How to encrypt your application data in Windows Phone

This article explains how to encrypt the data in `IsolatedStorage` so that it is much harder to access.

Introduction

IsolatedStorageExplorer Tool (ISETool.exe) can help the developers or testers to interact with the application data in `IsolatedStorage`.

For some plain and insensitive data this does not matter, however, if your application stores critical data such as passwords then it makes sense to further protect your valuable information.

This article explains how one can encrypt the data before storing to `IsolatedStorage` using the Windows Data Protection API (DPAPI) .

Utility for encrypting/decrypting strings

All essentials needed are wrapped in the WP7 Silverlight ProtectedData class in the System.Security.Cryptography namespace.

The cryptography classes use DPAPI. The encryption key for DPAPI is generated from the user's and the machine's credentials. This ensures that an encrypted string can't be decrypted on another machine or by a different user account. So even if you backup your encrypted machine state to the cloud – SkyDrive for example – it can't be decoded if the phone is lost and has to be replaced.

My code borrows the class `CryptoUtil` from this blog: Windows Phone Mango – Encrypt Data. The static functions `EncryptAndStore()` and `DecryptString()` save and load a string to/from a give file path in isolated storage.

I've added the function `GetHashCode()`, this is handy if one needs to calculate a hash code for a password. I use this function in my app for the following 3 stage process:

1. To calculate a hash value for the password, provided at the beginning by the user.
2. Store the hash value in `IsolatedStorage`.
3. Compare it to the hash value(s), generated, for the string(s) provided by a user in the subsequent authentication attempts.

Here's the code of the modified `CrytoUtil` class:

```
   public static class CryptoUtil
      {
      public static string salt = "!74jdsk0367r!ujekdijdwdkaw";
      public static string GetHashCode(string p)
         {
         var a = new SHA256Managed();
         return Convert.ToBase64String(a.ComputeHash(new
System.Text.UTF8Encoding().GetBytes(p)));
         }
      /// <summary>
      /// Encrypt a string and store it in the phone's isolated storage
      /// </summary>
      /// <param name="value"></param>
      /// <param name="path"></param>
      public static void EncryptAndStore(string value, string path)
         {
         // Convert the string to a byte[].
         byte[] PinByte = Encoding.UTF8.GetBytes(value);
         // Encrypt the string by using the Protect() method.
         byte[] ProtectedBytes = ProtectedData.Protect(PinByte, salt);
         // Store the encrypted string in isolated storage.
         CryptoUtil.WriteProtectedStringToFile(ProtectedBytes, path);
```


(C) Copyright Nokia 2014. All rights reserved.

```
        }
    /// <summary>
    /// Decrypt a string that is stored in the phone's isolated storage in the
provided path
    /// </summary>
    /// <param name="path"></param>
    /// <returns></returns>
    public static string DecryptString(string path)
        {
        using (IsolatedStorageFile file =
IsolatedStorageFile.GetUserStoreForApplication())
            {
            if (!file.FileExists(path)) return string.Empty;
            }
        // Retrieve the string from isolated storage.
        byte[] ProtectedPinByte = CryptoUtil.ReadStringFromFile(path);
        // Decrypt the string by using the Unprotect method.
        byte[] PinByte = ProtectedData.Unprotect(ProtectedPinByte, salt);
        // Convert the PIN from byte to string and display it in the text box.
        return Encoding.UTF8.GetString(PinByte, 0, PinByte.Length);
        }
    private static void WriteProtectedStringToFile(byte[] strinData, string path)
        {
        // Create a file in the application's isolated storage.
        using (IsolatedStorageFile file =
IsolatedStorageFile.GetUserStoreForApplication())
            {
            IsolatedStorageFileStream writestream = new IsolatedStorageFileStream(path,
System.IO.FileMode.Create, System.IO.FileAccess.Write, file);
            // Write stringData to the file.
            Stream writer = new StreamWriter(writestream).BaseStream;
            writer.Write(strinData, 0, strinData.Length);
            writer.Close();
            writestream.Close();
            }
        }
    private static byte[] ReadStringFromFile(string path)
        {
        // Access the file in the application's isolated storage.
        using (IsolatedStorageFile file =
IsolatedStorageFile.GetUserStoreForApplication())
            {
            IsolatedStorageFileStream readstream = new IsolatedStorageFileStream(path,
System.IO.FileMode.Open, FileAccess.Read, file);
            // Read the PIN from the file.
            Stream reader = new StreamReader(readstream).BaseStream;
            byte[] pinArray = new byte[reader.Length];
            reader.Read(pinArray, 0, pinArray.Length);
            reader.Close();
            readstream.Close();
            return pinArray;
            }
        }
    }
```


(C) Copyright Nokia 2014. All rights reserved.

Using encryption for more than a string

Sometimes you need to store more than just one string - in this case you can use the JSON serializer DataContractJsonSerializer (in System.Runtime.Serialization.Json namespace) to reduce complete objects to a string. You need to reference **System.Runtime.Serialization.dll** and **System.ServiceModel.Web.dll** for this namespace. In my case I needed to save the complete `DataModel`. I used the JSON format to store the data to a string and then encrypted the file to iso store.

Here's my code from the `Application_Closing()` handler:

```
      private void Application_Closing(object sender, ClosingEventArgs e)
      {
      var ser = new DataContractJsonSerializer(typeof(DataModel));
      MemoryStream stream1 = new MemoryStream();
      ser.WriteObject(stream1, data);
      string objects = Encoding.UTF8.GetString(stream1.GetBuffer(), 0,
 (int)stream1.Length);
      CryptoUtil.EncryptAndStore(objects, "keys");
      }
```

With the following code in `Application_Launching` you get your decrypted data back:

```
      private void Application_Launching(object sender, LaunchingEventArgs e)
      {
      reactivated = false;
      if (!IsolatedStorageFile.GetUserStoreForApplication().FileExists("keys"))
         data = new DataModel();
      string parms = CryptoUtil.DecryptString("keys");
      if (parms.Length > 0)
         using (var ms = new MemoryStream(Encoding.UTF8.GetBytes(parms)))
            {    //parse into jsonser
            var ser = new DataContractJsonSerializer(typeof(DataModel));
            data = (DataModel)ser.ReadObject(ms);
            }
      }
```

Sample Code

You can download a sample project (VS 2010) here: Media:EncryptionSample.zip (it is also available on SkyDrive here: EncryptionSample on SkyDrive).

It's called EncryptionSample and consists of two pages, **MainPage.xaml** and **DetailsPage.xaml**. **MainPage.xaml** lists sample data (created by VisualStudio's wizard) and a click on an item transfers to **DetailsPage.xaml**, where the properties of the selected item are shown.

On **DetailsPage.xaml** there is an ApplicationBar containing a Delete button. The code-behind for this button deletes the current entry from the app's data model and goes back to **MainPage.xaml**. There the list is now missing the deleted item.

Leaving the app either by the Back button or by bringing another app in front with the Windows button saves the current application state to isolated storage. The interesting code is in the App.Application_Deactivated() and App.Application_Closing() methods (-> set breakpoint here to see what happens).

Restarting the app loads the saved state again (App.Application_Launching() and App.Application_Activated()).

If you want to restore the initial set of sample data because you have deleted items you can use the **MainPage.xaml** ApplicationBar's Reset Data menu item.

This sample also contains code for the How to debug data binding problems on Windows Phone wiki article. Set a breakpoint in `DataBindingDebugConverter.Convert()` and you'll see the data items arrive in the **MainPage.xaml** listbox. On **MainPage.xmal**

(C) Copyright Nokia 2014. All rights reserved.

you'll find the necessary coding for the converter, as the article explains.

Further new code added to the sample shows a login dialog and uses the above mentioned `GetHashCode()` function to hash the password and verify it according to my article Login dialog for Windows Phone apps.

Conclusion

The given classes in make it easy to encrypt critical data on Windows Phone 7. It's very easy to use and I noticed no performance degradation.

(C) Copyright Nokia 2014. All rights reserved.