# 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 Sie eine verwaltete Klassenbibliothek erstellen, die DPAPI-Funktionen für Anwendungen bereitstellt, in denen Daten wie Datenbank-Verbindungszeichenfolgen und Kontoanmeldeinformationen verschlüsselt werden sollen.

# Vorgehensweise: Erstellen einer DPAPI-Bibliothek

In Webanwendungen müssen häufig sicherheitsrelevante Daten wie Datenbank-Verbindungszeichenfolgen und Anmeldeinformationen von Dienstkonten in Konfigurationsdateien gespeichert werden. Aus Sicherheitsgründen sollten solche Informationen niemals im Klartext gespeichert und vor der Speicherung immer verschlüsselt werden.

Diese Vorgehensweise erläutert, wie eine verwaltete Klassenbibliothek erstellt wird, die Aufrufe der DPAPI (Data Protection API) zum Ver- und Entschlüsseln von Daten einkapselt. Die Bibliothek kann anschließend auch von anderen verwalteten Anwendungen wie ASP.NET-Webanwendungen, Webdiensten und Enterprise Services-Anwendungen genutzt werden.

Weitere Informationen über die Verwendung der hiermit erstellten DPAPI-Bibliothek finden Sie in den nachstehenden Vorgehensweisen im Abschnitt "Referenz" dieses Handbuchs:

- "Vorgehensweise: Verwenden von DPAPI (Computerspeicher) von ASP.NET aus"
- "Vorgehensweise: Verwenden von DPAPI (Benutzerspeicher) von ASP.NET aus mit Enterprise Services"

## Hinweise

- Mit Microsoft® Windows® 2000 und den neueren Betriebssystemen wurde die Win32® Data Protection API (DPAPI) für das Ver- und Entschlüsseln von Daten eingeführt.
- DPAPI ist Teil der Cryptography API (Crypto API) und in **crypt32.dll** implementiert. Die Schnittstelle setzt sich aus zwei Methoden, **CryptProtectData** und **CryptUnprotectData**, zusammen.

- Die DPAPI ist insofern besonders nützlich, als sich hiermit das Schlüsselverwaltungsproblem erübrigt, das ansonsten mit Anwendungen einhergeht, die Kryptografie einsetzen. Mit der Verschlüsselung können Daten zwar gesichert werden, es müssen jedoch weitere Schritte unternommen werden, um auch die Sicherheit des Schlüssels zu gewährleisten. Die DPAPI verwendet das Kennwort des Benutzerkontos, das die DPAPI-Funktionen aufruft, um den Verschlüsselungsschlüssel abzuleiten. Damit verwaltet das Betriebssystem (und nicht die Anwendung) den Schlüssel.
- Die DPAPI kann entweder mit dem Computerspeicher oder mit dem Benutzerspeicher arbeiten (was jedoch das Laden eines Benutzerprofils voraussetzt). Standardmäßig verwendet die DPAPI den Benutzerspeicher; Sie können jedoch auch festlegen, dass der Computerspeicher verwendet werden soll, indem Sie das Flag CRYPTOPROTECT_LOCAL_MACHINE an die DPAPI-Funktionen übergeben.
- Mit dem Benutzerprofilansatz wird eine zusätzliche Sicherheitsschicht eingezogen, denn hiermit wird der Zugriff auf die geheimen Daten weiter eingeschränkt. Nur der Benutzer, der die Daten verschlüsselt, kann sie auch wieder entschlüsseln. Allerdings erfordert die Verwendung des Benutzerprofils zusätzlichen Entwicklungsaufwand, wenn die DPAPI von einer ASP.NET-Anwendung verwendet werden soll, denn Sie müssen explizite Schritte zum Laden und Entladen des Benutzerprofils unternehmen (da ASP.NET Benutzerprofile nicht automatisch lädt).
- Der Computerspeicheransatz ist einfacher zu entwickeln, da er keine Benutzerprofilverwaltung voraussetzt. Wird hierbei jedoch kein zusätzlicher Entropieparameter verwendet, ist dieser Ansatz weniger sicher, da jeder Benutzer des Computers Daten entschlüsseln kann. (Unter Entropie wird in diesem Zusammenhang ein Zufallswert verstanden, der das Dechiffrieren der Geheimdaten schwieriger macht.) Das Problem bei der Verwendung eines zusätzlichen Entropieparameters besteht darin, dass dieser Wert von der Anwendung ebenfalls sicher gespeichert werden muss, woraus sich ein weiteres Schlüsselverwaltungsproblem ergibt.

---

**Hinweis:** Wenn Sie die DPAPI mit dem Computerspeicher verwenden, ist die verschlüsselte Zeichenfolge für den jeweiligen Computer spezifisch, daher müssen die verschlüsselten Daten auf jedem Computer erzeugt werden. Sie können die verschlüsselten Daten nicht auf andere Computer in einer Farm oder in einem Cluster kopieren.

Wenn Sie DPAPI mit dem Benutzerspeicher verwenden, können die Daten auf jedem Computer mit einem servergespeicherten Benutzerprofil entschlüsselt werden.

---

## 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
- Microsoft Visual Studio® .NET als Entwicklungssystem

Die in dieser Vorgehensweise erläuterten Verfahren setzen zudem Kenntnisse des Entwicklungstools Microsoft Visual C#™ voraus.

## Zusammenfassung

Diese Vorgehensweise enthält folgende Verfahren:

1. Erstellen einer C#-Klassenbibliothek
2. Zuweisen eines starken Namens für die Assembly (optional)

# 1. Erstellen einer C#-Klassenbibliothek

Mit dem nachstehenden Verfahren wird eine C#-Klassenbibliothek erstellt, die Ver- und Entschlüsselungsmethoden bereitstellt. Hiermit werden Aufrufe der Win32-DPAPI-Funktionen gekapselt.

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

1. Starten Sie Visual Studio .NET, und erstellen Sie ein neues Visual C#-Klassenbibliotheksprojekt mit Namen **DataProtection**.
2. Benennen Sie **class1.cs** im Projektmappen-Explorer in **DataProtection.cs** um.
3. Benennen Sie in **DataProtection.cs** die Klasse **class1** in **DataProtector** um, und weisen Sie auch dem Standardkonstruktor den entsprechenden Namen zu.
4. Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf **DataProtection**, und klicken Sie dann auf **Eigenschaften**.
5. Klicken Sie auf den Ordner **Konfigurationseigenschaften**, und legen Sie **Unsichere Codeblöcke zulassen** auf **True** fest.
6. Klicken Sie auf **OK**, um das Dialogfeld **Eigenschaften** zu schließen.
7. Fügen Sie am Anfang von **DataProtection.cs** unterhalb der vorhandenen **using**-Anweisung die folgenden **using**-Anweisungen hinzu.

```
using System.Text;
using System.Runtime.InteropServices;
```

8. Fügen Sie am Anfang der **DataProtector**-Klasse die folgenden **DllImport**-Anweisungen hinzu, damit die Win32-DPAPI-Funktionen zusammen mit der Hilfsfunktion **FormatMessage** über **P/Invoke** aufgerufen werden können.

```
[DllImport("Crypt32.dll", SetLastError=true,
            CharSet=System.Runtime.InteropServices.CharSet.Auto)]
private static extern bool CryptProtectData(
                                ref DATA_BLOB pDataIn,
                                String szDataDescr,
                                ref DATA_BLOB pOptionalEntropy,
                                IntPtr pvReserved,
                                ref CRYPTPROTECT_PROMPTSTRUCT pPromptStruct,
                                int dwFlags,
                                ref DATA_BLOB pDataOut);
[DllImport("Crypt32.dll", SetLastError=true,
            CharSet=System.Runtime.InteropServices.CharSet.Auto)]
private static extern bool CryptUnprotectData(
                                ref DATA_BLOB pDataIn,
                                String szDataDescr,
                                ref DATA_BLOB pOptionalEntropy,
                                IntPtr pvReserved,
                                ref CRYPTPROTECT_PROMPTSTRUCT pPromptStruct,
                                int dwFlags,
                                ref DATA_BLOB pDataOut);
[DllImport("kernel32.dll",
            CharSet=System.Runtime.InteropServices.CharSet.Auto)]
private unsafe static extern int FormatMessage(int dwFlags,
                                            ref IntPtr lpSource,
                                            int dwMessageId,
                                            int dwLanguageId,
                                            ref String lpBuffer, int nSize,
```

```
                                                IntPtr *Arguments);
```

9. Fügen Sie die folgenden Strukturdefinitionen und Konstanten hinzu, die von den DPAPI-Funktionen verwendet werden.

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Unicode)]
internal struct DATA_BLOB
{
  public int cbData;
  public IntPtr pbData;
}

[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Unicode)]
internal struct CRYPTPROTECT_PROMPTSTRUCT
{
  public int cbSize;
  public int dwPromptFlags;
  public IntPtr hwndApp;
  public String szPrompt;
}
static private IntPtr NullPtr = ((IntPtr)((int)(0)));
private const int CRYPTPROTECT_UI_FORBIDDEN = 0x1;
private const int CRYPTPROTECT_LOCAL_MACHINE = 0x4;
```

10. Fügen Sie der Klasse einen Auflistungstyp vom Typ **public** mit Namen **Store** hinzu. Dieser wird verwendet, um anzugeben, ob die DPAPI in Verbindung mit Computerspeichern oder Benutzerspeichern verwendet werden soll.

```
public enum Store {USE_MACHINE_STORE = 1, USE_USER_STORE};
```

11. Fügen Sie der Klasse eine Membervariable vom Typ **private** mit Namen **Store** hinzu.

```
private Store store;
```

12. Ersetzen Sie den Standardkonstruktor der Klasse durch den folgenden Konstruktor, der einen **Store**-Parameter akzeptiert und den gelieferten Wert an die private Membervariable **store** übergibt.

```
public DataProtector(Store tempStore)
{
  store = tempStore;
}
```

13. Fügen Sie der Klasse die folgende **Encrypt**-Methode vom Typ **public** hinzu.

```
public byte[] Encrypt(byte[] plainText, byte[] optionalEntropy)
{
  bool retVal = false;

  DATA_BLOB plainTextBlob = new DATA_BLOB();
  DATA_BLOB cipherTextBlob = new DATA_BLOB();
  DATA_BLOB entropyBlob = new DATA_BLOB();
```

```
CRYPTPROTECT_PROMPTSTRUCT prompt = new CRYPTPROTECT_PROMPTSTRUCT();
InitPromptstruct(ref prompt);

int dwFlags;
try
{
  try
  {
    int bytesSize = plainText.Length;
    plainTextBlob.pbData = Marshal.AllocHGlobal(bytesSize);
    if(IntPtr.Zero == plainTextBlob.pbData)
    {
      throw new Exception("Unable to allocate plaintext buffer.");
    }
    plainTextBlob.cbData = bytesSize;
    Marshal.Copy(plainText, 0, plainTextBlob.pbData, bytesSize);
  }
  catch(Exception ex)
  {
    throw new Exception("Exception marshalling data. " + ex.Message);
  }
  if(Store.USE_MACHINE_STORE == store)
  {//Using the machine store, should be providing entropy.
    dwFlags = CRYPTPROTECT_LOCAL_MACHINE|CRYPTPROTECT_UI_FORBIDDEN;
    //Check to see if the entropy is null
    if(null == optionalEntropy)
    {//Allocate something
      optionalEntropy = new byte[0];
    }
    try
    {
      int bytesSize = optionalEntropy.Length;
      entropyBlob.pbData = Marshal.AllocHGlobal(optionalEntropy.Length);;
      if(IntPtr.Zero == entropyBlob.pbData)
      {
        throw new Exception("Unable to allocate entropy data buffer.");
      }
      Marshal.Copy(optionalEntropy, 0, entropyBlob.pbData, bytesSize);
      entropyBlob.cbData = bytesSize;
    }
    catch(Exception ex)
    {
      throw new Exception("Exception entropy marshalling data. " +
                          ex.Message);
    }
  }
  else
  {//Using the user store
    dwFlags = CRYPTPROTECT_UI_FORBIDDEN;
  }
  retVal = CryptProtectData(ref plainTextBlob, "", ref entropyBlob,
                            IntPtr.Zero, ref prompt, dwFlags,
                            ref cipherTextBlob);
  if(false == retVal)
```

```
    {
      throw new Exception("Encryption failed. " +
                          GetErrorMessage(Marshal.GetLastWin32Error()));
    }
  }
  catch(Exception ex)
  {
    throw new Exception("Exception encrypting. " + ex.Message);
  }
  byte[] cipherText = new byte[cipherTextBlob.cbData];
  Marshal.Copy(cipherTextBlob.pbData, cipherText, 0, cipherTextBlob.cbData);
  return cipherText;
}
```

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

```
public byte[] Decrypt(byte[] cipherText, byte[] optionalEntropy)
{
  bool retVal = false;
  DATA_BLOB plainTextBlob = new DATA_BLOB();
  DATA_BLOB cipherBlob = new DATA_BLOB();
  CRYPTPROTECT_PROMPTSTRUCT prompt = new CRYPTPROTECT_PROMPTSTRUCT();
  InitPromptstruct(ref prompt);
  try
  {
    try
    {
      int cipherTextSize = cipherText.Length;
      cipherBlob.pbData = Marshal.AllocHGlobal(cipherTextSize);
      if(IntPtr.Zero == cipherBlob.pbData)
      {
        throw new Exception("Unable to allocate cipherText buffer.");
      }
      cipherBlob.cbData = cipherTextSize;
      Marshal.Copy(cipherText, 0, cipherBlob.pbData, cipherBlob.cbData);
    }
    catch(Exception ex)
    {
      throw new Exception("Exception marshalling data. " + ex.Message);
    }
    DATA_BLOB entropyBlob = new DATA_BLOB();
    int dwFlags;
    if(Store.USE_MACHINE_STORE == store)
    {//Using the machine store, should be providing entropy.
      dwFlags = CRYPTPROTECT_LOCAL_MACHINE|CRYPTPROTECT_UI_FORBIDDEN;
      //Check to see if the entropy is null
      if(null == optionalEntropy)
      {//Allocate something
        optionalEntropy = new byte[0];
      }
      try
      {
        int bytesSize = optionalEntropy.Length;
        entropyBlob.pbData = Marshal.AllocHGlobal(bytesSize);
```

```
        if(IntPtr.Zero == entropyBlob.pbData)
        {
          throw new Exception("Unable to allocate entropy buffer.");
        }
        entropyBlob.cbData = bytesSize;
        Marshal.Copy(optionalEntropy, 0, entropyBlob.pbData, bytesSize);
      }
      catch(Exception ex)
      {
        throw new Exception("Exception entropy marshalling data. " +
                            ex.Message);
      }
    }
    else
    {//Using the user store
      dwFlags = CRYPTPROTECT_UI_FORBIDDEN;
    }
    retVal = CryptUnprotectData(ref cipherBlob, null, ref entropyBlob,
                                IntPtr.Zero, ref prompt, dwFlags,
                                ref plainTextBlob);
    if(false == retVal)
    {
      throw new Exception("Decryption failed. " +
                          GetErrorMessage(Marshal.GetLastWin32Error()));
    }
    //Free the blob and entropy.
    if(IntPtr.Zero != cipherBlob.pbData)
    {
      Marshal.FreeHGlobal(cipherBlob.pbData);
    }
    if(IntPtr.Zero != entropyBlob.pbData)
    {
      Marshal.FreeHGlobal(entropyBlob.pbData);
    }
  }
  catch(Exception ex)
  {
    throw new Exception("Exception decrypting. " + ex.Message);
  }
  byte[] plainText = new byte[plainTextBlob.cbData];
  Marshal.Copy(plainTextBlob.pbData, plainText, 0, plainTextBlob.cbData);
  return plainText;
}
```

15. Fügen Sie der Klasse nun die folgenden Hilfsmethoden vom Typ **private** hinzu.

```
private void InitPromptstruct(ref CRYPTPROTECT_PROMPTSTRUCT ps)
{
  ps.cbSize = Marshal.SizeOf(typeof(CRYPTPROTECT_PROMPTSTRUCT));
  ps.dwPromptFlags = 0;
  ps.hwndApp = NullPtr;
  ps.szPrompt = null;
}
```

```
private unsafe static String GetErrorMessage(int errorCode)
{
  int FORMAT_MESSAGE_ALLOCATE_BUFFER = 0x00000100;
  int FORMAT_MESSAGE_IGNORE_INSERTS = 0x00000200;
  int FORMAT_MESSAGE_FROM_SYSTEM  = 0x00001000;
  int messageSize = 255;
  String lpMsgBuf = "";
  int dwFlags = FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM |
               FORMAT_MESSAGE_IGNORE_INSERTS;
  IntPtr ptrlpSource = new IntPtr();
  IntPtr prtArguments = new IntPtr();
  int retVal = FormatMessage(dwFlags, ref ptrlpSource, errorCode, 0,
                             ref lpMsgBuf, messageSize, &prtArguments);
  if(0 == retVal)
  {
    throw new Exception("Failed to format message for error code " +
                        errorCode + ". ");
  }
  return lpMsgBuf;
}
```

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

## 2. Zuweisen eines starken Namens für die Assembly (optional)

Wenn die verwaltete DPAPI-Klassenbibliothek von einer Enterprise Services-Anwendung aufgerufen werden soll (die über einen starken Namen verfügen muss), muss auch die DPAPI-Klassenbibliothek einen starken Namen aufweisen. Mit diesem Verfahren wird der Klassenbibliothek ein starker Name zugewiesen.

Wenn die verwaltete DPAPI-Klassenbibliothek direkt von einer ASP.NET-Webanwendung aufgerufen werden soll (die nicht über einen starken Namen verfügt), können Sie dieses Verfahren übergehen.

▶ **So weisen Sie der Assembly einen starken Namen zu**

1. Öffnen Sie ein Befehlsfenster, und wechseln Sie in den Projektordner **DataProtection**.
2. Erzeugen Sie mithilfe des Dienstprogramms **sn.exe** ein Schlüsselpaar zum Signieren der Assembly.

```
sn -k dataprotection.snk
```

3. Kehren Sie zu Visual Studio .NET zurück, und öffnen Sie **Assemblyinfo.cs**.
4. Suchen Sie das **AssemblyKeyFile**-Attribut, und fügen Sie den Pfad zur Schlüsseldatei im Projektordner hinzu.

```
[assembly: AssemblyKeyFile(@"..\..\dataprotection.snk")]
```

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

## Referenzen

Weitere Informationen finden Sie in den folgenden zugehörigen Vorgehensweisen:

- "Vorgehensweise: Verwenden von DPAPI (Computerspeicher) von ASP.NET aus" im Abschnitt "Referenz" dieses Handbuchs
- "Vorgehensweise: Verwenden von DPAPI (Benutzerspeicher) von ASP.NET aus mit Enterprise Services" im Abschnitt "Referenz" dieses Handbuchs