Erstellen sicherer ASP.NET-Anwendungen

Authentifizierung, Autorisierung und sichere Kommunikation

Auf der Orientierungsseite finden Sie einen Ausgangspunkt und eine vollständige Übersicht zum *Erstellen sicherer ASP.NET-Anwendungen*.

Zusammenfassung

Nachstehend wird erläutert, wie eine verwaltete Klassenbibliothek für die Bereitstellung von Verschlüsselungsfunktionen für Anwendungen erstellt wird. Auf der Basis dieser Klassenbibliothek sind Anwendungen in der Lage, den jeweils geeigneten Verschlüsselungsalgorithmus auszuwählen. Zu den unterstützten Algorithmen gehören DES, Dreifach-DES (3DES, Triple DES), RC2 und Rijndael.

Vorgehensweise: Erstellen einer Verschlüsselungs-bibliothek

In dieser Vorgehensweise wird erläutert, wie eine generische Verschlüsselungsbibliothek erstellt wird, die zum Ver- und Entschlüsseln von Daten mithilfe der folgenden Algorithmen herangezogen werden kann:

- DES (Digital Encryption Standard)
- Dreifach-DES (3DES)
- Rijndael
- RC2

Eine Beispielanwendung, die auf die so erstellte Klassenbibliothek zugreift, finden Sie unter "Vorgehensweise: Speichern verschlüsselter Verbindungszeichenfolgen in der Registrierung" im Abschnitt "Referenz" dieses Handbuchs.

Anforderungen

Im Folgenden finden Sie eine Liste der empfohlenen Hardware und Software und eine Beschreibung der Netzwerkinfrastruktur, der Fähigkeiten und Kenntnisse sowie der Service Packs, die Sie benötigen.

- Microsoft® Windows® 2000 als Betriebssystem
- Microsoft Visual Studio® .NET als Entwicklungssystem

Die hierin erläuterten Verfahren setzen zudem Kenntnisse des Entwicklungstools Microsoft Visual C#™ voraus.

Zusammenfassung

Diese Vorgehensweise enthält folgende Verfahren:

1. Erstellen einer C#-Klassenbibliothek
2. Erstellen einer Konsolentestanwendung

1. Erstellen einer C#-Klassenbibliothek

Mit dem nachstehenden Verfahren wird eine C#-Klassenbibliothek erstellt, die Ver- und Entschlüsselungsfunktionen bereitstellt.

▶ **So erstellen Sie eine C#-Klassenbibliothek**

1. Starten Sie Visual Studio .NET, und erstellen Sie ein neues C#-Klassenbibliotheksprojekt mit Namen **Encryption**.
2. Benennen Sie **Class1.cs** im Projektmappen-Explorer in **EncryptTransformer.cs** um.
3. Benennen Sie in **EncryptTransformer.cs** die Klasse **Class1** in **EncryptTransformer** um.
4. Ändern Sie den Geltungsbereich der Klasse von **public** in **internal**.

```
internal class EncryptTransformer
```

5. Fügen Sie am Anfang der Datei die folgende **using**-Anweisung hinzu.

```
using System.Security.Cryptography;
```

6. Fügen Sie innerhalb des **Encryption**-Namespaces den folgenden Auflistungstyp hinzu.

```
public enum EncryptionAlgorithm {Des = 1, Rc2, Rijndael, TripleDes};
```

7. Fügen Sie der **EncryptTransformer**-Klasse die folgenden Membervariablen vom Typ **private** hinzu.

```
private EncryptionAlgorithm algorithmID;
private byte[] initVec;
private byte[] encKey;
```

8. Ersetzen Sie den Standardkonstruktor durch den folgenden Konstruktor.

```
internal EncryptTransformer(EncryptionAlgorithm algId)
{
  //Save the algorithm being used.
  algorithmID = algId;
}
```

9. Fügen Sie der Klasse die folgende Methode hinzu.

```
internal ICryptoTransform GetCryptoServiceProvider(byte[] bytesKey)
{
  // Pick the provider.
  switch (algorithmID)
  {
    case EncryptionAlgorithm.Des:
    {
```

```
  DES des = new DESCryptoServiceProvider();
  des.Mode = CipherMode.CBC;

  // See if a key was provided
  if (null == bytesKey)
  {
    encKey = des.Key;
  }
  else
  {
    des.Key = bytesKey;
    encKey = des.Key;
  }
  // See if the client provided an initialization vector
  if (null == initVec)
  { // Have the algorithm create one
    initVec = des.IV;
  }
  else
  { //No, give it to the algorithm
    des.IV = initVec;
  }
  return des.CreateEncryptor();
}
case EncryptionAlgorithm.TripleDes:
{
  TripleDES des3 = new TripleDESCryptoServiceProvider();
  des3.Mode = CipherMode.CBC;
  // See if a key was provided
  if (null == bytesKey)
  {
    encKey = des3.Key;
  }
  else
  {
    des3.Key = bytesKey;
    encKey = des3.Key;
  }
  // See if the client provided an IV
  if (null == initVec)
  { //Yes, have the alg create one
    initVec = des3.IV;
  }
  else
  { //No, give it to the alg.
    des3.IV = initVec;
  }
  return des3.CreateEncryptor();
}
case EncryptionAlgorithm.Rc2:
{
  RC2 rc2 = new RC2CryptoServiceProvider();
  rc2.Mode = CipherMode.CBC;
  // Test to see if a key was provided
```

```
      if (null == bytesKey)
      {
        encKey = rc2.Key;
      }
      else
      {
        rc2.Key = bytesKey;
        encKey = rc2.Key;
      }
      // See if the client provided an IV
      if (null == initVec)
      { //Yes, have the alg create one
        initVec = rc2.IV;
      }
      else
      { //No, give it to the alg.
        rc2.IV = initVec;
      }
      return rc2.CreateEncryptor();
    }
    case EncryptionAlgorithm.Rijndael:
    {
      Rijndael rijndael = new RijndaelManaged();
      rijndael.Mode = CipherMode.CBC;
      // Test to see if a key was provided
      if(null == bytesKey)
      {
        encKey = rijndael.Key;
      }
      else
      {
        rijndael.Key = bytesKey;
        encKey = rijndael.Key;
      }
      // See if the client provided an IV
      if(null == initVec)
      { //Yes, have the alg create one
        initVec = rijndael.IV;
      }
      else
      { //No, give it to the alg.
        rijndael.IV = initVec;
      }
      return rijndael.CreateEncryptor();
    }
    default:
    {
      throw new CryptographicException("Algorithm ID '" + algorithmID +
                                       "' not supported.");
    }
  }
}
```

10. Fügen Sie der Klasse die folgenden Eigenschaften hinzu.

```
internal byte[] IV
{
  get{return initVec;}
  set{initVec = value;}
}
internal byte[] Key
{
  get{return encKey;}
}
```

11. Fügen Sie dem Projekt eine neue Klasse mit Namen **DecryptTransformer** hinzu.
12. Fügen Sie am Anfang der Datei **DecryptTransformer.cs** die folgende **using**-Anweisung hinzu.

```
using System.Security.Cryptography;
```

13. Ändern Sie den Geltungsbereich der Klasse von **public** in **internal**.
14. Ersetzen Sie den Standardkonstruktor durch den folgenden Konstruktor.

```
internal DecryptTransformer(EncryptionAlgorithm deCryptId)
{
  algorithmID = deCryptId;
}
```

15. Fügen Sie der Klasse die folgende Variablen vom Typ **private** hinzu.

```
private EncryptionAlgorithm algorithmID;
private byte[] initVec;
```

16. Fügen Sie der Klasse die folgende Methode hinzu.

```
internal ICryptoTransform GetCryptoServiceProvider(byte[] bytesKey)
{
  // Pick the provider.
  switch (algorithmID)
  {
    case EncryptionAlgorithm.Des:
    {
      DES des = new DESCryptoServiceProvider();
      des.Mode = CipherMode.CBC;
      des.Key = bytesKey;
      des.IV = initVec;
      return des.CreateDecryptor();
    }
    case EncryptionAlgorithm.TripleDes:
    {
      TripleDES des3 = new TripleDESCryptoServiceProvider();
      des3.Mode = CipherMode.CBC;
      return des3.CreateDecryptor(bytesKey, initVec);
    }
```

```
      case EncryptionAlgorithm.Rc2:
      {
        RC2 rc2 = new RC2CryptoServiceProvider();
        rc2.Mode = CipherMode.CBC;
        return rc2.CreateDecryptor(bytesKey, initVec);
      }
      case EncryptionAlgorithm.Rijndael:
      {
        Rijndael rijndael = new RijndaelManaged();
        rijndael.Mode = CipherMode.CBC;
        return rijndael.CreateDecryptor(bytesKey, initVec);
      }
      default:
      {
        throw new CryptographicException("Algorithm ID '" + algorithmID +
                                         "' not supported.");
      }
    }
  } //end GetCryptoServiceProvider
```

17. Fügen Sie der Klasse die folgende Eigenschaft hinzu.

```
internal byte[] IV
{
  set{initVec = value;}
}
```

18. Fügen Sie dem Projekt eine neue Klasse mit Namen **Encryptor** hinzu.
19. Fügen Sie am Anfang von **Encryptor.cs** die folgenden **using**-Anweisungen hinzu.

```
using System.Security.Cryptography;
using System.IO;
```

20. Ersetzen Sie den Standardkonstruktor durch den folgenden Konstruktor.

```
public Encryptor(EncryptionAlgorithm algId)
{
  transformer = new EncryptTransformer(algId);
}
```

21. Fügen Sie der Klasse die folgenden Membervariablen vom Typ **private** hinzu.

```
private EncryptTransformer transformer;
private byte[] initVec;
private byte[] encKey;
```

22. Fügen Sie der Klasse die folgende **Encrypt**-Methode hinzu.

```
public byte[] Encrypt(byte[] bytesData, byte[] bytesKey)
{
  //Set up the stream that will hold the encrypted data.
  MemoryStream memStreamEncryptedData = new MemoryStream();
```

```
  transformer.IV = initVec;
  ICryptoTransform transform = transformer.GetCryptoServiceProvider(bytesKey);
  CryptoStream encStream = new CryptoStream(memStreamEncryptedData,
                                            transform,
                                            CryptoStreamMode.Write);
  try
  {
    //Encrypt the data, write it to the memory stream.
    encStream.Write(bytesData, 0, bytesData.Length);
  }
  catch(Exception ex)
  {
    throw new Exception("Error while writing encrypted data to the stream: \n"
                        + ex.Message);
  }
  //Set the IV and key for the client to retrieve
  encKey = transformer.Key;
  initVec = transformer.IV;
  encStream.FlushFinalBlock();
  encStream.Close();

  //Send the data back.
  return memStreamEncryptedData.ToArray();
}//end Encrypt
```

23. Fügen Sie der Klasse die folgenden Eigenschaften hinzu.

```
public byte[] IV
{
  get{return initVec;}
  set{initVec = value;}
}

public byte[] Key
{
  get{return encKey;}
}
```

24. Fügen Sie dem Projekt eine neue Klasse mit Namen **Decryptor** hinzu.
25. Fügen Sie am Anfang von **Decryptor.cs** die folgenden **using**-Anweisungen hinzu.

```
using System.Security.Cryptography;
using System.IO;
```

26. Ersetzen Sie den Standardkonstruktor durch den folgenden Konstruktor.

```
public Decryptor(EncryptionAlgorithm algId)
{
  transformer = new DecryptTransformer(algId);
}
```

27. Fügen Sie der Klasse die folgenden Membervariablen vom Typ **private** hinzu.

```
private DecryptTransformer transformer;
private byte[] initVec;
```

28. Fügen Sie der Klasse die folgende **Decrypt**-Methode hinzu.

```
public byte[] Decrypt(byte[] bytesData, byte[] bytesKey)
{
  //Set up the memory stream for the decrypted data.
  MemoryStream memStreamDecryptedData = new MemoryStream();

  //Pass in the initialization vector.
  transformer.IV = initVec;
  ICryptoTransform transform = transformer.GetCryptoServiceProvider(bytesKey);
  CryptoStream decStream = new CryptoStream(memStreamDecryptedData,
                                            transform,
                                            CryptoStreamMode.Write);
  try
  {
    decStream.Write(bytesData, 0, bytesData.Length);
  }
  catch(Exception ex)
  {
    throw new Exception("Error while writing encrypted data to the stream: \n"
                        + ex.Message);
  }
  decStream.FlushFinalBlock();
  decStream.Close();
  // Send the data back.
  return memStreamDecryptedData.ToArray();
} //end Decrypt
```

29. Fügen Sie der Klasse die folgende Eigenschaft hinzu.

```
public byte[] IV
{
  set{initVec = value;}
}
```

30. Klicken Sie im Menü **Erstellen** auf **Projektmappe erstellen**.

2. Erstellen einer Konsolentestanwendung

Mit diesem Verfahren wird eine einfache Konsolentestanwendung erstellt, um die Ver- und Entschlüsselungsfunktionen zu testen.

▶ **So erstellen Sie eine Konsolentestanwendung**

1. Fügen Sie dem aktuellen Projekt mit C# eine neue Konsolenanwendung mit Namen **EncryptionTester** hinzu.
2. Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf das Projekt **EncryptionTester**, und klicken Sie dann auf **Als Startprojekt festlegen**.
3. Benennen Sie **Class1.cs** im Projektmappen-Explorer in **EncryptionTest.cs** um.
4. Benennen Sie in **EncryptionTest.cs** die Klasse **Class1** in **EncryptionTest** um.

5. Fügen Sie dem **Encryption**-Projekt einen Projektverweis hinzu.
6. Fügen Sie am Anfang von **EncryptionTest.cs** die folgenden **using**-Anweisungen hinzu.

```
using System.Text;
using Encryption;
```

7. Fügen Sie der **Main**-Methode den folgenden Code hinzu.

```
// Set the required algorithm
EncryptionAlgorithm algorithm = EncryptionAlgorithm.Des;

// Init variables.
byte[] IV = null;
byte[] cipherText = null;
byte[] key = null;

try
{ //Try to encrypt.
  //Create the encryptor.
  Encryptor enc = new Encryptor(EncryptionAlgorithm.Des);
  byte[] plainText = Encoding.ASCII.GetBytes("Test String");

  if ((EncryptionAlgorithm.TripleDes == algorithm) ||
      (EncryptionAlgorithm.Rijndael == algorithm))
  { //3Des only work with a 16 or 24 byte key.
    key = Encoding.ASCII.GetBytes("password12345678");
    if (EncryptionAlgorithm.Rijndael == algorithm)
    { // Must be 16 bytes for Rijndael.
      IV = Encoding.ASCII.GetBytes("init vec is big.");
    }
    else
    {
      IV = Encoding.ASCII.GetBytes("init vec");
    }
  }
  else
  { //Des only works with an 8 byte key. The others uses variable length keys.
    //Set the key to null to have a new one generated.
    key = Encoding.ASCII.GetBytes("password");
    IV = Encoding.ASCII.GetBytes("init vec");
  }
  // Uncomment the next lines to have the key or IV generated for you.
  // key = null;
  // IV = null;

  enc.IV = IV;

  // Perform the encryption.
  cipherText = enc.Encrypt(plainText, key);
  // Retrieve the intialization vector and key. You will need it
  // for decryption.
  IV = enc.IV;
  key = enc.Key;
```

```
  // Look at your cipher text and initialization vector.
  Console.WriteLine("          Cipher text: " +
                    Convert.ToBase64String(cipherText));
  Console.WriteLine("Initialization vector: " + Convert.ToBase64String(IV));
  Console.WriteLine("                  Key: " + Convert.ToBase64String(key));
}
catch(Exception ex)
{
  Console.WriteLine("Exception encrypting. " + ex.Message);
  return;
}
try
{ //Try to decrypt.
  //Set up your decryption, give it the algorithm and initialization vector.
  Decryptor dec = new Decryptor(algorithm);
  dec.IV = IV;
  // Go ahead and decrypt.
  byte[] plainText = dec.Decrypt(cipherText, key);
  // Look at your plain text.
  Console.WriteLine("           Plain text: " +
                    Encoding.ASCII.GetString(plainText));
}
catch(Exception ex)
{
  Console.WriteLine("Exception decrypting. " + ex.Message);
  return;
}
```

8. Klicken Sie im Menü **Erstellen** auf **Projektmappe erstellen**.
9. Führen Sie die Testanwendung aus, um die Funktionstüchtigkeit der Klassen **Encryptor** und **Decryptor** zu prüfen.

Referenzen

Weitere Informationen finden Sie im Abschnitt "Referenz" dieses Handbuchs unter "Vorgehensweise: Speichern einer verschlüsselten Verbindungszeichenfolge in der Registrierung".