15 Sicherheit

Codezugriffssicherheit

Codezugriffssicherheit (*Code Access Security*, *CAS*) ist ein Mechanismus, der regelt, welche Teile das Programmcodes auf welche Systemressourcen zugreifen dürfen.

Sicherheitsrichtlinie

Eine Sicherheitsrichtlinie ist ein Satz von Regeln für den Zugriff auf Ressourcen.

Das Code-Access-Security-Policy-Tool (Caspol.exe)

Das Programm *Caspol.exe* ist ein kommandozeilenorientiertes Tool, das dem Administrator die Konfiguration von Sicherheitsrichtlinien ermöglicht. Sie finden es im Ordner *Microsoft.NET\Framework\v.x.y.z* unterhalb Ihres Windows-Verzeichnisses.

Das .NET-Framework-Configuration-Tool (Mscorcfg.msc)

Auch mit dem .NET-Framework-Configuration-Tool können Sie Sicherheitsrichtlinien konfigurieren. Klicken Sie dazu im Startmenü auf *Einstellungen / Systemsteuerung*, dann auf das Symbol *Verwaltung* und schließlich auf *Microsoft .NET Framework-Konfiguration*.

Abbildung 15.1: Sicherheitsrichtlinien im .NET-Framework-Configuration-Tool

Identitätsberechtigungen

Identitätsberechtigungen (*Identity Permissions*) sind Berechtigungen, die einer Assembly durch die Common Language Runtime aufgrund der Identität dieser Assembly erteilt werden.

Beweise

Unter *Beweisen* versteht man Informationen über die Identität einer Assembly. Dazu gehören z.B.:

- ✔ Herausgeber
- ✔ Site
- ✔ Zone

Mithilfe von Attributen kann der Programmcode bestimmte Beweise fordern.

Das Attribut [StrongNameIdentityPermission]

Das Attribut `[StrongNameIdentityPermission]` fordert einen bestimmten starken Namen.

Beispiel

Das folgende Beispiel zeigt eine Klasse, auf die nur von einer Assembly zugegriffen werden kann, die einen ganz bestimmten öffentlichen Schüssel enthält – die also mit dem zugehörigen privaten Schlüssel signiert wurde.

Sie finden das Beispiel auf der Buch-CD im Verzeichnis *TeilIII\IdentityPermsBsp*.

Um die gleiche Verzeichnisstruktur zu erhalten wie diejenige auf der CD, legen Sie zunächst eine leere Projektmappe an: Klicken Sie im Menü *Datei* auf *Neu / Leere Projektmappe*. Geben Sie der Projektmappe den Namen *IdentityPermsBsp*.

Server

Klicken Sie dann im Projektmappen-Explorer mit der rechten Maustaste auf die Projektmappe und wählen Sie aus dem Kontextmenü den Befehl *Hinzufügen / Neues Projekt*.

Im Dialogfeld *Neues Projekt* wählen Sie unter *Projekttypen* den Eintrag *Visual C#-Projekte* und klicken Sie im Feld *Vorlagen* auf *Klassenbibliothek*. Geben Sie dem Projekt einen Namen, z.B. *Server* und wählen Sie im Feld *Speicherort* ein Verzeichnis.

Schreiben Sie die folgende using-Direktive in die Datei *Class1*.cs:

```
using System.Security.Permissions;
```

Fügen Sie dann eine Methode zum Testen in die Klasse Class1 ein:

```
public static string GetInfo()
{
   return "Dies ist eine vertrauliche Information";
}
```

Erzeugen Sie ein Schlüsselpaar (falls noch nicht vorhanden) und signieren Sie die Assembly, indem Sie in der Datei *AssemblyInfo.cs* das Attribut [assembly: AssemblyKeyFile] setzen:

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

Kennzeichnen Sie nun die Klasse Class1 mit dem [StrongNameIdentityPermission]-Attribut:

```
[StrongNameIdentityPermission(SecurityAction.Demand,
PublicKey="0x00240000048000009400000006020000002400005253413100040000010001006B852F6E049BD98B9A861F0997A463A8E34F638E5B0CC2B025DC8A75791C84439130A3F30CCA4561154C86A67BE3E2BC3EEF8F3701170E349C53EF94253DB5E41EB48DA0CFC9047A1601E45DF3F3C2B659315DBF44574C310D948D411784720015D3BB6CF782C38BA354AB25DC218E291B41A5DD167235E093B57039334970E6")]
```

Ersetzen Sie den Schlüssel durch den von Ihnen erzeugten. Eine Klartextdarstellung des Schlüssels erhalten Sie aus der fertig kompilierten Assembly mithilfe des Programms *SecUtil.exe*. Kompilieren Sie den Server also einmal, indem Sie im Projektmappen-Explorer mit der rechten Maustaste auf das Projekt klicken und aus dem Kontextmenü den Befehl *Erstellen* wählen. Wenn Sie Visual Studio installiert haben, finden Sie *Secutil.exe* im Verzeichnis *FrameworkSDK\Bin* unterhalb des Visual-Studio-Installationsverzeichnisses. Starten Sie die Eingabeaufforderung, wechseln Sie in das Verzeichnis, in dem sich Ihre Server-DLL befindet und rufen Sie das Tool mit der folgenden Syntax auf:

```
"C:\Programme\Microsoft Visual Studio
.NET\FrameworkSDK\Bin\secutil" -hex -s Server.DLL
```

(Passen Sie den Pfad gegebenenfalls an Ihre Installation an.) Die Ausgabe können Sie aus dem Eingabeaufforderungsfenster herauskopieren oder in eine Datei umlenken.

Nachdem Sie das `[StrongNameIdentityPermission]`-Attribut eingefügt haben, kompilieren Sie den Server nochmals.

Client

Erstellen Sie ein zweites Projekt, welches versucht, die geschützte Assembly zu verwenden:

Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf die Projektmappe und wählen Sie aus dem Kontextmenü den Befehl *Hinzufügen / Neues Projekt*.

Im Dialogfeld *Neues Projekt* wählen Sie unter *Projekttypen* den Eintrag *Visual C#-Projekte* und klicken Sie im Feld *Vorlagen* auf *Konsolenanwendung*. Geben Sie dem Projekt einen Namen, z.B. *Client* und wählen Sie im *Feld Speicherort* ein Verzeichnis.

Legen Sie das neue Projekt als Startprojekt fest.

Fügen Sie einen Assemblyverweis auf den Server in das Projekt ein.

Ergänzen Sie die `Main`-Methode wie folgt:

```
static void Main(string[] args)
{
  try
  {
    Console.WriteLine(Server.Class1.GetInfo());
  }
  catch(Exception e)
  {
    Console.WriteLine(e.Message);
  }
}
```

Starten Sie den Client. Wie erwartet wird eine Ausnahme ausgelöst: der Client hat keinen Zugriff auf den Server.

Signieren Sie jetzt den Client mit dem korrekten Schlüssel, indem Sie auch hier das `[assembly: AssemblyKeyFile]`-Attribut angeben (mit demselben Schlüssel, den Sie für den Server verwendet haben).

Testen Sie das Programm nochmals: der Client kann jetzt auf die geschützte Assembly zugreifen.

Rollenbasierte Sicherheit

Bei der *rollenbasierten Sicherheit* müssen Benutzer sich mit Benutzernamen und Kennwort anmelden (*authentifizieren*). Häufig werden Berechtigungen nicht für einzelne Personen festgelegt, sondern für so genannte Rollen (z.B. Systemadministratoren, Entwickler, normale Benutzer, Gäste usw.). Die Zuordnung der Rolle kann z.B. über Benutzergruppen erfolgen.

In manchen Anwendungen wird rollenbasierte Sicherheit auch verwendet, um zu erzwingen, dass mehrere Rollen an einem Vorgang beteiligt sind (z.B. Bestätigung durch Vorgesetzte).

Als Beispiel soll hier eine Webanwendung mit rollenbasierter Sicherheit entwickelt werden.

Sie finden das Beispiel auf der Buch-CD im Verzeichnis *TeilIII\WebSicherheitBsp*.

Um ein Web-Forms-Projekt zu erstellen, klicken Sie im Menü *Datei* auf den Befehl *Neu / Projekt*. Im Dialogfeld *Neues Projekt* wählen Sie unter *Projekttypen* den Eintrag *Visual C#-Projekte*. Im Feld Vorlagen klicken Sie diesmal auf *Windows-Anwendung*. Legen *Sie* im Feld *Speicherort* die URL für das Projekt fest, z.B. *http://localhost/WebSicherheitBsp*.

Sie können die vom Assistenten erstellte Seite *WebForm1.aspx* umbenennen in *Geheim.aspx*: Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf die Seite und wählen Sie aus dem Kontextmenü den Befehl *Umbenennen*. Fügen Sie ein `Label` in die Seite ein, um beim Testen erst einmal überhaupt etwas zu sehen.

Fügen Sie eine zweite Seite ein, indem Sie im Projektmappen-Explorer mit der rechten Maustaste auf das Projekt klicken und aus dem Kontextmenü den Befehl *Hinzufügen / Neues Element hinzufügen* wählen. Im Dialogfeld *Neues Element hinzufügen* wählen Sie unter Vorlagen den Eintrag *WebForm*. Geben Sie der Seite den Namen *Oeffentlich.aspx*. Fügen Sie auch hier ein Label ein. Legen Sie die neue Seite als Startseite fest (über das Kontextmenü).

Authentifizierung

Verschiedene Verfahren werden bei Webanwendungen eingesetzt:

- ✔ Windows:
  - ✔ Basic-Authentifizierung (Standardauthentifizierung)
  - ✔ Digest-Authentifizierung
  - ✔ Betriebssystemverfahren
  - ✔ z.B. NTLM oder Kerberos
- ✔ anwendungsdefinierte Verfahren mit HTML-Formularen
- ✔ Passport-Authentifizierung
- ✔ Clientzertifikate

Authentifizierungseinstellungen im IIS

Die Sicherheitsmechanismen bei ASP-.NET-Webanwendungen arbeiten eng mit den Sicherheitsmechanismen des Internet Information Servers zusammen. Daher müssen Sie zunächst die Authenti-

fizierungseinstellungen im IIS konfigurieren. Starten Sie dazu den Internetdienste-Manager, indem Sie im Startmenü auf den Befehl *Einstellungen / Systemsteuerung / Verwaltung / Internetdienste-Manager* klicken. Selektieren Sie auf der Registerkarte *Struktur* Ihr Projekt (z.B. *WebSicherheitBsp*), klicken Sie im Feld rechts daneben mit der rechten Maustaste auf die Seite *Oeffentlich.aspx* und wählen Sie im Kontextmenü den Befehl *Eigenschaften*.

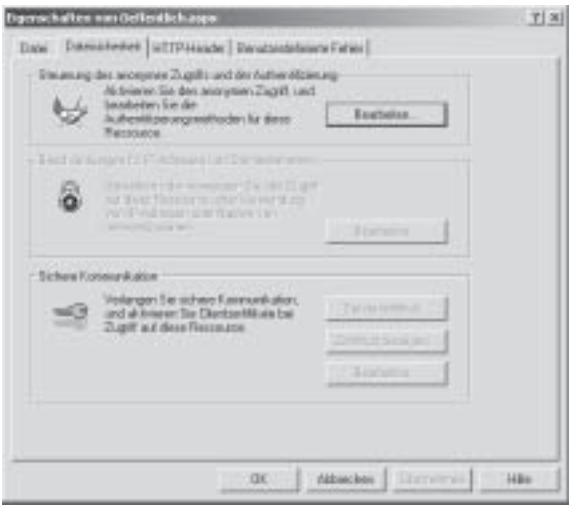

Abbildung 15.2: Dateisicherheit im IIS

Auf der Registerkarte *Dateisicherheit* klicken Sie unter *Steuerung des anonymen Zugriffs und der Authentifizierung* auf die Schaltfläche *Bearbeiten*.

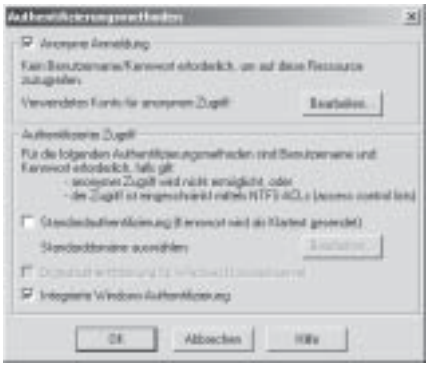

Abbildung 15.3: Das Dialogfeld *Authentifizierungsmethoden*

Anonyme Anmeldung

Wenn das Kontrollkästchen *Anonyme Anmeldung* selektiert ist, wird kein Anmeldedialog angezeigt. Die Anmeldung erfolgt automatisch mit der Benutzerkennung `IUSR_`*`Maschine`*, wobei *`Maschine`* der Rechnername des Server-Computers ist (oder unter der Kennung `ASPNET`, vgl. Abschnitt »Identitätswechsel« weiter unten).

Behalten Sie für die Seite *Oeffentlich.aspx* die anonyme Anmeldung bei.

Authentifizierter Zugriff

Zeigen Sie nun die Authentifizierungseigenschaften für die Seite *Geheim.aspx* an. Entfernen Sie hier das Häkchen im Kontrollkästchen *Anonyme Anmeldung* und testen Sie Ihre Webanwendung: Für diese Seite müssen Benutzer sich jetzt immer authentifizieren.

Integrierte Windows-Authentifizierung

Im Feld *Authentifizierter Zugriff* des Dialogfelds *Authentifizierungsmethoden* ist per Voreinstellung die *Integrierte Windows-Authentifizierung* ausgewählt.

In diesem Fall kann ein Benutzer sich mit seiner Windows-Benutzerkennung und dem dazugehörigen Passwort anmelden. Dazu muss das entsprechende Benutzerkonto auf dem Server natürlich existieren. Eine solche Lösung ist vor allem in Intranets praktisch.

Für den Zugriff auf die Dateien der Website gelten dann die auf Betriebssystemebene erteilten Berechtigungen. Windows speichert diese in einer so genannten *ACL* (*Access Control List*).

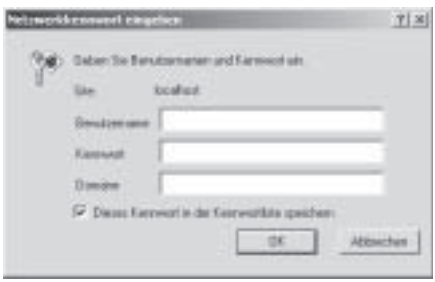

Abbildung 15.4: Der Anmeldedialog

Standard-Authentifizierung und Digest-Authentifizierung

Wählen Sie im Feld *Authentifizierter Zugriff* des Dialogfelds *Authentifizierungsmethoden* das Kontrollkästchen *Standardauthentifizierung (Kennwort wird als Klartext gesendet)* aus, wenn Sie den Standard-HTTP-Authentifizierungsmechanismus verwenden wollen. Der Browser zeigt dann einen Anmeldedialog an und sendet die dort eingegebenen Daten an den Webserver – allerdings tatsächlich im Klartext! Diese Methode ist also für Anwendungen mit hohen Sicherheitsanforderungen ungeeignet.

Auf einem Windows-Domänenserver können Sie stattdessen die sicherere *Digest-Authentifizierung* verwenden. Dabei wird nicht das Kennwort selbst, sondern ein aus diesem berechneter *Digest* an den Webserver gesendet. Der Server erzeugt nach derselben Methode

aus dem Originalkennwort einen Digest und vergleicht diesen mit dem empfangenen.

Hostbasierte Zugriffskontrolle

Unter Windows 2000 Server oder NT4 (nicht unter Windows 2000 Professional) können Sie im IIS den Zugriff für bestimmte Hosts oder Domänen regeln. Zeigen Sie dazu wieder die Eigenschaften der Website an und klicken Sie auf der Registerkarte *Verzeichnissicherheit* unter *Beschränkungen für IP-Adressen und Domänennamen* auf die Schaltfläche *Bearbeiten*.

Client-Zertifikate

Auch die Konfiguration für die Verwendung von Client-Zertifikaten erfolgt über die Registerkarte *Verzeichnissicherheit* des *Eigenschaften*-Dialogfelds.

Auswerten von Anmeldeinformationen

Die Eigenschaft `User` der Klasse `Page` enthält ein Principal-Objekt, welches Informationen über den Benutzer enthält.

Principal-Objekte

Principal-Objekte werden zur Überprüfung der Mitgliedschaft einer Identität (eines Benutzers) in einer Rolle (einer Benutzergruppe) mithilfe von Programmcode verwendet.

Die Schnittstelle `IPrincipal` definiert die Methoden und Eigenschaften von Principal-Objekten. Sie wird implementiert von den Klassen `GenericPrinciple` und `WindowsPrinciple`.

Identity

Die `Identity`-Eigenschaft eines Principal-Objekts enthält ein `IIdentity`-Objekt, welches die Identität des Principals darstellt.

IsInRole-Methode

Die Methode `IsInRole` bestimmt, ob der aktuelle Principal zu einer bestimmten Benutzergruppe (Rolle) gehört.


```
IPrincipal.IsInRole-Methode
bool IsInRole(
   string role
);
```

Parameter:

`role`

Der Name der Rolle, für die die Mitgliedschaft überprüft werden soll.

Rückgabewert:

`true`, wenn der aktuelle Principal ein Member der angegebenen Rolle ist, andernfalls `false`.

Allgemeine Principals

Die Klasse `GenericPrincipal` stellt einen allgemeinen Principal dar.

Windows-Principals

Die Klasse `WindowsPrincipal` ermöglicht es, die Mitgliedschaft eines Windows-Benutzers in einer Windows-Gruppe zu überprüfen.

Benutzerdefinierte Principals

Sie können auch eigene Principal-Klassen definieren. Diese müssen die Schnittstelle `IPrincipal` implementieren.

Identitätsobjekte

Die Schnittstelle `IIDentity` definiert Eigenschaften, welche die Identität eines Benutzers beschreiben. Sie wird von den Klassen

FormsIdentity, GenericIdentity, PassportIdentity und WindowsIdentity implementiert.

Name

Die Name-Eigenschaft enthält den Anmeldenamen.

IsAuthenticated

Die Eigenschaft IsAuthenticated gibt an, ob der Benutzer mit Namen und Kennwort angemeldet wurde.

AuthenticationType

Die schreibgeschützte Eigenschaft AuthenticationType gibt den Authentifizierungstyp an. Mögliche Typen sind z.B. Standardauthentifizierung, NTLM, Kerberos und Passport.

Beispiel

Fügen Sie drei weitere Labels in die Seite *Geheim.aspx* ein. Ergänzen Sie dann die Prozedur Page_Load wie folgt:

```
private void Page_Load(object sender,
System.EventArgs e)
{
  if (User.Identity.IsAuthenticated)
  {
    Label1.Text = "Hallo, " + User.Identity.Name;
    if (User.IsInRole("VORDEFINIERT\\Administrator"))
    {
      Label2.Text = "Sie sind ein Administrator";
    }
    else
    {
      Label2.Text = "Sie sind kein Administrator";
    }
    if (User.IsInRole("MeineDomain\\Bearbeiter"))
```

```
    {
        Label3.Text = "Sie sind ein Bearbeiter";
    }
    else
    {
      Label3.Text = "Sie sind kein Bearbeiter";
    }
    Label4.Text = "Authentifizierung: " +
    User.Identity.AuthenticationType;
  }
  else
  {
    Label1.Text = "Sie sind nicht authentifiziert !";
  }
}
```

Wenn Sie die anonyme Anmeldung deaktiviert haben, liefert `IsAuthenticated` immer `true`. Der letzte `else`-Zweig in obigem Beispiel kann also nur mit aktivierter anonymer Anmeldung erreicht werden.

Identitätswechsel

Unter *Identitätswechsel* versteht man die Möglichkeit, Code unter einer anderen Benutzerkennung auszuführen.

Um den Identitätswechsel für eine Webanwendung zu aktivieren, fügen Sie in die Konfigurationsdatei *web.config* die folgende Zeile ein:

```
<identity impersonate="true">
```

Um den Zielbenutzer festzulegen, geben Sie zusätzlich die Attribute `userName` und `password` an:

```
<identity impersonate="true"
          userName="MeineDomain\test"
          password="abc123" />
```

Wenn die Angaben zum Zielbenutzer fehlen, läuft ASP-.NET-Code unter der Benutzerkennung `IUSR_Maschine`. Ist der Identitätswechsel dagegen gar nicht aktiviert, so wird die vordefinierte Benutzerkennung `ASPNET` verwendet.

Authentifizierungsprovider

Ein *Authentifizierungsprovider* ist ein Modul, das den Code zur Authentifizierung enthält. ASP .NET unterstützt die Verfahren `Windows`, `Forms`, `Passport` oder die Einstellung `None` (keine Authentifizierung).

Der Windows-Authentifizierungsprovider

Zum Aktivieren des Authentifizierungsproviders `Windows` muss die Datei *Web.config* die folgende Zeile enthalten:

```
<authentication mode="Windows" />
```

Der Anwendungsassistent schreibt diese Zeile automatisch in die Konfigurationsdatei.

Der Forms-Authentifizierungsprovider

Zum Aktivieren des Authentifizierungsproviders `Forms` muss die Datei *Web.config* die folgende Zeile enthalten:

```
<authentication mode="Forms" />
```

Der Passport-Authentifizierungsprovider

Zum Aktivieren des Authentifizierungsproviders `Passport` muss die Datei *Web.config* die folgende Zeile enthalten:

```
<authentication mode="Passport" />
```

Autorisierung

Der Abschnitt `<authorization>` in der Datei *web.config* legt fest, welche Benutzer und Rollen Zugriff auf die Anwendung haben.

Beispiel:

```
<authorization>
  <allow roles="MeineDomain\Bearbeiter" />
  <deny users="MeineDomain\Klara,MeineDomain\Olga" />
</authorization>
```

Beachten Sie aber, dass Einstellungen in der Hauptkonfigurationsdatei *machine.config* Vorrang vor denjenigen in *web.config* haben.

Verschlüsselung

Kryptografische Verfahren ermöglichen das Verschlüsseln und Entschlüsseln von Daten, z.B. für die sichere Übertragung im Netzwerk. Die .NET-Klassenbibliothek stellt zu diesem Zweck eine Reihe von Klassen in dem Namespace `System.Security.Cryptography` zur Verfügung.

Symmetrische und unsymmetrische Schlüssel

Bei den Verschlüsselungsalgorithmen unterscheidet man zwischen symmetrischen und unsymmetrischen Verfahren. Bei der symmetrischen Verschlüsselung gibt es nur einen Schlüssel, der sowohl zum Verschlüsseln als auch zum Entschlüsseln verwendet wird. Dieser Schlüssel muss daher geheim gehalten werden.

Bei unsymmetrischen Verfahren arbeitet man mit zwei Schlüsseln: einem *öffentlichen Schlüssel* (*Public Key*) für die Verschlüsselung und einem *privaten Schlüssel* (*Private Key*) zum Entschlüsseln. Der Besitzer des privaten Schlüssels übermittelt den zugehörigen öffentlichen Schlüssel an Personen, die ihm verschlüsselte Nachrichten senden wollen. Selbst wenn dieser öffentliche Schlüssel Unbefug-

ten in die Hände fällt, können diese die Nachrichten nicht entschlüsseln – dafür wird der private Schlüssel benötigt. Unsymmetrische Verschlüsselung wird z.B. bei SSL (*Secure Sockets Layer*) eingesetzt.

Bei digitalen Signaturen wird die Herkunft und Unverfälschtheit von Informationen sichergestellt, indem Daten mit einem privaten Schlüssel signiert werden. Dazu bildet der Absender aus den Daten einen Hashwert (Digest) und verschlüsselt diesen mit seinem privaten Schlüssel. Der Empfänger kann dann mit dem dazugehörigen öffentlichen Schlüssel den Digest entschlüsseln und mit dem von ihm selbst ermittelten Digest vergleichen. Damit ist einerseits gewährleistet, dass die Daten tatsächlich vom Besitzer des privaten Schlüssels stammen, andererseits auch, dass die Daten unterwegs nicht manipuliert wurden.

Symmetrische Verfahren: die Klasse SymmetricAlgorithm

Die Klasse `SymmetricAlgorithm` ist die abstrakte Basisklasse für alle Klassen, die symmetrische Verschlüsselungsverfahren realisieren.

Die abgeleiteten Klassen sind:

- `DES`
- `RC2`
- `TripleDES`
- `Rijndael`

Alle diese Klassen verfügen über einen Standardkonstruktor und erben u.a. die Methode `CreateEncryptor`. Sie sind aber ebenfalls abstrakt, und erst die davon abgeleiteten Klassen stellen die tatsächliche Implementierung bereit. Im Beispiel wird die Klasse `RijndaelManaged` verwendet, die von `Rijndael` abgeleitet ist.

CreateEncryptor-Methode

Die Methode `CreateEncryptor` ist in den von `SymmetricAlgorithm` abgeleiteten Klassen überschrieben. Die im Beispiel verwendete Version erstellt ein symmetrisches Verschlüsselungsobjekt mit dem angegebenen Schlüssel (`Key`) und Initialisierungsvektor (`IV`). Eine weitere überladene Variante finden Sie in der Online-Hilfe.


```
SymmetricAlgorithm.CreateEncryptor-Methode
public abstract ICryptoTransform CreateEncryptor(
   byte[] rgbKey,
   byte[] rgbIV
);
Parameter:
rgbKey
Der geheime Schlüssel für den symmetrischen Algorithmus.
rgbIV
Der Initialisierungsvektor für den symmetrischen Algorithmus.
Rückgabewert:
Ein symmetrisches Verschlüsselungsobjekt.
```

CreateDecryptor-Methode

Auch die Methode `CreateDecryptor` ist in den abgeleiteten Klassen überschrieben. Die im Beispiel verwendete Version erstellt ein symmetrisches Entschlüsselungsobjekt mit dem angegebenen Schlüssel (`Key`) und Initialisierungsvektor (`IV`). Eine weitere überladene Variante finden Sie in der Online-Hilfe.


```
SymmetricAlgorithm.CreateDecryptor-Methode
public abstract ICryptoTransform CreateDecryptor(
   byte[] rgbKey,
   byte[] rgbIV
);
```

Parameter:
`rgbKey`
Der geheime Schlüssel für den symmetrischen Algorithmus.
`rgbIV`
Der Initialisierungsvektor für den symmetrischen Algorithmus.
Rückgabewert:
Ein symmetrisches Entschlüsselungsobjekt.

Die Klasse CryptoStream

Die Klasse `CryptoStream` repräsentiert einen Datenstrom, in den Informationen in verschlüsselter Form geschrieben oder aus dem sie gelesen werden.

Konstruktor

Der Konstruktor bekommt als Parameter einen `stream`, ein Verschlüsselungsobjekt und einen Modus (`Read` oder `Write`) übergeben.

```
CryptoStream-Konstruktor
public CryptoStream(
   Stream stream,
   ICryptoTransform transform,
   CryptoStreamMode mode
);
```

Parameter:
`stream`
Der `Stream`, für den die kryptographische Transformation ausgeführt werden soll.
`transform`
Die kryptographische Transformation, die für den `Stream` ausgeführt werden soll.
`mode`
Einer der `CryptoStreamMode`-Werte.

Die CryptoStreamMode-Aufzählung

Der Parameter `mode` des `CryptoStream`-Konstruktors kann einen der `CryptoStreamMode`-Werte `Read` (Lesen) oder `Write` (Schreiben) annehmen.

Beispiel

Das Beispiel zu diesem Abschnitt finden Sie auf der Buch-CD im Verzeichnis *TeilIII\KryptoBsp*.

Um das Projekt zu erstellen, klicken Sie im Menü *Datei* auf den Befehl *Neu / Projekt*. Im Dialogfeld *Neues Projekt* wählen Sie unter *Projekttypen* den Eintrag *Visual C#-Projekte* und klicken im Feld *Vorlagen* auf *Windows-Anwendung*. Geben Sie dem Projekt einen Namen, z.B. *KryptoBsp* und wählen Sie im Feld *Speicherort* ein Verzeichnis.

Fügen Sie eine `TextBox` (*txtGeheim*) sowie zwei `Buttons` (*btnVerschlüsseln* und *btnEntschlüsseln*) in das Formular *Form1* ein. Setzen Sie die Eigenschaft `MultiLine` der `TextBox` auf *True* und entfernen Sie den Inhalt der `Text`-Eigenschaft.

Fügen Sie außerdem in die Datei *Form1.cs* die folgenden `using`-Direktiven ein:

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

Verschlüsseln

Beim Klicken auf die erste Schaltfläche im Formular *Form1* soll der Text in der `TextBox` verschlüsselt in eine Datei geschrieben werden:

```
private void btnVerschlüsseln_Click(object sender,
System.EventArgs e)
{
  FileStream FS = new FileStream ("Krypt.dat",
```

```
  FileMode.OpenOrCreate);
  RijndaelManaged RMCrypto = new RijndaelManaged();

  byte[] Key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
  0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
  0x16};
  byte[] IV = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
  0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
  0x16};

  CryptoStream CS = new CryptoStream(FS,
  RMCrypto.CreateEncryptor(Key, IV),
  CryptoStreamMode.Write);

  StreamWriter SW = new StreamWriter(CS);
  SW.Write(txtGeheim.Text);
  SW.Close();

  CS.Close();
  FS.Close();
□
```

Der Schlüssel ist hier der Einfachheit halber im Programm hartcodiert.

Entschlüsseln

Beim Klicken auf die zweite Schaltfläche im Formular *Form1* soll der verschlüsselte Text aus der Datei gelesen und in der `TextBox` angezeigt werden:

```
private void btnEntschlüsseln_Click(object sender,
System.EventArgs e)
{
  byte[] Key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
  0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
  0x16};
```

```
    byte[] IV = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
    0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
    0x16};

    FileStream FS = new FileStream ("Krypt.dat",
    FileMode.Open);
    RijndaelManaged RMCrypto = new RijndaelManaged();

    CryptoStream CS = new CryptoStream(FS,
    RMCrypto.CreateDecryptor(Key, IV),
    CryptoStreamMode.Read);

    StreamReader SReader = new StreamReader(CS);
    txtGeheim.Text = SReader.ReadToEnd();
    SReader.Close();

    CS.Close();
    FS.Close();
}
```

Asymmetrische Verfahren: die Klasse AsymmetricAlgorithm

Die Klasse `AsymmetricAlgorithm` ist die abstrakte Basisklasse für alle Klassen, die asymmetrische Verschlüsselungsverfahren realisieren.

Die abgeleiteten Klassen sind:

- ✔ `DSA`
- ✔ `RSA`

Beide Klassen sind abstrakt, und erst die davon abgeleiteten Klassen stellen die tatsächliche Implementierung bereit. Im Folgenden wird die Klasse `RSACryptoServiceProvider` betrachtet, die von `RSA` abgeleitet ist.

Die Klasse RSACryptoServiceProvider

Die Klasse `RSACryptoServiceProvider` ermöglicht das Ver- und Entschlüsseln von Daten mit dem RSA-Algorithmus.

Encrypt-Methode

Die Methode `Encrypt` verschlüsselt Daten.


```
RSACryptoServiceProvider.Encrypt-Methode
public byte[] Encrypt(
   byte[] rgb,
   bool fOAEP
);
```

Parameter:

`rgb`

Die zu verschlüsselnden Daten.

`fOAEP`

`true`, um direkte RSA-Verschlüsselung mit OAEP-Padding durchzuführen (nur auf einem Computer unter Windows 2000 verfügbar, auf dem das 128-Bit-Verschlüsselungspaket installiert ist), andernfalls `false`, um PKCS#1-Padding in der Version 1.5 zu verwenden.

Rückgabewert:

Die verschlüsselten Daten.

Decrypt-Methode

Die Methode `Decrypt` entschlüsselt Daten.


```
RSACryptoServiceProvider.Decrypt-Methode
public byte[] Decrypt(
   byte[] rgb,
   bool fOAEP
);
```

Parameter:
`rgb`
Die zu entschlüsselnden Daten.
`fOAEP`
`true`, um direkte RSA-Entschlüsselung mit OAEP-Padding durchzuführen, andernfalls `false`.
Rückgabewert:
Die entschlüsselten Daten, d.h. der ursprüngliche Klartext, der vor der Verschlüsselung vorlag.

Beispiel

Das Beispiel zu diesem Abschnitt finden Sie auf der Buch-CD im Verzeichnis *TeilIII\AsymBsp*.

Um das Projekt zu erstellen, klicken Sie im Menü *Datei* auf den Befehl *Neu / Projekt*. Im Dialogfeld *Neues Projekt* wählen Sie unter *Projekttypen* den Eintrag *Visual C#-Projekte* und klicken im Feld *Vorlagen* auf *Windows-Anwendung*. Geben Sie dem Projekt einen Namen, z.B. *AsymBsp* und wählen Sie im Feld *Speicherort* ein Verzeichnis.

Fügen Sie wieder eine `TextBox` und zwei `Buttons` (btnVerschlüsseln und btnEntschlüsseln) in das Formular Form1 ein. Setzen Sie die Eigenschaft `MultiLine` der `TextBox` auf True und entfernen Sie den Inhalt der `Text`-Eigenschaft.

Schreiben Sie außerdem in die Datei Form1.cs die folgende `using`-Direktive ein:

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

Fügen Sie nun ein Feld vom Typ `RSACryptoServiceProvider` in die Formularklasse ein und initialisieren Sie es:

```
private RSACryptoServiceProvider RSACrypto =
new RSACryptoServiceProvider();
```

Verschlüsseln

Beim Klicken auf die erste Schaltfläche im Formular *Form1* soll der Text in der `TextBox` verschlüsselt werden:

```
private void btnVerschlüsseln_Click(object sender,
System.EventArgs e)
{
  byte[] textBytes =
  Encoding.Unicode.GetBytes(textBox1.Text);
  byte[] cryptBytes =
  RSACrypto.Encrypt(textBytes, false);
  textBox1.Text =
  Encoding.Unicode.GetString(cryptBytes);
}
```

Der verschlüsselte Text wird wieder in der `TextBox` angezeigt. Da es sich bei den durch die Verschlüsselung erzeugten Bytefolgen natürlich überwiegend nicht um Textzeichen handelt, gibt es hier allerdings nicht viel zu sehen.

Entschlüsseln

Beim Klicken auf die zweite Schaltfläche im Formular *Form1* soll der Text wieder entschlüsselt werden:

```
private void btnEntschlüsseln_Click(object sender,
System.EventArgs e)
{
  byte[] cryptBytes =
  Encoding.Unicode.GetBytes(textBox1.Text);
  byte[] textBytes =
  RSACrypto.Decrypt(cryptBytes,false);
  textBox1.Text =
  Encoding.Unicode.GetString(textBytes);
}
```