.NET - Classe de cryptage simple

Version simplifiée d'une classe de cryptage

Dans tous les projets plus ou moins sensible, il est très souvent demandé de 'crypter' les données qui transitent (par exemple, dans une URL par l'utilisation des QueryString). Voila donc un exemple totalement commenté et que j'utilise dans tous mes projets actuels et surtout un exemple d'application l'utilisant.

Classe de Cryptage Simple des Données

Introduction

Dans le développement d'applications, il arrive toujours un moment où il faut échanger des données à l'extérieur de cette application (donc pas d'utilisation de variable de session, ...), on doit donc passer en Mode POST ou GET, qui n'est pas fiable en matière de sécurité des données.

Il est donc toujours très utile de pouvoir 'crypter' un minimum ces données sortantes. Un exemple de la mise en place récente de cette classe a été l'ajout du cryptage simple dans les HttpHandler pour le téléchargement de fichier (Cf. **l'article sur les HttpHandler**). Ainsi le nom du fichier appelé dans l'URL n'était pas visible de l'utilisateur.

Nous allons donc dans cet article présenter simplement la classe (elle est entièrement commentée) et ensuite modifier le fichier que nous avions dans l'article précédent (les HttpHandler) pour ajouter le cryptage.

Le cryptage

Tout d'abord voyons ce qu'est le cryptage, il s'agit de transformer une donnée d'origine selon une méthode précise (utilisant pour cela une ou plusieurs clés) pour qu'un utilisateur n'ayant pas la clé (ou les clés) qui a (ont) servi à ce cryptage ne puisse pas retrouver la donnée d'origine.

Une présentation plus précise de ce qu'est le cryptage et son évolution jusqu'à nos jours est ici :

- **Principe du Cryptage PGP (FR)**

La méthode qui va être utilisée dans la classe présentée est celle de la Cryptographie conventionnelle, c'est à dire le cryptage de clé secrète ou de clé symétrique. Ainsi dans cet outil nous définirons une seule et unique clé qui sera utilisée par les deux partie (la partie qui codera, et celle qui décodera). Ces deux parties peuvent être dans la même application ou non (et l'intérêt est la).

Voyons maintenant plus précisément cette classe.

Classe de Cryptage Simple

Le langage qui a été utilisé pour créer cette classe est le VB.NET. Vous pouvez donc facilement l'intégrer dans la librairie de classe présentée ici même :

- **Librairie de Classe 'ASP.NET Chaînes Utilitaires' (FR)**

De même, elle est indépendante du FrameWork installé et fonctionne aussi bien avec le FrameWork 1.0 que le FrameWork 1.1.

```
'*************************************************
' $Archive: $
' $Author: $
' $Date: $ $Revision: $
'  Description : Liste des Fonctions pour le cryptage et décryptage des données
' *************************************************
Imports System
Imports System.IO
Imports System.Text
Imports System.Security
Imports System.Security.Cryptography

Public Class EncryptData
   Public Shared mbytKey(7) As Byte
   Public Shared mbytIV(7) As Byte

   ' ------------------------------------------------------
   Public Shared Function InitKey(ByVal strKey As String) As Boolean
      'Fonction de génération des clés pour les variables interne
      Try
         ' Conversion de la clé en Tableau d'Octets
         Dim bp(strKey.Length - 1) As Byte
         Dim aEnc As ASCIIEncoding = New ASCIIEncoding()
         aEnc.GetBytes(strKey, 0, strKey.Length, bp, 0)

         'Hashage de la clé en utilisant SHA1
         Dim sha As SHA1CryptoServiceProvider = New SHA1CryptoServiceProvider()
         Dim bpHash() As Byte = sha.ComputeHash(bp)

         Dim i As Integer
         ' Utilisation de la base 64-bits pour la valeur de la Clé
         For i = 0 To 7
            mbytKey(i) = bpHash(i)
         Next i

         For i = 8 To 15
            mbytIV(i - 8) = bpHash(i)
         Next
         Return True

      Catch e As Exception
         'Erreur durant les opérations
         Return False
      End Try
   End Function

   ' ------------------------------------------------------
   Public Shared Function EncryptData(ByVal strKey As String, ByVal strData As String) As String
      Dim strResult As String

      '1. La chaine de doitpas dépasser 90 Ko, Sinon le Buffer sera dépassé.
      'Voir la raison au point 3
      If strData.Length > 92160 Then
         strResult = "Erreur: Les données sont trop volumineuses. Ne pas dépasser 90Ko."
         Return strResult
      End If

      '2. Génération de la Clé
      If Not (InitKey(strKey)) Then
         strResult = "Erreur: Impossible de générer la clé de Cryptage."
         Return strResult
      End If
```

```
    '3. Préparation de la Chaine
    ' Les premiers 5 Characteres de la Chaine sont formatté pour stocker l'actuel
    ' longueur des Données. C'est une méthode simple pour conserver la taille originale
    ' des données, Sans avoir à compliquer les calculs.
    strData = String.Format("{0,5:00000}" & strData, strData.Length)


    '4. Cryptage des Données
    Dim rbData(strData.Length - 1) As Byte
    Dim aEnc As New ASCIIEncoding()
    aEnc.GetBytes(strData, 0, strData.Length, rbData, 0)

    Dim descsp As DESCryptoServiceProvider = New DESCryptoServiceProvider()

    Dim desEncrypt As ICryptoTransform = descsp.CreateEncryptor(mbytKey, mbytIV)


    '5. Préparation des streams:
    ' mOut est le Stream de Sortie.
    ' mStream est le Stream d'Entrée.
    ' cs est la Stream de Transformation.
    Dim mStream As New MemoryStream(rbData)
    Dim cs As New CryptoStream(mStream, desEncrypt, CryptoStreamMode.Read)
    Dim mOut As New MemoryStream()

    '6. Début du Cryptage
    Dim bytesRead As Integer
    Dim output(1023) As Byte
    Do
        bytesRead = cs.Read(output, 0, 1024)
        If Not (bytesRead = 0) Then
            mOut.Write(output, 0, bytesRead)
        End If
    Loop While (bytesRead > 0)

    '7. Renvoie le résultat Crypté encodé en Base 64
    ' Dans ce cas, Le résultat Actuel est Converti en Base 64 et il peut être
    ' transporté au travers du protocol HTTP sans déformation.
    If mOut.Length = 0 Then
        strResult = ""
    Else
        strResult = Convert.ToBase64String(mOut.GetBuffer(), 0, CInt(mOut.Length))
        '8. Modification de la chaine d'entrée pour récupérer les +
        ' qui ne sont pas passable via Querystring
        strResult = strResult.Replace("+", "-||-||-")
    End If
    Return strResult

End Function

' ----------------------------------------------------
Public Shared Function DecryptData(ByVal strKey As String, ByVal strData As String) As String

    Dim strResult As String

    '1. Modification de la chaine d'entrée pour récupérer les +
    ' qui ne sont pas passable via Querystring
    strData = strData.Replace("-||-||-", "+")

    '2. Génération de la Clé
    If Not (InitKey(strKey)) Then
        strResult = "Erreur: Impossible de générer la clé de Cryptage."
        Return strResult
    End If

    '2. Initialisation du Provider
    Dim nReturn As Integer = 0
    Dim descsp As New DESCryptoServiceProvider()
    Dim desDecrypt As ICryptoTransform = descsp.CreateDecryptor(mbytKey, mbytIV)

    '3. Préparation des streams:
    ' mOut est le Stream de Sortie.
    ' cs est la Stream de Transformation.
    Dim mOut As New MemoryStream()
    Dim cs As New CryptoStream(mOut, desDecrypt, CryptoStreamMode.Write)
```

```
        '4. Mémorisation pour revenir de la Base 64 vers un tableau d'Octets pour récupérer le
        'le Stream de base Crypté
        Dim bPlain(strData.Length - 1) As Byte
        Try
            bPlain = Convert.FromBase64CharArray(strData.ToCharArray(), 0, strData.Length)
        Catch e As Exception
            strResult = "Erreur: Les données en Entrée ne sont pas en base 64."
            Return strResult
        End Try

        Dim lRead As Long = 0
        Dim lReadNow As Long = 0
        Dim lTotal As Long = strData.Length

        Try
            '5. Réalise le Décryptage
            Do While (lTotal >= lRead)
                cs.Write(bPlain, 0, bPlain.Length)
                lReadNow = CLng(((bPlain.Length / descsp.BlockSize) * descsp.BlockSize))
                lRead = lReadNow + lRead
            Loop

            Dim aEnc As New ASCIIEncoding()
            strResult = aEnc.GetString(mOut.GetBuffer(), 0, CInt(mOut.Length))

            '6. Nettoie la chaine pour ne renvoyer que les données signicatives
            ' Rappelez-vous cela dans la fonction de chiffrage, les 5 premiers caractère
            ' permettent de conserver la taille originale de la chaine de caracteres
            ' C'est la méthode la plus simple de mémoriser la taille des données d'origine,
            ' sans passer par des calculs complèxes.

            Dim strLen As String = strResult.Substring(0, 5)
            Dim nLen As Integer = CInt(strLen)
            strResult = strResult.Substring(5, nLen)
            nReturn = CInt(mOut.Length)

            Return strResult
        Catch e As Exception
            strResult = "Erreur: Impossible de décrypter. Eventuellement "
            strResult &= "à cause d'une clé non conforme ou de données corrompue."
        End Try
        Return strResult
    End Function

    ' --------------------------------------------------
End Class
```

On a donc 3 fonctions dans cette classe :

- **InitKey** : Cette fonction permet d'initialiser la clé à partir d'une chaîne de caractères qu'on a choisi d'utiliser comme clé de cryptage. C'est cette chaîne qu'il faudra que les deux parties connaissent pour échanger les données.
- **EncryptData** : Permet de crypter la donnée voulue avec la clé transmise.
- **DecryptData** : Permet de décrypter la donnée voulue avec la clé transmise.

Maintenant que l'on a la classe et les fonctions qui y sont, voyons comment on peut utiliser celle-ci et plus particulièrement dans notre exemple de téléchargement de fichier PDF.

Utilisation de la classe de cryptage

Je vous rappelle le cadre de notre exemple qui est ici :

- **Exemple de Téléchargement de Fichier PDF Via HttpHandlers (FR)**

Rappel :

Ainsi dans cet exemple, nous avons pour le moment un outil qui permet de simuler un appel d'un fichier PDF. C'est-à-dire qu'on appelle une page (ASHX), en lui passant en paramètre le nom du fichier (via QueryString), qui va se charger de vérifier que le fichier voulu existe et dans le cas l'envoyer en flux HTTP au client en donnant les entêtes HTTP pour Acrobat Reader.

Objectif :

On voit très bien que si on ne veut pas que l'utilisateur de cette page connaisse le nom du fichier d'origine (cas, par exemple, où plusieurs utilisateurs différents peuvent appeler un fichier qui leur est propre mais ne doivent pas pouvoir appeler celui d'un autre utilisateur et donc ne doivent pas connaître la méthode de définition des noms de fichiers), il faut absolument que ce nom transmis via la QueryString soit masqué et donc de préférence crypté.

Méthode :

On va donc modifier la classe du Gestionnaire HTTP pour ajouter le module de décryptage, il ne restera plus qu'à utiliser à l'inverse le cryptage dans la page où on appellera ce gestionnaire.

On va donc déjà ajouter deux clés dans le fichier Web.Config, afin de pouvoir correctement utiliser le HttpHandler :

```
<configuration>
  <appSettings>
...
    <!-- Fourni la clé qui va être utilisée lors de l'appel
         de la fonction de cryptage et Décryptage -->
    <add key="CleCryptageFichier" value="DotNetQueDuBonheur"/>

    <!-- Fourni Le chemin permettant d'atteindre le répertoire des PDF -->
    <add key="RepertoireRacinePDF" value="C:\MonRepertoirePDF\"/>
...
  </appSettings>
...
 </system.web>
...
  <httpHandlers>
   <add verb="*"
     path="DL_PDF.ashx"
     type="DotNetQueDuBonheur.Handlers.PDFHandler, DotNetQueDuBonheur"/>
  </httpHandlers>
...
 </system.web>
</configuration>
```

Maintenant voyons la classe du Gestionnaire HTTP :

```
'****************************************
' $Archive: /DotNetQueDuBonheur/Classes/PdfHandler.vb $
' $Author: Fabrice69 $
' $Date: 18/03/04 01:25 $  $Revision: 3 $
'  Description : Gestion de la sortie des fichier PDF sans donner l'URL réelle d'accès a ces fichier
'     Ajout d'une couche de cryptage des noms
' ****************************************
Imports System.IO

Namespace Handlers
' ----------------------------------------------------
Public Class PDFHandler
 Implements IHttpHandler
 ' ----------------------------------------------------
 Public Sub ProcessRequest(ByVal context As HttpContext) Implements IHttpHandler.ProcessRequest
  ' A partir du nom du fichier PDF crée le Flux Binaire
  ' afin de le fournir à la page

  Dim nomPDF As String = Trim(context.Request("nom"))
  Dim nomFichier As String
  Dim Cle As String
  Cle = System.Configuration.ConfigurationSettings.AppSettings("CleCryptageFichier")
  nomFichier = System.Configuration.ConfigurationSettings.AppSettings("RepertoireRacinePDF")
  nomFichier & = EncryptData. DecryptData(Cle, (Trim(nomPDF))

  If nomPDF = "" OrElse Not File.Exists(nomFichier) Then
   context.Response.Write("Fichier Inexistant")
  Else
   Dim f As New FileStream(nomFichier, FileMode.Open)
   context.Response.AddHeader("content-type", "application/pdf")
   Dim buffer(f.Length) As Byte
   f.Read(buffer, 0, f.Length)
   f.Close()
   context.Response.BinaryWrite(buffer)
  End If
 End Sub
 ' ----------------------------------------------------
 Public ReadOnly Property IsReusable() As Boolean Implements IHttpHandler.IsReusable
  Get
   Return True
  End Get
 End Property
 ' ----------------------------------------------------
 End Class
End Namespace
```

Maintenant dans la page qui va être affichée à l'utilisateur (qui ne doit donc pas connaître le nom du fichier d'origine), on aura plus qu'à placer un HyperLink en lui fournissant comme URL :

```
Cle  = System.Configuration.ConfigurationSettings.AppSettings("CleCryptageFichier")
nomFichier  = EncryptData. EncryptData(Cle, (NomReelFichierPDF)
MonURL  = "/DotNetQueDuBonheur/DL_PDF.ashx?nom=" & nomFichier
```

Conclusion

Cette classe n'est qu'un exemple rapide de ce qui peut être fait en matière de cryptage de données, de plus la clé utilisée est symétrique dans cet exemple et chaque partie devra posséder la même. Si vous souhaiter développer un réel outil de cryptage, il faudra certainement choisir l'utilisation de la clé privée et clé publique.

Je vous laisse découvrir dans l'article cité au départ quelle sont les spécificités de ce choix technique :

- **Principe du Cryptage PGP (FR)**

Quelques articles ou exemples sur ce sujet :

- **Create a Simple, Reusable Infrastructure for Public Key Encryption Using VB.NET (US)**
- **Exemple d'implémentation de cryptage dans un WebService (FR)**
- **Example of Public/Private Key (US)**
- **Exemple avec une Clé Symétrique en C# (US)**
- **Exemple de Clé Publique/Privée en PERL (US)**

En vous souhaitant de bons projets de développement.

Romelard Fabrice (alias F___)