SSIS – An Effortless Two Step Approach to Protect Sensitive Information in Xml Configuration Files

Marudhamaran Gunasekaran

Contents

Acknowledgements

Writing a tool or an article becomes best when it is a communal effort. While I conceived and developed the library and tool, I want to acknowledge a few people who contributed in various ways.

Balabhadra Pavan Kumar, Dheeraj Dhamija, Balakrishna Allidi for constantly proving feedback, improvement aspects, and for reviewing sections of the support guide.

Manohara Mahadevappa for putting up with me during early development cycles and suggesting a bit of user friendly features.

Narasimha A Prakash, Sriram Seshan for being good enough to support me whenever I approached them.

I. Introduction

One of the ways of saving configuration entries for SSIS packages is an Xml configuration file, probably because they are simple editable text files, portable, and so on. When developing ASP.Net or Windows applications, most of us are cautious enough to encrypt the sensitive information that is saved in their configuration files. Reasons? May be it is thought that ASP.Net applications may reside on a web server that is exposed to outside world and hence they might be susceptible to attacks of sorts, and when that happens, we would not want to expose sensitive information in plain text, so we encrypt them. And the .Net framework provides it as an out-of-the-box functionality. In the most common scenarios that we have known, SSIS packages run as scheduled jobs inside a server that is located far inside the corporate firewall with ports closed, just sitting there in a good hope that *'I am not vulnerable to attacks, so I can just be here with an xml configuration file that has connections strings, besides others things, in plain text'*. As protection comes with security applied in various layers, the kind of attitude to rely on good hope and on firewall may just not be enough. In the wake of recent attacks in high profile organizations, if by any miracle, someone infiltrates and steals the data, we don't want to expose further information in plain text. Do we? The risk could even come from a new developer that accidentally performs some unwanted activities on data (by using the information in the xml configuration file saved as plain text) that is meant to be protected from accidental damage, both foreign and domestic. That is the motivation to write this library and executable to help us encrypt information in our SSIS xml configuration files and hence this article. The entire process is done in two simple steps. If you are an experienced developer, read the prerequisites section and proceed directly to section *V* and *VI*.

II. Competitive approaches

That said, there are best practices that you can adhere to, like protecting the configuration using Access Control Lists that are tied to your Active Directory efforts, storing configurations in a SQL server database somewhere. Below are two blog entries that talk about our present options.

1. SSIS: Storing passwords
2. SSIS: Encrypted Configurations
3. BI xPress Secure Configuration Manager

While there are pros and cons for approaches, they might not suit your need due to restrictions of company's policies. And most developers prefer to encrypt and decrypt using a custom script task. While this is a viable option, it is a lot of effort to write a separate encryption module and unit test it, and not all developers are expert programmers that care about strong cryptography and key management. Most of the times the symmetric key used for encryption/decryption is hardcoded in the script task and the package itself is protected by a password. The encryption algorithm that the proposed library (referred to as ***SSISCipherUtil.dll*** from now on) uses state-of-the-art methods of encrypting information that hands over the cryptographic key management to the Windows operating system itself. As a matter of fact the ASP.Net out of the box functionality supports encrypting configuration information using DPAPI and RSA. SSISCipherUtil.dll comes with those two options. DPAPI associates cryptographic key with the windows user accounts and RSA uses key containers. That's just a one liner. More at msdn - DPAPI, RSA.

When you have an encrypted configuration that you would like to be ported across multiple servers, use RSA, export the key pair from one server as xml, import it another server, then destroy the key pair xml file. If the package just sits on one server use RSA or DPAPI.

III. Prerequisites

1. Windows XP or later workstations, Windows Server 2003 or later server operating systems.
2. SSISCipherUtil.dll requires .Net framework 2.0 or later – is the library that helps in encryption/decryption during package setup and at package runtime.
3. SSISCipherBoy.exe requires .Net framework 3.5 or later – is the tool that helps you encrypt/decrypt a configuration entry at design time.
4. Auto code generation functionality of SSISCipherBoy.exe of requires a computer with Business Intelligence Development Studio (BIDS) installation.
5. Requires Administrator Privileges. *Run as administrator* option if UAC is switched on.
6. Supports only String encryption/decryption. No other data types are allowed.
7. Supports only Package.Variables and Package.Connections collections.
8. Supported Package *ProtectionLevel* values are *DontSaveSensitive*, *EncryptAllWithPassword, EncryptSensitiveWithPassword. EncryptAllWithUserKey*, *EncryptSensitiveWithUserKey* are supported only when the user account that modifies the package is used to run the package, in the same machine.

Below is the screenshot of the main window of ***SSISCipherboy.exe***. Let's explore the features one by one in later sections.

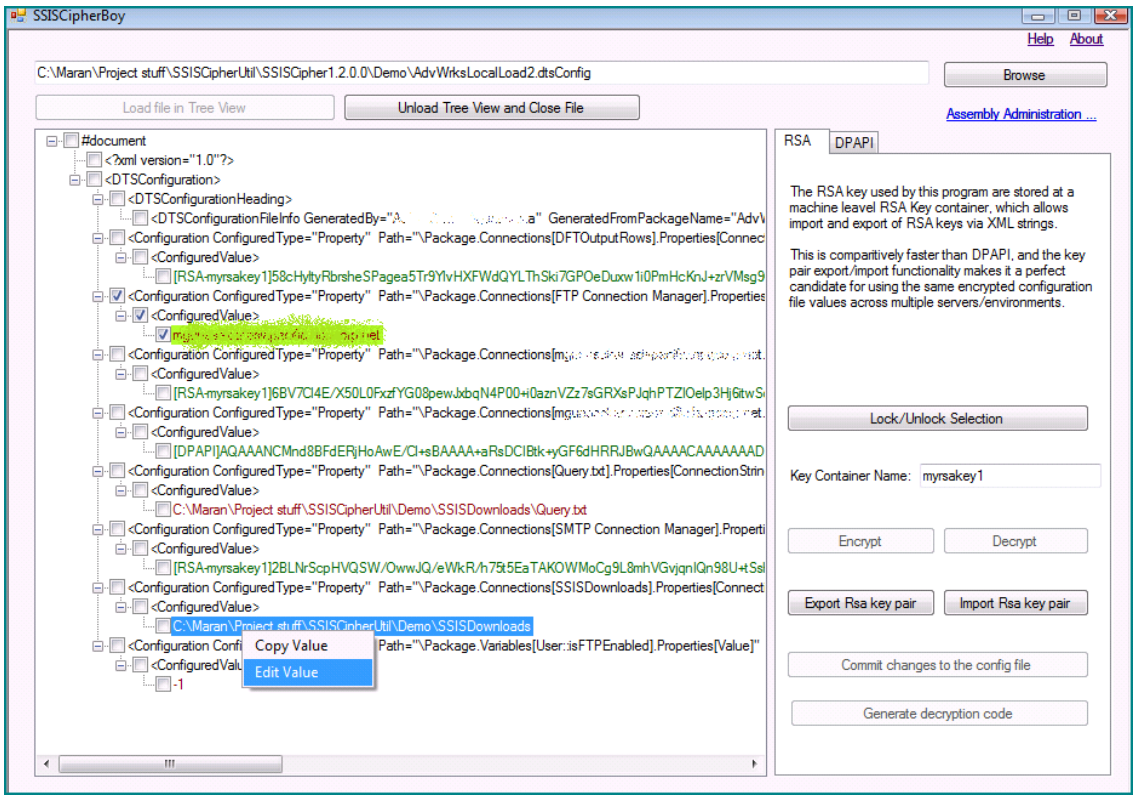

Fig. 3.1

IV. Preparing your package for configuration

If you here reading this article, I assume that you are an experienced developer that has configured and managed SSIS packages before. However, for the sake of completeness below are msdn references that explain saving SSIS configurations in xml files:

1. Package Configurations
2. Understanding Integration Services Package Configurations

Most important thing, if you don't *uncheck* the *Enable package configurations* in *SSIS Package Configurations Organize* and if there is a configuration set up for the package, then the config files passed to the package using the */config* option of *dtexec.exe* will not be effective. This might seem counter intuitive but that's how SQL Server 2008 Integration Services works.

In the following steps, we will be running an unmodified package named ***AdvWrksLocalLoad2.dtsx*** with its original configuration ***AdvWrksLocalLoad2.dtsConfig***, then run the modified package ***AdvWrksLocalLoad2-Mod.dtsx*** with an encrypted configuration ***AdvWrksLocalLoad2.dtsConfig***, and then run the latter on a Windows 2008 R2 server (just like a production environment).

V. Step 1 – Using SSISCipherBoy.exe – Select and encrypt configuration entries

Now, without further delay, let's pick a package and let's encrypt its configuration entries. All the samples along with the tools are zipped and attached to this article. If someone's interested in looking through the source code of ***SSISCipherBoy.exe*** and ***SSISCipherUtil.dll*** or enhancing its functionally or aesthetically, please contact me, I'd be pleased to share them.

Below is a list of files that we will be working with for demonstration.

AdvWrksLocalLoad2.dtsConfig	11/19/2012 9:52 PM	DTSCONFIG File
AdvWrksLocalLoad2.dtsx	11/19/2012 4:05 PM	DTSX File
AdvWrksLocalLoad3.dtsx	11/19/2012 4:05 PM	DTSX File
AdvWrksLocalLoad4.dtsx	11/19/2012 4:05 PM	DTSX File
AdvWrksLocalLoad5.dtsx	11/19/2012 4:05 PM	DTSX File
InnerVariablesTest.dtsx	11/17/2012 9:05 PM	DTSX File
MultipleEntryPoints.dtsx	11/15/2012 6:59 PM	DTSX File
TestEventFiring.dtsx	11/10/2012 4:11 PM	DTSX File

Fig. 5.1

AdvWrksLocalLoad2.dtsx in ***Business Intelligence Development Studio*** is a package that does a few things, and the task the package performs is irrelevant here because we are concentrating on encrypting its configuration values in the xml configuration file. ***AdvWrksLocalLoad2.dtsx*** is protected with the password ***test*** *[ProtectionLevel=EncryptAllWithPassword and PackagePassword=test].*

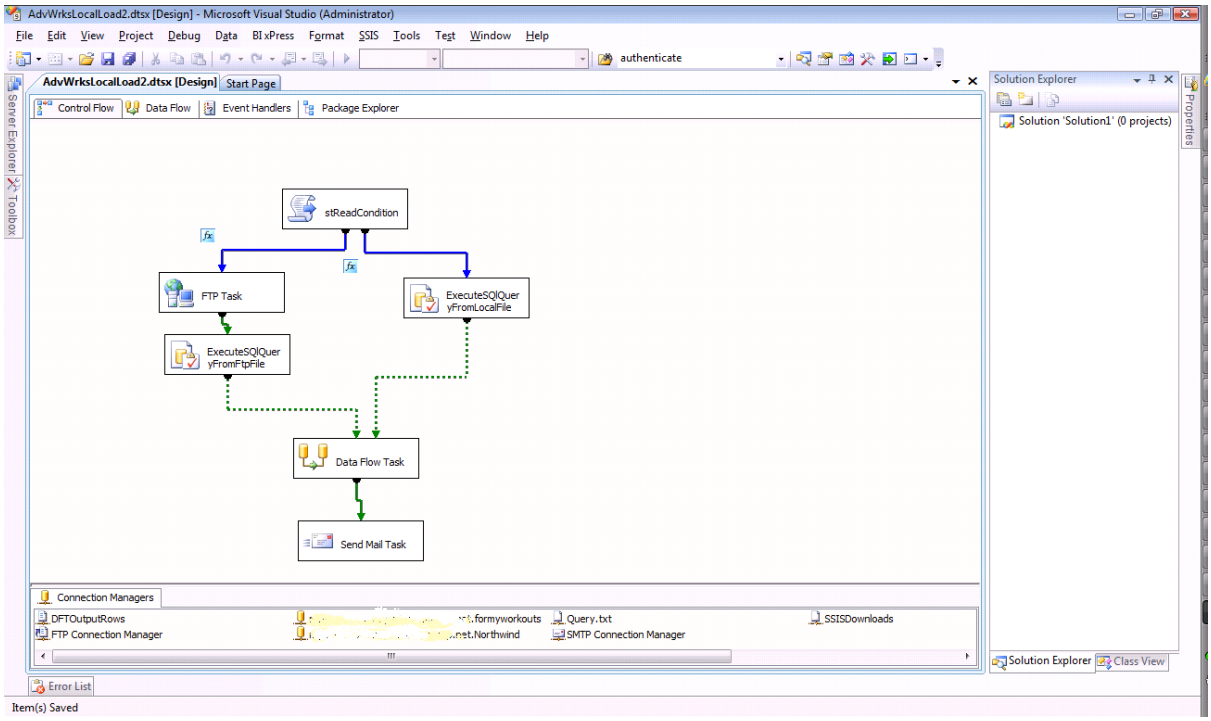

Fig. 5.2

AdvWrksLocalLoad2.dtsConfig is the configuration file for ***AdvWrksLocalLoad2.dtsx*** that appears like below in Internet Explorer. As you could see the connection strings are mere plain text.

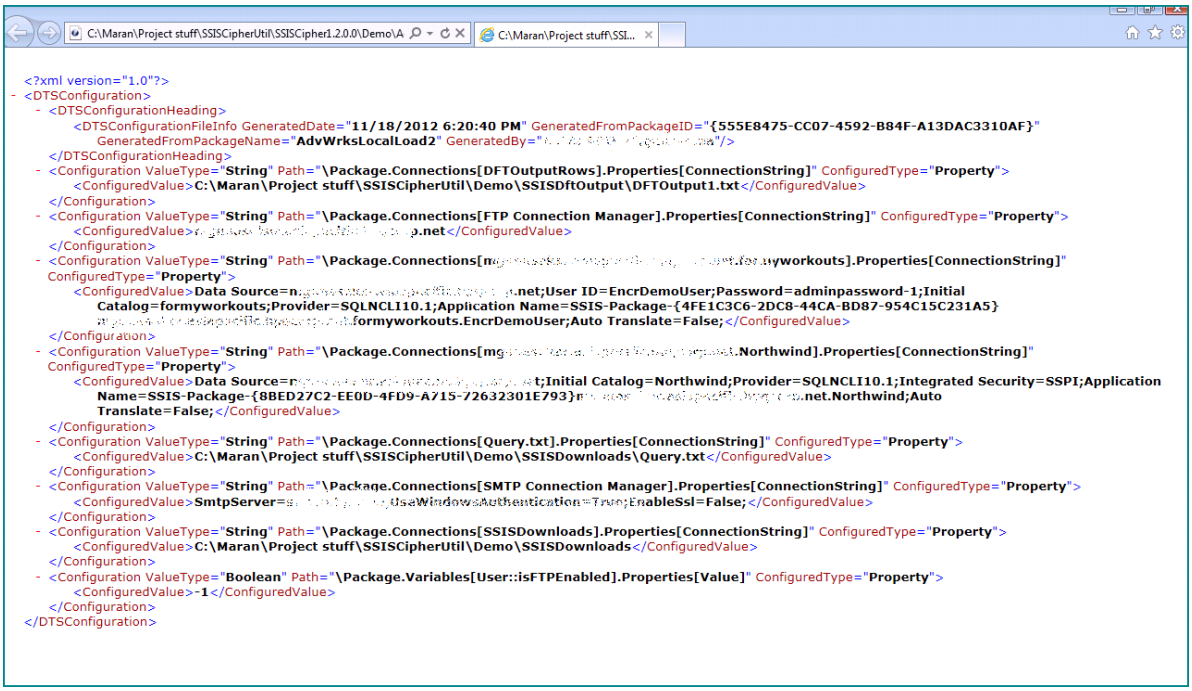

Fig. 5.3

Let's encrypt them. ***Run SSISCipherBoy.exe as administrator***. This program requires administrator privileges. On Windows systems with User Access Control (UAC) switched on, running as administrator is a mandatory requirement. Otherwise the program will not load properly; you will have to go to Task Manager to end it.

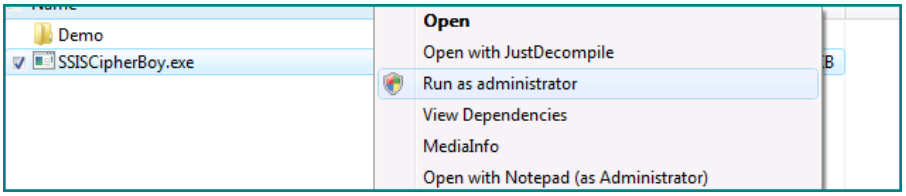

Fig. 5.4

When you run this program on a computer for the first time, it will install ***SSISCipherUtil.dll*** to the computer's ***Global Assembly Cache*** (GAC).

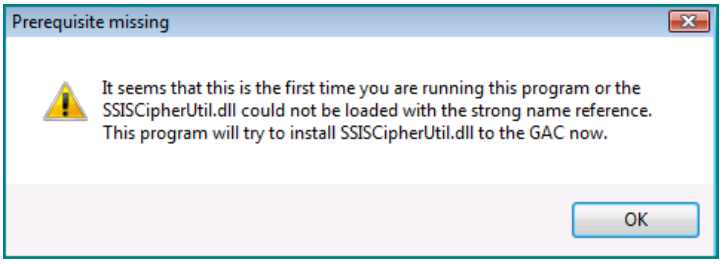

Fig 5.5

Hit, ***OK*** to that warning, and proceed with the success message. Once the program is opened, let's ***drag and drop*** the configuration file named ***AdvWrksLocalLoad2.dtsConfig***. Use the ***Browse*** button to locate a configuration file if drag and drop does not work. The configuration file will be loaded in a tree view.

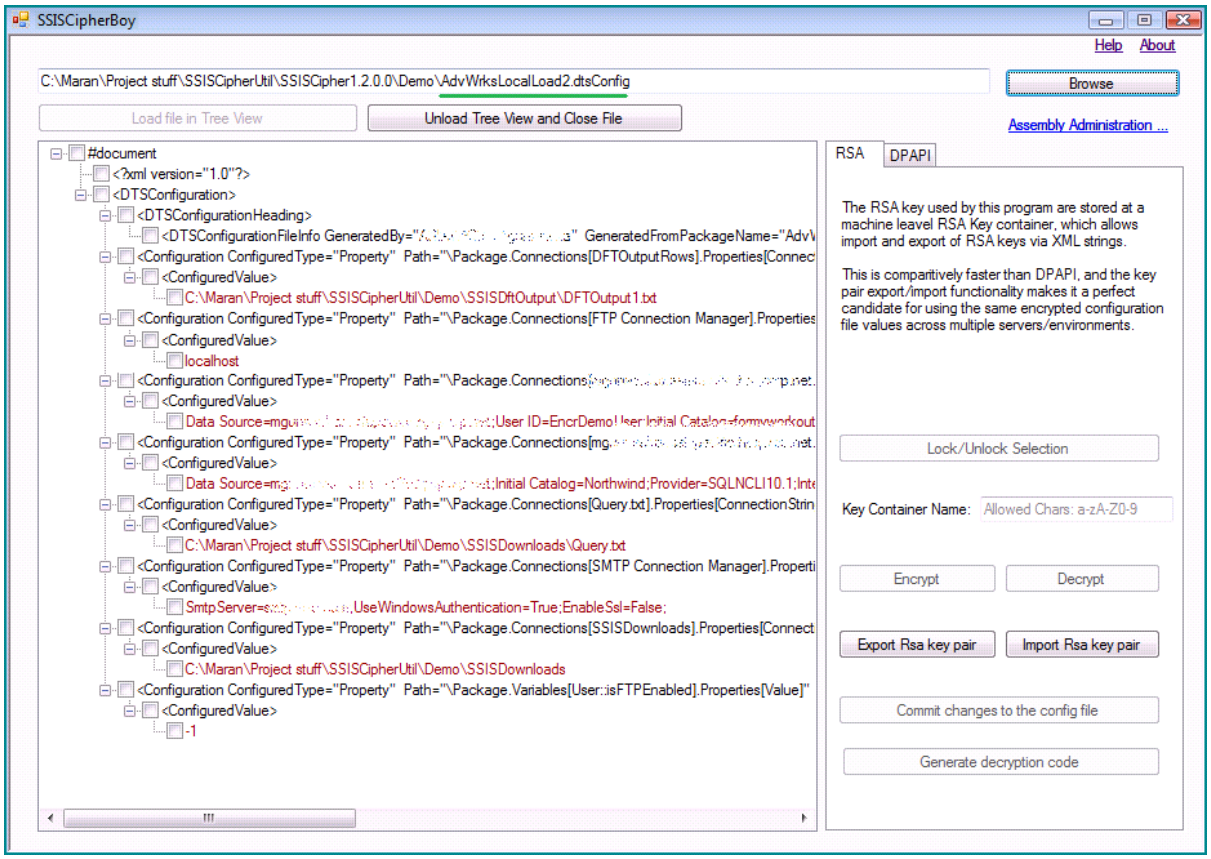

Fig. 5.6

Encrypting the values is as simple as selecting them and hitting the Encrypt button. However, if you pay attention to the third item, there is a ***User ID=EncrDemoUser***, however, there is no ***Password*** provided for it in the configuration file. When you are exporting connections strings that contain a password, then SSIS ***Package Configurations Organizer*** does not export the *Password* property in the configuration file. It knows that a *Password* is sensitive information and it does not export the P*assword* for *ConnectionString*. However, you could add it at your own risk – Just what we have been doing all along.

Before we do anything to this configuration file, lets test run it once.

```
dtexec.exe /file "C:\Maran\Project
stuff\SSISCipherUtil\SSISCipher1.2.0.0\Demo\AdvWrksLocalLoad2.dtsx" /config
"C:\Maran\Project
stuff\SSISCipherUtil\SSISCipher1.2.0.0\Demo\AdvWrksLocalLoad2.dtsConfig" /DE test
```

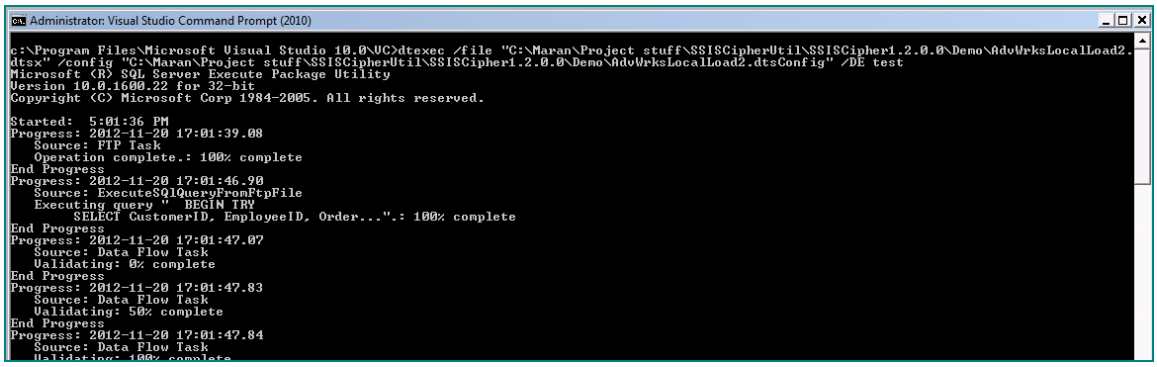

Fig 5.7

The package runs just fine. Let's get back to SSISCipherBoy.exe, ***right click*** on the item missing the *Password* property and click ***Edit Value*** to add a Password and hit the ***Save*** button.

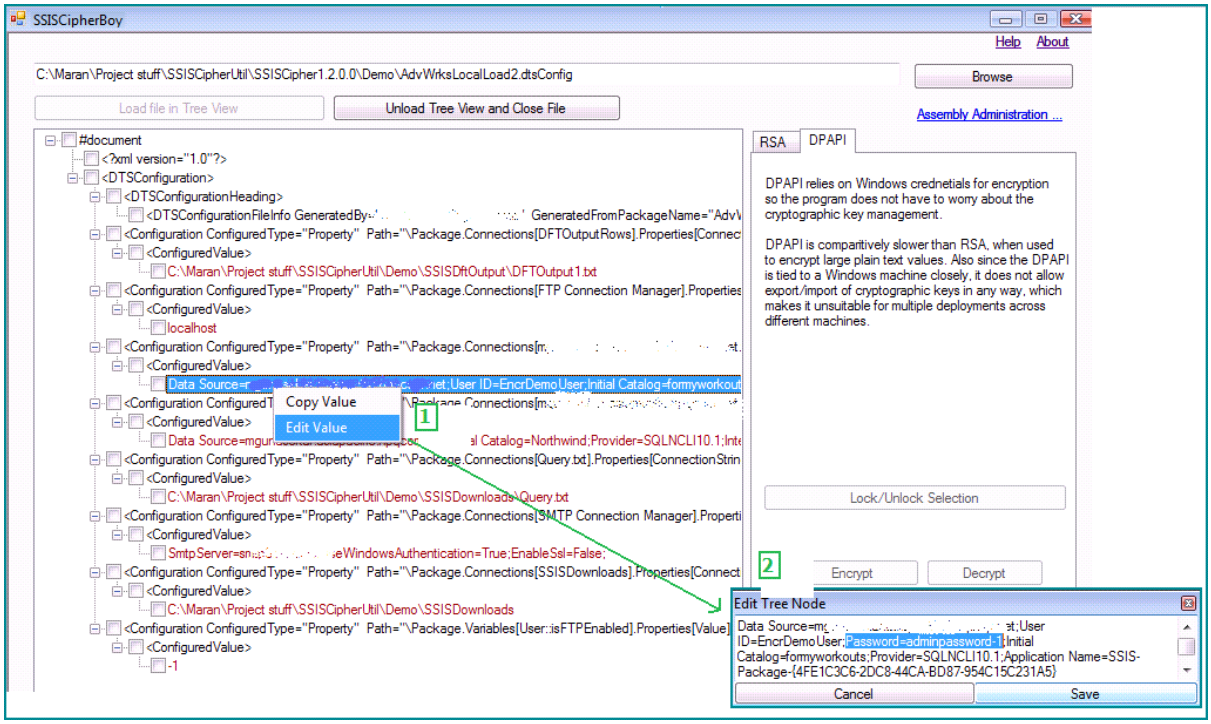

Fig. 5.8

Note: Nothing is saved to the original *.dtsConfig* file until you press the ***Commit changes to the config file*** button. Commit changes button gets enabled after the ***Encrypt***/***Decrypt*** buttons are clicked.

Once done, we have got two choices of encryption algorithm. To keep things simple: entries encrypted with DPAPI can only be decrypted on the computer on which it is encrypted on, however entries encrypted with RSA can be decrypted on any computer running windows if we export and import the key container used for encryption. *So, if we encrypt a configuration entry on our development machine, in order to use the same configuration file across other environments, you will have to export and import the RSA key pair on the target machines using this tool SSISCipherUtil.exe.* More about the algorithms are discussed in the Glossary section *X-f*.

Lets pick the entries that are sensitive, click the ***Lock/Unlock Selection*** button, provide a ***Key Container Name*** as *myrsakey1* (you are free to provide any relevant name for the key container) and hit the Encrypt button. Wait for the success message, and now all the selected entries will be encrypted for you.

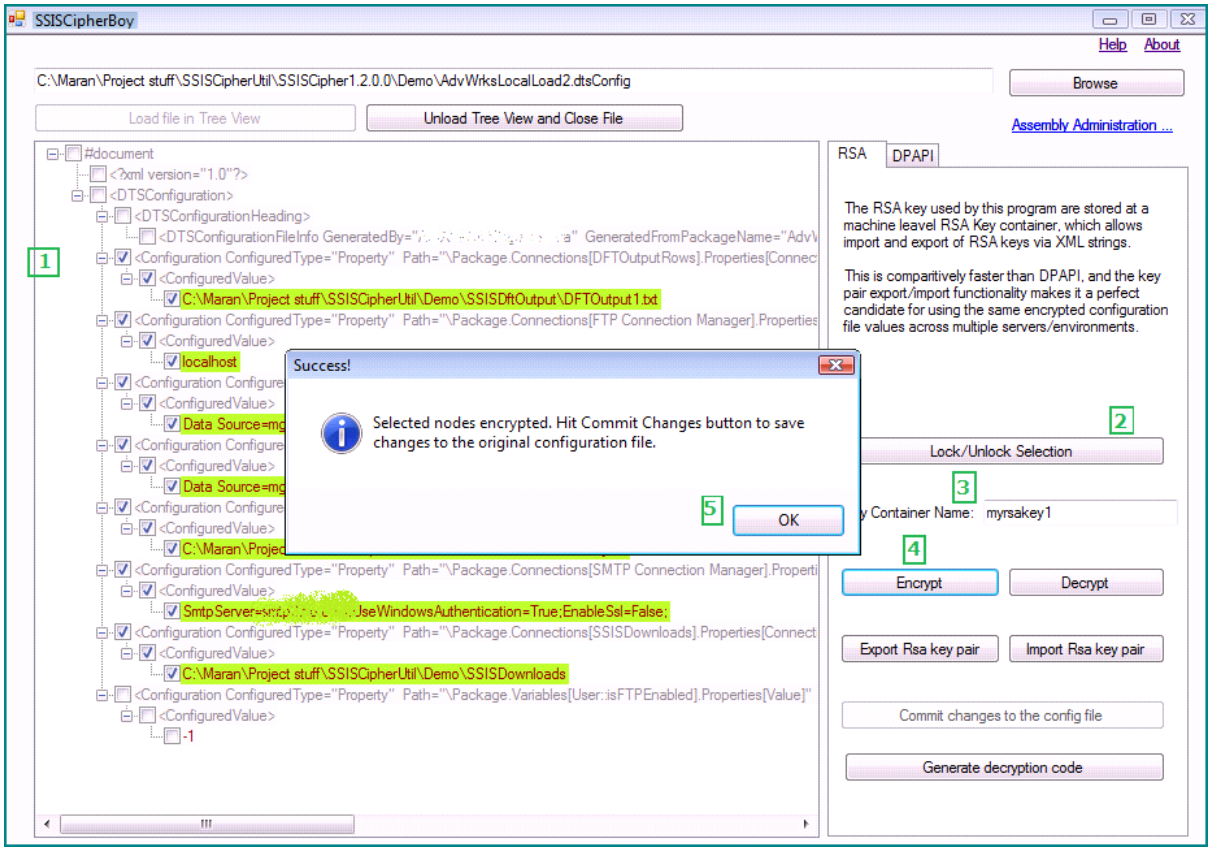

Fig. 5.9

Hit ***Commit changes to the config file*** button to save the encrypted data back to the file system.

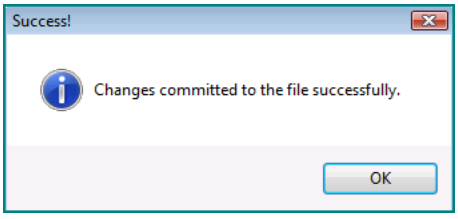

Fig. 5.10

You may, try clicking the ***Lock/Unlock Selection*** button and try ***Encrypt***/***Decrypt***/***Edit Value*** and other operations; however until the time you click ***Commit changes to the config file***, the changes that appear in the tree view will not be saved to the file system.

That's it! You sensitive values are now encrypted. Next step is to prepare your package to decrypt the package at runtime. Don't close the tool already; we will have to use it to generate decryption code.

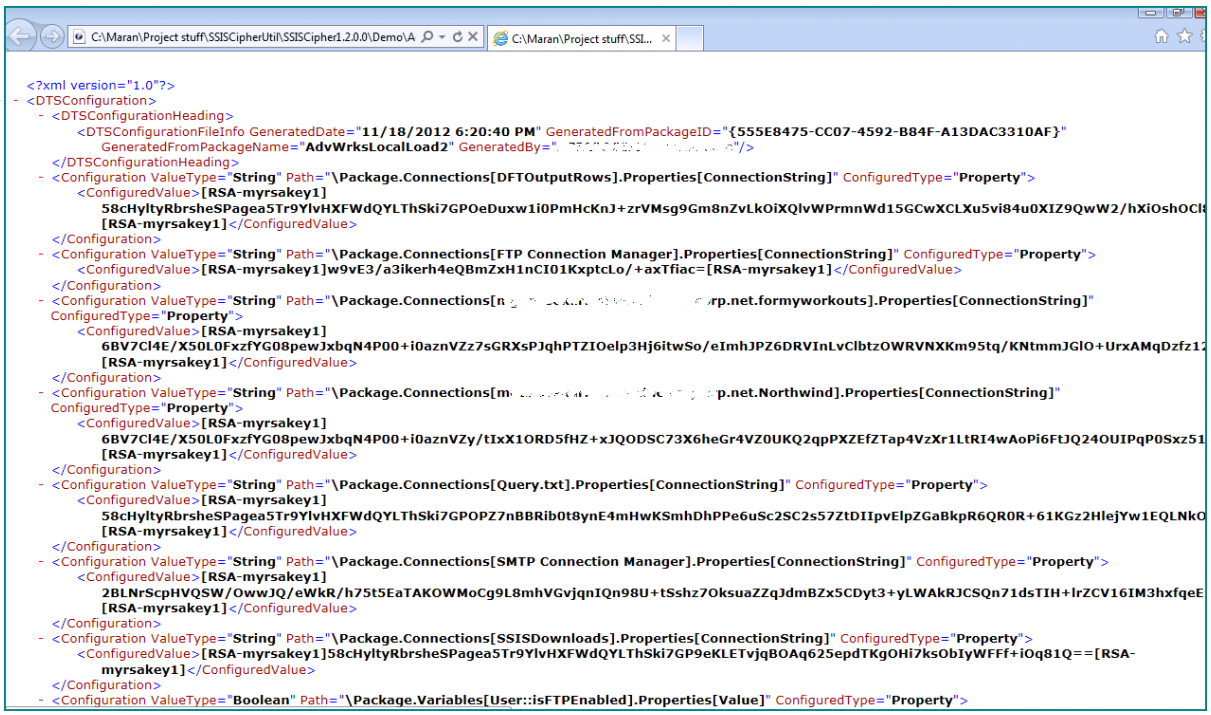

Fig. 5.11

VI. Step 2 – Using SSISCipherBoy.exe – Prepare the package to ready to decrypt information

In our example, the package ***AdvWrksLocalLoad2.dtsx*** uses the configuration file ***AdvWrksLocalLoad2.dtsConfig***. That is the configuration file that we encrypted in the preceding step. The dts runtime in no way knows that the configuration file is encrypted; neither there is a way we could tell it in a command line option. If you have *DelayValidation=false* (which is the default) in the package, then as soon as it encounters an invalid connection string (a connection string that does not have a name value pair like Initial Catalog, Server, User ID..), it throws exception saying that the validation failed and quits the execution of the job. Let's see how to make a package ready to decrypt information during runtime.

As the ***AdvWrksLocalLoad2.dtsConfig*** is open in ***SSISCipherBoy.exe***, check all the encrypted nodes, hit ***Lock/Unlock Selection*** and hit ***Generate decryption code*** button. This action displays a tip, hitting ***OK***; brings you to the ***DecryptorCode*** window.

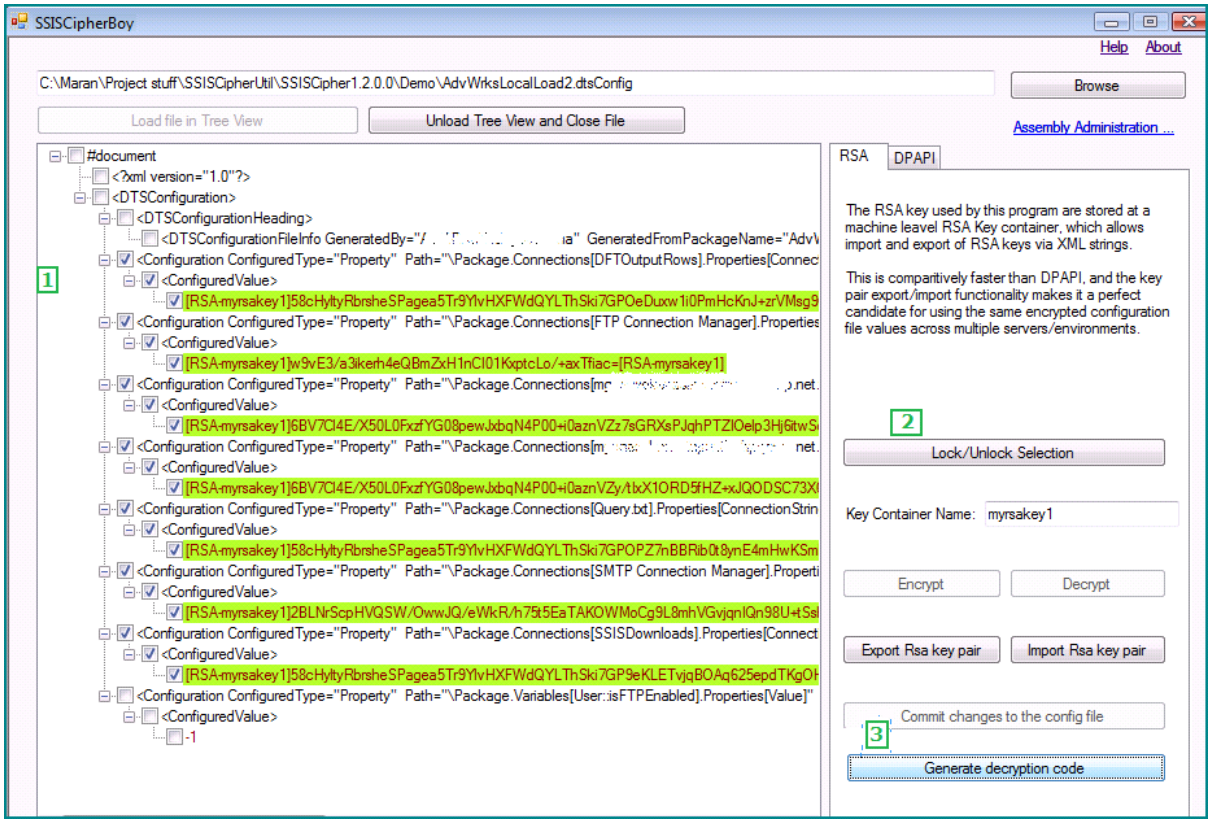

Fig. 6.1

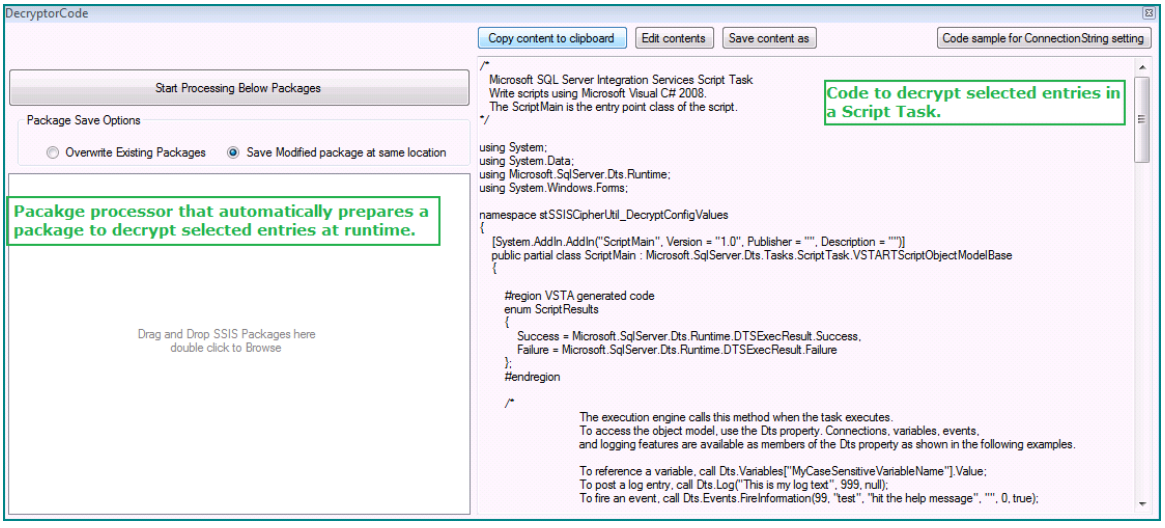

Fig. 6.2

You are left with two options here.

#1. Manually create a ScriptTask to decrypt information

You can copy the decryption code from the right pane, create a ***ScriptTask*** before all other tasks in the target package and replace the ***scriptmain.cs*** code with the decryptor code and set the ***DelayValidation=true***. If *DelayValidation=false*, then an error will be thrown because of invalid connection strings even before any of the tasks are executed. More about manually adding ScriptTask, at Glossary *X-d*.

#2. Make the package ready for decryption automatically

Drag and Drop a package in to the Package processor, click the ***Start Processing Below Packages*** button, and let the tool do the rest of the job for you. If ***Drag and Drop*** does not work ***double click*** on the package processor area to ***browse*** and pick a package. More about how it works, at Glossary *X-e.*

I am going to demonstrate option #2 here and if you are interested in option #1, refer to Glossary *X-d*.

With the ***Decryptorcode*** window open, add ***AdvWrksLocalLoad2.dtsx*** to the package processor and hit ***Start Processing Below Packages*** button. Hit ***OK*** to the big warning message. If the package is password protected, you will be prompted to enter a password. Enter the password as ***test*** and hit ***Done.***

Note: To process multiple package sharing same configuration file that was encrypted, refer to Glossary *X-a*.

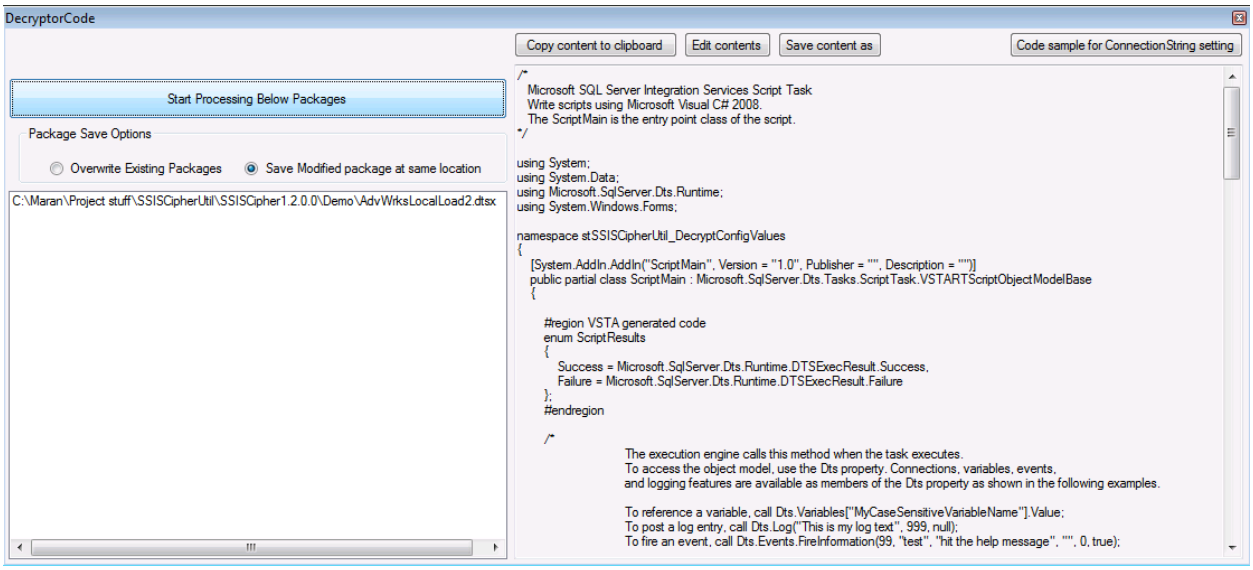

Fig. 6.3

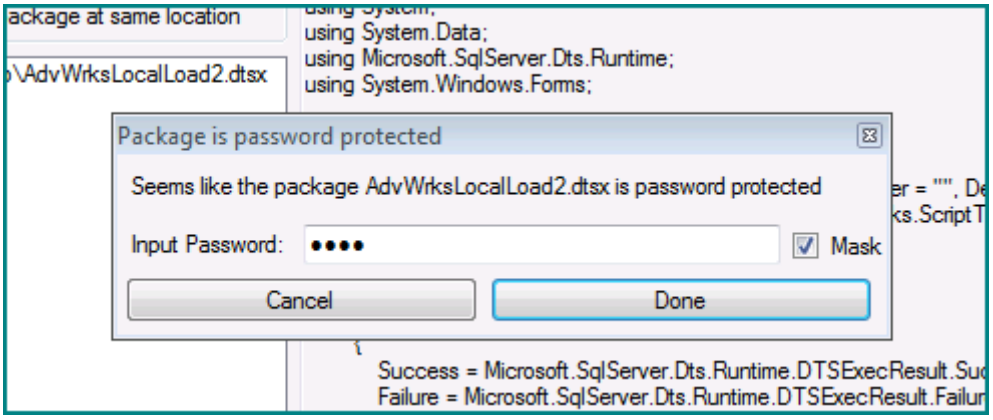

Fig. 6.4

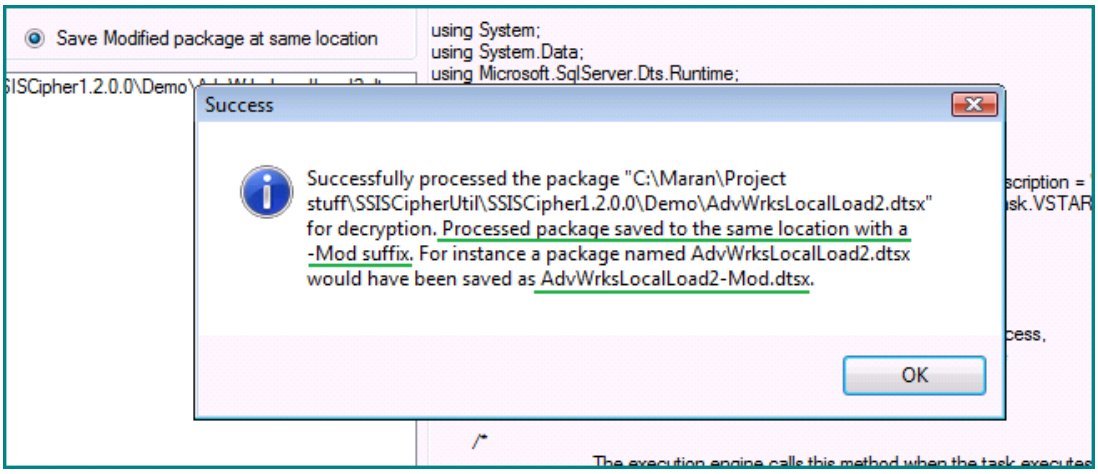

Fig. 6.5

Accept the success message and close the tool. Let's run the modified package with the encrypted configuration file. The modified package will be saved with a ***–Mod*** suffix.

```
dtexec.exe /file "C:\Maran\Project
stuff\SSISCipherUtil\SSISCipher1.2.0.0\Demo\AdvWrksLocalLoad2-Mod.dtsx" /config
"C:\Maran\Project
stuff\SSISCipherUtil\SSISCipher1.2.0.0\Demo\AdvWrksLocalLoad2.dtsConfig" /DE test
```

The package runs as expected. It prints some base64 encoded strings to the command window, which can be neglected.

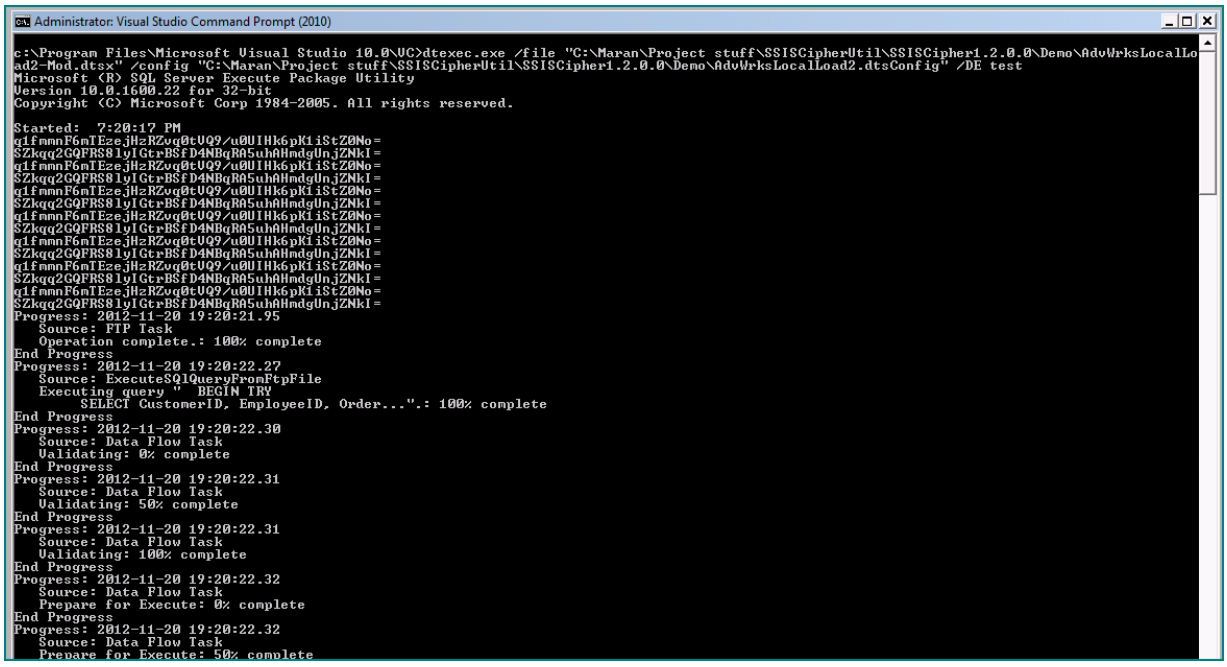

Fig. 6.6

That's about it. Now lets try deploying the modfied package ***AdvWrksLocalLoad2-Mod.dtsx*** and the config file ***AdvWrksLocalLoad2.dtsConfig*** to a Windows 2008 R2 server and test run it.

VII. Deploying package with encrypted configuration to server – Using SSISCipherBoy.exe – Export/Import RSA key pair

As mentioned before, we can use the same encrypted configuration file on any other Windows workstations or servers.

- You would need ***SSISCipherUtil.dll*** installed to the ***Global Assembly Cache*** (GAC) on the server. ***GAC*** is located at ***C:\Windows\assembly***. You can drag and drop the SSISCipherUtil.dll to that folder. Or let the tool do that for you. If you could remember, when the ***SSISCipherBoy.exe*** runs for the first time on a computer, it aumatically installs ***SSISCipherUtil.dll*** to the GAC. Easiest option, let's to that. Refer to the glossary *X-b* and *X-c* on how to install an assembly (dll) to GAC.

- Importing the *RSA key pair* (that was used to encrypt the configuration) to the the server.

Note: You can even use this tool to encryt and modify packages directly on an staging or production server, if you have done necessary testing in the development environments. The point is, if you want to use the same encrypted configuration file across all machines, then you would need to export/import the key container on all other machines.

I have logged on to a Windows server and copied the modified package - ***AdvWrksLocalLoad2-Mod.dtsx*** - and its config file - ***AdvWrksLocalLoad2.dtsConfig*** - to a location, ***SSISCipherBoy.exe*** on to my desktop. (You can place it anywhere though) (*DTPOutput1.txt* and *Query.txt* are files used by the package)

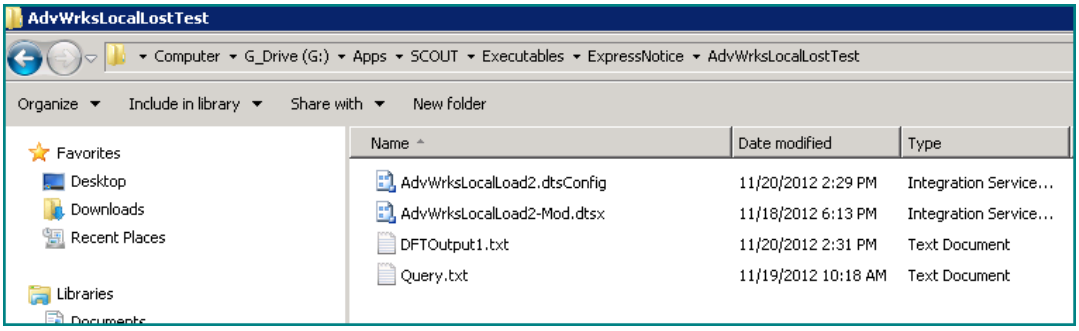

Fig. 7.1

1. Right click ***SSISCipherBoy.exe*** and select ***Run as administrator***. Accept the User Access Control prompts, if any. SSISCipherUtil.dll will be automatically installed to the GAC, and the program loads fine.

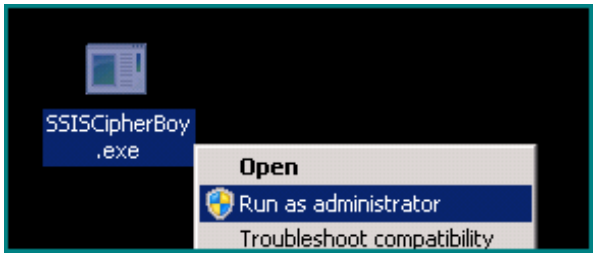

Fig. 7.2

2. From our workstation, or from the development machine where we encrypted the configuration, let's export the RSA key container named ***myrsakey1*** and import it to the server. Open ***SSISCipherBoy.exe*** on your workstation or development machine (make sure to ***Run as administrator***), provide the key container name that we used for encryption ***myrsakey1*** and hit ***Export Rsa key pair***. Save the file as ***myrsakey1.xml***. And copy it to the server.

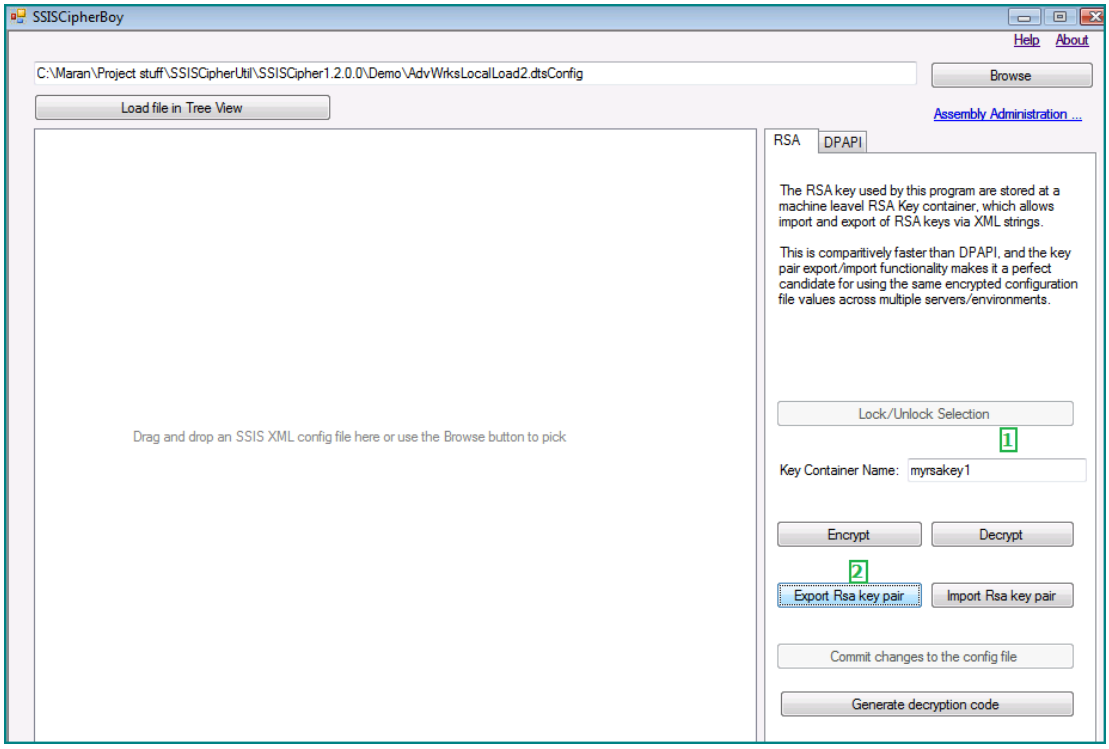

Fig. 7.3

Copy the ***myrsakey1.xml*** file to the server and use the ***Import Rsa key pair*** to import it. You should provide the same name ***myrsakey1*** in the ***Key Container Name*** text box.

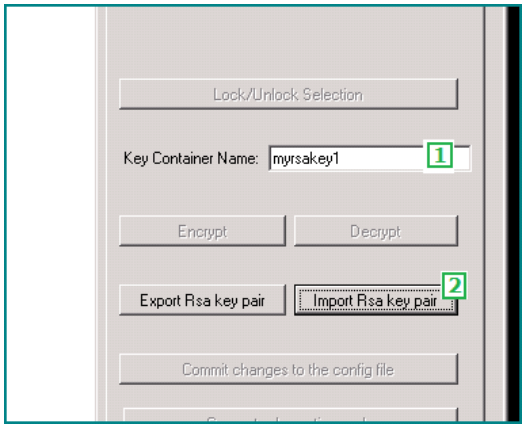

Fig. 7.4

<u>After, successful import of RSA key pair, Destroy the file myrsakey1.xml and SSISCipherUtil.exe from the server.</u>

That's it! I am going to test run the package using the below command and you will see it working like it did in the development machine.

```
dtexec.exe /file
"G:\Apps\SCOUT\Executables\ExpressNotice\AdvWrksLocalLostTest\AdvWrksLocalLoad2-
Mod.dtsx" /config
"G:\Apps\SCOUT\Executables\ExpressNotice\AdvWrksLocalLostTest\AdvWrksLocalLoad2.dt
sConfig" /DE test
```

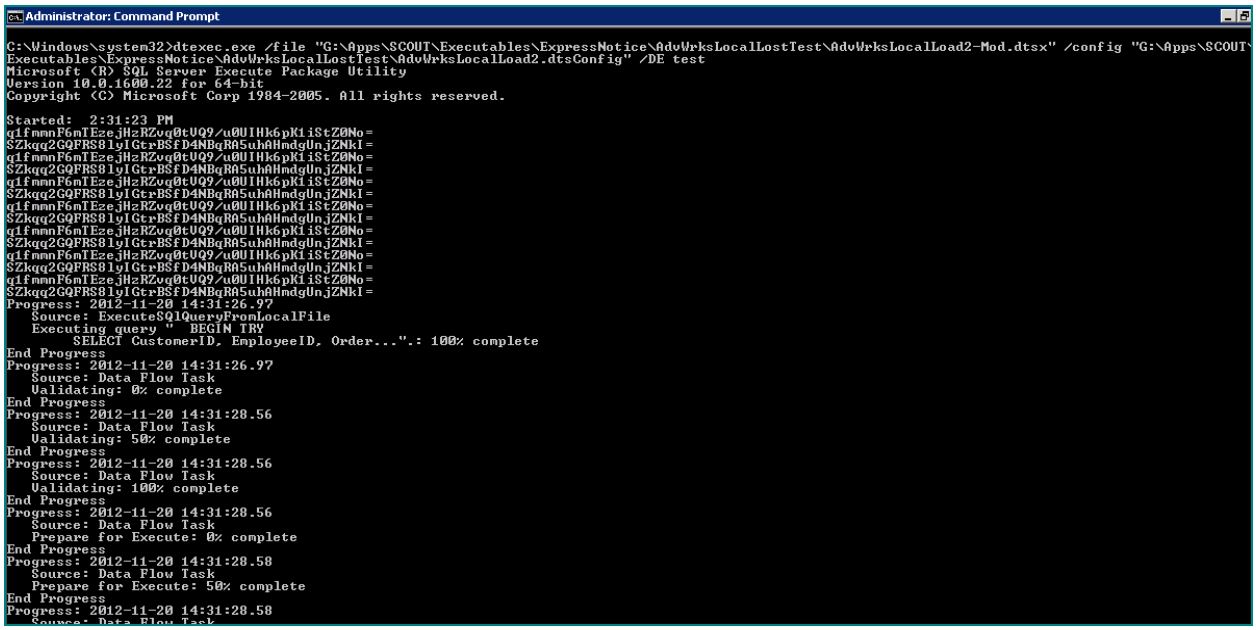

Fig. 7.5

VIII. Common errors and debugging options

- If you get any *CryptographicException* during decryption, it is possible that

  1. You are using DPAPI and trying to decrypt information on some other computer.
  2. You are using RSA but you have not exported and imported the key pair (that was used to encrypt) yet.
  3. The encrypted string is modified in some way or other and hence decryption failed.

- If you get *Connection Manager validations/runtime* errors, it is possible that

  1. Decryption failed
  2. *ConnectionString* does not contain *Password* property
  3. *DelayValidation* is set to *false* and you are getting an error because the *ConnectionString* property is encrypted. Since it is encrypted, it won't contain strings like *User ID, Password, Initial Catalog, Server etc.*, hence the validation error.
  4. *EnableConfiguration* is set to *true* (i.e., *Enable package configurations* check box is checked in the *Package Configuration Organizer* wizard. And the design time associated configuration file is either missing or not encrypted).

- To debug, the easiest way is to add a script task at the top of the package, and try accessing the Connection Manager's ***ConnectionString*** property to see if values are getting decrypted properly. For instance, try logging them, or try a ***MessageBox.Show(). AdvWrksLocalLoad2.dtsx*** has a ***ScriptTask*** in the beginning that just alerts the *ConnectionString* values for a few connections.

```
public void Main()
{
    // TODO: Add your code here
    Dts.TaskResult = (int)ScriptResults.Success;

    MessageBox.Show("Inside stReadCondition"
        + Environment.NewLine + Dts.Connections["mg[illegible].formyworkouts"].ConnectionString
        + Environment.NewLine + Dts.Connections["m[illegible]et.Northwind"].ConnectionString
        + Environment.NewLine + Dts.Connections["SMTP Connection Manager"].ConnectionString
        + Environment.NewLine + Dts.Connections["Query.txt"].ConnectionString
        + Environment.NewLine + Dts.Variables["isFTPEnabled"].Value
        + Environment.NewLine + Dts.Connections["SSISDownloads"].ConnectionString);
}
```

Fig. 8.1

Or even better, add the package to a an Integration Services Project, set a break point on the ScriptTask and look if the values are getting set correctly.

- If the *Package Processor* of ***DecryptorCode*** window throws any errors, close all Visual Studio instances, and try ***Start Processing Below Packages*** button again.

- Rule of thumb, when running ***SSISCipherBoy.exe*** always ***Run as administrator***.

IX. What now?

Hence you protect sensitive information in your SSIS Xml Config files. While the aforementioned methods describe an automated way of achieving encryption decryption, if any of the tasks fail, you could always try the manual lengthy process, or if you are interested in more information, the glossary section should suffice. Report any bugs, enhancement requests, set up assistance at http://renouncedthoughts.wordpress.com.

X. Glossary

a. Using SSISCipherBoy.exe – Processing multiple packages sharing same config

Under the heading "*Step 2 – Using SSISCipherBoy.exe – Make the package ready to decrypt information*" we saw how to process a package and make it ready for decryption. In that demonstration we added a package named ***AdvWrksLocalLoad2.dtsx*** that used a configuration named ***AdvWrksLocalLoad2.dtsConfig***, which is a very common scenario. Another, common scenario is that we have one configuration file that will be shared by many packages. To make the job easier in these situations, the ***DecryptCode*** functionality allows you to add multiple packages to the Package Processor and process them all at one shot.

Let's imagine that the configuration file named ***AdvWrksLocalLoad2.dtsConfig*** is used by multiple packages listed in the figure below.

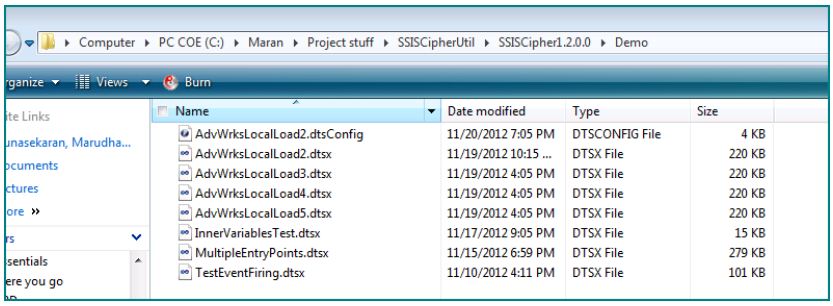

Fig. a.1

In order to process them, ***Drag and drop*** *all the packages* or ***double click*** *and select all the packages* to the Package Processor of ***DecryptCode*** window; hit the ***Start Processing Below Packages*** button. Supply password if required, hit ***OK*** on the success message that appears after processing each package. If *Drag and drop* does not work *double click* and add multiple packages.

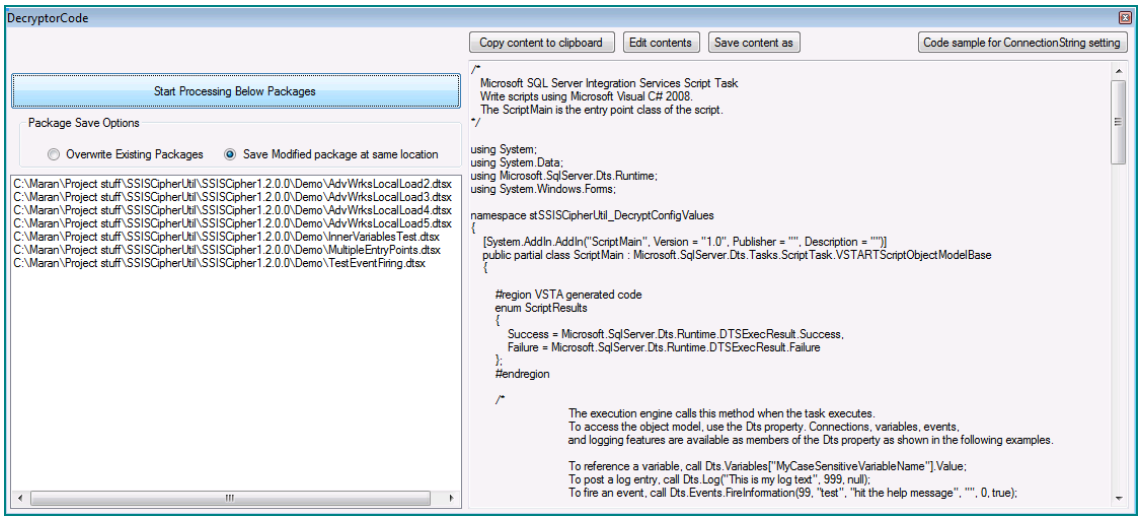

Fig. a.2

After processing, the processed packages are stored with a ***-Mod*** suffix on the same folder which looks like below.

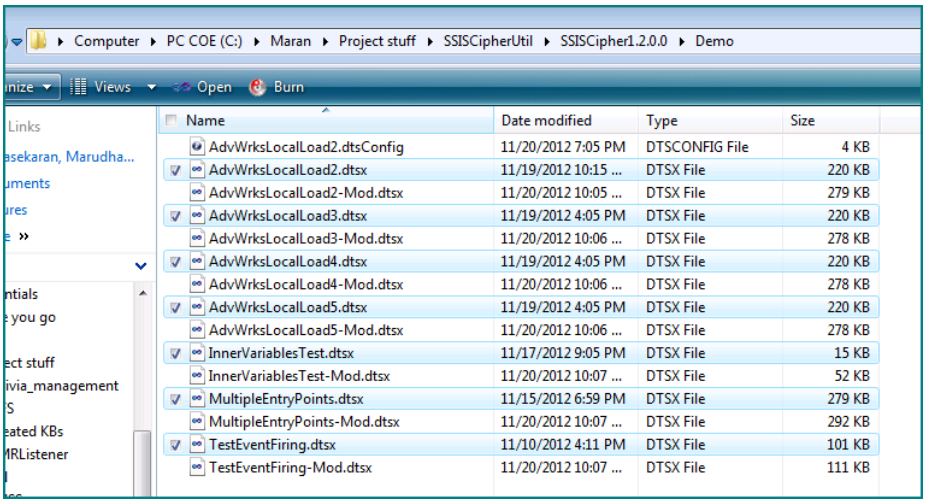

Fig. a.3

b. Using SSISCipherBoy.exe – Dump SSISCipherUtil.dll to GAC / local directory

The encryption and decryption functionality is provided by the ***SSISCipherUtil.dll***. The SSIS package in our example above that was modified to decrypt information at runtime, uses this dll. And the tool ***SSISCipherBoy.exe*** uses the same dll for all its primary operations. However, if you wanted to locate the ***SSISCipherUtil.dll*** in the file system, the only location you would find it is in the ***GAC***. In case the assembly installation fails when the SSISCipherBoy.exe loads, you would be required to manually install the SSISCipherUtil.dll to GAC. Or when the package processor fails for some reason, you would want to create a Decryptor ScriptTask yourself manually. And when you create that Decryptor ScriptTask, you would need to ***Add a reference*** to the SSISCipherUtil.dll in Visual Studio in order to decrypt values at runtime. In order to accomplish any of these aforementioned tasks, you would need SSISCipherUtil.dll on some known file system location.

Run SSISCipherBoy.exe as administrator. Click the ***Assembly Administration …*** link at the top right corner.

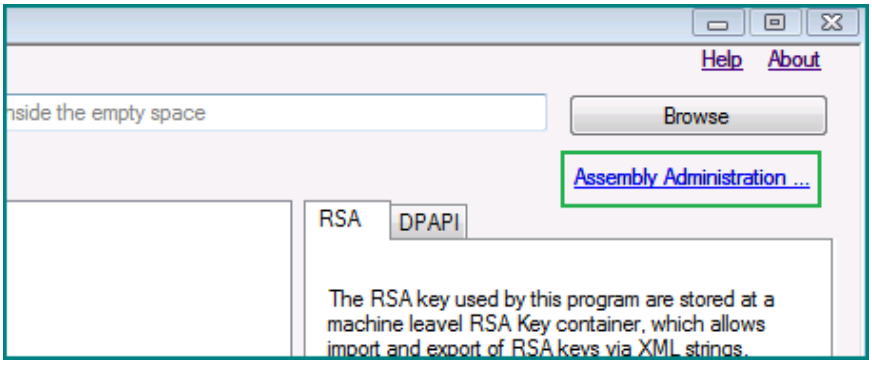

Fig. b.1

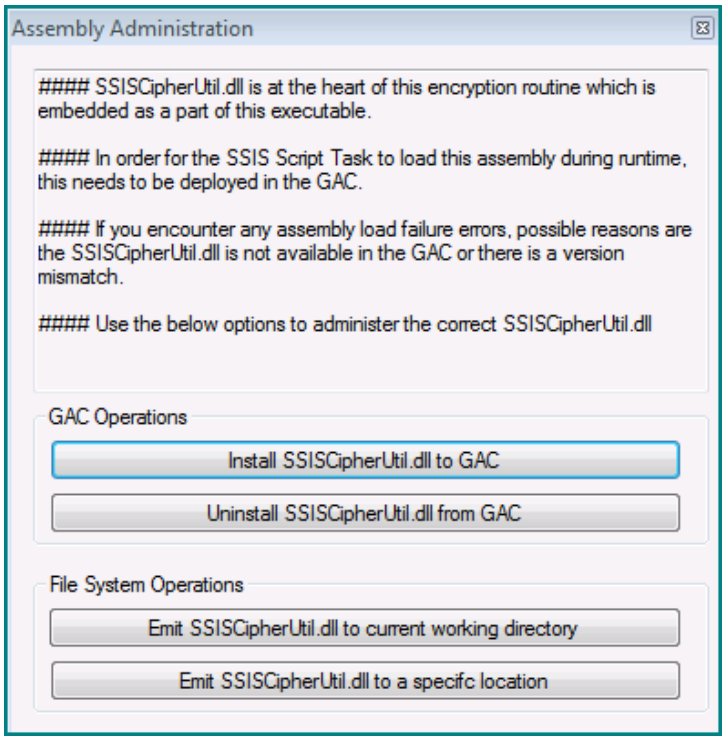

Fig b.2

Use the buttons to perform the required operations.

c. Manually installing SSISCipherUtil.dll to GAC

Many might have observed that when adding a reference to a third party dll in a *ScriptTask*, we are bound to get *AssemblyLoadException* or *FileNotFoundException* if the referenced dll is not installed to the GAC. ***SSISCipherBoy.exe*** tries to install ***SSISCipherUtil.dll*** to the ***GAC*** at startup. If that does not work you can try installing it to the GAC using the ***Assembly Administration*** window. Remember that you need Administrator privileges to perform this task. So you run SSISCipherBoy.exe with administrator privileges at all times. If none of them works, the last resort is to emit the SSIScipherUtil.dll to a specific location and then manually install it to the GAC.

Installing to the GAC is simple.

1→ Emit the SSISCipherUtil.dll to some file location (like explained in Glossary *X-b*)

2→ Navigate to ***C:\Windows\assembly*** and ***Drag and Drop*** the ***SSISCipherUtil.dll*** to that folder ***C:\Windows\assembly***. That's it. You may also try to use the *GacUtil.exe* which is a program specifically meant for this purpose.

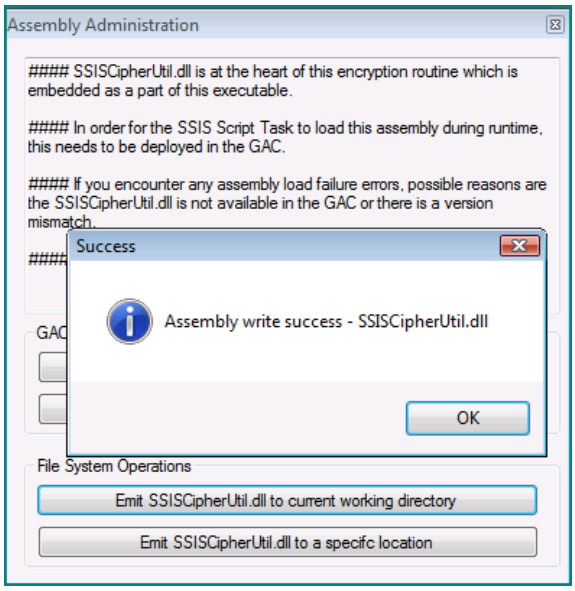

Fig. c.1

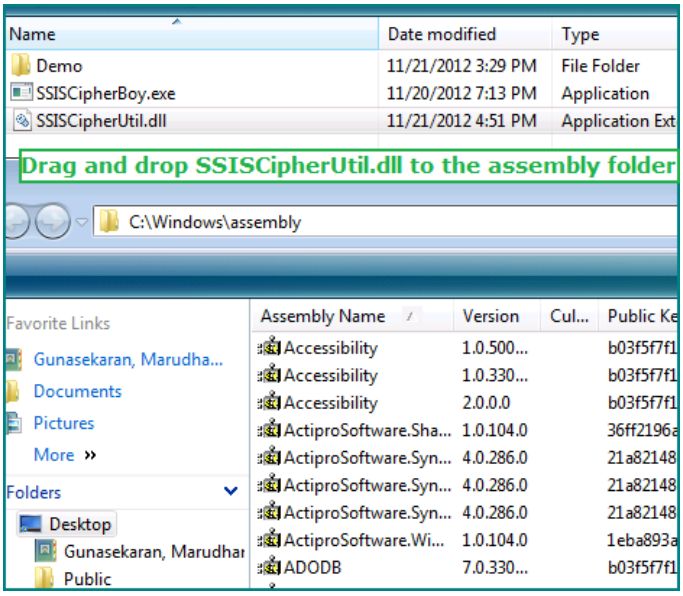

Fig. c.2

d. Manually create a ScriptTask to decrypt information

The package processor creates a ***ScriptTask*** inside a sequence container in the ***OnPreExecute EventHandler*** to decrypt information. If the Package Processor fails for some unknown reasons, you can try processing the package again or create a ***ScriptTask*** manually to decrypt information.

1→ Create a ***ScriptTask*** on top of all other tasks in the ***Control Flow*** Tab of the Package Designer. Provide a relevant name.

2→ ***Double click*** to open, hit the ***Edit Script*** button. Wait for the ***Visual Studio*** to load the project.

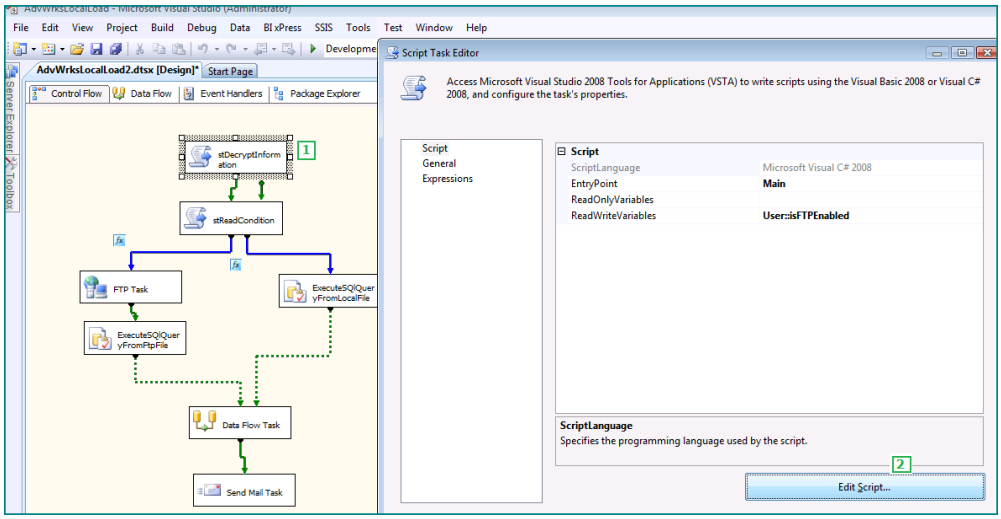

Fig. d.1

3→ ***Add reference*** to the ***SSISCipherUtil.dll***. (If you don't have ***SSISCipherUtil.dll*** at you file system yet, use the ***Assembly Administration …*** link in *SSISCipherBoy.exe* and write the dll to some comfortable location – as in Glossary *X-b*)

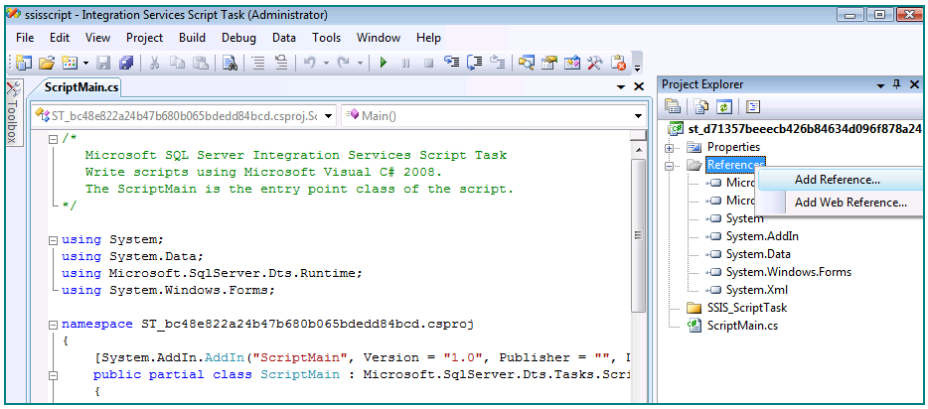

Fig. d.2

4→ Once the reference is added, replace the entire ***ScriptMain.cs*** code with the code from the ***DecryptorCode*** window of SSISCipherUtil.exe. Refer to the steps in *Step 2 – Using SSISCipherBoy.exe – Make the package ready to decrypt information* to copy the decryptor code.

5→ ***Build*** the project; close the ***ScriptTask*** after successful build. Hit ***OK*** on the ***Script Task Editor***. ***Save*** the dtsx package.

That completes everything, and now your package is modified to decrypt information at runtime. Let the ScriptTask and the SSISCipherUtil.dll do rest of the work for you.

e. What does the automatic Package Processor do?

When you add an SSIS package to the Package Processor of the DecryptorCode window, it does the following things:

1. Tries to *load* the package at the specified location.
2. If the package is protected with a password using *EncryptAllWithPassword* or *EncryptSensitiveWithPassword*, it prompts to enter the Password.
3. On successful load of the package, it checks to see if there is an *OnPreExecute EventHandler* attached to this package.
4. If an *OnPreExecute EventHandler* is attached to the package, it gets all the Tasks inside the *OnPreExecute EventHandler* and adds the *DecryptionSequence* on top of all those tasks.
5. If an OnPreExecute EventHandler is not created for the package, it simply creates an OnPreExecute EventHandler and adds the DecryptionSequence to it.
6. Inside the *DecryptionSequence*, there are two script tasks.
7. First is a *dummy ScriptTask* that leads to the second ScriptTask that does the decryption.
8. The *decryption ScriptTask* and the *dummy ScriptTask* are connected by a *PrecedenceConstraint* that allows the decryption ScriptTask to run only once during the execution of the package.

   i.e., The decryption ScriptTask runs only during the OnPreExecute event of the *package*, when other tasks in the package fires an OnPreExecute event, the decryption ScriptTask is not called. This is done with the below Expression and Constraint check

   ```
   @[System::SourceID]== @[System::PackageID]
   ```
9. Sets *DelayValidate=true* for the package.
10. Sets *EnableConfiguration=false* for the package. (the equivalent of un-checking *Enable Package Configurations* in *Configuration* wizard)

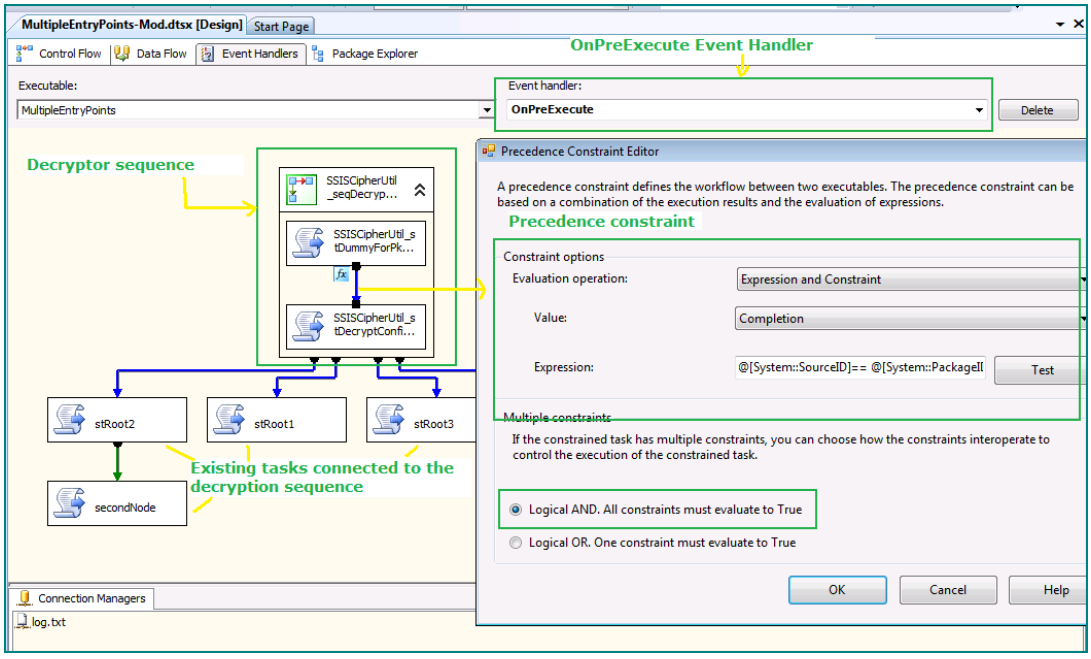

Fig. e.1

f. How does the cipher algorithm work?

SSISCipherUtil.dll supports DPAPI and RSA as we saw earlier. If you have a basic understanding of cryptography, please read forward, otherwise you may want to review the basics of Symmetric key cryptography, asymmetric key cryptography, and hashing. The following paragraphs are meant to give an overview of how DPAPI and RSA are implemented in SSISCipherUtil.dll. They might not depict the exact flow of how DPAPI and RSA work in SSISCipherUtil.dll. The source code is the only way to identify the exact implementation.

DPAPI

DPAPI – known as the Windows Data Protection API associates the cryptographic key used for encryption and decryption with the Windows User account and uses a machine wide key store. An application level entropy value is passed to the DPAPI method, so that only SSISCipherUtil.dll knows how to decrypt values that were encrypted by SSISCipherUtil.dll. The entropy value is an output of the Rfc2898DeriveBytes method that takes SHA-256 hash of some application wide constant values as input and salt.

RSA

RSA – an asymmetric cipher algorithm that in the .Net framework, not meant to encrypt inputs that are larger than the key size specified. In SSISCipherUtil.dll, the RSA algorithm is used to generate an exportable/importable public-private key pair stored at the RSA machine level key store. Later the public key components of the key pair are used to derive an entropy value, and the private key components are used to derive an input. The input and the entropy are later hashed with SHA-256 and sent to Rfc2898DeriveBytes. The output of Rfc2898DeriveBytes is used as a master key for RijndaelManaged algorithm which is a symmetric algorithm that works under the covers to encrypt and decrypt when using RSA.

C:\Users\All Users\Microsoft\Crypto\RSA\MachineKeys is the location in which the RSA key pairs are stored by *Windows*. That means, when you export/import an RSA key pair, the key pair with the specified key container name is accessed from this location.

References and Further Reading

1. SSIS: Storing Passwords - http://consultingblogs.emc.com/jamiethomson/archive/2007/04/26/SSIS_3A00_-Storing-passwords.aspx
2. SSIS: Encrypted Configurations - http://consultingblogs.emc.com/jamiethomson/archive/2007/05/04/SSIS_3A00_-Encrypted-Configurations.aspx
3. BI xPress Secure Configuration Manager - http://pragmaticworks.com/Products/BI-xPress/Features/SecureConfigurationManager.aspx
4. Understanding how SSIS configurations are applied - http://dougbert.com/blog/post/understand-how-ssis-package-configurations-are-applied.aspx

5. Integration Services Error and Message Reference - http://msdn.microsoft.com/en-us/library/ms345164.aspx

6. Dynamic Package Generation Samples - http://sqlsrvintegrationsrv.codeplex.com/releases/view/17647

7. Samples for creating SSIS packages programmatically - http://blogs.msdn.com/b/mattm/archive/2008/12/30/samples-for-creating-ssis-packages-programmatically.aspx

8. Programmatically recompiling a ScriptTask - http://social.msdn.microsoft.com/Forums/en/sqlintegrationservices/thread/27535ce7-c72c-4ea5-a27a-b34c12c73312

9. RSA class - http://msdn.microsoft.com/en-us/library/system.security.cryptography.rsa.aspx

10. Windows Data Protection - http://msdn.microsoft.com/en-us/library/ms995355.aspx

11. CspKeyContainerInfo class - http://msdn.microsoft.com/en-us/library/system.security.cryptography.cspkeycontainerinfo.aspx