Encrypting and Decrypting Configuration Section in web.config

When it comes to securing IIS web applications on [Windows Server 20008 R2](https://technet.microsoft.com/en-us/library/dd349801(v=ws.10).aspx) or [Windows Server 2012 R2](http://www.microsoft.com/en-us/server-cloud/products/windows-server-2012-r2/) one typically thinks of firewalls, access control lists (ACL), and using an application pool identity. These security measures will protect a site from external threats . However, .Net configuration files which typically store username and password data are text files so anyone with admin access to the server can read their contents.  The only way to prevent prying eyes from seeing app.config or web.config passwords is to encrypt them. Fortunately encrypting the [connectionStrings section](https://msdn.microsoft.com/library/bf7sd233%28v=vs.100%29.aspx?f=255&MSPPError=-2147217396" \t "_blank) of a config file is straight foward. You can also encrypt other configuration sections in addition to **connectionStrings** section. Encrypting and decrypting config files can be performed programatically using .NET Framework methods or by using the [ASP.NET IIS Registration tool](https://msdn.microsoft.com/library/k6h9cz8h%28v=vs.100%29.aspx)(**aspnet\_regiis.exe**).  With the encryption commands you can target either the path to the config file or reference an IIS application name. In my examples I will be encrypting and decrypting the **connectionStrings**section with the .NET Framework 4.

[**NOTE**: use “connectionStrings” instead of “section” in below commands when you want to encrypt or decrypt connectionStrings section of web.config.]

Encrypting Configuration Sections

You will find aspnet\_regiis.exe in the C:\Windows\Microsoft.NET\Framework\version\ folder.  With the .NET Framework you can use the builtin protected configuration providers [RSAProtectedConfigurationProvider](https://msdn.microsoft.com/en-us/library/system.configuration.rsaprotectedconfigurationprovider.aspx)  or [DPAPIProtectedConfigurationProvider](https://msdn.microsoft.com/en-us/library/system.configuration.dpapiprotectedconfigurationprovider.aspx) to encrypt and decrypt sections of your config files. You can also create your own provider. The general synatax to encrypt a config section is as follows:

**aspnet\_regiis.exe -pef section physical\_directory -prov provider  
or  
aspnet\_regiis.exe -pe section -app virtual\_directory -prov provider**

It is important to note when using **aspnet\_regiis.exe** to encrypt or decrypt config files and you specify a physical path (rather than a web app name) the command is hardcoded for a file named “web.config”.  If you are trying to run the command against an app.config you will first need to rename that file to web.config before running the command. Rename it back afterwards before using it. For this reason I find it easier to create a .bat file hardcoded with the necessary command syntax to encrypt my configs and then a 2nd .bat file to decrypt my configs.

On my Windows 2012 R2 server I have setup an IIS 8.5 site called domain1.com. For the example below I am using the builtin DPAPI provider to encrypt a web.config in c:\domains\domain1.com. The encrypted web.config is shown below.

**aspnet\_regiis.exe -pef "connectionStrings" "c:\domains\domain1.com"   
-prov "DataProtectionConfigurationProvider"**

[](http://www.peterviola.com/wp-content/uploads/2016/02/encrypted-config.jpg)

Decrypting Configuration Sections

Following steps above we have now encrypted the **connectionStrings section** of the web.config for domain1.com. Naturally We also need to be able to decrypt it. When decrypting a config section you do not need to specify the **protected configuration provider**. Just like when encrypting a config file we can target either a file path or IIS web application name. Here is the syntax to decrypt a configuration file section:

**aspnet\_regiis.exe –pdf section physical\_directory   
or  
aspnet\_regiis.exe –pd section -app virtual\_directory**

In my example below I decrypt the connectionStrings section of my web.config in c:\domains\domain1.com. As a reminder again when using the **–pdf**option we do not need to specify “web.config” in the syntax.

**aspnet\_regiis.exe –pdf "connectionStrings" "c:\domains\domain1.com"**

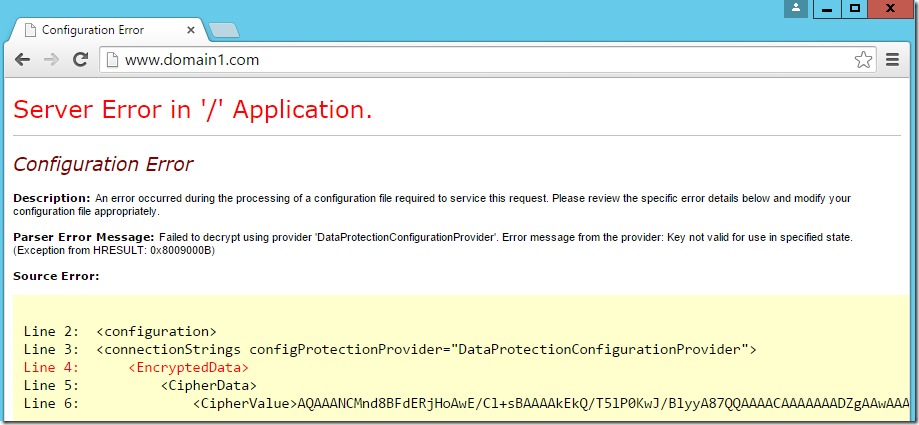
[](http://www.peterviola.com/wp-content/uploads/2016/02/decrypt-config-connectstrings.png)

After running the above command, the **connectionStrings** section of the web.config is decrypted as shown below. Once I am done editing my connection string I will follow best practices and encrypt the**connectionStrings** section again.

[](http://www.peterviola.com/wp-content/uploads/2016/02/web-config-decrypted.png)

Failed to decrypt using provider error

It is important to note that when encrypting your config files the encryption key is stored locally on the server which means if you need to move your encrypted config file to another server you will need to either decrypt the config file first before moving it to the new server or export the key prior to moving and install it on the new server.  If you move an encrypted config file to a server without exporting the encryption key you will receive an error like below indicating: **Failed to decrypt using provider … Key not valid for use in specified state**.

[](http://www.peterviola.com/wp-content/uploads/2016/02/crypto-error.jpg)

Creating an RSA Key Container

Fortunately moving encryption keys between servers is straight forward. We can create our own **RSA key container**, export it to a file, and then move it from server to server as needed. This is ideal for multi node web farm solutions where applications are deployed across multiple servers.  Use the following syntax to create an RSA key container. Be sure to include the **–exp** option so the container can be exported:

**aspnet\_regiis -pc "MyFarmKey" –exp**

[](http://www.peterviola.com/wp-content/uploads/2016/02/creating-mycrypto.jpg)

Adding the configProtectedData section to your config

Next you will add the following **configProtectedData** section to your web.config.

<configProtectedData>

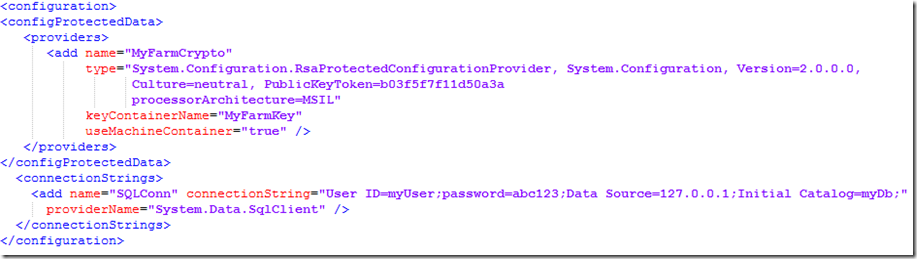
<providers>

<add name="MyFarmCrypto" type="System.Configuration.RsaProtectedConfigurationProvider,   
System.Configuration, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a processorArchitecture=MSIL" keyContainerName="MyFarmKey" useMachineContainer="true" />

</providers>

</configProtectedData>

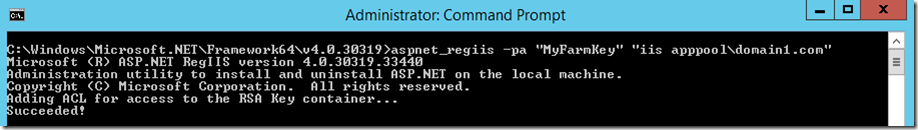
Below is how my web.config looks now that I have added the **configProtectedData** section.

[](http://www.peterviola.com/wp-content/uploads/2016/02/myfarmkey-config.png)

Assigning permissions to the RSA key container

Before the new RSA key container is ready to be used by my site domain1.com, I need to assign the [application pool identity](http://www.iis.net/learn/manage/configuring-security/application-pool-identities) permission to access it. On the server in my example the application pool identity for domain1.com is **ApplicationPoolIdentity**. I use the following syntax to assign this user to the new RSA key container:

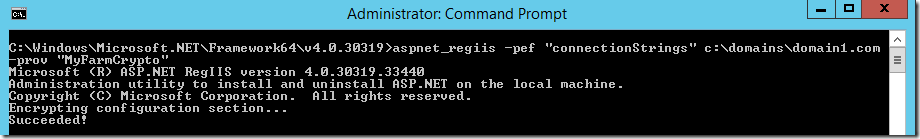
**aspnet\_regiis -pa "MyFarmKey" "iis apppool\domain1.com"**

[](http://www.peterviola.com/wp-content/uploads/2016/02/myfarmkey-identity.png)

Encrypting a config with an RSA key container

After adding the **configProtectedData section** to the web.config and granting permission to the RSA key container for domain1.com’s application pool identity, I’ll run the encryption command again using the new “MyFarmCrypto” RSA key container:

**aspnet\_regiis.exe -pef "connectionStrings" "c:\domains\domain1.com"  
 -prov "MyFarmCrypto"**

[](http://www.peterviola.com/wp-content/uploads/2016/02/image-4.png)

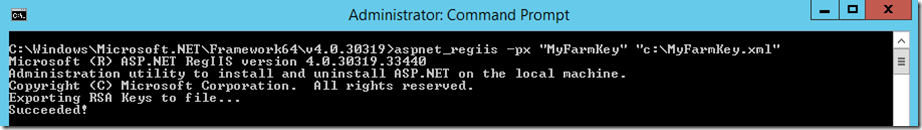
In the image above we see the encryption succeeded. Note in the command syntax above we are specifying the configProtectionProvider name **MyFarmCrypto** and not the RSA key container name. If you mix that up you’ll get an error. We can see below how domain1.com’s web.config now looks after being encrypted with the new RSA key container.

[](http://www.peterviola.com/wp-content/uploads/2016/02/farmcrypto-encrypted.png)

Exporting and Importing an RSA Key Container

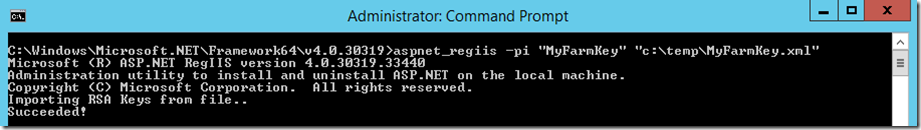
Now that we’ve successfully created and tested our new**RSA key Container** we need to export it to a file. Once it’s saved in a file we can then copy it to other servers for installation as needed. It is important to remember to use **–pri** option to include the private key when the export file is created otherwise you will not be able to decrypt information on the next server .

**aspnet\_regiis -px "MyFarmKey" "c:\MyFarmKey.xml" –pri**

[](http://www.peterviola.com/wp-content/uploads/2016/02/export-farmkey.png)

Having logged into another server and copied the MyFarmKey.xml file to c:\temp I will import the key fil using the following command:

**aspnet\_regiis -pi "MyFarmKey" "c:\temp\MyFarmKey.xml"**

[](http://www.peterviola.com/wp-content/uploads/2016/02/import-farmkey.png)

For security purposes, after importing the key on a new server, delete the key .xml file from the server to ensure someone unauthorized doesn’t use it decrypt data. This of course assumes that you have backed up the file off server somewhere safe.

To permanently delete the RSA key container from a server you should run this command:

**aspnet\_regiis -pz "MyFarmKey"**

Summary

The .NET Framework offers powerful encryption tools to secure sensitive information like usernames and passwords in application connection strings. When encrypting a config file on a server the private key used to decrypt the information is local to the server. Creating an RSA key container will enable you to encrypt information with the same private key across multiple servers.

Note: One can integrate above commands into an HTML Application (.hta) in order to avoid going to Command prompt every time he/she wants to encrypt/decrypt some section in Web.Config file.

For example,

To Encrypt Section, write below line in HTML Application.

objShell.Run "%comspec% /k %windir%\Microsoft.NET\Framework\v2.0.50727\aspnet\_regiis.exe -pef ConfigSection PathToWebConfig -prov ProviderName", NORMAL\_WINDOW,True

To Decrypt Section, write below line in HTML Application.

objShell.Run "%comspec% /k %windir%\Microsoft.NET\Framework\v2.0.50727\aspnet\_regiis.exe -pdf ConfigSectionName PathToWebConfig ", NORMAL\_WINDOW,True