André Obelink (MCSD) is werkzaam als technical manager en senior developer/architect bij AcouSoft Informatisering B.V.

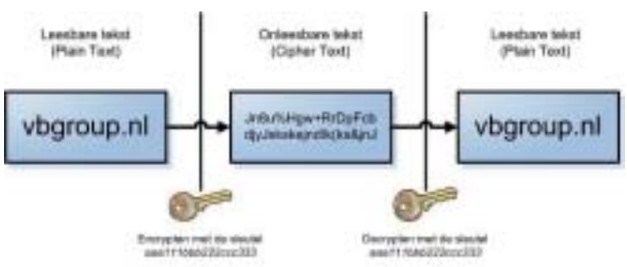
Introductie .NET cryptografie

"VERSLEUTELEN VAN DATA IS GEEN FEATURE, MAAR EEN REQUIREMENT."

Het Microsoft .NET platform en zijn ontwikkelomgevingen bieden je een eenvoudige manier om gedistribueerde applicaties te ontwikkelen. Steeds vaker zal een dergelijke applicatie bepaalde delen functionaliteit via het internet beschikbaar stellen. Het is dan ook belangrijker dan ooit om goed na te denken hoe je die applicatie en de bijbehorende data kunt beveiligen. Een onderdeel van die beveiliging kan het versleutelen of encrypten van data zijn. In dit artikel maak je kennis met de diverse mogelijkheden binnen het .NET Framework om data te encrypten en decrypten.

In het verleden kon je als ontwikkelaar vaak vertrouwen op de relatief veilige omgeving van het LAN. Je kon er min of meer vanuit gaan dat gebruikers die op het netwerk toegang hadden ook bepaalde bestanden mochten inzien of wijzigen. Met het instellen van rechten op de desbetreffende gebruikersaccounts was men goed in staat aan de meeste beveiligingswensen te voldoen. De besturingssystemen van vandaag de dag zijn gelukkig een stuk geavanceerder geworden, waarbij ook de toegang tot gedistribueerde (Internet) applicaties veel beter beveiligd kan worden. Toch blijven er situaties bestaan waarbij deze middelen niet voldoende toereikend zijn. In deze gevallen kun je besluiten om data te encrypten.

Microsoft heeft al een geruime tijd de Crypto API beschikbaar gesteld. Met deze interface kunnen ontwikkelaars deze techniek toepassen in eigen applicaties. De Crypto API is echter lastig te begrijpen en toe te passen. Ontwikkelaars zonder een gedegen C++ achtergrond raken snel het spoor bijster. Om ook de Visual Basic 6 gebruikers tegemoet te komen heeft Microsoft nog niet zo heel lang geleden de CAPICOM gereleased. Dit ActiveX-control fungeert als een soort 'wrapper' om de Crypto API. In dit object is slechts een deel van de hele API voorhanden. Men kan inderdaad veel eenvoudiger encrypten en decrypten, maar men heeft niet de beschikking over alle mogelijkheden die de oorspronkelijke API wel biedt. Indien men toch bepaalde functionaliteit wenst, moet men uitwijken naar C++ of controls van derden. Het .NET Framework daarentegen biedt je een aantal objecten waarmee je data op een eenvoudige wijze kan encrypten. Er wordt een groot aantal encryptiestandaarden ondersteund die alle op een eenduidige en consistente manier zijn geïmplementeerd. Een deel van de objecten zijn 'wrappers' om de Crypto API, de andere zijn compleet nieuwe 'managed' implementaties.



Afbeelding 1. 'Private Key' encryptie

Wat is cryptografie?

Cryptografie beveiligt de data tegen lezen en eventueel aanpassen wanneer je deze transporteert over zogenaamde 'insecure channels'. Hierbij kun je denken aan het versturen of distribueren van informatie over het internet of op bijvoorbeeld USB memorysticks. De data wordt geëncrypt met een bepaald algoritme en bij de ontvanger vervolgens weer gedecrypt. Mocht de informatie onverhoopt toch in handen komen van derden, dan kunnen zij hier in principe niets mee, omdat zij de encryptie-algoritmes en sleutels, of zogenaamde keys, niet kennen.

Cryptografie wordt gebruikt voor de volgende doeleinden:

* Het beschermen van vertrouwelijke gegevens, dus het onleesbaar maken van informatie.
* Het beschermen van de integriteit van de gegevens, dus ervoor zorgen dat men kan vaststellen dat in de tussentijd de informatie niet is gewijzigd.
* Het vaststellen van de herkomst van de data, men dient er namelijk zeker van te zijn dat de betreffende informatie afkomstig is van een bepaalde partij.

In wezen is het basisprincipe van encryptie niet heel ingewikkeld. Stel je wil de tekst 'vbgroup.nl' versleutelen met de sleutel '123'. Het versleutelen van data kan er dan als volgt uit zien. We tellen de ASCII-waarden van de verschillende letters op met de repeterende ASCII-waarden van de sleutel.

vbgroup.nl : 118,98,103,114,111,117,112,46,110,108
123 : 49, 50, 51

v	b	g	r	o	u	p	.	n	l
1	2	3	1	2	3	1	2	3	1
118	98	103	114	111	117	112	46	110	108
+49	+50	+51	+49	+50	+51	+49	+50	+51	+40
167	148	154	163	165	168	161	96	161	148

Je kunt je voorstellen dat wanneer men deze ASCII-codes weer omzet naar karakters, er een onleesbare tekst tevoorschijn komt. Door nu de gebruikte ASCII-waarden er repeterend af te trekken, kunnen we de originele tekst ook weer eenvoudig herleiden. Vanzelfsprekend zijn de encryptiestandaarden zoals die vandaag de dag worden gebruikt, veel ingewikkelder.
Om op een verantwoorde wijze aan de slag te gaan met encryptie en met de classes beschikbaar in het .NET Framework, dien je

wel inzicht te hebben in een aantal basisprincipes en kenmerken van encryptie. Wanneer je op de hoogte bent van de verschillende encryptie-typen, kun je beter een keuze maken welk type encryptie, specifiek voor jouw toepassing, het geschiktst is. Een bijkomend voordeel is dat je ook de desbetreffende classes en de bijbehorende code beter kan begrijpen.

Verschillende typen encryptie

Grofweg kun je een viertal verschillende typen encryptie onderscheiden: 'Private key', 'Public key', 'Hashing' en 'Digital signing'. Elk type wordt hieronder kort uitgelegd.

Private key encryption

Met een 'private key' encryption, ook wel bekend onder de term 'symmetric cryptography', wordt data versleuteld met één sleutel of zogenaamde 'key'. Doordat er maar één key is, moeten beide partijen deze key geheim houden; vandaar de naam 'private key'. Dezelfde key wordt dus gebruikt om zowel te encrypten als te decrypten. Deze wijze van encrypten is erg snel en is zeer geschikt voor het versleutelen van grotere hoeveelheden data. In afbeelding 1 is deze vorm schematisch weergegeven. Voorbeelden van private key encryption zijn Rijndael en DES.

Public key encryption

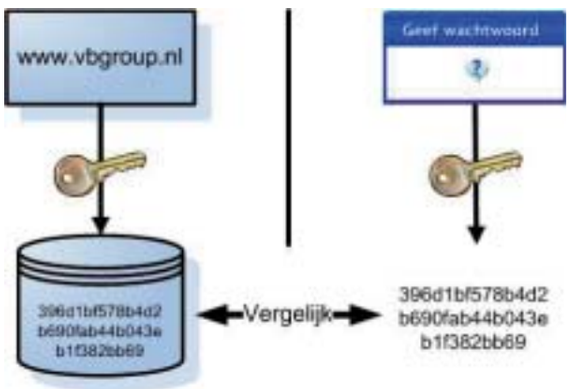
Met een 'public key' encryption, ook wel asymmetric cryptography genoemd, wordt data versleuteld met een tweetal keys. Een van de keys mag publiek worden gemaakt, terwijl de andere key geheim moet blijven. De ene partij versleutelt het bericht met de public key. De andere partij kan het vervolgens ontsleutelen met de private key. De public en private key zijn rekenkundig met elkaar verbonden. Deze wijze van encrypten wordt op dit moment gezien als de meest veilige. Public key encryption kost wel wat meer tijd dan private key encryption. Een voorbeeld van public key encryption is RSA.

Hashing

Deze wijze van encryption converteert data van een variabele lengte naar een vaste lengte, ook wel 'hash' genoemd. Er wordt wel eens gesteld dat hashes statistisch gezien uniek zijn. Wanneer er ook maar een letter anders is, komt het resultaat totaal niet meer overeen met de oorspronkelijk hash. Deze wijze wordt vaak toegepast in een soort eenrichtingsverkeer. Het is in principe niet mogelijk om een hash te ontsleutelen naar de oorspronkelijke informatie. Het wordt veel gebruikt voor controle van wachtwoorden en dergelijke. Men kan hier volstaan met het vergelijken van de hash, dus met het geëncrypte resultaat. In het eerste codevoorbeeld wordt deze vorm nader bekeken.

Cryptographic signing

Public key algoritmes kunnen ook worden gebruikt voor zogenaamde digitale handtekeningen (digital signatures). Hiermee kan de identiteit van de verzender worden vastgesteld, mits men



Afbeelding 2. Hashing van een wachtwoord

Afbeelding 3. Validatie van een wachtwoord

natuurlijk zijn of haar public key vertrouwt. Om de identiteit te verifiëren kan men de digitale handtekening vergelijken met de public key die je reeds eerder van de verzender hebt gekregen.

Aan de slag

Na deze wat theoretische inleiding is het de hoogste tijd eens te kijken hoe we deze kennis kunnen gebruiken in combinatie met het .NET Framework. De codevoorbeelden zijn geschreven in Visual Basic .NET, maar deze zijn eenvoudig te porten naar andere .NET-talen. Zoals reeds eerder beschreven wordt 'hashing' veel toegepast voor het controleren van wachtwoorden. Bij het invoeren van een wachtwoord zijn we in principe alleen geïnteresseerd of het ingevoerde wachtwoord overeenkomt met het reeds bekende wachtwoord. Wat nu het feitelijke wachtwoord is, is niet relevant; als beide maar identiek zijn.

Het .NET Framework herbergt in de namespace 'System.Security.Cryptography' een aantal hashing algoritmes. Een ervan is het 'Secure Hash Algorithm' (SHA). De SHA-provider maakt het mogelijk om tot 512-bit hashes te genereren. In grote lijnen komt het er op neer dat het object een byte-array met de te encrypten waarde accepteert en vervolgens ook een byte-array met de hash retourneert.
In het voorbeeld staat er op een form een Textbox (txtPassword) en een Button (btnValidatePassword). De gebruiker voert een wachtwoord in, valideert deze met behulp van de knop 'Controleer' en krijgt een melding of het ingevoerde wachtwoord wel of niet juist is.

In codevoorbeeld 1 kun je zien hoe we dit met behulp van de .NET-classes eenvoudig kunnen realiseren. Hier zijn de volgende stappen voor nodig:

* Converteer het ingevoerde wachtwoord naar een byte-array met de functie StringToBytes().
* Deze byte-array wordt door de SHA1Managed provider 'gehasht' naar een encrypted byte-array.
* Deze encrypted byte-array wordt vervolgens weer omgezet naar een string met de functie BytesToString().
* Deze string wordt vergeleken met een waarde die, in dit geval fictief, in de database is opgeslagen.

Door deze techniek wordt het wachtwoord 'www.vbgroup.nl' versleuteld tot: '396d1bf578b4d2b690fab44b043eb1f382bb69'

Je moet je er wel van bewust zijn dat deze vorm van encrypten ook gevaren met zich meebrengt. Op het internet zijn programma's beschikbaar die door middel van 'trial and error' de oorspronkelijke tekst proberen te achterhalen. Ze proberen allerlei woorden en woordcombinaties, net zolang tot ze dezelfde hash kunnen produceren. Hoewel de kans klein is dat ze een wachtwoord raden, is het toch aan te raden om wachtwoorden te voorzien van speciale tekens zoals bijvoorbeeld een underscore of procentteken. Wordt het wachtwoord via internet getransporteerd dan kun je natuurlijk deze hash ook nog weer eens encrypten. Hierdoor is de kans dat eventuele onbevoegden het wachtwoord kunnen ontcijferen wel heel erg klein geworden.

```
Imports System.Security.Cryptography
Imports System.Text

' Converteer een string naar een byte array
Private Function StringToBytes(ByVal strString As String) As
  Byte()
  Dim objUnicodeEncoding As New UnicodeEncoding

  Return objUnicodeEncoding.GetBytes(strString)

End Function

' Converteer een Byte array naar een (hexadecimale) string
Private Function BytesToString(ByVal bytBytes() As Byte) As
  String
  Dim bytByte As Byte
  Dim strBuffer As New StringBuilder(40)

  For Each bytByte In bytBytes
    strBuffer.AppendFormat(bytByte.ToString("x").ToLower)
  Next

  Return strBuffer.ToString

End Function

' Hash de meegegeven string naar een nieuwe encrypte string
' in dit voorbeeld wordt dus het wachtwoord versleuteld
Private Function GetHash(ByVal strStringToHash As String) As
  String

  Dim bytStringToHash() As Byte
  Dim bytHash() As Byte
  Dim objSHA1Managed As New SHA1Managed

  ' Converteer wachtwoord eerst naar een Byte array en
  ' encrypt deze naar een nieuwe 'gehashte' Byte array
  bytStringToHash = StringToBytes(strStringToHash)
  bytHash = objSHA1Managed.ComputeHash(bytStringToHash)

  ' Converteer 'gehashte' Byte array terug naar een string
  Return BytesToString(bytHash)

End Function

Private Sub btnValidatePassword_Click(ByVal sender As _
              System.Object, ByVal e As System.EventArgs) _
              Handles btnValidatePassword.Click

  Dim strPasswordStored As String

  ' Dit wachtwoord kan bijvoorbeeld opgeslagen zitten in een
  ' een database of een configbestand.
  strPasswordStored = "396d1bf578b4d2b690fab44b043eb1f382bb69"

  ' Het juiste wachtwoord is: 'www.vbgroup.nl'. Let op:
  ' de versleutelde waarden worden met elkaar vergeleken!
  If strPasswordStored = GetHash(txtPassword.Text) Then
    MessageBox.Show("Wachtwoord is juist.")
  Else
    MessageBox.Show("Wachtwoord is onjuist.")
  End If

End Sub
```

Codevoorbeeld 1

Een andere vorm van encryptie is 'private key encryption'. Informatie wordt zowel versleuteld en ontsleuteld met dezelfde sleutel. Zowel de private key, als de public key algoritmes, maken het dus mogelijk om gegevens onleesbaar te maken, deze te transporteren en vervolgens weer te ontsleutelen tot een leesbare tekst. Deze vormen van encryptie zijn dus uitermate geschikt om bestanden of andere data onleesbaar te maken.

Een voorbeeld van 'private encryption'

In het .NET Framework is een aantal classes beschikbaar waarmee je 'private key' encryptie relatief eenvoudig kunt implementeren. Al deze encryptiemethoden werken voor de ontwikkelaar in grote lijnen hetzelfde. Inhoudelijk wijken ze van elkaar af doordat er verschillende algoritmes gebruikt worden en doordat de maximale lengtes van de sleutels van elkaar verschillen. Hoe langer de sleutel, hoe meer combinaties er mogelijk zijn, hoe lastiger het is om als onbevoegde de data te kunnen ontsleutelen.

Het .NET Framework kent een aantal bekende private key encryption-classes. Voorbeelden zijn:
* Data Encryption Standard (DES) - 64 bit key
* Ron's Code 2 (RC2) - 40 tot 128 bit key
* TripleDES - 128 tot 192 bits key
* Rijndael - 128 tot 256-bits key

Je kunt van elke provider bepalen welke lengtes er zijn toegestaan. Gebruik hiervoor de LegalKeySize() method.

```
Dim objRijndael As New RijndaelManaged
Dim objKeySizes() As KeySizes = objRijndael.LegalKeySizes()

With objKeySizes(0)
  MessageBox.Show("Minimum lengte: " & .MinSize & ", " & _
   "Maximum lengte: " & .MaxSize & ", " & _
   "Tussenliggend verschil: " & .SkipSize)
End With
```

Codevoorbeeld 2

Het algoritme van een private key encryption vereist dat er een key van een bepaalde lengte is, maar dat er ook een zogenaamde 'Initialization Vector' (IV) beschikbaar is. Het voert in dit artikel te ver om uitgebreid in te gaan op de exacte functie van beide variabelen, maar het komt erop neer dat de combinatie van de key de versleuteling uniek maakt. De IV zorgt ervoor dat bepaalde blokken niet vaker terug komen in de versleutelde data. In codevoorbeeld 3 wordt een wachtwoord gehasht tot een byte-array en vervolgens als sleutel gebruikt. Tevens wordt een byte-array gecreëerd om te dienen als IV. De variabele mbytKey wordt gevuld door het toepassen van de reeds eerder besproken 'hash-techniek' op basis van het meegegeven wachtwoord.

```
Dim objSHA As New SHA256Managed
Dim strKey = "abcdefghijklmnop"

' Key : 32 Bytes = 256 bits
mbytKey = objSHA.ComputeHash(ConvertStringToBytes(strPassword))

' IV  : 16 Bytes = 128 bits
mbytIV = Encoding.ASCII.GetBytes(strKey)
```

Codevoorbeeld 3

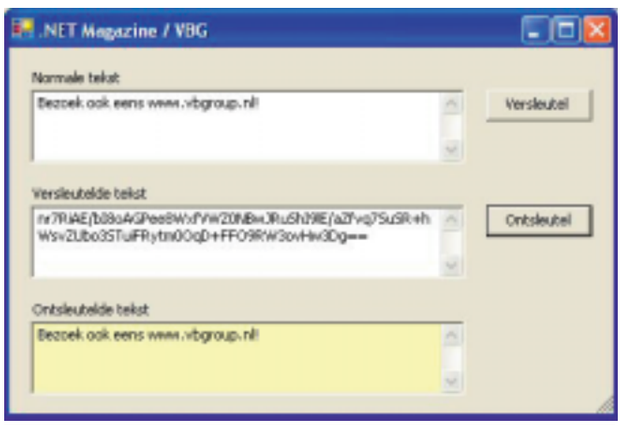
Onmisbaar voor het encrypten en decrypten zijn zogenaamde 'streams'. Een stream is soort abstracte weergave (in bytes) van bijvoorbeeld een bestand. Deze streams kunnen je helpen data in het geheugen op te slaan, maar bijvoorbeeld ook het schrijven naar een bestand. In dit voorbeeld (zie codevoorbeeld 4) worden er, los van enkele conversiefuncties, maar twee encryptiefuncties gebruikt. De eerste functie, EncryptText(), doet feitelijk niets meer

```
Public Function EncryptText(ByVal strPlainText As String) As String

   Dim stmEncryptStream As MemoryStream
   Dim bytPlainText() As Byte = ConvertStringToBytes(strPlainText)
   Dim stmPlainStream As New MemoryStream(bytPlainText)

   stmEncryptStream = EncryptStream(stmPlainStream)

   Return Convert.ToBase64String(stmEncryptStream.ToArray)

End Function


Private Function EncryptStream(ByVal stmPlainStream As _
                   MemoryStream) As MemoryStream

   Dim bytPlainStream() As Byte = stmPlainStream.ToArray()
   Dim objCryptoStream As CryptoStream
   Dim objEncryptedStream As New MemoryStream
   Dim objRijndaelManaged As New RijndaelManaged

   With objRijndaelManaged
      .Key = mbytKey
      .IV = mbytIV
   End With

   objCryptoStream = New CryptoStream(objEncryptedStream,
                   objRijndaelManaged.CreateEncryptor, _
                   CryptoStreamMode.Write)

   With objCryptoStream
      .Write(bytPlainStream, 0, stmPlainStream.Length)
      .FlushFinalBlock()
      .Close()
   End With

 Return objEncryptedStream

End Function
```

Codevoorbeeld 4

of minder dan de te versleutelen tekst in een stream te plaatsen en de functie EncryptStream() aan te roepen. De reden dat deze functies zijn opgesplitst, heeft er mee te maken dat men de functionaliteit eenvoudig zou kunnen hergebruiken om bijvoorbeeld een functie EncryptFile() te maken. Je maakt een andere stream (FileStream) aan en roept vervolgens EncryptStream() aan. Het encrypten werkt namelijk precies hetzelfde.

De nog leesbare data in stmPlainStream wordt met behulp van een CryptoStream versleuteld en weer teruggeven in stmEncrypted-



Afbeelding 4. De source-code in praktijk

```
Private Function DecryptStream(ByVal stmEncryptedStream As _
   MemoryStream) As MemoryStream

 Dim intBytesCountIn As Integer
 Dim objCryptoStream As CryptoStream
 Dim bytEncryptedStream(stmEncryptedStream.Length - 1) As Byte
 Dim objPlainStream As New MemoryStream
 Dim objRijndaelManaged As New RijndaelManaged

 With objRijndaelManaged
      .Key = mbytKey
      .IV = mbytIV
 End With

 objCryptoStream = New CryptoStream(stmEncryptedStream, _
      objRijndaelManaged.CreateDecryptor(), _
      CryptoStreamMode.Read)

 With objCryptoStream
    intBytesCountIn = .Read(bytEncryptedStream, 0, _
     stmEncryptedStream.Length)
     .Close()
 End With

 objPlainStream.Write(bytEncryptedStream, 0, intBytesCountIn)

 Return objPlainStream

End Function
```

Codevoorbeeld 5

Stream. Deze stream kan vervolgens worden geconverteerd naar een string en worden weggeschreven naar een bestand of database. Het ontsleutelen van data gaat ongeveer op een soortgelijke manier.

Het fundamentele verschil is het gebruik van de CreateEncryptor() en CreateDecryptor() method op het Rijndael-object.

To encrypt or not to encrypt?

Zodra jouw applicatie gevoelige informatie verwerkt of opslaat, dien je serieus na te denken of de data niet versleuteld moet worden. Het versleutelen van data met behulp van classes in het .NET Framework is vrij eenvoudig. De techniek mag je in ieder geval niet weerhouden om het toe te passen. Het grootste probleem is het aanmaken en versturen van de sleutels. Gebruik daarom bij voorkeur 'gehashte' sleutel in Initialization Vectors. In praktijk zie je vaak dat de IV wordt aangemaakt op basis van een hash van het gebruikte wachtwoord. In dit kader kan ook de 'public key' encryptie uitkomst bieden. Deze vorm is niet als een codevoorbeeld aan bod gekomen. Ik hoop echter dat de bovenstaande code je voldoende heeft uitgenodigd om hier zelf mee te experimenteren.

Nuttige internetadressen

- http://www.devx.com/security
- http://msdn.microsoft.com/security/
- http://www.mycrypto.net/encryption/
- http://www.microsoft.com/billgates/speeches/2004/02-24rsa.asp

André Obelink (MCSD) (http://www.acousoft.nl) is sinds 1995 als eindredacteur betrokken bij de Nederlandse Visual Basic Groep. (http://www.vbgroup.nl). Voor vragen of opmerkingen kun je hem mailen op andre@obelink.com.

De source-code van dit artikel is downloaden van de site van Microsoft Nederland (http://www.microsoft.nl/netmagazine) of van de website van de Visual Basic Groep (http://www.vbgroup.nl)