28.3.1 Anwendungen

Die folgenden Anwendungen der Komponente *gb.openssl* werden vorgestellt:

- Übersicht der auf dem System unterstützten Cipher-Algorithmen erzeugen und anzeigen
- Übersicht der auf dem System unterstützten Digest-Algorithmen erzeugen und anzeigen
- Ermittlung von Hash-Werten von Texten (Klasse Digest)
- Ermittlung von Hash-Werten von Texten (Klasse HMac)
- Ermittlung der Schlüssel- und InitVektoren-Länge ausgewählter Cipher-Algorithmen
- Verschlüsselung von Texten
- Entschlüsselung chiffrierter Texte
- Ermittlung einer Prüfsumme für eine frei wählbare Datei
- Projekt mit allen Klassen, die unter → Kapitel 28.3.0 beschrieben wurden

28.3.1.1 Übersicht der auf dem System unterstützten Cipher-Algorithmen

In der Eigenschaft *Cipher.List* wird ein Array zurückgegeben, dass alle von der OpenSSL-Bibliothek auf dem System unterstützten Cipher-Algorithmen enthält. Mit diesem Quelltext-Ausschnitt:

```
Dim sName As String
For Each sName In Cipher.List.Sort()
  Print sName; " | ";
Next
```

wurden die folgenden Algorithmen-Namen (sortiert) in der Konsole der IDE ausgegeben:

```
AES-128-CBC | AES-128-CBC | AES-128-CBC-HMAC-SHA1 | AES-128-CBC-HMAC-SHA1 | AES-128-CFB | AES-
128-CFB | AES-128-CFB1 | AES-128-CFB1 | AES-128-CFB8 | AES-128-CFB8 | AES-128-CTR | ...
```

Die Abbildung von einem Algorithmus-Namen auf den verwendeten Algorithmus wird durch OpenSSL intern geregelt. Die Komponente gb.openssl hat darauf keinen Einfluss. Beachten Sie, dass die Liste auf Ihrem System andere Elemente enthalten kann.

28.3.1.2 Übersicht der auf dem System unterstützten Digest-Algorithmen

In der Eigenschaft *Digest.List* wird ein String-Array zurückgegeben, dass alle von der OpenSSL-Bibliothek auf dem System unterstützten Hash-Algorithmen enthält. Mit dem folgenden Quelltext-Ausschnitt wurde die sortierte Liste der Algorithmen-Namen in der Konsole der IDE ausgegeben:

```
Dim sName As String
For Each sName In Digest.List.Sort()
  Print sName; " | ";
Next
```

```
DSA | DSA-SHA | MD4 | MD5 | RIPEMD160 | SHA | SHA1 ... | SHA512 | ecdsa-with-SHA1 | whirlpool
```

28.3.1.3 Ermittlung von Hash-Werten der Klasse Digest

Für alle Beispiele wird folgender kurze Text verwendet:

```
sOriginalText = "RSA ist ein asymmetrisches kryptographisches Verfahren."
```

Mit dem "SHA256"-Algorithmus ergab sich

```
Print Digest["SHA256"](sOriginalText)
Print Base64$(Digest["SHA256"](sOriginalText))            *
Print Base64$(Digest["SHA256"].Hash(sOriginalText))       **
```

ein Hash-Wert (Message Digest oder Digest) für den oben stehenden Text im binären Rohdaten-Format und darunter im (lesbaren) Base64-Format:

```
???a???R^X5????6???~??????^T?[?r?
vrKYYbfp41IYNeDd2to2s6DIfpObn6SuhBSRW95y8wo=
```

In den beiden Fällen * und ** wird intern die gleiche Funktion aufgerufen. Die Syntax ist absolut gleichwertig.

28.3.1.4 Ermittlung von Hash-Werten der Klasse HMac

Für die Ermittlung der Prüfsumme (Message Authentication Code (MAC)) für den Beispiel-Text wurde folgender Quelltext-Ausschnitt genutzt:

```
Dim sKey, sData, iMethod As String

sKey = "a1Bc+d2D*"
sData = sOriginalText
iMethod = HMac.RipeMD160 ' Konstante der Klasse HMac (Typ Integer)

Print Base64$(HMac(sKey, sData, iMethod))
```

In der Konsole wurde ein lesbarer MAC im Base64-Format angezeigt:

```
1IK0gQGM976LoRt0L7BME2ipdLU=
```

28.3.1.5 Ermittlung der Schlüssel- und InitVektoren-Länge ausgewählter Cipher-Algorithmen

Die genaue Kenntnis der Schlüssel-Länge und der Länge der InitVektoren für ausgewählte Cipher-Algorithmen ist *notwendige* Voraussetzung sowohl für die Verschlüsselung als auch für die Entschlüsselung mit den Methoden Encrypt() und Decrypt().

Der kommentierte Quelltext-Ausschnitt zur Ermittlung und Anzeige der Schlüssel-Länge sowie der Länge der InitVektoren ist einem Projekt zur Demonstration der Arbeit mit den Klassen der Komponente *gb.openssl* entnommen. Das Projekt-Archiv finden Sie im Download-Bereich.

```
[1] Public aCipherList As String[]
[2] Public Sub btnShowCipherKeyAndIVLength_Click()
[3]    Dim i As Integer
[4]
[5]    txaText.Clear
[6]    Wait
[7]    If Not aCipherList Then
[8]       aCipherList = RemoveMultiple(Cipher.List)
[9]    Endif
[10]   For i = 0 To aCipherList.Max
[11]     txaText.Text &= aCipherList[i] & " : IvLength = " & Cipher[aCipherList[i]].IvLength
[12]     txaText.Text &= "   |  KeyLength = " & Cipher[aCipherList[i]].KeyLength & gb.NewLine
[13]   Next
[14] End ' btnShowCipherKeyAndIVLength_Click()
```

Kommentar:

- In der Zeile 8 wird das aCipherList-Array – wenn es nicht existiert – mit den Elementen von Cipher.List (Datentyp String-Array) gefüllt, nachdem mögliche Dubletten entfernt wurden. Ab Gambas-Revision 6684 werden die Listen sortiert und redundanzfrei ausgegeben. Die Funktion RemoveMultiple() kann dann entfallen.
- In einer TextArea werden die auf dem System vorhandenen Cipher-Algorithmen mit den korrespondierenden Längen für InitVektor (IvLength) und Schlüssel (KeyLength) angezeigt [Zeilen 11 bis 14].

Hier sehen Sie einen Ausschnitt aus der Anzeige, wie Sie beim Autor auf seinem System zu sehen war:

```
AES-256-CTR   : IvLength = 16   |  KeyLength = 32
AES-256-ECB   : IvLength = 0    |  KeyLength = 32
AES-256-XTS   : IvLength = 16   |  KeyLength = 64
BF-CBC : IvLength = 8   |  KeyLength = 16
BF-CFB : IvLength = 8   |  KeyLength = 16
BF-ECB : IvLength = 0   |  KeyLength = 16
BF-OFB : IvLength = 8   |  KeyLength = 16
CAMELLIA-128-CBC : IvLength = 16   |  KeyLength = 16
```

28.3.1.6 Verschlüsselung von Texten

Mit der Klasse *Cipher* kann man Texte nach zwei unterschiedlichen Ansätzen verschlüsseln:

(A1) Einsatz von Passwort und *optionalem* Salt (Methode: EncryptSalted())
(A2) Einsatz eines Schlüssels und eines Initialisierungsvektors (Methode: Encrypt())

- Bei beiden Verfahren müssen Sie sich für die Anwendung eines Cipher-Algorithmus entscheiden, der auf dem System zur Verfügung steht.
- Das Passwort und den Wert von Salt können Sie beim Ansatz A1 frei festlegen. Der Salt muss allerdings eine Länge von genau 8 Bytes haben. Sonst wird durch Anfügen von Null-Bytes oder durch Trunkieren diese Länge erzwungen. Wird kein Salt übergeben, dann wird eine zufällige Sequenz verwendet.
- Für den Schlüssel und den Initialisierungsvektors beim Ansatz A2 müssen Sie die für den gewählten Cipher-Algorithmus vorgeschriebenen Längen beachten – sind aber sonst frei in der Wahl der verwendeten Zeichen. Der Initialisierungsvektor hat wesentlichen Einfluss auf die Güte der Verschlüsselung.

Ansatz A1 – Einsatz von Passwort und optionalem Salt

```
[1] Public Sub btnCipherSalt_TextToFile_Click()
[2]   Dim sPlainText, sCipherText, sPassword, sSalt As String
[3]
[4]   Reset()
[5]   sAlgorithmus = "aes-256-cfb" ' Festlegung des Chiffrier-Algorithmus
[6]   sPlainText = sOriginalText
[7]   sPassword = GetPassword() ' Passwort aus einem Passwort-Formular
[8]   sSalt = "123abc#*"     ' Frei definierte Zeichenkette (8Byte)
[9]
[10]  sCipherText = Cipher[sAlgorithmus].EncryptSalted(sPlainText, sPassword, sSalt)
[11]
[12]  File.Save(csPath, sCipherText)
[13]
[14]  ' Print Cipher["aes-256-cfb"].EncryptSalted(sPlainText, sPassword, sSalt)
[15]  ' Print Base64(Cipher["aes-256-cfb"].EncryptSalted(sPlainText, sPassword, sSalt))
[16]
[17]End ' btnCipherSalt_TextToFile_Click()
```

Kommentar:

- In der Zeile 10 wird der Klartext unter Festlegung der EncryptSalted()-Methode der Klasse Cipher sowie des Chiffrier-Algorithmus, des Passworts und des Wertes von Salt chiffriert.
- Der chiffrierte Text wird in einer Datei [Zeile 12] abgespeichert, deren Pfad in der Variablen *csPath* gespeichert wurde.
- Die Zeilen 14 und 15 dienten bei der Erprobung des Projektes für Kontrollzwecke.
- Unter dem diesem Link: *http://de.wikipedia.org/wiki/Salt_(Kryptologie)* finden Sie eine gute Beschreibung zum Thema *Salt*.

- Das zu verwendende Passwort wird über ein Passwort-Formular interaktiv erfasst:

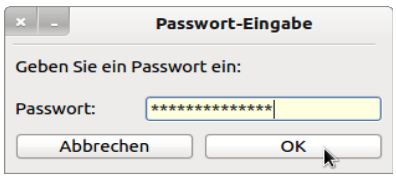

Abbildung 28.3.1.6.1: Passwort-Dialog

Ansatz A2 – Einsatz eines Schlüssels und eines Initialisierungsvektors

- Die Verschlüsselung eines Textes nach dem Ansatz A2 erfordert einen höheren Aufwand gegenüber A1, denn Sie müssen sich nach der Entscheidung für den Verschlüsselungsalgorithmus um die Ermittlung der Länge des zu diesem Algorithmus gehörenden Schlüssels und um die Länge des entsprechenden Initialisierungsvektors kümmern (→ Kapitel 28.3.1.5).

- Außerdem müssen Sie explizit mit den drei Eigenschaften (Cipher, Key und InitVector) Klasse CipherText arbeiten, was beim Ansatz A1 nicht notwendig ist!

```
[1] Public Sub btnCipherKIV_TextToFile_Click()
[2]   Dim hCipherText As CipherText
[3]   Dim sPlain, sKey, sInitVector, sAlgorithmus As String
[4]   Dim iInitVector, iKey As Integer
[5]
[6]   Reset()
[7]   sAlgorithmus = "AES-128-XTS" ' Cipher-Algorithmus
[8] ' iInitVector = Cipher[sAlgorithmus].IvLength
[9] ' iKey = Cipher[sAlgorithmus].KeyLength
[10]' Print iInitVector, iKey
[11]
[12]  sPlain = sOriginalText ' Klartext
[13]  sInitVector = "0123456789vector" ' Länge = 16 für "AES-128-XTS"
[14]  sKey = "0123456789abcdef0123456789abcdef" ' Länge = 32 für "AES-128-XTS"
[15]
[16]  hCipherText = Cipher[sAlgorithmus].Encrypt(sPlain, sKey, sInitVector)
[17]  File.Save(ckivPath, hCipherText.Cipher) ' Speichern des (binären) Geheimtextes in einer Datei
[18]  txaText.Text = "GEHEIMTEXT IM BASE64-FORMAT =\n" & Base64$(hCipherText.Cipher) ' Lesbarer Geheimtext
[19]
[20]End ' btnCipherKIV_TextToFile_Click()
```

Kommentar:

- Beachten Sie den Datentyp 'CipherText' der Variable hCipherText!
- In der Zeile 7 wird der Chiffrier-Algorithmus festgelegt, zu dem Sie die entsprechenden Längen von InitVektor und Schlüssel kennen müssen.
- Die Zeile 13 zeigt, wie ein InitVektor der Länge 16 definiert wird.
- Von doppelter Länge ist der Schlüssel, der in der Zeile 14 festgelegt wird.
- In der Zeile 16 wird ein CipherText-Objekt als Rückgabewert von *Cipher[...].Encrypt(..)* generiert.
- Gespeichert wird von diesem Objekt nur der Wert der Eigenschaft *hCipherText.Cipher* in Zeile 17, weil nur dieser den verschlüsselten Geheimtext enthält. Der Datei-Pfad ist in der Variablen *ckivPath* gespeichert.
- In einer TextArea können Sie sich den Geheimtext im Base64-Format in lesbarer Form ansehen.

28.3.1.7 Entschlüsselung von Texten

Die Entschlüsselung von Texten, die als Geheimtext in einer Datei gespeichert wurden, wird von den beiden Methoden Cipher[...].Decrypt(..) und Cipher[...].DecryptSalted(..) geleistet. Auch das Entschlüsseln folgt zwei unterschiedlichen Ansätzen:

(B1) Einsatz Passwort
(B2) Einsatz Schlüssel und Initialisierungsvektor

Ansatz B1

Es wird davon ausgegangen, dass eine Datei mit einem Geheimtext vorliegt und der Datei-Pfad sowie das notwendige Passwort und der benutzte Chiffrier-Algorithmus bekannt sind.

```
[1] Public Sub btnCipherSalt_FileToText_Click()
[2]   Dim sPassword, sPlainText, sAlgorithmus As String
[3]
[4]   If Not Exist(csPath) Then Return
[5]   txaText.Clear
[6]   sAlgorithmus = "aes-256-cfb"
[7]   sPassword = GetPassword() ' Passwort aus einem Passwort-Formular wie bei A1
[8]
[9]   sPlainText = Cipher[sAlgorithmus].DecryptSalted(File.Load(csPath), sPassword)
[10]  txaText.Text = sPlainText ' Anzeige Klartext
[11]
[12]End ' btnCipherSalt_FileToText_Click()
```

Kommentar:

- Als Chiffre-Algorithmus muss der gleiche verwendet werden, mit dem verschlüsselt wurde.
- Das erforderliche Passwort [Zeile 7] wird über ein Passwort-Formular eingelesen.
- Die Methode *Cipher[algorithmus].DecryptSalted(geheimtext, password)* liefert als Funktionswert einen String, der sofort verarbeitet oder angezeigt werden kann.

Ansatz B2

Für das Entschlüsseln über die Methode Cipher[..].Decrypt(..) gelten ähnliche Betrachtungen zum Algorithmus, zum InitVektor und zum Schlüssel. Es wird gefordert, dass eine Datei mit einem Geheimtext vorliegt und der Datei-Pfad sowie der benutzte Chiffrier-Algorithmus und der zu verwendende InitVektor und der Schlüssel bekannt sind.

```
[1] Public Sub btnCipherKIV_FileToText_Click()
[2]   Dim hCipherText As CipherText
[3]   Dim sCipher, sKey, sInitVector As String
[4]
[5]   If Not Exist(ckivPath) Then Return
[6]   txaText.Clear
[7]   Wait 1
[8]
[9]   sCipher = File.Load(ckivPath)
[10]  sKey = "0123456789abcdef0123456789abcdef" ' Länge = 32
[11]  sInitVector = "0123456789vector" ' Länge = 16
[12]
[13]  hCipherText = New CipherText(sCipher, sKey, sInitVector)
[14]  txaText.Text = Cipher["AES-128-XTS"].Decrypt(hCipherText)
[15]
[16]End ' btnCipherKIV_FileToText_Click()
```

Kommentar:

- Die Variable sCipher nimmt den Geheimtext auf, der aus der Geheimtext-Datei stammt.
- In der Zeile 13 wird ein CipherText-Objekt generiert, dem der Geheimtext, der Schlüssel und der InitVektor als Ininialisierungswerte übergeben werden.
- Im neuen CipherText-Objekt sind alle für die Entschlüsselung benötigten Daten in einem Objekt gebündelt.
- Die Methode *Cipher[algorithmus].Decrypt(ciphertext)* liefert als Funktionswert in der Zeile 14 einen String mit dem Klartext, der sofort in einer TextArea angezeigt wird.

Hinweise:

- Für eine *Demonstration der Klasse Cipher* mag es im Zusammenhang mit den zwei Methoden Cipher[..].Encrypt(..) und Cipher[..].Decrypt(..) unkritisch sein, den InitVektor und den Schlüssel im Quelltext abzuspeichern.
- Generell muss die Empfehlung ernst genommen werden, den InitVektor und den Schlüssel im praktischen Einsatz nicht im Quelltext abzuspeichern, sondern in geeigneter Form interaktiv zur Laufzeit in geeigneter Weise einzulesen!

28.3.1.8 Ermittlung der Prüfsumme für eine frei wählbare Datei

Der Quelltext für die Ermittlung der MD5-Prüfsumme für eine frei wählbare Datei enthält

- einen Dialog zur Datei-Auswahl,
- die Berechnung der Prüfsumme nach einem fest *vorgegebenen* Hash-Algorithmus und
- die Anzeige der Prüfsumme in drei unterschiedlichen Formaten → Abbildung 28.3.1.9.1.

```
[1] Public Sub btnDigestFromFile_Click()
[2]   Dim sPath As String
[3]   Dim vRawData As Variant
[4]
[5]   Dialog.Title = "Wählen Sie eine Datei aus!"
[6]   Dialog.Path = sFilePath
[7]   If Dialog.OpenFile() Then Return
```

```
[8]
[9]   sPath = Dialog.Path
[10]
[11]  vRawData = Digest["MD5"].Hash(File.Load(sPath))
[12]
[13]  txaText.Clear
[14]  txaText.Text = "MD5-Prüfsumme für die Datei '" & File.Name(Dialog.Path) & "'\n\n"
[15]  txaText.Text &= "Base64-Format:" & gb.NewLine
[16]  txaText.Text &= Base64$(vRawData) & gb.NewLine & gb.NewLine
[17]  txaText.Text &= "Format: hexadezimal:" & gb.NewLine
[18]  txaText.Text &= StringToHex(vRawData) & gb.NewLine & gb.NewLine ' Funktion StringToHex(..)
[19]  txaText.Text &= "Format: binär:" & gb.NewLine
[20]  txaText.Text &= vRawData
[21]
[22]End ' btnDigestFromFile_Click()
```

28.3.1.9 Projekt zur Demonstration der Komponente gb.openssl

Im Projekt werden alle Klassen der Komponente *gb.openssl* sowie deren Methoden und Eigenschaften eingesetzt. Hier wird Ihnen nur die GUI vorgestellt:

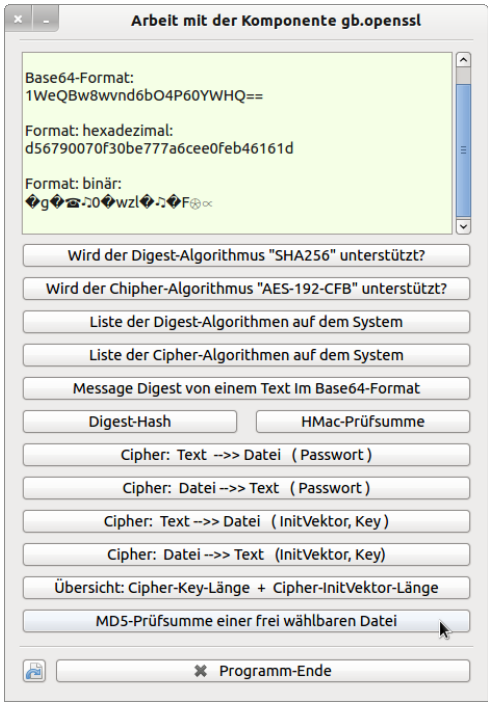

Abbildung 28.3.1.9.1: Projekt-Oberfläche - MD5-Prüfsumme

Das gut dokumentierte Projekt-Archiv finden Sie im Download-Bereich.