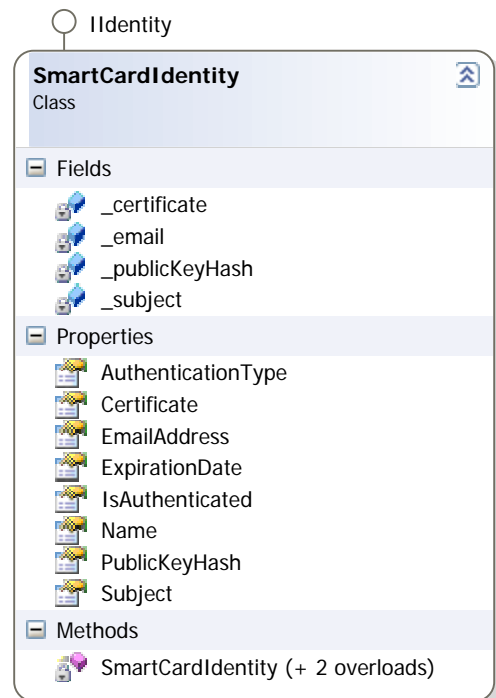
        public string Name
        {
            get { return _subject; }
        }

        public string PublicKeyHash
        {
            get { return _publicKeyHash; }
        }

        public string EmailAddress
        {
            get { return this._email; }
            set { this._email = value; }
        }

        public X509Certificate Certificate
        {
            get { return this._certificate; }
        }
    }
}

```




```

public DateTime ExpirationDate
{
    get
    {
        return DateTime.Parse(
            this._certificate.GetExpirationDateString()
        );
    }
}

public string Subject
{
    get { return this._subject; }
}

public SmartCardIdentity(HttpClientCertificate certificate)
{
    this._certificate = new X509Certificate(certificate.Certificate);
    this._publicKeyHash = CryptoUtility.GetPublicKeyHash(certificate);
    this._subject = certificate.Subject;
}

public SmartCardIdentity(X509Certificate certificate)
{
    this._certificate = certificate;
    this._publicKeyHash = CryptoUtility.GetPublicKeyHash(certificate);
    this._subject = certificate.GetName();
}
}
}

```

```

VB.Net
Imports System.Web
Imports System.Security.Cryptography.X509Certificates
Imports System.Security.Principal

Public Class SmartCardIdentity
    Implements IIdentity

    Private _subject As String
    Private _email As String
    Private _publicKeyHash As String
    Private _certificate As X509Certificate

    Public ReadOnly Property PublicKeyHash() As String
        Get
            Return _publicKeyHash
        End Get
    End Property

    Public ReadOnly Property AuthenticationType() As String _
        Implements IIdentity.AuthenticationType
        Get
            Return "SmartCard"
        End Get
    End Property

    Public ReadOnly Property IsAuthenticated() As Boolean _
        Implements IIdentity.IsAuthenticated
        Get
            ' The fact that this identity object exists means that
            ' the user was Authenticated
            Return True
        End Get
    End Property

    Public ReadOnly Property Name() As String _
        Implements IIdentity.Name
        Get
            Return _subject
        End Get
    End Property
End Class

```

```

        End Get
    End Property

    Public Property EmailAddress() As String
        Get
            Return Me._email
        End Get
        Set(ByVal Value As String)
            Me._email = Value
        End Set
    End Property

    Public ReadOnly Property Certificate() As X509Certificate
        Get
            Return Me._certificate
        End Get
    End Property

    Public ReadOnly Property ExpirationDate() As DateTime
        Get
            Return DateTime.Parse(Me._certificate.GetExpirationDateString())
        End Get
    End Property

    Public ReadOnly Property Subject() As String
        Get
            Return Me._subject
        End Get
    End Property

    Public Sub New(ByVal certificate As HttpClientCertificate)
        Me._certificate = New X509Certificate(certificate.Certificate)
        Me._publicKeyHash = CryptoUtility.GetPublicKeyHash(certificate)
        Me._subject = certificate.Subject
    End Sub

    Public Sub New(ByVal certificate As X509Certificate)
        Me._certificate = certificate
        Me._publicKeyHash = CryptoUtility.GetPublicKeyHash(certificate)
        Me._subject = certificate.GetName()
    End Sub

End Class

```

SmartCardPrincipal Implementation

The **SmartCardPrincipal** Class – Inherits from *System.Security.Principal.IPrincipal*

From the MSDN documentation, a principal object represents the security context of the user on whose behalf the code is running, including that user's identity (IIdentity) and any roles to which they belong. The Principal is the object that gets interrogated when a PrincipalPermission Demand is made to make sure it is in the proper role to perform the requested operation.

When *SmartCardPrincipal.IsInRole()* is called, our Smart Card aware object will return whether or not the Principal is in the requested role. On first run, this class will populate a *Hashtable* with the roles for the current user from the database. The key to lookup the Users' Roles in the database will be a SHA256 hash of the Users' public key.

Also notice that both *IsElevatedUser* and *IsInRole()* is virtual/Overridable – This allows for custom role resolution if you have a different methodology to resolve roles other than database role resolution provided.

```

C#
using System;
using System.Configuration;
using System.Web;

```

Implementing Smart Card Authentication with ASP.NET

```
using System.Web.Security;
using System.Collections;
using System.Security.Principal;

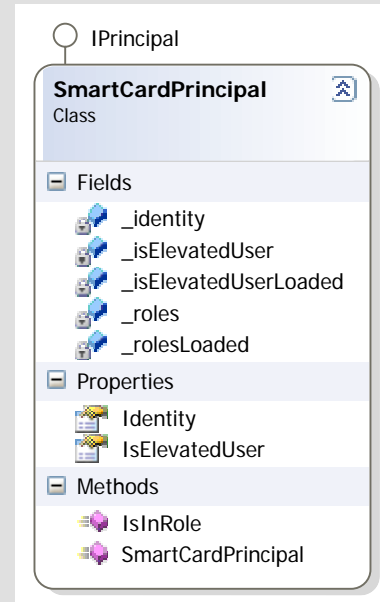
namespace SmartCardAuthentication
{
    public class SmartCardPrincipal : IPrincipal
    {
        private SmartCardIdentity _identity;
        private Hashtable _roles;
        private bool _isElevatedUser;
        private bool _rolesLoaded;
        private bool _isElevatedLoaded;

        public IIdentity Identity
        {
            get
            {
                return this._identity;
            }
        }

        public SmartCardPrincipal(SmartCardIdentity identity)
        {
            this._identity = identity;
            this._rolesLoaded = false;
            this._isElevatedLoaded = false;
        }

        public virtual bool IsElevatedUser
        {
            get
            {
                if (!_isElevatedLoaded)
                {
                    lock (this)
                    {
                        // Evaluate the user against the database
                        // to see if they have an elevated account
                        DataAccess dAccess = new DataAccess();
                        this._isElevatedUser =
                            dAccess.AuthenticateUser(_identity.PublicKeyHash);
                    }
                    this._isElevatedLoaded = true;
                }
                return this._isElevatedUser;
            }
        }

        public virtual bool IsInRole(string role)
        {
            if (!_rolesLoaded)
            {
                lock (this)
                {
                    if (this.IsElevatedUser)
                    {
                        DataAccess dAccess = new DataAccess();
                        _roles = dAccess.GetPrincipalRoles(_identity.PublicKeyHash);
                        _roles.Add("User", "User");
                    }
                    else
                    {
                        _roles = new Hashtable(1);
                        _roles.Add("User", "User");
                    }
                    _rolesLoaded = true;
                }
            }
            return _roles.Contains(role);
        }
    }
}
```



```
}
}
```

VB.Net

```
Imports System.Configuration
Imports System.Web
Imports System.Web.Security
Imports System.Collections
Imports System.Security.Principal

Public Class SmartCardPrincipal
    Implements IPrincipal

    Private _identity As SmartCardIdentity
    Private _roles As Hashtable
    Private _rolesLoaded As Boolean
    Private _isElevatedLoaded As Boolean

    Public ReadOnly Property Identity() As IIdentity _
        Implements IPrincipal.Identity
        Get
            Return Me._identity
        End Get
    End Property

    Public Sub New( _
        ByVal identity As SmartCardIdentity _
    )
        Me._identity = identity
        Me._rolesLoaded = False
        Me._isElevatedLoaded = False
    End Sub

    Public Overridable ReadOnly Property IsElevatedUser() As Boolean
        Get
            If (Not _isElevatedLoaded) Then
                SyncLock (Me)
                    ' Authenticate the user against the database
                    Dim dAccess As New DataAccess
                    _isElevatedUser = dAccess.AuthenticateUser(Me._identity.PublicKeyHash)

                    _isElevatedLoaded = True
                End SyncLock
            End If

            Return _isElevatedUser
        End Get
    End Property

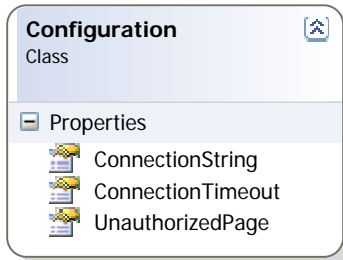
    Public Overridable Function IsInRole(ByVal role As String) As Boolean _
        Implements IPrincipal.IsInRole
        If (Not _rolesLoaded) Then
            SyncLock (Me)
                If _identity.IsElevatedUser Then
                    ' If the user is elevated, retrieve roles from the database
                    Dim dAccess As New DataAccess
                    _roles = dAccess.GetPrincipalRoles(_identity.PublicKeyHash)
                    ' Finally add the Default Role
                    _roles.Add("User", "User")
                Else
                    _roles = New Hashtable(1)
                    ' Regular user
                    _roles.Add("User", "User")
                End If
                _rolesLoaded = True
            End SyncLock
        End If
        Return _roles.Contains(role)
    End Function
End Class
```

Additional Implementation Details

There are several more classes involved in this implementation, but they go beyond the scope of the HttpModule. If you download the sample code, you can take a look at them. They are barely implemented so I'd recommend you use them at your own risk.

The Configuration Class

The *Configuration* class resolves some things like 401 error page, the database connection string, and the database connection timeout. The class had the following structure:

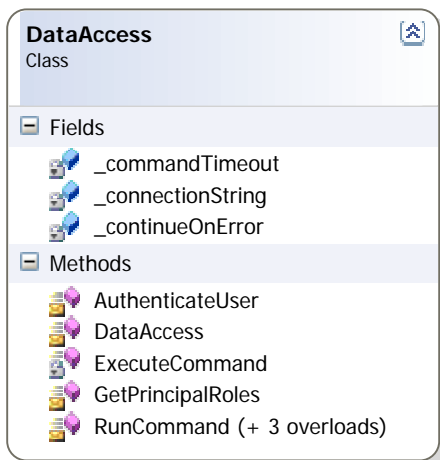


These values are retrieved from the web.config file:

```
<configuration>
  <system.web>
    ...
  </system.web>
  <appSettings>
    <add key="SmartCardAuthentication_UnauthorizedPage"
        value="C:\WINNT\help\iisHelp\common\401-1.htm"/>
    <add key="SmartCardAuthentication_ConnectionString"
        value="integrated security=SSPI;data source=Server;initial catalog=Database"/>
    <add key="SmartCardAuthentication_ConnectionTimeout"
        value="180" />
  </appSettings>
</configuration>
```

The DataAccess Class

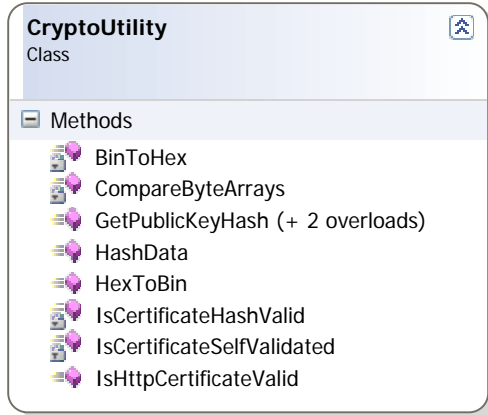
The *DataAccess* class had some methods for quickly retrieving data from the database.



The CryptoUtility Class

Implementing Smart Card Authentication with ASP.NET

This class performed operations (hash compare, BinToHex() / HexToBin() conversions, etc.) on the X509 certificates. It also generates the Public Key hash used to retrieve users out of the database, and validates some simple properties on the X509 certificate for an internal sanity check. IIS should already be catching these problems before it gets to our code, but it can't hurt to check again.



Some Final Tests

Once you have the SmartCardAuthenticationModule code up and running, a simple way to test it is as follows:

```
C#  
private void Page_Load(object sender, EventArgs e)  
{  
    SmartCardIdentity smartCardIdentity = (SmartCardIdentity)Me.User.Identity;  
    SmartCardPrincipal smartCardPrincipal = (SmartCardPrincipal)Me.User;  
  
    Response.Write("Name: " + smartCardIdentity.Name + "<br>");  
    Response.Write("Is Authenticated: " + smartCardIdentity.IsAuthenticated + "<br>");  
    Response.Write("Authentication Type: " + smartCardIdentity.AuthenticationType + "<br>");  
    Response.Write("Elevated User: " + SmartCardPrincipal.IsElevatedUser + "<br>");  
    Response.Write("Is in role Administrator: " + this.User.IsInRole("Administrator"));  
}
```

```
VB.Net  
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
  
    Dim smartCardIdentity As SmartCardIdentity = DirectCast(Me.User.Identity,  
SmartCardIdentity)  
    Dim smartCardPrincipal As SmartCardPrincipal = DirectCast (Me.User, SmartCardPrincipal)  
  
    Response.Write("Name: " & smartCardIdentity.Name & "<br>")  
    Response.Write("Is Authenticated: " & smartCardIdentity.IsAuthenticated & "<br>")  
    Response.Write("Authentication Type: " & smartCardIdentity.AuthenticationType & "<br>")  
    Response.Write("Elevated User: " & SmartCardPrincipal.IsElevatedUser & "<br>")  
    Response.Write("Is in role Administrator: " & Me.User.IsInRole("Administrator"))  
End Sub
```

Implementing Authorization in ASP.NET

Once you have your *SmartCardPrincipal* setup, there are several ways to implement authorization with the *IPrincipal* using Code Access Security (CAS) for authorization within ASP.NET.

We can configure Role base authorization using the web.config file, using `PrincipalPermission Demands`, or `IPrincipal.IsInRole()` checks in code.

- **Declarative**

Principal Permissions can be used to decorate methods that will demand upstream callers in the stack have a particular Role.

```
C#  
using System.Security.Permissions;  
  
...  
[PrincipalPermission(SecurityAction.Demand, Role="Administrator"),  
PrincipalPermission(SecurityAction.Demand, Role="Auditors")]  
public void DoSomethingImportant()  
{  
}
```

```
VB.Net  
Imports System.Security.Permissions  
  
...  
<PrincipalPermission(SecurityAction.Demand, Role:="Administrator"), _  
PrincipalPermission(SecurityAction.Demand, Role:="Auditors")> _  
Public Sub DoSomethingImportant()  
End Sub
```

- **Imperative**

Principal Permissions can be used to make demands programmatically to upstream callers in the stack have a particular Role.

```
C#  
using System.Security.Permissions;  
  
...  
public void DoSomethingImportant()  
{  
    PrincipalPermission permCheck = new PrincipalPermission(Nothing, "Administrators");  
    permCheck.Demand();  
}
```

```
VB.Net  
Imports System.Security.Permissions  
  
...  
Public Sub DoSomethingImportant()  
    Dim permCheck As New PrincipalPermission(Nothing, "Administrators")  
    permCheck.Demand()  
End Sub
```

- **IPrincipal.IsInRole() Check**

We can check if the `IPrincipal` is in the role we require (which is exactly what the `PrincipalPermission` class does by using the `IPrincipal` stored in the `Thread.CurrentPrincipal`):

```
C#  
if (myPrincipal.IsInRole("Administrators"))  
{  
    ...  
}
```

```
VB.Net  
If myPrincipal.IsInRole("Administrators") Then  
    ...  
End If
```

- **Web.Config**

To allow all Administrators and deny everyone else to a folder called 'Admin', and to allow only Auditors into a folder called 'Reports', we'd add the following to the web.config

Specify access permissions to files and/or folders in the web.config:

```
<configuration>
  <system.web>
    ...
  </system.web>
  <location path="Admin">
    <system.web>
      <authorization>
        <allow roles="Administrator" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
  <location path="Reports">
    <system.web>
      <authorization>
        <allow roles="Auditor" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

Conclusion

ASP.NET provides a powerful, yet simple way to implement custom authentication functionality in the HTTP Pipeline using HTTP Modules. IIS also has robust support for Client Certificates and when combined, Http Modules in ASP.NET and IIS make a great platform for developing sites that need to use Smart Cards for authentication and authorization.

Appendix A – Further Reading

1. <http://support.microsoft.com/kb/307985>
INFO: ASP.NET HTTP Modules and HTTP Handlers Overview
This article introduces the ASP.NET HTTP modules and HTTP handlers
2. <http://msdn.microsoft.com/msdnmag/issues/02/09/HTTPIPipelines/>
This article introduces the architecture of the pipeline and shows how you can use it to add sophisticated functionality to an ASP.NET-based app.
3. <http://msdn.microsoft.com/msdnmag/issues/02/05/asp/default.aspx>
A (brief) look at HTTP modules in ASP.NET.
4. <http://support.microsoft.com/kb/887289>
HTTP module to check for canonicalization issues with ASP.NET
To aid customers in protecting their ASP.NET applications, Microsoft has made available an HTTP module that implements canonicalization best practices.
5. <http://msdn2.microsoft.com/en-us/library/aa479332.aspx>
Using HTTP Modules and Handlers to Create Pluggable ASP.NET Components

In this article, Scott Mitchell and Atif Aziz show how you can use HTTP modules and handlers to add error logging to your ASP.NET applications. (22 printed pages)
6. <http://msdn2.microsoft.com/en-us/library/ms972974.aspx>
URL Rewriting in ASP.NET (using HTTP Handlers)

Examines how to perform dynamic URL rewriting with Microsoft ASP.NET. URL rewriting is the process of intercepting an incoming Web request and automatically redirecting it to a different URL. Discusses the various techniques for implementing URL rewriting, and examines real-world scenarios of URL rewriting. (31 printed pages)
7. <http://support.microsoft.com/kb/313070>
HOW TO: Configure Client Certificate Mappings in Internet Information Services (IIS) 5.0
8. <http://support.microsoft.com/kb/272175/EN-US/>
HOW TO: Configure Active Directory Certificate Mapping
9. <http://support.microsoft.com/kb/216906/EN-US/>
Comparing IIS 5.0 Certificate Mapping and Native Windows 2000 Active Directory Certificate Mapping
10. <http://www.google.com/microsoft?q=HTTP+modules+&hq=microsoft&btnG=Google+Search>
Search Google for more!

Disclaimer

There is no warranty expressed, written, or implied for any code or methodology presented in this document. It is for informational purposes only.