CryptLib

Security Toolkit

API Handbuch

Version 2.4.2

CryptLib Programmierhandbuch

Copyright

Warenzeichen

Inhaltsverzeichnis

Kapitel

1

Einführung

'Kryptographie ist die Wissenschaft, die sich mit der Absicherung von Nachrichten beschäftigt' (Zitat aus 'Angewandte Kryptographie' von Bruce Schneier, Seite 1, Addison-Wesley GmbH, 1996).

Die Entwicklung neuer Verfahren und Methoden zur Absicherung von Nachrichten wurde in der Vergangenheit vor allem durch Anforderungen im militärischen Bereich vorangetrieben. Schon die Römer setzten simple Chiffrierverfahren ein, um den Klartext von Nachrichten vor ihren Feinden zu verbergen. Die so genannte 'Caesar-Chiffrierung' ist ein Beispiel dafür.

In den letzten Jahren haben sich kryptographische Anwendungen ihren Weg in eine Vielzahl von Geschäftsbereichen gebahnt. Einer der Hauptgründe hierfür ist die Tatsache, dass die weltweite Vernetzung von Computern durch das Internet und die damit entstandenen neuen Möglichkeiten zur Kommunikation natürlich Fragen bezüglich der Sicherheit von übermittelten Informationen aufwerfen. Dabei spielen sowohl persönliche Interessen, wie sie etwa beim Online-Banking, als auch geschäftliche Interessen, wie sie etwa beim Abwickeln von Transaktionen zwischen Geschäftspartnern über das Internet auftreten, eine Rolle.

Der zunehmende Bedarf an kryptographischen Verfahren hat dazu geführt, dass die Entwicklungen auf diesem Gebiet stark vorangetrieben wurden. Die Tatsache, dass die breite Öffentlichkeit heute Zugang zu geprüften, sicheren und einfach anzuwendenden kryptographischen Verfahren hat, kann durchaus als Resultat dieser Entwicklung angesehen werden.

Private Computeranwender können Daten heute problemlos und komfortabel mit als sicher geltenden Methoden schützen. 'Sicher' bedeutet in diesem Zusammenhang, dass selbst der Einsatz von Rechenleistung, die zurzeit als unrealistisch groß angesehen werden muss, keine systematische Möglichkeit bietet, den Chiffriertext in angemessener Zeit in den Klartext zurück zu überführen. Dies kann nur gelingen, wenn eine bestimmte geheime Information bekannt ist, die als 'Schlüssel' bezeichnet wird.

Im Laufe der Zeit haben sich einige Verfahren als Standards etabliert. Dies liegt daran, dass diese Verfahren öffentlich dokumentiert und daher gut getestet und auf Sicherheit geprüft werden konnten. Dabei ist festzustellen, dass die Sicherheit dieser Verfahren allein auf der Wahl der verwendeten Schlüssel basiert. Die Kenntnis der benutzten Algorithmen schwächt die Sicherheit dieser Verfahren nicht.

Mit CryptLib bietet die XPS Software GmbH Programmierern eine Bibliothek mit standardisierten Verfahren aus der Kryptographie zur Einbindung in selbst entwickelte Applikationen an. CryptLib ist für die Betriebssysteme Win32, Linux, OS/2, OS/400, IBM iSeries, VSE/ESA, MVS/ESA, OS/390 und IBM zSeries verfügbar. Die Funktionalität erstreckt sich von Verfahren zur Hashwertbildung, symmetrischer und asymmetrischer Verschlüsselung über die Verarbeitung von X.509 Zertifikaten, die Erstellung und Prüfung digitaler Signaturen bis hin zur Unterstützung der Public Key Cryptography Standards PKCS#7 (S/MIME) und PKCS#12 (public Key).

Kapitel

2

Schlüsselgenerierung

Allgemein

Zufallszahlen spielen in der Kryptographie eine große Rolle. Die Erzeugung eines neuen Schlüssels startet sowohl bei symmetrischer als auch bei asymmetrischer Verschlüsselung mit dem Generieren einer Zufallszahl. Ist diese Zufallszahl vorhersehbar, kann mit dem entsprechenden Verfahren auch der Schlüssel berechnet werden. Von der Geheimhaltung der Schlüssel hängt das ganze Sicherheitssystem ab. Aus diesem Grund ist es möglich, durch den Parameter *seed* einen eigenen Initialisierungswert vorzugeben.

Mit der Funktion **GenerateRSAKey** kann ein RSA Public-/Private Schlüsselpaar erzeugt werden. Die Funktion **GenerateKey** kann zur Erzeugung von zufälligen Schlüsseln, Initialisierungs-Vektoren (iv) oder sonstigen Zufallswerten benutzt werden.

Funktionen

GenerateKey

Ein zufälliger Schlüssel für die symmetrische Ver- bzw. Entschlüsselung wird generiert.

Syntax	**void GenerateKey(BYTE *key, int keylength, BYTE *seed, int seedlength);**	
Returncode	Keiner.	
Parameter	**Beschreibung**	**Verwendung**
key	Speicheradresse des Rückgabebereiches für den zu erzeugenden symmetrischen Schlüssel.	Ausgabe
keylength	Länge des zu erzeugenden Schlüssels.	Eingabe
seed	Speicheradresse der variablen Daten, die in die Schlüsselgenerierung mit einfließen.	Eingabe
seedlength	Länge der variablen Daten.	Eingabe

Beispiel:

```
#include "XPSCRYPT.H"

int main(void)
{
  BYTE block[32] = "XPS Software GmbH, Haar/Muenchen";
  BYTE seed1[16] = {0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01,0x00};
  BYTE seed2[16] = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f};
  BYTE key[32]   = {0};
  BYTE iv[16]    = {0};

  CIPHERCTX ctx     = {0};
  int       iOutlen = 0;
  int       rc      = 0;

  GenerateKey( key, sizeof(key), seed1, sizeof(seed1) );
```

```
  GenerateKey( iv,  sizeof(iv),  seed2, sizeof(seed2) );

  ctx.algorithm = AES;
  ctx.mode      = CBC;
  ctx.keylength = 256;
  ctx.key       = (BYTE *)key;
  ctx.iv        = (BYTE *)iv;

  /* encryption */
  InitCTX( &ctx );
  iOutlen = Encrypt( &ctx, block, sizeof(block), block, sizeof(block) );
  CleanupCTX( &ctx );
  return 0;
}
```

GenerateRSAKey

Ein zufälliges RSA Public-/Private-Key Paar wird generiert.

Syntax	**void GenerateRSAKey(RSA_PRIVATE_KEY *privkey, RSA_PUBLIC_KEY *pubkey, BYTE *seed, int seedlength, int keylength);**	
Returncode	Keiner.	
Parameter	**Beschreibung**	**Verwendung**
privkey	Speicheradresse des Rückgabebereiches für den zu erzeugenden privaten RSA Schlüssel.	Ausgabe
pubkey	Speicheradresse des Rückgabebereiches für den zu erzeugenden öffentlichen RSA Schlüssel.	Ausgabe
seed	Speicheradresse der variablen Daten, die in die Schlüsselgenerierung mit einfließen.	Eingabe
seedlength	Länge der variablen Daten.	Eingabe
keylength	Länge des zu erzeugenden Schlüssels (maximal 4096).	Eingabe

Beispiel:

```
#include "XPSCRYPT.H"

int main(void)
{
  BYTE  block[32]       = "XPS Software GmbH, Haar/Muenchen";
  BYTE  KeyRandom[16]   = {0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01,0x00};
  BYTE *output;

  RSA_PRIVATE_KEY rsa_priv;
  RSA_PUBLIC_KEY  rsa_pub;
  CIPHERCTX       ctx     = {0};
  int             iOutlen = 0;
  int             rc      = 0;

  GenerateRSAKey( &rsa_priv, &rsa_pub, KeyRandom, sizeof(KeyRandom), 1024 );

  ctx.algorithm    = RSA;
  ctx.mode         = PUBLIC;
  ctx.key          = (BYTE *)&rsa_pub;

  /* rsa encryption        */
  InitCTX( &ctx );
  iOutlen = GetResultLength( &ctx, sizeof(block) );
  output  = malloc( iOutlen );
  iOutlen = Encrypt( &ctx, block, sizeof(block), output, iOutlen );
  CleanupCTX( &ctx );
  free( output );
  return 0;
}
```

Kapitel

3

Verschlüsselung

Allgemein

Man unterscheidet zwei Arten von Verschlüsselungsverfahren:

Symmetrische Verfahren

Bei symmetrischen Verfahren wird zum Verschlüsseln und zum Entschlüsseln der gleiche Schlüssel verwendet. Daraus folgt, dass Sender und Empfänger den gleichen Schlüssel verwenden müssen. Verwendet man den Operationsmodus CBC, so benötigt der Chiffre zusätzlich einen so genannten Initialisierungsvektor (iv). Der Initialisierungsvektor ist immer genau so groß wie die Blocklänge des Chiffre (DES, TripleDES, RC2, RC4, Blowfish = 8 Byte, AES = 16 Byte).

Asymmetrische Verfahren (PublicKey-Verfahren)

Sind beide Schlüssel verschieden, spricht man von einem asymmetrischen Verfahren. Ein Schlüssel wird zum Verschlüsseln, der andere zum Entschlüsseln verwendet. Beide Schlüssel werden gemeinsam bei der Schlüsselgenerierung erzeugt und es ist unmöglich, von einem der beiden Schlüssel auf den anderen zu schließen. Aus diesem Grund ist es zulässig, einen der beiden Schlüssel öffentlich bekannt zu geben (PublicKey). Mit diesem öffentlichen Schlüssel ist man dazu in der Lage, dem Besitzer des Schlüsselpaares eine verschlüsselte Nachricht zu senden. Dieser verwendet dann den geheimen Schlüssel (PrivateKey) um die Nachricht zu entschlüsseln.

CryptLib stellt eine Reihe von symmetrischen Verschlüsselungsroutinen zur Verfügung (**AES**, **DES**, **TripleDES**, **RC2**, **RC4**, **Blowfish**). Außerdem ist die asymmetrische Verschlüsselungsroutine **RSA** Public-/Private-Key implementiert. Die Art der gewünschten Ver-/Entschlüsselung wird beim Initialisieren der Kryptokontexts festgelegt (Funktion *InitCTX*). Danach ist nur noch eine der für jede Verschlüsselungsart identischen Routinen *Encrypt* bzw. *Decrypt* aufzurufen, um die Ver-/Entschlüsselung durchzuführen.

Funktionen

InitCTX

Initialisierung des Kryptokontexts.

Syntax	**int InitCTX(PCIPHERCTX ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Ver-/Entschlüsseln benötigten Kontexts.	Eingabe

Felder	Beschreibung
ctx.algorithm	Algorithmus zum Chiffrieren der Daten. Folgende Algorithmen werden unterstützt: AES Advanced Encryption Standard (Rijndael) DES Data Encryption Standard mit ctx.keylength = 56 TripleDES EDE2 Data Encryption Standard mit ctx.keylength = 112 TripleDES EDE3 Data Encryption Standard mit ctx.keylength = 168 RC2 Rivest Cipher No. 2 RC4 Rivest Cipher No. 4 Blowfish Schneier RSA Public-/Private-Key Verfahren von Rivest, Shamir und Adleman
ctx.mode	Verarbeitungsmodus. Unterstützte Modi bei symmetrischer Verschlüsselung (AES, DES, RC2, RC4, Blowfish): ECB Electronic-Codebook-Mode CBC Cipher-Block-Chaining Unterstützte Modi bei asymmetrischer Verschlüsselung (RSA): PUBLIC (Public-Key Ver-/Entschlüsselung) unter *ctx.key* muss die Adresse einer RSA_PUBLIC_KEY Struktur übergeben werden PRIVATE (Private-Key Ver-/Entschlüsselung) unter *ctx.key* muss die Adresse einer RSA_PRIVATE_KEY Struktur übergeben werden
ctx.key	Speicheradresse des Schlüssels.
ctx.keylength	Länge des Schlüssels. Folgende Schlüssellängen werden unterstützt: AES 128, 192, 256 DES 56, 112, 168 RC2 40, 64, 128 RC4 40, 64, 128 Blowfish 128 RSA 512, 1024, 2048, 4096
ctx.iv	Speicheradresse des Initialisierungs-Vektors (iv) bei symmetrischer Verschlüsselung.

Encrypt

Verschlüsseln von Daten. Die Art der Verschlüsselung ist abhängig vom Parameter *ctx.algorithm* bei der Funktion InitCTX.

Syntax	**int Encrypt(PCIPHERCTX ctx, BYTE *input, int inputlength, BYTE *output, int outputlength);**	
Returncode	Länge der verschlüsselten Daten oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Verschlüsseln benötigten Kontexts.	Eingabe
input	Speicheradresse der zu verschlüsselnden Daten.	Eingabe
inputlength	Länge der zu verschlüsselnden Daten.	Eingabe
output	Speicheradresse des Rückgabebereiches für die verschlüsselten Daten.	Ausgabe
outputlength	Länge des unter *output* angegebenen Speicherbereiches.	Eingabe

Decrypt

Entschlüsseln von Daten. Die Art der Entschlüsselung ist abhängig vom Parameter *ctx.algorithm* bei der Funktion InitCTX.

Syntax	int Decrypt(PCIPHERCTX ctx, BYTE *input, int inputlength, BYTE *output, int outputlength);	
Returncode	Länge der entschlüsselten Daten oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Entschlüsseln benötigten Kontexts.	Eingabe
input	Speicheradresse der zu entschlüsselnden Daten.	Eingabe
inputlength	Länge der zu entschlüsselnden Daten.	Eingabe
output	Speicheradresse des Rückgabebereiches für die entschlüsselten Daten.	Ausgabe
outputlength	Länge des unter *output* angegebenen Speicherbereiches.	Eingabe

GetResultLength

Ermitteln des Speicherbedarfs für die zu ver-/entschlüsselnden Daten.
Hinweis: Bei Verwendung von RSA ist diese Funktion nur zur Ermittlung der Ausgabelänge für die Verschlüsselung möglich. Bei der Entschlüsselung ist die Ausgabelänge immer kleiner oder gleich der Eingabelänge!

Syntax	int GetResultLength(PCIPHERCTX ctx, int inputlength);	
Returncode	Länge der ver-/entschlüsselten Daten oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Ver-/Entschlüsseln benötigten Kontexts.	Eingabe
inputlength	Länge der zu ver-/entschlüsselnden Daten.	Eingabe

ResetCTX

Rücksetzen des Initialisierungs-Vektors (iv) bei symmetrischer Ver-/Entschlüsselung.

Syntax	int ResetCTX(PCIPHERCTX ctx);	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des Kontexts.	Eingabe

CleanupCTX

Freigeben des bei den Ver-/Entschlüsselungsaktionen benötigten Speichers.

Syntax	int CleanupCTX(PCIPHERCTX ctx);	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des Kontexts.	Eingabe

Beispiel (AES):

```
#include <stdlib.h>
#include "XPSCRYPT.H"

int main(void)
{
  BYTE block[32] = "XPS Software GmbH, Haar/Muenchen";
  BYTE seed1[16] = {0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01,0x00};
```

```
  BYTE seed2[16] = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f};
  BYTE key[32]   = {0};
  BYTE iv[16]    = {0};

  CIPHERCTX ctx     = {0};
  int       iOutlen = 0;
  int       rc      = 0;

  GenerateKey( key, sizeof(key), seed1, sizeof(seed1) );
  GenerateKey( iv,  sizeof(iv),  seed2, sizeof(seed2) );

  ctx.algorithm = AES;
  ctx.mode      = CBC;
  ctx.keylength = 256;
  ctx.key       = (BYTE *)key;
  ctx.iv        = (BYTE *)iv;

  /* encryption            */
  InitCTX( &ctx );
  iOutlen = Encrypt( &ctx, block, sizeof(block), block, sizeof(block) );
  if( iOutlen < 0 )
  {
    printf( "Encrypt error %d\n", iOutlen );
    exit( -1 );
  }
  /* now check decryption: */
  rc = ResetCTX( &ctx );
  iOutlen = Decrypt( &ctx, block, iOutlen, block, sizeof(block) );
  if( iOutlen < 0 )
  {
    printf( "Decrypt error %d\n", iOutlen );
    exit( -1 );
  }
  CleanupCTX( &ctx );
  return 0;
}
```

Beispiel (TripleDES):

```
#include <stdlib.h>
#include "XPSCRYPT.H"

int main(void)
{
  BYTE block[32] = "XPS Software GmbH, Haar/Muenchen";
  BYTE key[24]   = {0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01,0x00,
                    0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x08};
  BYTE iv[8]     = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07};

  CIPHERCTX ctx     = {0};
  int       iOutlen = 0;
  int       rc      = 0;

  ctx.algorithm = DES;
  ctx.mode      = CBC;
  ctx.keylength = 168;
  ctx.key       = (BYTE *)key;
  ctx.iv        = (BYTE *)iv;

  /* encryption            */
  InitCTX( &ctx );
  iOutlen = Encrypt( &ctx, block, sizeof(block), block, sizeof(block) );
  if( iOutlen < 0 )
  {
    printf( "Encrypt error %d\n", iOutlen );
    exit( -1 );
  }
  /* now check decryption: */
  ResetCTX( &ctx );
  iOutlen = Decrypt( &ctx, block, sizeof(block), block, sizeof(block) );
  if( iOutlen < 0 )
  {
    printf( "Decrypt error %d\n", iOutlen );
    exit( -1 );
  }
  CleanupCTX( &ctx );
  return 0;
}
```

Beispiel (RSA):

```
#include <stdlib.h>
#include "XPSCRYPT.H"

int main(void)
```

```
{
  BYTE  block[32]       = "XPS Software GmbH, Haar/Muenchen";
  BYTE  KeyRandom[16]   = {0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01,0x00};
  BYTE *output;
  BYTE  output2[32]    = {0};

  RSA_PRIVATE_KEY rsa_priv;
  RSA_PUBLIC_KEY  rsa_pub;
  CIPHERCTX       ctx     = {0};
  int             iOutlen = 0;
  int             rc      = 0;
  GenerateRSAKey( &rsa_priv, &rsa_pub, KeyRandom, sizeof(KeyRandom), 1024 );

  ctx.algorithm    = RSA;
  ctx.mode         = PUBLIC;
  ctx.key          = (BYTE *)&rsa_pub;

  /* encryption            */
  InitCTX( &ctx );
  iOutlen = GetResultLength( &ctx, sizeof(block) );
  output  = malloc( iOutlen );
  iOutlen = Encrypt( &ctx, block, sizeof(block), output, iOutlen );
  if( iOutlen < 0 )
  {
    printf( "Encrypt error %d\n", iOutlen );
    exit( -1 );
  }
  CleanupCTX( &ctx );

  /* now check decryption: */
  ctx.algorithm    = RSA;
  ctx.mode         = PRIVATE;
  ctx.key          = (BYTE *)&rsa_priv;

  InitCTX( &ctx );
  iOutlen = Decrypt( &ctx, output, iOutlen, output2, sizeof(output2) );
  if( iOutlen < 0 )
  {
    printf( "Decrypt error %d\n", iOutlen );
    exit( -1 );
  }
  CleanupCTX( &ctx );
  free( output );
  return 0;
}
```

Kapitel

4

Digitale Signatur

Allgemein

Eine der wichtigsten Anwendungsmöglichkeiten asymmetrischer Verschlüsselungsverfahren ist die Implementierung digitaler Signaturen. Dabei wird aus den Daten, die elektronisch unterschrieben werden sollen, eine charakteristische Prüfsumme (Hashwert) gebildet und diese dann mit dem geheimen Schlüssel verschlüsselt. Der so entstandene Schlüsseltext kann mit dem dazugehörigen öffentlichen Schlüssel wieder entschlüsselt werden.

CryptLib unterstützt digitale Signaturen mit dem Algorithmus **RSA**. Als Hashfunktion können **MD2**, **MD5, SHA-1, SHA-224, SHA-256, SHA-386, SHA-512** und **RipeMD160** benutzt werden.

Die Prozedur zum Erzeugen einer digitalen Signatur umfasst folgende Schritte:

- Initialisieren des Kontexts mit der Funktion *SignInit* zur Festlegung des Hashtyps.
- Hinzufügen der zu signierenden Daten mit der Funktion *SignUpdate*. Diese Aktion kann beliebig oft ausgeführt werden.
- Zum Abschluss muss die Funktion *SignFinal* aufgerufen werden, die die digitale Signatur zurückgibt.

Die Prozedur zum verifizieren einer digitalen Signatur umfasst folgende Schritte:

- Initialisieren des Kontexts mit der Funktion *VerifyInit* zur Festlegung des Hashtyps.
- Hinzufügen der zu prüfenden Daten mit der Funktion *VerifyUpdate*. Diese Aktion kann beliebig oft ausgeführt werden.
- Zum Abschluss muss die Funktion *VerifyFinal* aufgerufen werden, die die digitale Signatur verifiziert.

Funktionen

SignInit

Initialisierung des Signaturkontexts.

Syntax	**int SignInit(SIGNATURE_CTX *ctx, int digestAlgo);**
Returncode	0 oder Fehlercode (< 0).

Parameter	Beschreibung	Verwendung
ctx	Speicheradresse des zum Signieren benötigten Kontexts.	Eingabe
digestAlgo	Der zu verwendende Hashtyp. Unterstützt werden die Hashtypen MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160.	Eingabe

SignUpdate

Hinzufügen der zu signierenden Daten.

Syntax	**int SignUpdate(SIGNATURE_CTX *ctx, unsigned char *inputdata,int datalength);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Signieren benötigten Kontexts.	Eingabe
inputdata	Speicheradresse der zu signierenden Daten.	Eingabe
intputlength	Länge der zu signierenden Daten.	Eingabe

SignFinal

Erstellen der digitalen Signatur mit Hilfe des privaten Schlüssels.

Syntax	**int SignFinal(SIGNATURE_CTX *ctx, unsigned char *signdata, int *signlength, RSA_PRIVATE_KEY privkey);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Signieren benötigten Kontexts.	Eingabe
signdata	Speicheradresse des Rückgabebereichs für die erstellte Signatur.	Ausgabe
signlength	Speicheradresse für die Rückgabe der Länge der erstellten Signatur.	Ausgabe
privkey	Privater RSA Schlüssel des Unterzeichners.	Eingabe

VerifyInit

Initialisierung des Verifizierungskontexts.

Syntax	**int VerifyInit(SIGNATURE_CTX *ctx, int digestAlgo);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Verifizieren benötigten Kontexts.	Eingabe
digestAlgo	Der zu verwendende Hashtyp. Unterstützt werden die Hashtypen MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160.	Eingabe

VerifyUpdate

Hinzufügen der zu prüfenden Daten.

Syntax	**int VerifyUpdate(SIGNATURE_CTX *ctx, unsigned char *inputdata, int datalength);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Verifizieren benötigten Kontexts.	Eingabe
inputdata	Speicheradresse der zu verifizierenden Daten.	Eingabe
inputlength	Länge der zu verifizierenden Daten.	Eingabe

VerifyFinal

Prüfen der digitalen Signatur mit Hilfe des öffentlichen Schlüssels.

Syntax	**int VerifyFinal(SIGNATURE_CTX *ctx, unsigned char *signdata, int signlength, RSA_PUBLIC_KEY pubkey);**	
Beschreibung	Prüfen der digitalen Signatur mit Hilfe des öffentlichen Schlüssels.	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Verifizieren benötigten Kontexts.	Eingabe
signdata	Speicheradresse der Signatur.	Eingabe
signlength	Länge der Signatur.	Eingabe
pubkey	Öffentlicher RSA Schlüssel des Unterzeichners.	Eingabe

Beispiel:

```
#include <stdlib.h>
#include "XPSCRYPT.H"
#define keylen 1024

int main(void)
{
  BYTE  text1[32]       = "XPS Software GmbH, Haar/Muenchen";
  BYTE  text2[]         = "Muenchener Strasse 17";
  BYTE  randomSeed[16]  = {0x0f,0x0e,0x0d,0x0c,0x0b,0x0a,0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01,0x00};
  BYTE  sign[keylen/8]  = {0};
  int   signlen;
  int   rc;

  SIGNATURE_CTX   ctx;
  RSA_PRIVATE_KEY rsa_priv;
  RSA_PUBLIC_KEY  rsa_pub;

  GenerateRSAKey( &rsa_priv, &rsa_pub, randomSeed, sizeof(randomSeed), keylen );

  SignInit  ( &ctx, SHA1 );
  SignUpdate( &ctx, text1, sizeof(text1) );
  SignUpdate( &ctx, text2, sizeof(text2) );
  SignFinal ( &ctx, sign, &signlen, &rsa_priv );

  VerifyInit  ( &ctx, SHA1 );
  VerifyUpdate( &ctx, text1, sizeof(text1) );
  VerifyUpdate( &ctx, text2, sizeof(text2) );
  rc = VerifyFinal( &ctx, sign, signlen, &rsa_pub );

  printf( "\nVerify completed: RC=%d\n", rc );
  return 0;
}
```

Kapitel

5

Hashfunktionen

Allgemein

Hashfunktionen spielen im Zusammenhang mit Sicherheitsverfahren eine sehr wichtige Rolle. Sie werden immer dann benötigt, wenn aus Eingabedaten beliebiger Länge ein eindeutiger Hashwert erzeugt werden soll. Diese Prüfsumme wird dann zur Überprüfung der Unversehrtheit von Daten verwendet.

CryptLib unterstützt die Message Digest Funktionen **MD2**, **MD5, SHA-1, SHA-224, SHA-256, SHA-386, SHA-512** und **RipeMD160**. Außerdem wird **HMAC** (Keyed-Hashing for Message Authentication) unterstützt.

Die Prozedur zum Erzeugen eines Hashwerts umfasst folgende Schritte:

- Initialisieren des Kontexts mit der Funktion *DigestInit* zur Festlegung des Hashtyp.
- Hinzufügen der zu hashenden Daten mit der Funktion *DigestUpdate*. Diese Aktion kann beliebig oft ausgeführt werden.
- Zum Abschluss muss die Funktion *DigestFinal* aufgerufen werden, die den Hashwert zurückgibt.

Funktionen

DigestInit

Initialisierung des Hashkontexts.

Syntax	**int DigestInit(DIGEST_CTX *ctx, int digestAlgo);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des Rückgabebereichs für den zum Hashen benötigten Kontexts.	Ausgabe
digestAlgo	Der zu verwendende Hashtyp. Unterstützt werden die Hashtypen MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160.	Eingabe

DigestUpdate

Hinzufügen der zu hashenden Daten.

Syntax	int DigestUpdate(DIGEST_CTX *ctx, unsigned char *inputdata, int datalength);	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Hashen benötigten Kontexts.	Eingabe
inputdata	Speicheradresse der zu hashenden Daten.	Eingabe
inputlength	Länge der zu hashenden Daten.	Eingabe

DigestFinal

Erstellen des Hashwertes.

Syntax	int DigestFinal(DIGEST_CTX *ctx, unsigned char *hashdata, int *hashlength);	
Beschreibung	Erstellen des Hashwertes.	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des zum Hashen benötigten Kontexts.	Eingabe
hashdata	Speicheradresse des Rückgabebereiches für den zu erstellenden Hashwert.	Ausgabe
hashlength	Speicheradresse für die Rückgabe der Länge des erstellten Hashwerts.	Ausgabe

Beispiel:

```
#include <stdlib.h>
#include "XPSCRYPT.H"

int main(void)
{
  BYTE  text1[32]       = "XPS Software GmbH, Haar/Muenchen";
  BYTE  text2[]         = "Muenchener Strasse 17";
  char  md5Hash[16]     = {0};
  char  sha1Hash[20]    = {0};
  char  ripeHash[20]    = {0};
  int   digestlen;

  DIGEST_CTX   ctx;

  DigestInit  ( &ctx, MD5 );
  DigestUpdate( &ctx, text1, sizeof(text1) );
  DigestUpdate( &ctx, text2, sizeof(text2) );
  DigestFinal ( &ctx, md5Hash, &digestlen );

  DigestInit  ( &ctx, SHA1 );
  DigestUpdate( &ctx, text1, sizeof(text1) );
  DigestUpdate( &ctx, text2, sizeof(text2) );
  DigestFinal ( &ctx, sha1Hash, &digestlen );

  DigestInit  ( &ctx, RIPEMD160 );
  DigestUpdate( &ctx, text1, sizeof(text1) );
  DigestUpdate( &ctx, text2, sizeof(text2) );
  DigestFinal ( &ctx, ripeHash, &digestlen );
  return 0;
}
```

HMAC

Erstellen eines MAC. Ein Message-Authentication-Code oder MAC ist eine schlüsselabhängige Einweg-Hashfunktion. Daher kann der MAC nur mit Hilfe eines Schlüssels erzeugt oder verifiziert werden. Dadurch wird verhindert, dass ein Unbefugter eine durch einen MAC geschützte Nachricht verändert und den Hashwert neu berechnet. Ein MAC dient somit zur Sicherstellung von Datenintegrität ohne die Daten verschlüsseln zu müssen.

Syntax	int HMAC(BYTE *input, int inputlength, BYTE *key, int keylength, BYTE

	***digestMessage, int digestAlgo);**	
Returncode	Länge des HMAC oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
input	Speicheradresse der Eingabedaten.	Eingabe
inputlength	Länge der Eingabedaten.	Eingabe
key	Speicheradresse des Schlüssels.	Eingabe
keylength	Länge des Schlüssels.	Eingabe
digestMessage	Speicheradresse des Rückgabebereiches für den erstellten Hashwert.	Ausgabe
digestAlgo	Der zu verwendende Hashtyp. Unterstützt werden die Hashtypen MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160.	Eingabe

Kapitel

6

X.509 Zertifikate

Allgemein

Zertifikate gibt es seit der Erfindung von Public-Key-Algorithmen. Sie werden auch als elektronische Ausweise bezeichnet. Ein Zertifikat ist nichts anderes als ein signierter Datensatz. Beim Ausstellen eines Zertifikats sind zwei Parteien beteiligt: ein Zertifikatsaussteller und ein Zertifikatsantragsteller. Beide müssen ein asymmetrisches Schlüsselpaar besitzen. Will der Antragsteller zertifiziert werden, so übergibt er seinen öffentlichen Schlüssel dem Aussteller. Dieser bildet einen Datensatz, der sich aus dem Namen des Ausstellers, dem Namen des Antragstellers und dessen öffentlichen Schlüssel zusammensetzt. Dieser Datensatz wird anschließend mit dem privaten Schlüssel des Ausstellers signiert. Zusammen mit der Signatur bildet er das Zertifikat. Der Aussteller bescheinigt somit, dass der Zertifikatsinhaber und dessen öffentlicher Schlüssel zusammengehören. Mit dem öffentlichen Schlüssel des Zertifizierers kann der elektronische Ausweis jederzeit auf seine Unverfälschtheit überprüft werden.

CryptLib bietet Funktionen an, mit denen man die Unverfälschtheit eines Zertifikats überprüfen, sowie sämtliche Daten eines Zertifikats auslesen kann.

Funktionen

ImportCertificate

Einlesen eines Zertifikatobjekts sowie Prüfung auf formale Richtigkeit.

Syntax	**int ImportCertificate(void *certObject, void **certctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certObject	Speicheradresse eines X.509 Zertifikats. Sowohl binäre als auch Base64-verschlüsselte Formate werden unterstützt.	Eingabe
certctx	Adresspointer zur Aufnahme der Speicheradresse des erstellten Zertifikatskontexts. Dieser wird für die weitere Bearbeitung des Zertifikats benötigt.	Ausgabe

GetPublicKey

Extrahieren des öffentlichen Schlüssels aus dem Zertifikat.

Syntax	**int GetPublicKey(void *certctx, RSA_PUBLIC_KEY *pubkey, int outlen);**
Returncode	Länge des öffentlichen Schlüssels oder Fehlercode (< 0).

Parameter	Beschreibung	Verwendung
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
pubkey	Speicheradresse des Rückgabebereiches für den öffentlichen Schlüssel.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes für den öffentlichen Schlüssel.	Eingabe

GetCryptAlgo

Extrahieren des Verschlüsselungs-Algorithmus des öffentlichen Schlüssels.

Syntax	**int GetCryptAlgo(void *certctx, char *algo, int outlen);**	
Returncode	Länge der Bezeichnung des Verschlüsselungs-Algorithmus oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
algo	Speicheradresse des Rückgabebereiches für die Bezeichnung des Verschlüsselungs-Algorithmus.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetCryptKeylen

Extrahieren der Länge des öffentlichen Schlüssels.

Syntax	**int GetCryptKeylen(void *certctx);**	
Returncode	Länge des öffentlichen Schlüssels oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe

GetVersionInfo

Extrahieren der Versionsnummer des Zertifikats.

Syntax	**int GetVersionInfo(void *certctx, char *version, int outlen);**	
Returncode	Länge der Versionsnummer oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
version	Speicheradresse des Rückgabebereiches für die extrahierte Versionsnummer.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetSerialNumber

Extrahieren der Seriennummer des Zertifikats.

Syntax	**int GetSerialNumber(void *certctx, BYTE *serial, int outlen);**	
Returncode	Länge der Seriennummer oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**

certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
serial	Speicheradresse des Rückgabebereiches für die extrahierte Seriennummer.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetIssuerDN

Extrahieren der Zertifikatsausstellerdaten.

Syntax	**int GetIssuerDN(void *certctx, char *dn, int outlen);**	
Returncode	Länge der Zertifikatsausstellerdaten oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
dn	Speicheradresse des Rückgabebereiches für die extrahierten Ausstellerdaten. Die einzelnen Elemente (CN, O, OU usw.) sind durch Kommata getrennt.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetSubjectDN

Extrahieren der Zertifikatsinhaberdaten.

Syntax	**int GetSubjectDN(void *certctx, char *dn, int outlen);**	
Returncode	Länge der Zertifikatsinhaberdaten oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
dn	Speicheradresse des Rückgabebereiches für die extrahierten Inhaberdaten. Die einzelnen Elemente (CN, O, OU usw.) sind durch Kommata getrennt.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetSignatureAlgo

Extrahieren des vom Zertifikatsaussteller zur Signatur verwendeten Algorithmus.

Syntax	**int GetSignatureAlgo(void *certctx, char *algo, int outlen);**	
Returncode	Länge der Bezeichnung des Verschlüsselungs-Algorithmus oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
algo	Speicheradresse des Rückgabebereiches für die Bezeichnung des Verschlüsselungs-Algorithmus.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetSignature

Extrahieren der Zertifikatssignatur.

Syntax	**int GetSignature(void *certctx, BYTE *signature, int outlen);**

Returncode	Länge der Signatur oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
signature	Speicheradresse des Rückgabebereiches für die extrahierte Signatur.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetStartDate

Extrahieren des Gültigkeitsbeginns des Zertifikats.

Syntax	**int GetStartDate(void *certctx, char *startdate, int outlen);**	
Returncode	Länge des Datums oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
startdate	Speicheradresse des Rückgabebereiches für den Gültigkeitsbeginn. Das Datum wird in der Form DD.MM.YYYY HH:MM:SS zurückgegeben.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetEndDate

Extrahieren des Gültigkeitsendes des Zertifikats.

Syntax	**int GetEndDate(void *certctx, char *enddate, int outlen);**	
Returncode	Länge des Datums oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
startdate	Speicheradresse des Rückgabebereiches für das Gültigkeitsende. Das Datum wird in der Form DD.MM.YYYY HH:MM:SS zurückgegeben.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetIssuerDNBlob

Extrahieren der Zertifikatsausstellerdaten in binärer Form einschließlich der ASN.1 Steuerzeichen. Dieser Blob kann dazu verwendet werden, den weiteren Zertifizierungspfad zu verfolgen.

Syntax	**int GetIssuerDNBlob(void *certctx, BYTE **blob);**	
Returncode	Länge des Zertifikatsaussteller-Blobs oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
blob	Adresspointer zur Aufnahme der Speicheradresse des Rückgabebereiches des extrahierten Zertifikatsaussteller-Blobs.	Ausgabe

GetSubjectDNBlob

Extrahieren der Zertifikatsinhaberdaten in binärer Form einschließlich der ASN.1 Steuerzeichen. Dieser Blob kann dazu verwendet werden, den weiteren Zertifizierungspfad zu verfolgen.

Syntax	int GetSubjectDNBlob (void *certctx, BYTE **blob);	
Returncode	Länge des Zertifikatsinhaber-Blobs oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
blob	Adresspointer zur Aufnahme der Speicheradresse des Rückgabebereiches des extrahierten Zertifikatsinhaber-Blobs.	Ausgabe

GetIssuerDNByType

Extrahieren des unter *type* spezifizierten Zertifikatsausstellerelements.

Syntax	int GetIssuerDNByType(void *certctx, int type, char *data, int outlen);	
Returncode	Länge der Zertifikatsausstellerdaten oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
type	DN-Typ des Ausstellers. Folgende Typen sind möglich: DN_C Country DN_SP State/Province DN_L Locality DN_O OrganizationName DN_OU OrganizationUnit DN_CN CommonName DN_EMAIL E-Mail DN_STREET Street DN_PHONE Phone DN_POSTAL PostalCode DN_TITLE Title	Eingabe
data	Speicheradresse des Rückgabebereiches für die extrahierten Ausstellerdaten.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetSubjectDNByType

Extrahieren des unter *type* spezifizierten Zertifikatsinhaberelements.

Syntax	int GetSubjectDNByType(void *certctx, int type, char *data, int outlen);	
Returncode	Länge der Zertifikatsinhaberdaten oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
type	DN-Typ des Inhabers. Folgende Typen sind möglich: DN_C Country DN_SP State/Province DN_L Locality DN_O OrganizationName DN_OU OrganizationUnit DN_CN CommonName DN_EMAIL E-Mail DN_STREET Street	Eingabe

	DN_PHONE Phone DN_POSTAL PostalCode DN_TITEL Titel	
data	Speicheradresse des Rückgabebereiches für die extrahierten Inhaberdaten.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

GetFirstExtension

Die Standardfelder der X.509 Zertifikate reichen für viele Anwendungen nicht aus. Aus diesem Grund wurde die Syntax der Version 3 um eine Extension-Komponente erweitert. Mit dieser ist es möglich, beliebige Daten in einem Zertifikat anzugeben.

Syntax	**int GetFirstExtension(void *certctx, CERTEXT *ext);**	
Returncode	Länge des Extension-Bereichs (0 falls nicht vorhanden) oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
ext	Speicheradresse des Rückgabebereiches für die extrahierte Extension.	Ausgabe
Felder	**Beschreibung**	
ext.oid_binary	Binärer Wert des OIDs (Object Identifier).	
ext.oid_char	Character Wert des OIDs (Object Identifier).	
ext.isCritical	Diese Komponente markiert eine Erweiterung als kritisch oder unkritisch. Kritische Erweiterungen müssen immer beachtet werden. Stößt ein Programm auf eine solche und kennt sie nicht, darf das Zertifikat nicht verwendet werden. Nichtkritische Erweiterungen sind unproblematisch. Sie können gegebenenfalls ignoriert werden.	
ext.fieldType	Der Feldtyp dieser Extension. Folgende Feldtypen sind möglich: BER_BOOLEAN BER_INTEGER BER_BITSTRING BER_OCTETSTRING BER_OBJECT_IDENTIFIER BER_OBJECT_DESCRIPTOR BER_EXTERNAL BER_REAL BER_ENUMERATED BER_EMBEDDED_PDV BER_STRING_UTF8 BER_RELATIVE_OID BER_STRING_NUMERIC BER_STRING_PRINTABLE BER_STRING_T61 BER_STRING_GRAPHIC BER_STRING_ISO646 BER_STRING_VIDEOTEXT BER_STRING_GENERAL BER_STRING_UNIVERSAL BER_CHAR_STRING BER_STRING_BMP	
ext.value	Integer Wert bei den Typen BER_BOOLEAN, BER_INTEGER, BER_ENUMERATED.	
ext.data	Speicheradresse des binären Datenbereichs der Extension.	
ext.datalen	Länge des binären Datenbereichs der Extension.	

GetNextExtension

Extrahieren der nächsten Zertifikats-Extension.

Syntax	**int GetNextExtension(void *certctx, CERTEXT *ext);**	
Returncode	Länge des Extension-Bereichs (0 falls nicht vorhanden) oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
ext	Speicheradresse des Rückgabebereiches für die extrahierte Extension. Die Struktur des Rücktgabebereiches ist unter 'GetFirstExtension' beschrieben.	Ausgabe

GetExtensionByOID

Extrahieren der Zertifikats-Extension mit diesem Object-Identifier (OID).

Syntax	**int GetExtensionByOID(void *certctx, BYTE *oid, CERTEXT *ext);**	
Returncode	Länge des Extension-Bereichs (0 falls nicht vorhanden) oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
oid	Speicheradresse des binären Object-Identifier, der extrahiert werden soll.	Eingabe
ext	Speicheradresse des Rückgabebereiches für die extrahierte Extension. Die Struktur des Rücktgabebereiches ist unter 'GetFirstExtension' beschrieben.	Ausgabe

GetFingerPrint

Erzeugen eines Fingerprints (Hashwert) des Zertifikats. Der Fingerprint dient zur visuellen Überprüfung eines Zertifikats.

Syntax	**int GetFingerPrint(void *certctx, int digestAlgo, BYTE *data, int outlen);**	
Returncode	Länge des Fingerprints oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
digestAlgo	Der zu verwendende Hashtyp. Unterstützt werden die Hashtypen MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160.	Eingabe
data	Speicheradresse des Rückgabebereiches für den erzeugten Fingerprint.	Ausgabe
outlen	Größe des verfügbaren Speicherplatzes.	Eingabe

VerifyCertificate

Überprüfung der Gültigkeit eines Zertifikats.

Syntax	**int VerifyCertificate(void *certctx, RSA_PUBLIC_KEY pubkey);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe
pubkey	Öffentlicher Schlüssel des Zertifikatsausstellers.	Eingabe

CleanupCertificate

Freigeben des von den Zertifikatsroutinen benötigten Speichers.

Syntax	**int CleanupCertificate(void *certctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
certctx	Speicheradresse des Zertifikatskontexts.	Eingabe

Beispiel:

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "XPSCRYPT.H"

int  getRootPublicKey( char *file, RSA_PUBLIC_KEY * );
void printExtension( CERTEXT *extension );

typedef struct _dn_tab
{
  char name[21];
  int  type;
}DN_TAB;


/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
/*     parm1: certificate                                       *
/*     parm2: root-certificate [OPTIONAL]                       *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
// ------------------------------------------------------------ //
{
  RSA_PUBLIC_KEY  pubkey   = {0};
  BYTE           *cert;
  BYTE           *certctx;
  int             iCount, iRc, i1;
  unsigned char   temp[512]  = {0};
  unsigned char   temp2[128] = {0};
  unsigned char   oid[] = {0x06,0x09,0x60,0x86,0x48,0x01,0x86,0xF8,0x42,0x01,0x04};
  char           *ptr1, *ptr2;
  CERTEXT         extension;


  DN_TAB          DN_TAB[11] =
  {
    "CN=",     DN_CN,
    "O=",      DN_O,
    "OU=",     DN_OU,
    "L=",      DN_L,
    "SP=",     DN_SP,
    "STREET=", DN_STREET,
    "POSTAL=", DN_POSTAL,
    "C=",      DN_C,
    "EMAIL=",  DN_EMAIL,
    "PHONE=",  DN_PHONE,
    "TITEL=",  DN_TITEL
  };

  iCount = readFile( argv[1], &cert );
  if( iCount < 1 )
  {
    printf( "Certificate \"%s\" not found.\n", argv[1] );
    return( -1 );
  }

  iRc = ImportCertificate( cert, &certctx );
  if( iRc != 0 )
  {
    printf( "invalid certificate \"%s\": rc=%d\n", iRc );
    return( -1 );
  }

  if( argc > 2 )     // if parm2 -> verify certificate
  {
    iRc = getRootPublicKey( argv[2], &pubkey );
    if( iRc != 0 )
      return( -1 );

    iRc = VerifyCertificate( certctx, &pubkey );
    if( iRc != 0 )
      printf("Verify Error: rc=%d\n\n", iRc);
  }

  iCount = GetVersionInfo( certctx, temp, sizeof(temp) );
  printf("Version           = %s\n", temp);

  iCount = GetSerialNumber( certctx, temp, sizeof(temp) );
  printf("Serialnr          = %02X", temp[0]);
  for( i1 = 1; i1 < iCount; i1++ )
    printf("%02X", temp[i1]);
  printf("\n");
```

```
  iCount = GetSignatureAlgo( certctx, temp, sizeof(temp));
  printf("SignatureAlgo     = %s\n", temp);

  iCount = GetIssuerDN( certctx, temp, sizeof(temp) );
  strcpy( temp2, "\nIssuer            = " );
  ptr1 = temp;
  while( iCount )
  {
    unsigned char   temp3[128] = {0};
    ptr2 = strchr( ptr1, ',' );
    if( ptr2 )
    {
      memcpy( temp3, ptr1, ptr2-ptr1 );
      strcat( temp2, temp3 );
      iCount -= (ptr2-ptr1);
    }
    else
    {
      memcpy( temp3, ptr1, iCount );
      strcat( temp2, temp3 );
      iCount = 0;
    }
    printf( "%s\n", temp2 );
    strcpy( temp2, "                    " );
    ptr1 = ptr2+2;
  }
  printf("\n");

  iCount = GetStartDate( certctx, temp, sizeof(temp) );
  printf("StartDate         = %s\n", temp);

  iCount = GetEndDate( certctx, temp, sizeof(temp) );
  printf("EndDate           = %s\n", temp);

  strcpy( temp2, "\nSubject           = ");
  for( i1 = 0; i1 < 11; i1++ )
  {
    iCount = GetSubjectDNByType( certctx, DN_TAB[i1].type, temp, sizeof(temp) );
    if( iCount > 0 )
      printf("%s%s%s\n", temp2, DN_TAB[i1].name, temp);
    strcpy( temp2, "                    ");
  }
  printf("\n");

  iCount = GetCryptAlgo( certctx, temp, sizeof(temp) );
  printf("Algorithm         = %s", temp);

  iCount = GetCryptKeylen( certctx );
  printf("(%d)\n", iCount);

  iCount = GetPublicKey( certctx, (RSA_PUBLIC_KEY *)temp, sizeof(RSA_PUBLIC_KEY) );
  DumpData( "PublicKey", temp, iCount );

  iCount = GetSignatureAlgo( certctx, temp, sizeof(temp) );
  printf("\nSignature-Algo    = %s\n", temp);

  iCount = GetSignature( certctx, temp, sizeof(temp) );
  DumpData( "Signature", temp, iCount );

  iCount = GetFirstExtension( certctx, &extension );
  if( iCount == 0 )
  {
    printExtension( &extension );
    while( iCount == 0 )
    {
      iCount = GetNextExtension( certctx, &extension );
      if( iCount == 0 )
       printExtension( &extension );
    }
  }

  iCount = GetExtensionByOID( certctx, oid, &extension );
  if( iCount == 0 )
    printExtension( &extension );

  iCount = GetFingerprint( certctx, SHA1, temp, sizeof(temp) );
  DumpData( "SHA1 Fingerprint", temp, iCount );
  iCount = GetFingerprint( certctx, MD5, temp, sizeof(temp) );
  DumpData( "MD5 Fingerprint", temp, iCount );

  CleanupCertificate( certctx );
  CleanupFile( cert );
  return 0;
}

/* ------------------------------------------------------------ *
 *                                                              */
   void printExtension
/*                                                              *
```

```
 * ------------------------------------------------------------ */
( CERTEXT    *extension)
// ------------------------------------------------------------ //
{
  printf("\n\nExtension         = %s\n", extension->oid_char );
  printf("   isCritical     = %d\n", extension->isCritical);
  printf("   Type           = %d\n", extension->fieldType);
  if( extension->datalen == 0 )
    printf("   value          = %d\n", extension->value);
  else
    DumpData( "   data", extension->data, extension->datalen );
}

/* ------------------------------------------------------------ *
 *                                                              */
   int getRootPublicKey
/*                                                              *
 * ------------------------------------------------------------ */
( char            *rootFile,
  RSA_PUBLIC_KEY  *pubkey
)
// ------------------------------------------------------------ //
{
  BYTE  *buffer;
  BYTE  *rootctx;
  int    iRc, iCount;

  iCount = readFile( rootFile, &buffer );
  if( iCount == 0 )
  {
    printf( "file \"%s\" not found\n", rootFile );
    return( -1 );
  }

  iRc = ImportCertificate( buffer, &rootctx );
  if( iRc != 0 )
  {
    printf( "invalid root certificate \"%s\": rc=%d\n", iRc );
    return( -1 );
  }

  iRc = GetPublicKey( rootctx, pubkey, sizeof(RSA_PUBLIC_KEY) );
  if( iRc < 1 )
  {
    printf( "invalid root public-key: rc=%d\n", iRc );
    return( -1 );
  }

  CleanupCertificate( rootctx );
  CleanupFile( buffer );
  return( 0 );
}
```

Kapitel

7

S/MIME Objekte (PKCS#7)

Allgemein

PKCS#7 wird als *Cryptographic Message Syntax Standard* bezeichnet und beschreibt eine Syntax, nach der Daten durch kryptographische Maßnahmen wie digitale Signaturen oder Verschlüsselung geschützt werden können. CryptLib unterstützt folgende Inhaltstypen (Content Types):

Data	wird zur Modellierung von Daten verwendet und bietet keinerlei kryptographische Funktionalität.
Signed-data	beschreibt ein Format um die Integrität von Daten und die Sender-Authentizität durch Verwendung von digitalen Signaturen und Zertifikaten zu gewährleisten.
Enveloped-data	wird zur empfängerspezifischen Verschlüsselung von Daten verwendet, um zu verhindern, dass ein Unbefugter den Inhalt lesen kann. (Vertraulichkeit).
Encrypted-data	wird zur Datenverschlüsselung verwendet.

Funktionen

ImportPKCS7Data

Einlesen eines PKCS#7-Data Objekts sowie Prüfung auf formale Richtigkeit.
Der Inhaltstyp *Data* beschreibt eine beliebige Folge von Datenbytes.
Die Daten können mit den Funktionen *GetFirstData* sowie *GetNextData* extrahiert werden.

Syntax	**int ImportPKCS7Data(BYTE *pkcs7object, int lengthobject, void **pkcs7ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7Object	Speicheradresse eines PKCS#7-Data Objekts. Unterstützt werden sowohl binäre, Base64-verschlüsselte als auch S/MIME Formate.	Eingabe
lengthobject	Länge des PKCS#7-Data Objekts.	Eingabe
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des importierten PKCS#7-Kontexts. Dieser wird für die weitere Bearbeitung des Objekts benötigt.	Ausgabe

ImportSignedData

Einlesen eines PKCS#7 Signed-data Objekts sowie Prüfung auf formale Richtigkeit. Der Inhaltstyp *Signed-data* definiert eine Syntax zur Berechnung und Übertragung von digitalen Signaturen. Die Anzahl der Parteien, die eine Nachricht signieren, ist nicht eingeschränkt. Das heißt, es ist erlaubt, dass dieselben Daten von mehr als nur einer Partei unterzeichnet werden.
Die Daten können mit den Funktionen *GetFirstData* sowie *GetNextData* extrahiert werden. Mit den Funktionen *GetFirstSigner* sowie *GetNextSigner* können die einzelnen Unterzeichner ausgelesen werden. Die Funktionen *VerifySigner* sowie *VerifyAllSigner* können zur Überprüfung der Integrität der Daten verwendet werden. Mit der Funktion *AddSignerCert* können Zertifikatsaussteller hinzugefügt werden.

Syntax	**int ImportSignedData(BYTE *pkcs7object, int lengthobject, void **pkcs7ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7Object	Speicheradresse eines PKCS#7 Signed-data Objekts. Unterstützt werden sowohl binäre, Base64-verschlüsselte als auch S/MIME Formate.	Eingabe
lengthobject	Länge des PKCS#7 Signed-data Objekts.	Eingabe
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des importierten PKCS#7-Kontexts. Dieser wird für die weitere Bearbeitung des Objekts benötigt.	Ausgabe

ImportEnvelopedData

Einlesen eines PKCS#7 Enveloped-data Objekts sowie Prüfung auf formale Richtigkeit. Der Inhaltstyp *Enveloped-data* definiert eine Syntax, nach der die Nachricht 'empfänger-spezifisch' verschlüsselt wird, also Informationen über den beabsichtigten Empfänger miteinbezieht. Dazu wird eine Technik verwendet, die man als 'Digital Enveloping' bezeichnet. In Äquivalenz zum *Signed-data*-Typ, bei dem eine Nachricht von mehreren Unterzeichnern signiert werden kann, erlaubt es der *Enveloped-data*-Typ, mehrere Empfänger einzubeziehen. Informationen über einen Empfänger werden im Hilfstyp 'RecipientInfo' zusammengefasst.
Die Daten können mit den Funktionen *GetFirstData* sowie *GetNextData* extrahiert werden.

Syntax	**int ImportEnvelopedData(BYTE *pkcs7object, int lengthobject, BYTE *pkcs12object, int lengthpkcs12, BYTE *pwd, int lengthpwd, void **pkcs7ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7Object	Speicheradresse eines PKCS#7 Enveloped-data Objekts. Unterstützt werden sowohl binäre, Base64-verschlüsselte als auch S/MIME Formate.	Eingabe
lengthobject	Länge des PKCS#7 Enveloped-data Objekts.	Eingabe
pkcs12Object	Speicheradresse eines PKCS#12-Objekts. PKCS#12-Objekte stellen eine Syntax für den Austausch von Schlüsseln und Zertifikaten zur Verfügung. Ein PKCS#12-Objekt enthält Schlüsseltaschen (key-bags) und Zertifikatstaschen (certificate-bags). Mit Hilfe der Zertifikatstaschen kann ermittelt werden, ob im PKCS#7-Objekt eine RecipientInfo für diesen Benutzer enthalten ist. Falls eine RecipientInfo enthalten ist, kann dann aus der Schlüsseltasche der private Schlüssel des Anwenders extrahiert werden. Mit dem privaten Schlüssel wird der *Content-Encryption*-Schlüssel dekodiert. Mit diesem wiederum können dann die Daten entschlüsselt werden.	Eingabe
lengthpkcs12	Länge des PKCS#12-Objekts.	Eingabe
pwd	PKCS#12-Objekte sind mit einem Passwort gesichert. Hier ist die Speicheradresse des Passwortes zu übergeben.	Eingabe
lengthpwd	Länge des Passwortes.	Eingabe
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des importierten PKCS#7-Kontexts. Dieser wird für die weitere Bearbeitung benötigt.	Ausgabe

ImportEncryptedData

Einlesen eines PKCS#7 Encrypted-data Objekts sowie Prüfung auf formale Richtigkeit. Der Inhaltstyp *Encrypted-data* beschreibt eine Syntax zur Datenverschlüsselung. Im Unterschied zum Typ *Enveloped-data* wird hier davon ausgegangen, dass der Empfänger bereits im Besitz des *Content-Encryption*-Schlüssels ist und dieser daher nicht eigens angegeben werden muss. Dieser Typ eignet sich vor allem für Applikationen, in denen Daten verschlüsselt auf ein Speichermedium geschrieben werden. Ein prominentes Anwendungsbeispiel ist der *Personal-Information-Exchange-Syntax*-Standard PKCS#12.
Die Daten können mit den Funktionen *GetFirstData* sowie *GetNextData* extrahiert werden.

Syntax	**int ImportEncryptedData(BYTE *pkcs7object, int lengthobject, BYTE *pwd, int lengthpwd, void **pkcs7ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7Object	Speicheradresse eines PKCS#7 Encrypted-data Objekts. Unterstützt werden sowohl binäre, Base64-verschlüsselte, als auch S/MIME Formate.	Eingabe
lengthobject	Länge des PKCS#7 Encrypted-data Objekts.	Eingabe
pwd	Speicheradresse des Passwortes, mit dem der *Content-Encryption*-Schlüssel verschlüsselt wurde.	Eingabe
lengthpwd	Länge des Passwortes.	Eingabe
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des importierten PKCS#7-Kontexts. Dieser wird für die weitere Bearbeitung des Objekts benötigt.	Ausgabe

CreatePKCS7Data

Erstellen eines PKCS#7-Data-Objektes. Ein PKCS#7-Data-Objekt wird zur Modellierung von Daten verwendet und bietet keinerlei kryptographische Funktionalität. Daten werden mit der Funktion *AddPKCS7Data* hinzugefügt.

Syntax	**int CreatePKCS7Data(int option, void **pkcs7ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
option	HEADER_INCLUDED Bei Angabe dieser Option wird der ContentType hinzugefügt.	Eingabe
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des erstellten PKCS#7-Kontexts.	Ausgabe

CreateSignedData

Erstellen eines PKCS#7 Signed-data-Objektes. Der Inhaltstyp *Signed-data* definiert eine Syntax zur Berechnung und Übertragung von digitalen Signaturen. Die Anzahl der Parteien, die eine Nachricht signieren, ist unbeschränkt. Das heißt, es ist erlaubt, dass dieselben Daten von mehr als nur einer Partei unterzeichnet werden. Unterzeichner werden mit der Funktion *AddSigner,* Daten mit der Funktion *AddPKCS7Data* hinzugefügt. Mit der Funktion *AddSignerCert* können zusätzliche Zertifikate hinzugefügt werden.

Syntax	**int CreateSignedData(int option, void **pkcs7ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**

option	HEADER_INCLUDED	Bei Angabe dieser Option wird der ContentType hinzugefügt.	Eingabe
	DATA_IMPLICIT	Bei Angabe dieser Option werden die Daten in das Signed-data Objekt miteinbezogen. Das bedeutet, dass dann das content-Feld für die Aufnahme des Nachrichteninhalts vorhanden ist. Wenn diese Option nicht gesetzt wird, fehlt das content-Feld und die Daten müssen auf andere Art und Weise übertragen werden.	
	CERT_IMPLICIT	Bei Angabe dieser Option werden sämtliche in der PKCS#12-Datei enthaltenen Zertifikate des Unterzeichners in das Signed-data-Objekt mit aufgenommen.	
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des erstellten PKCS#7-Kontexts.		Ausgabe

CreateEnvelopedData

Erstellen eines PKCS#7 Enveloped-data-Objektes. Der Inhaltstyp *Enveloped-data* definiert eine Syntax, nach der die Nachricht 'empfänger-spezifisch' verschlüsselt wird, also Informationen über den beabsichtigten Empfänger miteinbezieht. Dazu wird eine Technik verwendet, die man 'Digital Enveloping' bezeichnet. In Äquivalenz zum *Signed-data*-Typ, bei dem eine Nachricht von mehreren Unterzeichnern signiert werden kann, erlaubt es der *Enveloped-data*-Typ, mehrere Empfänger einzubeziehen. Empfänger werden mit der Funktion *AddRecipient,* Daten mit der Funktion *AddPKCS7Data* hinzugefügt.

Syntax	**int CreateEnvelopedData(int option, void **pkcs7ctx, int encryptionAlgo);**		
Returncode	0 oder Fehlercode (< 0).		
Parameter	**Beschreibung**		**Verwendung**
option	HEADER_INCLUDED	Bei Angabe dieser Option wird der ContentType hinzugefügt.	Eingabe
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des erstellten PKCS#7-Kontexts.		Ausgabe
encryptionAlgo	Algorithmus mit dem die Daten verschlüsselt werden sollen. Folgende Algorithmen werden unterstützt:		Eingabe
	desEDE3CBC	Triple DES 168Bit	
	desCBC	DES 56Bit	
	rc2CBC_128	RC2 128Bit	
	rc2CBC_64	RC2 64Bit	
	rc2CBC_40	RC2 40Bit	
	rc4_128	RC4 128Bit	
	rc4_64	RC4 64Bit	
	rc4_40	RC4 40Bit	

CreateEncryptedData

Erstellen eines PKCS#7 Encrypted-data-Objektes. Der Inhaltstyp *Encrypted-data* beschreibt eine Syntax zur Datenverschlüsselung. Im Unterschied zum Typ *Enveloped-data* wird hier davon ausgegangen, dass der Empfänger bereits im Besitz des *Content-Encryption*-Schlüssels ist und dieser daher nicht eigens angegeben werden muss. Dieser Typ eignet sich vor allem für Applikationen, bei denen Daten verschlüsselt auf ein Speichermedium geschrieben werden. Die Daten werden mit der Funktion *AddPKCS7Data* hinzugefügt.

Syntax	**int CreateEncryptedData(int option, void **pkcs7ctx, int pbeAlgo, BYTE *pwd, int lengthpwd);**

Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
option	HEADER_INCLUDED Bei Angabe dieser Option wird der ContentType hinzugefügt.	Eingabe
pkcs7ctx	Adresspointer zur Aufnahme der Speicheradresse des erstellten PKCS#7-Kontexts.	Ausgabe
pbeAlgo	Algorithmus mit dem die Daten verschlüsselt werden sollen. Folgende Algorithmen werden unterstützt: pbe3DES_3Key Triple DES 168Bit pbe3DES_2Key Triple DES 112Bit pbeRC2_128 RC2 128Bit pbeRC2_40 RC2 40Bit pbeRC4_128 RC4 128Bit pbeRC4_40 RC4 40Bit	Eingabe
pwd	Speicheradresse des Passwortes, das zur Schlüsselgenerierung benötigt wird.	Eingabe
lengthpwd	Länge des Passwortes.	Eingabe

AddPKCS7Data

Import Funktionen:
Bei *Signed-data* Objekts können die Daten IMPLICIT oder EXPLICIT verarbeitet werden. Wird der Modus IMPLICIT festgelegt, werden die Daten in das *Signed-data*-Objekt miteinbezogen. Genauer ausgedrückt bedeutet dies, dass dann das Feld *content* für die Aufnahme des Nachrichteninhalts vorhanden ist. Wird jedoch der Modus EXPLICIT gewählt, fehlt das *content*-Feld und die Daten müssen auf andere Art und Weise übertragen werden. Diese Funktion bietet die Möglichkeit, Daten an das *Signed-data*-Objekt zu übergeben.
Create Funktionen:
Hiermit werden bei den Funktion *CreatePKCS7Data, CreateSignedData, CreateEnvelopedData* sowie *CreateEncryptedData* Daten zum PKCS#7-Objekt hinzugefügt.

Syntax	**int AddPKCS7Data(void *pkcs7ctx, BYTE *data, int lengthdata);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
data	Speicheradresse der zu übergebenden Daten.	Eingabe
lengthdata	Länge der Daten.	Eingabe

AddSigner

Ein *Signed-data* Objekt muss von einem oder mehreren Unterzeichnern signiert werden. Mit dieser Funktion wird aus der *PKCS#12*-Datei, die den geheimen Schlüssel sowie das X.509 Zertifikat des Unterzeichners enthält, eine *SignerInfo* Struktur erstellt, mit der das Objekt signiert werden kann. Ausserdem kann mit dem Zertifikat das Feld *issuerAndSerialNumber erzeugt werden*, das dem Empfänger die Eindeutigkeit des Unterzeichners beweist.

Syntax	**int AddSigner(void *pkcs7ctx, BYTE *pkcs12obj, int iLenpkcs12, BYTE *pwd, int lengthpwd);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe

pkcs12Object	Speicheradresse des PKCS#12-Objekts des Unterzeichners.	Eingabe
lengthobject	Länge des PKCS#12-Objekts.	Eingabe
pwd	Speicheradresse des Passwortes, mit dem das PKCS#12-Objekt verschlüsselt wurde.	Eingabe
lengthpwd	Länge des Passwortes.	Eingabe

AddSignerExtern

Die Funktion *AddSignerExtern* arbeitet wie die Funktion *AddSigner* mit dem Unterschied, dass die Signatur nicht von XPS-CryptLib, sondern von einer zu übergebenden Callback-Funktion erstellt wird. Außerdem ist in einer weiteren Callback-Funktion das X.509 Zertifikat an XPS-CryptLib zu übergeben.
Mit dieser Funktion kann erreicht werden, dass die Signatur z. B. von einer Smartcard erstellt wird, für das Erstellen des PKCS#7-Objekts jedoch XPS-CryptLib verantwortlich ist.

Syntax	**int AddSignerExtern(void *pkcs7ctx, void *userdata, int (*SignData)(), int (*ReadX509Cert)());**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
userdata	Speicheradresse eines Benutzerbereichs, der an die Callback Routinen übergeben wird.	Eingabe
*int (*SignData)()*	Callback Routine zum Signieren des PKCS#7-Objektes.	Eingabe
*int (*ReadX509Cert)()*	Callback Routine zum Einlesen des X.509 Zertifikates.	Eingabe

Syntax	**int (*SignData)(void *userdata, BYTE *hash, int iLenHash, BYTE *signature, int *iLenSign);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
userdata	Speicheradresse eines Benutzerbereichs, der an die Funktion *AddSignerExtern* übergeben wurde.	Eingabe
hash	Speicheradresse des zu signierenden Hashwertes.	Eingabe
iLenHash	Länge des zu signierenden Hashwertes.	Eingabe
signature	Speicheradresse der Signatur. Die Callback-Funktion muss die Signatur in diesem Bereich speichern. Die maximale Länge der Signatur beträgt 512 Byte. Eine Signaturlänge von 512 Byte entspricht einem RSA-Key von 4096 Bit.	Ausgabe
iLenSign	Länge der Signatur. Die Länge darf maximal 512 Byte betragen.	Ausgabe

Syntax	**int (*ReadX509Cert)(void *userdata, BYTE **cert, int *iLenCert);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
userdata	Speicheradresse eines Benutzerbereichs, der an die Funktion *AddSignerExtern* übergeben wurde.	Eingabe
cert	Speicheradresse des X.509 Zertifikates.	Ausgabe

iLenCert	Länge des X.509 Zertifikates.	Ausgabe

AddRecipient

Beim Erstellen eines *EnvelopedData* Objekts müssen Informationen über den beabsichtigten Empfänger mit einbezogen werden. Das heißt, die Nachricht wird 'empfänger-spezifisch' verschlüsselt. Mit dieser Funktion wird das X.509 Zertifikat eines Empfängers übergeben. In dem Zertifikat ist der öffentliche Schlüssel enthalten, der zur Verschlüsselung des symmetrischen *Content-Encryption*-Schlüssels verwendet wird.

Syntax	**int AddRecipient(void *pkcs7ctx, BYTE *cert, int lengthcert);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
cert	Speicheradresse des X.509 Zertifikates des Empfängers.	Eingabe
lengthcert	Länge des Zertifikates.	Eingabe

AddSignerCert

Import Funktionen:
Bei Verifizierung von *Signed-data* Objekts sollte die Zertifikatshierarchie der Unterzeichner überprüft werden. Falls die Aussteller-Zertifikate im PKCS#7-Objekt nicht enthalten sind, bietet diese Funktion die Möglichkeit, die Zertifikate an das *Signed-data*-Objekt zu übergeben.
Create Funktionen:
Bei der Funktion *CreateSignedData* werden alle Zertifikate der PKCS#12-Datei der Unterzeichner hinzugefügt. Falls weitere Zertifikate zu dem Objekt hinzugefügt werden sollen, kann dies mit dieser Funktion geschehen.

Syntax	**int AddSignerCert(void *pkcs7ctx, BYTE *cert, int lengthcert);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
cert	Speicheradresse des X.509 Zertifikates des Unterzeichners.	Eingabe
lengthcert	Länge des Zertifikates.	Eingabe

AddTrustedSigner

Bei der Verifizierung von *Signed-data* Objekten kann das Zertifikat des Unterzeichners auf einen vertrauenswürdigen Aussteller (Trusted Signer) überprüft werden. Mit dieser Funktion können vertrauenswürdige Unterzeichner geladen werden.

Syntax	**int AddTrustedSigner(void *pkcs7ctx, BYTE *cert, int lengthcert);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
cert	Speicheradresse des X.509 Zertifikates des vertrauenswürdigen Ausstellers.	Eingabe
lengthcert	Länge des Zertifikates.	Eingabe

ForceTrustedSigner

Mit dieser Funktion können vor der Verifizierung von *Signed-data* Objekten Zertifikate mit der gleichen Identität (Issuer- und Subject-BLOB) ausgetauscht werden, die sich bereits im PKCS#7-Object befinden.

Syntax	**int ForceTrustedSigner(void *pkcs7ctx, BYTE *cert, int lengthcert);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
cert	Speicheradresse des X.509 Zertifikates des vertrauenswürdigen Ausstellers.	Eingabe
lengthcert	Länge des Zertifikates.	Eingabe

GetFirstSigner

Bei *Signed-data* Objekten können die Informationen über die Unterzeichner (*SignerInfos)* des Objekts extrahiert werden. Mit dieser Funktion wird die erste SignerInfo-Struktur ausgelesen.

Syntax	**int GetFirstSigner(void *pkcs7ctx, SIGNERINFO **signer);**	
Returncode	0 falls Unterzeichner vorhanden, sonst kleiner 0.	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
signer	Adresspointer zur Aufnahme der Speicheradresse der ausgelesenen SIGNERINFO. Die Struktur enthält Informationen über Zertifikat, Serial-Nr., Zertifikatsaussteller, Attribute, Message Digest, Digest Algorithm sowie die Signatur.	Ausgabe

GetNextSigner

Bei *Signed-data* Objekten können die Informationen über die Unterzeichner (*SignerInfos)* des Objekts extrahiert werden. Mit dieser Funktion wird die nächste SignerInfo-Struktur ausgelesen.

Syntax	**int GetNextSigner(void *pkcs7ctx, SIGNERINFO **signer);**	
Returncode	0 falls Unterzeichner vorhanden, sonst kleiner 0.	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
signer	Adresspointer zur Aufnahme der Speicheradresse der ausgelesenen SIGNERINFO. Die Struktur enthält Informationen über Zertifikat, Serial-Nr., Zertifikatsaussteller, Attribute, Message Digest, Digest Algorithm sowie die Signatur.	Ausgabe

GetSigningAlgo

Bei *Signed-data* Objekten können die Art der Verschlüsselung und der Hash-Algorithmus des Dokuments extrahiert werden. Diese Funktion benötigt als Eingabe eine SignerInfo-Struktur, die bei Ausführung der Funktionen *GetFirstSigner* und *GetNextSigner* erzeugt wird.

Syntax	**int GetSigningAlgo(void *pkcs7ctx, SIGNERINFO *signer, char *AlgoInfo, int AlgoLen);**	
Returncode	Stringlänge der *AlgoInfo*.	
Parameter	**Beschreibung**	**Verwendung**

pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
signer	Adresspointer der Speicheradresse der zuvor ausgelesenen SIGNERINFO.	Eingabe
AlgoInfo	Speicheradresse an der die gewünschte Information gespeichert werden soll. Die ermittelte Information wird als Zeichenkette zurückgegeben, bei der der Verschlüsselungsalgorithmus und die Hashmethode durch einen Schrägstrich getrennt sind. Beispiel: "rsaEncryption/sha-1".	Ein-/Ausgabe
AlgoLen	Länge des von der *AlgoInfo* belegten Speicherbereichs.	Eingabe

GetSigningTime

Bei *Signed-data* Objekten kann der Zeitpunkt der Signatur des Dokuments extrahiert werden. Diese Funktion benötigt als Eingabe eine SignerInfo-Struktur, die bei Ausführung der Funktionen *GetFirstSigner* und *GetNextSigner* erzeugt wird.

Syntax	**int GetSigningTime(void *pkcs7ctx, SIGNERINFO *signer, char *SigningTime, int SigningTimeLen);**	
Returncode	Stringlänge der *SigningTime*.	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
signer	Adresspointer der Speicheradresse der zuvor ausgelesenen SIGNERINFO.	Eingabe
SigningTime	Speicheradresse an der die gewünschte Information gespeichert werden soll. Der Zeitpunkt der Erstellung der Signatur wird im folgenden Format zurückgegeben: "YYMMDDHHMMZ". Die einzelnen Positionen haben dabei die folgenden Bedeutungen: YY letzte 2 Stellen des Jahres MM Monat (01 bis 12) DD Tag des Monats (01 bis 31) HH Stunde (00 bis 23) MM Minuten (00 bis 59) Z Das Zeichen "Z" bezeichnet Greenwich Mean Time (GMT).	Ein-/Ausgabe
SigningTimeLen	Länge des von der *SigningTime* belegten Speicherbereichs.	Eingabe

GetNextSignerCert

Mit dieser Funktion wird das nächste Ausstellerzertifikat des aktuellen Unterzeichners ausgelesen.

Syntax	**int GetNextSignerCert(void *pkcs7ctx, BYTE **cert);**	
Returncode	0 falls kein Zertifikat mehr vorhanden, sonst Länge des Zeritifikats.	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
cert	Adresspointer zur Aufnahme der Speicheradresse der ausgelesenen Unterzeichner Zertifikats.	Ausgabe

VerifySigner

Das *Signed-data* Objekt wird auf Gültigkeit überprüft. Bei dieser Funktion wird jedoch nicht die Gesamtheit überprüft, sondern nur die Gültigkeit dieses bestimmten Unterzeichners. Die SIGNERINFO-Struktur muss vorher mit der Funktion *GetFirstSigner/GetNextSigner* ermittelt worden sein.

Syntax	**int VerifySigner(void *pkcs7ctx, SIGNERINFO **signer, int checkcertchain);**	
Returncode	0 falls Unterzeichner verifiziert, sonst Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
signer	Adresspointer der Speicheradresse der zu verifizierenden SIGNERINFO Struktur.	Eingabe
checkcertchain	Bei Angabe von 0 werden der Zertifikatspfad des Unterzeichners sowie vertrauenswürdige Aussteller nicht überprüft. Bei Angabe von 1 wird der Zertifikatspfad bis zum Wurzelzertifikat überprüft. Ausserdem muss der Zertifikatsaussteller mit der Funktion *AddTrustedSigner* vorher geladen worden sein.	Eingabe

VerifyAllSigner

Alle Unterzeichner des *Signed-data* Objekts werden auf Gültigkeit überprüft.

Syntax	**int VerifyAllSigner(void *pkcs7ctx, int checkcertchain);**	
Returncode	0 falls alle Unterzeichner verifiziert, sonst Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
checkcertchain	Bei Angabe von 0 werden der Zertifikatspfad des Unterzeichners sowie vertrauenswürdige Aussteller nicht überprüft. Bei Angabe von 1 wird der Zertifikatspfad bis zum Wurzelzertifikat überprüft. Ausserdem muss der Zertifikatsaussteller mit der Funktion *AddTrustedSigner* vorher geladen worden sein.	Eingabe

GetFirstPKCS7Data

Die Daten des PKCS#7-Objekts werden ausgelesen. Diese Funktion ist bei allen unterstützten PKCS#7-Typen vorhanden.

Syntax	**int GetFirstPKCS7Data(void *pkcs7ctx, BYTE **data);**	
Returncode	Länge der Daten, bei 0 keine Daten vorhanden, sonst Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
data	Adresspointer zur Aufnahme der Speicheradresse der ausgelesenen Daten.	Ausgabe

GetNextPKCS7Data

Die nächsten Daten des PKCS#7-Objekts werden ausgelesen. Diese Funktion ist bei allen unterstützten PKCS#7-Typen vorhanden.

Syntax	**int GetNextPKCS7Data(void *pkcs7ctx, BYTE **data);**	
Returncode	Länge der Daten, bei 0 keine Daten mehr vorhanden, sonst Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**

pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
data	Adresspointer zur Aufnahme der Speicheradresse der ausgelesenen Daten.	Ausgabe

CreateObject

Mit dieser Funktion wird das Erstellen von PKCS#7-Objekten abgeschlossen.

Syntax	**int CreateObject (void *pkcs7ctx, BYTE **object);**	
Returncode	Länge des PKCS#7-Datenobjekts oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe
data	Adresspointer zur Aufnahme der Speicheradresse des erstellten PKCS#7-Objekts.	Ausgabe

CleanupPKCS7

Freigeben des von den PKCS#7-Routinen benötigten Speichers.

Syntax	**int CleanupPKCS7(void *pkcs7ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs7ctx	Speicheradresse des PKCS#7-Kontexts.	Eingabe

Beispiel (Create PKCS7Data):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
/*                                                              *
 * write PKCS#7 Data object                                     *
 *                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  BYTE       *object;
  void       *ctx          = {0};
  int         iRc;
  int         option       = HEADER_INCLUDED;
  char        data1[]      = "XPS Software GmbH";
  char        data2[]      = "Muenchner Str. 17";
  char        filename[]   = "pk7data.p7m";

  iRc = CreatePKCS7Data( option, &ctx );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  AddPKCS7Data( ctx, data1, strlen(data1) );
  AddPKCS7Data( ctx, data2, strlen(data2) );

  iRc = CreateObject( ctx, &object );
  while( iRc < 0 )
  {
    printf( "Object invalid: rc=%d\n", iRc );
    return( iRc );
  }
```

```
  writeFile( filename, object, iRc, 0 );

  CleanupPKCS7( ctx );

  return( 0 );
}
```

Beispiel (Read PKCS7Data):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

#define true  1
#define false 0

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * extract PKCS#7 Data object                                   *
 *                                                              *
 *                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  char       *buffer;
  char       *data;
  void       *ctx         = {0};
  int         iRc, iCount;

  iCount = readFile( argv[1], &buffer );
  if( iCount == 0 )
  {
    printf( "file \"%s\" not found\n", argv[1] );
    return( -1 );
  }

  iRc = ImportPKCS7Data( buffer, iCount, &ctx );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = GetFirstPKCS7Data( ctx, &data );
  while( iRc )
  {
    DumpData( "PKCS#7-Data:", data, iRc );
    iRc = GetNextPKCS7Data( ctx, &data );
  }

  CleanupPKCS7( ctx );
  return( 0 );
}
```

Beispiel (Create SignedData):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * write PKCS#7 signedData object                               *
 *                                                              *
 * Parameter-1: pkcs#12-file  privateKey Signer   (xpsuser1.p12) *
 * Parameter-2: password pkcs#12-privateKey file  (xpsuser1)    *
 *                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  BYTE       *buffer, *object;
  void       *ctx, *pemctx;
  BYTE       *PEM;
  int         iRc, iCount1;
```

```
  int         option       = DATA_IMPLICIT | CERT_IMPLICIT | HEADER_INCLUDED;
  char        filename[]   = "pk7sd.p7m";
  char        data1[]      = "XPS Software GmbH";
  char        data2[]      = "Muenchner Str. 17";
  char        data3[]      = "85540 Haar";

  iCount1 = readFile( argv[1], &buffer );
  if( iCount1 == 0 )
  {
    printf( "pkcs#12-object \"%s\" not found\n", argv[1] );
    return( -1 );
  }

  iRc = CreateSignedData( option, &ctx );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  AddPKCS7Data( ctx, data1, sizeof(data1)-1 );
  AddPKCS7Data( ctx, data2, sizeof(data2)-1 );
  AddPKCS7Data( ctx, data3, sizeof(data3)-1 );

  iRc = AddSigner( ctx, buffer, iCount1, argv[2], sizeof(argv[2]) );
  if( iRc != 0 )
  {
    printf( "Signer invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = CreateObject( ctx, &object );
  while( iRc < 0 )
  {
    printf( "Object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = ASN2PEM( object, iRc, filename, &PEM, &pemctx );
  writeFile( filename, PEM, iRc, createFile );

  CleanupFile( buffer );
  CleanupPEM( pemctx );
  CleanupPKCS7( ctx );

  return( 0 );
}
```

Beispiel (Read SignedData):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

#define true  1
#define false 0


/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * read PKCS#7 signedData object                                *
 *                                                              *
 * Parameter-1: pkcs#7-signedData file           (pk7sd.p7m)    *
 * Parameter-2: X.509 Certificate trustedSigner  (xpstest.cer)  *
 *                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  char       *buffer;
  char       *cbuffer;
  char       *data;
  void       *ctx         = {0};
  int         iRc, iCount1, iCount2;
  PSIGNERINFO signer;

  if( argc < 3 )
  {
    printf( "Parameter missing: xpsread2 pk7sd.p7m xpstest.cer\n" );
    return( -1 );
  }

  /* read pkcs#7-object */
```

```
  iCount1 = readFile( argv[1], &buffer );
  if( iCount1 == 0 )
  {
    printf( "pkcs#7-file \"%s\" not found\n", argv[1] );
    return( -1 );
  }

  /* read trusted signer */
  iCount2 = readFile( argv[2], &cbuffer );
  if( iCount2 == 0 )
  {
    printf( "trusted-signer \"%s\" not found\n", argv[2] );
    return( -1 );
  }

  iRc = ImportSignedData( buffer, iCount1, &ctx );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  AddTrustedSigner( ctx, cbuffer, iCount2 );

  iRc = GetFirstSigner( ctx, &signer );
  if( iRc < 0 )
    return( iRc );

  while( iRc == 0 )
  {
    DumpData( "Signer Certificate:", signer->pCert, signer->iLenCert );
    iRc = GetNextSigner( ctx, &signer );
  }

  iRc = VerifyAllSigners( ctx, true );
  printf( "\nVerify All Signers: rc=%d\n", iRc );

  iRc = VerifySigner( ctx, signer, true );
  printf( "\nVerify Last Signer: rc=%d\n", iRc );

  iRc = GetFirstPKCS7Data( ctx, &data );
  while( iRc )
  {
    DumpData( "PKCS#7-Data:", data, iRc );
    iRc = GetNextPKCS7Data( ctx, &data );
  }

  CleanupPKCS7( ctx );
  return( 0 );
}
```

Beispiel (Create EnvelopedData):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * write PKCS#7 envelopedData object                            *
 *                                                              *
 * Parameter-1: X.509-file  publicKey Recipient   (xpsuser1.cer) *
 *                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  BYTE       *object;
  void       *ctx, *pemctx;
  BYTE       *cert;
  BYTE       *PEM;
  int         iRc, iCount1;
  int         option        = HEADER_INCLUDED;
  char        data1[]       = "XPS Software GmbH";
  char        data2[]       = "Muenchner Str. 17";
  char        data3[]       = "85540 Haar";
  char        filename[]    = "pk7env.p7m";

  iCount1 = readFile( argv[1], &cert );
  if( iCount1 == 0 )
  {
    printf( "x509-cert \"%s\" not found\n", argv[1] );
    return( -1 );
```

```
  }

  /* iRc = CreateEnvelopedData( option, &ctx, rc2CBC_128 ); */
  iRc = CreateEnvelopedData( option, &ctx, desEDE3CBC );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  AddPKCS7Data( ctx, data1, strlen(data1) );
  AddPKCS7Data( ctx, data2, strlen(data2) );
  AddPKCS7Data( ctx, data3, strlen(data3) );

  iRc = AddRecipient( ctx, cert, iCount1 );
  if( iRc != 0 )
  {
    printf( "Recipient invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = CreateObject( ctx, &object );
  while( iRc < 0 )
  {
    printf( "Object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = ASN2PEM( object, iRc, filename, &PEM, &pemctx );
  writeFile( filename, PEM, iRc, createFile );

  CleanupFile( cert );
  CleanupPEM( pemctx );
  CleanupPKCS7( ctx );

  return( 0 );
}
```

Beispiel (Read EnvelopedData):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

#define true  1
#define false 0

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * decrypt envelopedData object:                                *
 *                                                              *
 * Parameter-1: pkcs#7 envelopedData file          (pk7env.p7m)   *
 * Parameter-2: pkcs#12-privateKey file (Recipient) (xpsuser1.p12) *
 * Parameter-3: password privateKey file            (xpsuser1)     *
/*                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  char       *pkcs7obj;
  char       *pkcs12obj;
  char       *data;
  void       *ctx         = {0};
  int         iRc, iCount1, iCount2;
  char        pw[32]      = {0};


  if( argc < 4 )
  {
    printf( "Parameter missing: xpsread3 pk7env.p7m p12-file password\n" );
    return( -1 );
  }

  strcpy( pw, argv[3] );
  iCount1 = readFile( argv[1], &pkcs7obj );           // pkcs7-file
  if( iCount1 == 0 )
  {
    printf( "file \"%s\" not found\n", argv[1] );
    return( -1 );
  }

  iCount2 = readFile( argv[2], &pkcs12obj );          // pkcs12-file
  if( iCount2 == 0 )
```

```
  {
    printf( "file \"%s\" not found\n", argv[2] );
    return( -1 );
  }

  iRc = ImportEnvelopedData( pkcs7obj, iCount1, pkcs12obj, iCount2,
                             pw, sizeof(pw), &ctx );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = GetFirstPKCS7Data( ctx, &data );
  while( iRc )
  {
    DumpData( "PKCS#7-Data:", data, iRc );
    iRc = GetNextPKCS7Data( ctx, &data );
  }

  CleanupPKCS7( ctx );
  return( 0 );
}
```

Beispiel (Create EncryptedData):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * write PKCS#7 encryptedData object                            *
 *                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  BYTE       *object;
  void       *ctx, *pemctx;
  BYTE       *PEM;
  int         iRc;
  int         option       = HEADER_INCLUDED;
  char        data1[]      = "XPS Software GmbH";
  char        data2[]      = "Muenchner Str. 17";
  char        data3[]      = "85540 Haar";
  char        filename[]   = "pk7enc.p7m";
  char        password[]   = "testpassword";

  iRc = CreateEncryptedData( option, &ctx, pbe3DES_3Key,
                             password, strlen(password) );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  AddPKCS7Data( ctx, data1, strlen(data1) );
/*  AddPKCS7Data( ctx, data2, strlen(data2) ); */
/*  AddPKCS7Data( ctx, data3, strlen(data3) ); */

  iRc = CreateObject( ctx, &object );
  while( iRc < 0 )
  {
    printf( "Object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = ASN2PEM( object, iRc, filename, &PEM, &pemctx );
  writeFile( filename, PEM, iRc, createFile );

  CleanupPKCS7( ctx );
  CleanupPEM( pemctx );

  return( 0 );
}
```

Beispiel (Read EncryptedData):

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include "xpscrypt.h"

#define true  1
#define false 0

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * decrypt encryptedData object:                                *
 *                                                              *
 * Parameter-1: pkcs#7 encryptedData file           (pk7enc.p7m)   *
/*                                                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  char       *buffer;
  char       *data;
  void       *ctx          = {0};
  int         iRc, iCount;
  char        password[]   = "testpassword";

  iCount = readFile( argv[1], &buffer );
  if( iCount == 0 )
  {
    printf( "file \"%s\" not found\n", argv[1] );
    return( -1 );
  }

  iRc = ImportEncryptedData( buffer, iCount, password, strlen(password), &ctx );
  if( iRc != 0 )
  {
    printf( "pkcs#7-object invalid: rc=%d\n", iRc );
    return( iRc );
  }

  iRc = GetFirstPKCS7Data( ctx, &data );
  while( iRc )
  {
    DumpData( "PKCS#7-Data:", data, iRc );
    iRc = GetNextPKCS7Data( ctx, &data );
  }

  CleanupPKCS7( ctx );
  return( 0 );
}
```

Kapitel

8

PKCS#12 private Key

Allgemein

PKCS#12-Objekte (*Personal-Information-Exchange-Syntax-Standard*) stellen eine Syntax für den Austausch von Schlüsseln und Zertifikaten zur Verfügung. Ein PKCS#12-Objekt enthält Schlüsseltaschen (key-bags) und Zertifikatstaschen (certificate-bags). PKCS#12 ist der Standard zum sicheren Speichern von Privaten Schlüsseln und Zertifikaten. Es wird vor allem von Internet Browsern wie Netscape (Exportformat .p12) und Microsoft Internet Explorer (Exportformat .pfx) verwendet.

Funktionen

ImportPKCS12

Einlesen eines PKCS#12-Objekts sowie Prüfung auf formale Richtigkeit.

Syntax	**int ImportPKCS12(BYTE *pkcs12object, int lengthobject, BYTE *pwd, int lengthpwd, void **pkcs12ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs12Object	Speicheradresse des PKCS#12-Objekts.	Eingabe
lengthobject	Länge des PKCS#12-Objekts.	Eingabe
pwd	Speicheradresse des Passwortes, mit dem das PKCS#12-Objekts verschlüsselt wurde.	Eingabe
lengthpwd	Länge des Passwortes.	Eingabe
pkcs12ctx	Adresspointer zur Aufnahme der Speicheradresse des eingelesenen PKCS#12-Objekts. Dieses wird für die weitere Bearbeitung des Objekts benötigt.	Ausgabe

GetPrivateKey

Extrahieren des geheimen Schlüssels aus dem PKCS#12-Objekt.

Syntax	**int GetPrivateKey(void *pkcs12ctx, RSA_PRIVATE_KEY *privkey);**	
Returncode	Länge der privateKey Struktur oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs12ctx	Speicheradresse des PKCS#12-Kontexts.	Eingabe
privkey	Speicheradresse, an der der extrahierte geheime Schlüssel gespeichert	Ausgabe

	werden soll.	

GetFirstCert

Extrahieren des Benutzerzertifikates aus dem PKCS#12-Objekt.

Syntax	**int GetFirstCert(void *pkcs12ctx, BYTE **x509cert);**	
Returncode	Länge des X.509 Zertifikates oder 0, falls kein Zertifikat vorhanden.	
Parameter	**Beschreibung**	**Verwendung**
pkcs12ctx	Speicheradresse des PKCS#12-Kontexts.	Eingabe
x509cert	Adresspointer zur Aufnahme der Speicheradresse des extrahierten X.509 Zertifikats.	Ausgabe

GetNextCert

Extrahieren des nächsten Zertifikates aus dem PKCS#12-Objekt. Ein PKCS#12-Objekt kann neben dem Benutzerzertifikat sämtliche Ausstellerzertifikate bis hin zum Wurzelzertifikat enthalten.

Syntax	**int GetNextCert (void *pkcs12ctx, BYTE **x509cert);**	
Returncode	Länge des X.509 Zertifikates oder 0, falls kein Zertifikat mehr vorhanden.	
Parameter	**Beschreibung**	**Verwendung**
pkcs12ctx	Speicheradresse des PKCS#12-Kontexts.	Eingabe
x509cert	Adresspointer zur Aufnahme der Speicheradresse des extrahierten X.509 Zertifikats.	Ausgabe

CleanupPKCS12

Freigeben des bei den PKCS#12-Routinen benötigten Speichers.

Syntax	**int CleanupPKCS12(void *pkcs12ctx);**	
Returncode	0 oder Fehlercode (< 0).	
Parameter	**Beschreibung**	**Verwendung**
pkcs12ctx	Speicheradresse des PKCS#12-Kontexts.	Eingabe

Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "XPSCRYPT.H"

/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * get private key from PKCS#12-File (.p12)                     *
 *                                                              *
 * Parameter 1: Name of PKCS#12-File        (xpsuser1.p12)      *
 * Parameter 2: Password                    (xpsuser1)          *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
// ------------------------------------------------------------ //
{
  RSA_PRIVATE_KEY privkey   = {0};
  BYTE           *p12obj    = 0;
```

```
  BYTE           *x509Cert  = 0;
  void           *ctx;
  int             iCount, iRc;

  if( argc < 3 )
  {
    printf( "Parameter missing: pk12test p12-file password\n" );
    return( -1 );
  }

  iCount = readFile( argv[1], &p12obj );
  if( iCount < 1 )
  {
    printf( "PKCS12 Object \"%s\" not found.\n", argv[1] );
    return( -1 );
  }

  iRc = ImportPKCS12( p12obj, iCount, argv[2], strlen(argv[2]), &ctx );
  if( iRc != 0 )
  {
    printf( "PKCS#12 Error: rc=%d\n", iRc );
    return( -1 );
  }

  iRc = GetPrivateKey( ctx, &privkey );
  DumpData( "Private Key:", (BYTE *)&privkey, sizeof(privkey) );

  iCount = GetFirstCert( ctx, &x509Cert );

  while( iCount != 0 )
  {
    DumpData( "X.509 Certificate:", x509Cert, iCount );
    iCount = GetNextCert( ctx, &x509Cert );
  }

  CleanupPKCS12( ctx );
  CleanupFile( p12obj );
  return 0;
}
```

Kapitel

9

SSL/TLS

Allgemein

Secure Sockets Layer (SSL) oder Transport Layer Security (TLS) ist ein hybrides Verschlüsselungsprotokoll für sichere Datenübertragungen im Internet. TLS ist die standardisierte Weiterentwicklung von SSL 3.0 (TLS 1.0 steht neu für SSL 3.1). SSL wird unter dem Namen TLS weiterentwickelt. Hier wird die Abkürzung SSL für beide Bezeichnungen verwendet.

SSL wurde von der Firma Netscape entwickelt. Das SSL-Protokoll gewährleistet, dass Daten während der Übertragung nicht gelesen oder manipuliert werden können.

Funktionen

SSL_Init

Initialisieren des SSL-Kontexts. Mit dieser Funktion wird festgelegt, ob eine Client- oder eine Server-Session aufgebaut werden soll. Bei einer Client-Session kann das Protokoll (SSL oder TLS) festgelegt werden. Bei einer Server-Session kann bestimmt werden, ob von den Clients ein Zertifikat verlangt werden soll oder nicht.

Syntax	**int SSL_Init(BYTE **ctx, int iProtocolSide, int iOption);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Adresspointer zur Aufnahme der Speicheradresse des SSL-Objekts. Dieses wird für die weitere Verarbeitung des Objekts benötigt.	Ausgabe
iProtocolSide	Hier wird festgelegt, ob es sich um einen SSL-Client oder einen SSL-Server handelt. Folgende Werte sind möglich: **SSL_ClientSide**: Es soll eine SSL-Client Verbindung aufgebaut werden. **SSL_ServerSide**: Es soll eine SSL-Server Verbindung aufgebaut werden.	Eingabe
iOption	**bei iProtocolSide = SSL_ClientSide:** Verarbeitungsprotokoll SSL_Version_3_0 oder TLS_Version_1_0. **bei iProtocolSide = SSL_ServerSide:** TRUE(1) – Ein Client-Zertifikat wird verlangt. FALSE(0) – Ein Client-Zertifikat wird nicht verlangt.	Eingabe

SSL_Set_PrivateKey

Privaten Schlüssel für die SSL-Verbindung zuordnen.

Diese Funktion ist sowohl für eine Client- als auch für eine Server-Session zwingend nötig, da beim SSL-Handshake ein RSA public-/private-Key Austausch erfolgt. XPS-CryptLib kann sowohl .p12 als auch .pfx-Dateien verarbeiten.

Syntax	**int SSL_Set_PrivateKey(BYTE **ctx, BYTE *pkcs12obj, int iLenpkcs12, BYTE *pwd, int lengthpwd);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
pkcs12obj	Speicheradresse des PKCS#12-Objekts.	Eingabe
iLenpkcs12	Länge des PKCS#12-Objekts.	Eingabe
pwd	Speicheradresse des Passwortes, mit dem das PKCS#12-Objekt verschlüsselt wurde.	Eingabe
lengthpwd	Länge des Passwortes.	Eingabe

SSL_Add_x509Cert

Ein weiteres X.509 Zertifikat hinzufügen.

Beim SSL-Handshake werden Zertifikate mit der Gegenseite ausgetauscht. XPS-CryptLib sendet alle Zertifikate, die in der pkcs12-Datei, die der Session mit der Funktion SSL_Set_PrivateKey zugeordnet wurde, vorhanden sind. Fehlen hier jedoch Zertifikate (z. B. Root-Zertifikat), können diese mit dieser Funktion hinzugefügt werden.

Diese Funktion ist sowohl für Client- als auch für Server-Verbindungen möglich. Server-Verbindungen senden immer ihre Zertifikate, Client-Verbindungen nur, wenn sie vom Server dazu aufgefordert werden.

Syntax	**int SSL_Add_x509Cert(BYTE **ctx, BYTE *pX509, int iLenX509);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
pX509	Speicheradresse des X.509 Zertifikates.	Eingabe
iLenX509	Länge des X.509 Zertifikates.	Eingabe

SSL_Set_Cipher

Ein SSL-Client bestimmt seine symmetrische Verschlüsselung.

Nach dem Handshake werden beim SSL-Protokoll die Daten mit einem symmetrischen Verfahren verschlüsselt. Der SSL-Client kann durch mehrmaliges Aufrufen dieser Methode seine bevorzugten Verschlüsselungsroutinen bestimmen. Der SSL-Server wird dann die erste Routine, die er unterstützt, zum Verschlüsseln auswählen.

Syntax	**int SSL_Set_Cipher(BYTE **ctx, CipherSuite Cipher);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
Cipher	Folgende Ciphers werden unterstützt:	Eingabe

	SSL_RSA_WITH_RC4_MD5_EXP SSL_RSA_WITH_RC4_128_MD5 SSL_RSA_WITH_RC4_128_SHA SSL_RSA_WITH_RC2_CBC_MD5_EXP SSL_RSA_WITH_RC2_CBC_128_MD5 SSL_RSA_WITH_DES_CBC_MD5 SSL_RSA_WITH_DES_CBC_SHA SSL_RSA_WITH_3DES_EDE_CBC_SHA SSL_RSA_WITH_3DES_EDE_CBC_MD5 TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA	

SSL_Add_DN

Erlaubte Sessionpartner bestimmen.

Mit dieser optionalen Funktion kann sowohl der SSL-Client als auch der SSL-Server bestimmen mit welchen Partnern eine Verbindung aufgebaut werden darf.

Bei einer Client-Verbindung werden nur Server zugelassen, die als Zertifikats-Aussteller mit dieser Funktion geladen wurden.

Bei einer Server-Verbindung werden nur Clients zugelassen, die von diesem Zertifikats-Aussteller signiert wurden.

Falls diese Funktion nicht ausgeführt wird, wird keine Prüfung der Zertifikate unternommen.

Syntax	**int SSL_Add_DN(BYTE **ctx, BYTE *pX509, int iLenX509);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
pX509	Speicheradresse des X.509 Zertifikates.	Eingabe
iLenX509	Länge des X.509 Zertifikates.	Eingabe

SSL_Handshake

Sessionkey mit Partner aushandeln.

Der symmetrische Algorithmus wird bestimmt und der zu verwendende Schlüssel wird ausgehandelt. Diese Funktion muss sowohl der SSL-Client als auch der SSL-Server ausführen. Der SSL-Client muss dabei den TCP/IP-Socket, den er mit ***connect*** verbunden hat, als Parameter übergeben. Der SSL-Server muss den TCP/IP-Socket, für den er den ***accept*** ausgeführt hat, übergeben.

Syntax	**int SSL_Handshake(BYTE **ctx, SOCKET socket);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
socket	TCP/IP-socket mit dem die Verbindung hergestellt wurde.	Eingabe

SSL_Get_Peer_Cert

Extrahieren des Partner-Benutzerzertifikates.

Syntax	int SSL_Get_Peer_Cert(BYTE **ctx, BYTE **pX509);	
Returncode	Länge der Zertifikats oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
pX509	Adresspointer zur Aufnahme der Speicheradresse des ausgelesenen Zertifikats.	Ausgabe

SSL_GetNext_Peer_Cert

Extrahieren des nächsten Partner-Benutzerzertifikates.

Syntax	int SSL_GetNext_Peer_Cert(BYTE **ctx, BYTE **pX509);	
Returncode	Länge der Zertifikats oder 0 (kein Zertifikat mehr vorhanden) oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
pX509	Adresspointer zur Aufnahme der Speicheradresse des ausgelesenen Zertifikats.	Ausgabe

SSL_Read

Daten lesen, entschlüsseln und auf Gültigkeit überprüfen.

Syntax	int SSL_Read(BYTE **ctx, BYTE *data, int iLength);	
Returncode	Länge der Daten oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
data	Speicheradresse der gelesenen und entschlüsselten Daten.	Eingabe
iLength	Länge der entschlüsselten Daten.	Eingabe

SSL_Write

Daten verschlüsseln, signieren und zum Partner übertragen.

Syntax	int SSL_Write(BYTE **ctx, BYTE *data, int iLength);	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
data	Speicheradresse der zu übertragenden Daten.	Eingabe
iLength	Länge der zu übertragenden Daten.	Eingabe

SSL_Close_Session

Schließen der SSL-Session.

Session-bezogene Daten werden freigegeben, der TCP/IP-Socket bleibt geöffnet. Der Sessionkey wird verworfen. Bei Client-Verbindungen kann mit SSL_Resume_Session mit diesem Socket eine neue SSL-Verbindung aufgebaut werden.

Syntax	**int SSL_Close_Session(BYTE **ctx);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe

SSL_Resume_Session

Wiederherstellung der SSL-Session.

Client-Verbindungen können Sessions, die mit SSL_Close_Session beendet wurden, mit dieser Funktion wiederherstellen, d.h. es wird mit den bereits vorhandenen Daten ein neuer Handshake durchgeführt.

Syntax	**int SSL_Resume_Session(BYTE **ctx);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
socket	TCP/IP-Socket mit dem die Verbindung hergestellt wurde.	Eingabe

SSL_Cleanup

Freigabe der SSL-Session.

Sämtliche Speicherbereiche der SSL-Session werden freigegeben. Der TCP/IP-Socket wird nicht geschlossen.

Syntax	**int SSL_Cleanup(BYTE **ctx);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe

SSL_Get_Last_Error

Mit dieser Funktion kann nach dem Auftreten eines Fehlers die zugehörige Nachricht extrahiert werden.

Syntax	**int SSL_Get_Last_Error(BYTE **ctx, BYTE *msg, int msglen);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des SSL-Kontexts.	Eingabe
msg	Speicheradresse des Nachrichtenbereichs.	Eingabe
msglen	Länge des Nachrichtenbereichs.	Eingabe

Beispiel SSL-Client:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock.h>
#include <fcntl.h>
#include <io.h>
#include <sys/stat.h>
#include <time.h>

#include "xpscrypt.h"

int main( int argc, char* argv[] )
{
  int     iRc, i1;
  SOCKET  sockfd;
  WSADATA wsadata      = {0};
  short   sServerPort  = 5555;
  int    *piTemp;
  BYTE   *ctx;
  BYTE    buffer[4096]  = {0};

  struct  sockaddr_in   addr;
  struct  hostent *he;

  BYTE   *p12obj      = 0;
  BYTE   *x509Cert    = 0;
  BYTE   *pcert       = 0;
  char    pwd[8]      = "xpsuser1";
  char    szData01[8] = "test";
  int     protocol    = TLS_Version_1_0;

  /*************************************************/
  /* Initialize the socket address structure       */
  /*************************************************/
  iRc =  (short)WSAStartup( 0x0101, &wsadata );

  he = gethostbyname("localhost");

  addr.sin_family      = AF_INET;
  piTemp               = (int *)*he->h_addr_list;
  addr.sin_addr.s_addr = *piTemp;
  addr.sin_port        = htons(sServerPort);

  /*************************************************/
  /* Create an AF_INET stream socket               */
  /*************************************************/
  sockfd = socket( PF_INET, SOCK_STREAM, IPPROTO_TCP );
  if (sockfd < 0)
  {
   printf("socket() failed");
      return (-1);
  }

  /*************************************************/
  /* Connect to the server                         */
  /*************************************************/
  iRc = connect(sockfd,
              (struct sockaddr *)&addr,
              sizeof(struct sockaddr));
  if( iRc < 0 )
  {
   printf("connect() failed");
      return(-1);
  }

  i1 = readFile( "xpsuser1.p12", &p12obj );
  if( i1 < 1 )
  {
    printf( "PKCS12 Object \"%s\" not found.\n", "xpsuser1.p12" );
    return( -1 );
  }

  SSL_Init( &ctx, SSL_ClientSide, protocol );

  SSL_Set_Cipher( &ctx, TLS_RSA_WITH_AES_256_CBC_SHA );
  SSL_Set_Cipher( &ctx, TLS_RSA_WITH_AES_128_CBC_SHA );
  SSL_Set_Cipher( &ctx, SSL_RSA_WITH_3DES_EDE_CBC_SHA );

  iRc = SSL_Set_PrivateKey( &ctx, p12obj, i1, pwd, strlen(pwd) );
  if( iRc != 0 )
    return( iRc );
  CleanupFile( p12obj );

  iRc = SSL_Handshake( &ctx, sockfd );
```

```
  if( iRc != 0 )
  {
    char szTemp[80];

    SSL_Get_Last_Error( &ctx, szTemp, sizeof(szTemp) );
    printf("%s\n", szTemp);
    return( iRc );
  }

  iRc = SSL_Get_Peer_Cert( &ctx, &pcert );
  while( iRc > 0 )
  {
    DumpData( "X.509 Cert", pcert, iRc );
    iRc = SSL_GetNext_Peer_Cert( &ctx, &pcert );
  }

  iRc = SSL_Write( &ctx, (char *)&szData01, 5 );
  if( iRc < 0 )
  {
    printf("send() failed");
       return(-1);
  }
  printf("write: %s\n", szData01 );

  iRc = SSL_Read( &ctx, buffer, sizeof(buffer) );
  if( iRc <= 0 )
    return( iRc );
  printf( "read: %s - Len=%d\n", buffer, iRc );

  SSL_Close_Session( &ctx );
  closesocket(sockfd);
  SSL_Cleanup( &ctx );

  WSACleanup();
  printf("Socket  e n d e d\n");
  return( 0 );
}
```

Beispiel SSL-Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock.h>
#include <time.h>
#include "xpscrypt.h"

int main( int argc, char* argv[] )
{
  int     iRc, i1;
  SOCKET  sd, asd;
  WSADATA wsadata               = {0};
  short   sServerPort           = 5555;

  struct  sockaddr_in   addr     = {0};
  struct  sockaddr      addr_acc = {0};
  char    szInput[100]           = {0};
  char    szOutput[100]          = {0};
  BYTE   *p12obj                 = 0;
  BYTE   *x509Cert               = 0;
  char    pwd[16]                = "xpssrv";
  char    file[32]               = "xpssrv.p12";
  int     iReadLen;
  int     iLen                   = sizeof(addr_acc);
  BYTE   *ctx;
  char    szTemp[80];
  int     iOption                = 1;

  /*************************************************/
  /* Initialize the socket address structure       */
  /*************************************************/
  iRc =  (short)WSAStartup( 0x0101, &wsadata );
  memset(&addr, 0, sizeof(addr));
  addr.sin_family      = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port        = htons(sServerPort);

  sd = socket( PF_INET, SOCK_STREAM, IPPROTO_TCP );
  if (sd < 0)
  {
   printf("socket() failed");
    return(-1);
  }
  iRc = bind( sd, (struct sockaddr *)&addr, sizeof(addr));
  if (iRc < 0)
  {
   printf("bind() failed");
```

```
    return(-1);
  }

  iRc = listen( sd,1000 );
  if (iRc != 0)
  {
    printf("listen() failed");
    return(-1);
  }

//  iOption |= 0x80000000;
  SSL_Init( &ctx, SSL_ServerSide, iOption );

  i1 = readFile( file, &p12obj );
  if( i1 < 1 )
  {
    printf( "PKCS12 Object \"%s\" not found.\n", file );
    return( -1 );
  }
  iRc = SSL_Set_PrivateKey( &ctx, p12obj, i1, pwd, strlen(pwd) );
  if( iRc != 0 )
    return( iRc );
  CleanupFile( p12obj );

  printf("SSL Server started at port %d\n", sServerPort);

  while (1)
  {
    asd = accept( sd, &addr_acc, &iLen );
    if (asd == 0)
    {
      printf("accept() failed\n");
      return(-1);
    }
    printf("accept() ok sd=%d\n",asd);


    iRc = SSL_Handshake( &ctx, asd );
    if( iRc != 0 )
    {
      char szTemp[80];

      SSL_Get_Last_Error( &ctx, szTemp, sizeof(szTemp) );
      printf("%s\n", szTemp);
    }
    else
    {
      while( TRUE )
      {
        memset( szInput, 0, sizeof(szInput) );
        iReadLen = SSL_Read( &ctx, (char *)szInput, sizeof(szInput) );
        if( iReadLen < 0 )
        {
          SSL_Get_Last_Error( &ctx, szTemp, sizeof(szTemp) );
          printf("%s\n", szTemp);
             break;
        }

        if( iReadLen == 0 )
          continue;

        sprintf(szOutput, "ECHO: %s", szInput );
        printf("%s\n", szOutput);

        iRc = SSL_Write( &ctx, (char *)szOutput, strlen(szOutput) );
        if( iRc < 0 )
        {
          SSL_Get_Last_Error( &ctx, szTemp, sizeof(szTemp) );
          printf("%s\n", szTemp);
             break;
        }
      }

      closesocket(asd);
      printf("close sd=%d\n",asd);
    }

  }

  /*********************************************/
  /* Close down the socket                     */
  /*********************************************/
  closesocket(sd);
  SSL_Cleanup( &ctx );
  WSACleanup();
  return( 0 );
}
```

Kapitel

10

GZIP

Allgemein

CryptLib bietet die Möglichkeit mit der Funktion *gzip* Daten zu komprimieren sowie mit der Funktion *gunzip* die Daten wieder zu entpacken.

Funktionen

gzip

Daten werden im gzip-Format gepackt.

Syntax	**int gzip(char *input, int inputlength, char **output, int *outputlength, char *fileName);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
input	Speicheradresse der zu komprimierenden Daten.	Eingabe
inputlength	Länge der zu komprimierenden Daten.	Eingabe
output	Adresspointer zur Aufnahme der Speicheradresse der komprimierten Daten.	Ausgabe
outputlength	Speicheradresse eines Feldes, in dem die Länge der komprimierten Daten zurückgegeben wird.	Ausgabe
fileName	Dateiname für den ZIP-Header. Der Name muss mit x'00' terminiert sein.	Eingabe

gunzip

Mit gzip gepackten Daten werden entpackt.

Syntax	**int gunzip(char *input, int inputlength, char **output, int *outputlength, char *fileName);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
input	Speicheradresse der komprimierten Daten.	Eingabe
inputlength	Länge der komprimierten Daten.	Eingabe
output	Adresspointer zur Aufnahme der Speicheradresse der entkomprimierten Daten.	Ausgabe
outputlength	Speicheradresse eines Feldes, in dem die Länge der entkomprimierten Daten zurückgegeben wird.	Ausgabe

fileName	Speicheradresse, an die der Dateiname aus dem ZIP-Header kopiert wird. Der Dateiname ist x'00' terminiert.	Ausgabe

Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"

/* ------------------------------------------------------------ *
   int main
/*                                                              *
 * gzip/gunzip Routines                                         *
* ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  char  *pZipData;
  char  *pUnzipData;
  char   fileName[_MAX_PATH]     = {0};
  int    iZipLen = 0, iUnzipLen = 0;

  char HAMLET[] =
  {
   "To be, or not to be: that is the question:\n"
   "Whether 'tis nobler in the mind to suffer\n"
   "The slings and arrows of outrageous fortune,\n"
   "Or to take arms against a sea of troubles,\n"
   "And by opposing end them. To die, to sleep-\n"
   "No more- and by a sleep to say we end\n"
   "The heartache, and the thousand natural shocks\n"
   "That flesh is heir to| 'Tis a consummation\n"
   "Devoutly to be wished. To die, to sleep-\n"
   "To sleep-perchance to dream; ay, there's the rub,\n"
   "For in that sleep of death what dreams may come'\n"
   "When we have shuffled off this mortal coil,\n"
   "Must give us pause. There's the respect\n"
   "That makes calamity of so long life:\n"
   "For who would bear the whips and scorns of time,\n"
   "Th' oppressor's wrong, the proud man's contumey,\n"
   "The pangs of despised love, the law's delay,\n"
   "The insolence of office, and the spurns\n"
   "That patient merit of th' unworthy takes,\n"
   "When he himself might his quietus make\n"
   "With a bare bodkin? Who would fardels bear,\n"
   "To grunt and sweat under a weary life,\n"
   "But that the dread of something after death,\n"
   "The undiscovered country, from whose bourn\n"
   "No traveler returns, puzzles the will,\n"
   "And makes us rather bear those ills we have,\n"
   "Than fly to others that we know not of?\n"
   "Thus conscience does make cowards of us all,\n"
   "And thus the native hue of resolution\n"
   "Is sicklied o'er with the pale cast of thought,\n"
   "And enterprises of great pitch and moment,\n"
   "With this regard their currents turn awry,\n"
   "And lose the name of action.-Soft you now,\n"
   "The fair Ophelia|-Nymph, in thy orisons\n"
   "Be all my sins remembered.\n"
  };

  gzip( HAMLET, sizeof(HAMLET), &pZipData, &iZipLen, "hamlet.txt" );
  DumpData( "Zip Data:", pZipData, iZipLen );

  gunzip( pZipData, iZipLen, &pUnzipData, &iUnzipLen, fileName );
  /* writeFile("hamlet.txt", pUnzipData, iUnzipLen, createFile ); */
  DumpData( fileName, pUnzipData, iUnzipLen );

  CleanupGzip( pZipData );
  CleanupGzip( pUnzipData );
  return( 0 );
}
```

Kapitel

11

Hilfsfunktionen

Allgemein

CryptLib enthält einige Hilfsfunktionen, die den Entwickler bei der Programmierung kryptographischer Anwendungen unterstützen. Hierzu zählen Funktionen zur Konvertierung der Darstellung von ASN.1-Objekten von binär nach US-ASCII und umgekehrt, Funtionen zum Lesen und Schreiben von Dateien sowie Funktionen zum Übersetzen von Daten von EBCDIC nach ASCII und umgekehrt.

Funktionen

ASN2PEM

BER/DER-kodierte ASN.1-Objekte liegen in binärer Form vor. Es gibt jedoch Übertragungsprotokolle, die nicht in der Lage sind, binäre Daten transparent wiederzugeben. Um BER/DER-kodierte Daten für derartige Applikationen 'lesbar' zu machen, müssen sie von der binären Form in die US-ASCII-Repräsentation übersetzt werden. Dies geschieht mit der, im RFC 1521 definierten, *Base64*-Methode.
Mit dieser Funktion kann ein binäres Objekt in ein Base64-Objekt umgewandelt werden.

Syntax	**int ASN2PEM(BYTE *input, int inputlength, char *smime, BYTE **output, void **ctx);**	
Returncode	Lände des Base64-Objekts oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
input	Speicheradresse eines binären ASN.1-Objekts.	Eingabe
inputlength	Länge des ASN.1-Objekts.	Eingabe

smime	Falls ein S/MIME-Name übergeben wird, wird vor dem PEM-Objekt folgender S/MIME-Header eingefügt:
	
	`Content-Disposition: attachment; filename="smime.p7m"`
	`Content-Type: application/x-pkcs7-mime; name="smime.p7m"`
	`Content-Transfer-Encoding: base64`
		Eingabe

output	Adresspointer zur Aufnahme der Speicheradresse des erstellten Base64-Objekts.	Ausgabe
ctx	Adresspointer zur Aufname der Speicheradresse eines erstellten Kontexts. Dieser wird zum Freigeben des verwendeten Speichers benötigt.	Ausgabe

PEM2ASN

Mit dieser Funktion kann ein Base64-Objekt in ein binäres Objekt zurückgewandelt werden.

Syntax	int PEM2ASN(BYTE *input, int inputlength, BYTE **output, void **ctx);	
Returncode	Lände des binären Objects oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
input	Speicheradresse eines Base64-Objekts.	Eingabe
inputlength	Länge des Base64-Objekts.	Eingabe
output	Adresspointer zur Aufnahme der Speicheradresse des erstellten binären Objekts.	Ausgabe
ctx	Adresspointer zur Aufnahme der Speicheradresse eines erstellten Kontexts. Dieser wird zum Freigeben des verwendeten Speichers benötigt.	Ausgabe

CleanupPEM

Freigeben des von den Base64-Routinen benötigten Speichers.

Syntax	int CleanupPEM(void *ctx);	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
ctx	Speicheradresse des Kontexts.	Eingabe

Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xpscrypt.h"


/* ------------------------------------------------------------ *
 *                                                              */
   int main
/*                                                              *
 * Translate  ASN.1 DER coded file to base64 coded PEM file, or *
 *            base64 coded PEM file to DER coded ASN.1 file     *
 *                                                              *
 * Parameter:                                                   *
 *            ASN.1/PEM file                                    *
 *            '1' = ASN.1 to PEM                                *
 *            '2' = PEM   to ASN.1                              *
 * ------------------------------------------------------------ */
( int    argc,
  char **argv )
/* ------------------------------------------------------------ */
{
  BYTE     *buffer;
  char     *PEM         = 0;
  int       count       = 0;
  int       func        = 0;
  void     *ctx;

  count = readFile( argv[1], &buffer );
  if( count == 0 )
  {
    printf( "File \"%s\" not found", argv[1] );
    return( -1 );
  }

  if( argv[2][0] == '1' )
    func = 1;
  else
    func = 2;

  if( func == 1 )
    count = ASN2PEM( buffer, count, &PEM, &ctx );
  else
    count = PEM2ASN( buffer, count, &PEM, &ctx );

  writeFile( "PEMout", PEM, count, createFile );

  CleanupPEM( ctx );
```

```
    return( count );
}
```

readFile

Mit dieser Funktion kann eine Datei eingelesen werden.

Syntax	**int readFile(char *fileName, BYTE **buffer);**	
Returncode	Länge der Datei oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
filename	Name der einzulesenden Datei.	Eingabe
buffer	Adresspointer zur Aufnahme der Speicheradresse der gelesenen Datei.	Ausgabe

writeFile

Mit dieser Funktion kann eine Datei auf die Festplatte geschrieben werden.

Syntax	**int writeFile(char *fileName, BYTE *buffer, int bufferlength, int fOption);**	
Returncode	0 oder Fehlercode (<0).	
Parameter	**Beschreibung**	**Verwendung**
filename	Name der zu schreibenden Datei.	Eingabe
buffer	Speicheradresse der zu schreibenden Daten.	Eingabe
bufferlength	Länge der zu schreibenden Daten.	Eingabe
fOption	Schreiboption. Folgenden Optionen sind möglich: createFile Die Datei wird neu erstellt. appendFile Die Daten werden an eine bereits bestehende Datei angehängt.	Eingabe

cleanupFile

Freigeben des Datenspeichers.

Syntax	**void cleanupFile(BYTE *buffer);**	
Returncode	Keiner.	
Parameter	**Beschreibung**	**Verwendung**
buffer	Speicheradresse des freizugebenden Speichers.	Eingabe

EBCDIC_to_ASCII

Mit dieser Funktion können Daten vom EBCDIC-Format in das ASCII-Format konvertiert werden.

Syntax	**void EBCDIC_to_ASCII(BYTE *data, int datalength);**	
Returncode	Keiner.	
Parameter	**Beschreibung**	**Verwendung**
data	Speicheradresse der zu konvertierenden EBCDIC-Daten.	Ein-/Ausgabe
datalength	Länge der EBCDIC-Daten.	Eingabe

ASCII_to_EBCDIC

Mit dieser Funktion können Daten vom ASCII-Format in das EBCDIC-Format konvertiert werden.

Syntax	**void ASCII_to_EBCDIC(BYTE *data, int datalength);**	
Returncode	Keiner.	
Parameter	**Beschreibung**	**Verwendung**
data	Speicheradresse der zu konvertierenden ASCII-Daten.	Ein-/Ausgabe
datalength	Länge der ASCII-Daten.	Eingabe

Kapitel

12

Fehlercodes

err_algorithm **-100**

Beschreibung: Der bei der Funktion *InitCTX* angegebene Algorithmus wird nicht unterstützt. Unterstützte Algorithmen sind **AES,** DES**, RC2, RC4, Blowfish** sowie **RSA**.

err_keylength **-101**

Beschreibung: Die bei der Funktion *InitCTX* angegebene Schlüssellänge wird nicht unterstützt.

err_mode **-102**

Beschreibung: Der bei der Funktion *InitCTX* angegebene Modus wird nicht unterstützt. Unterstützte Modi sind bei symmetrischer Verschlüsselung **ECB** und **CBC**, bei asymmetrischer Verschlüsselung **PUBLIC** und **PRIVATE**.

err_buildkey **-103**

Beschreibung: Bei der Funktion *InitCTX* ist ein Fehler bei der Algorithmus-Initialisierung aufgetreten.

err_buildiv **-104**

Beschreibung: Bei der Funktion *InitCTX* ist ein Fehler bei der Bildung des Initialisierungsvektors aufgetreten.

err_CTX **-105**

Beschreibung: Der bei der Funktion angegebene Kontext ist ungültig oder nicht initialisiert.

err_outlen **-106**

Beschreibung: Der bei der Funktion angegebene Speicherbereich ist zu klein.

err_contentenc **-200**

Beschreibung: Bei der Base64 zu PEM Umsetzung ist ein Fehler aufgetreten.

err_data **-201**

Beschreibung: Die Daten, die der RSA-Funktion übergebenen wurden, sind nicht korrekt.

err_digalgo **-202**

Beschreibung: Der Hashtyp, der bei der RSA-Funktion angegebene wurde, wird nicht unterstützt.

err_encoding **-203**

Beschreibung: Bei der PEM zu Base64 Umsetzung ist ein Fehler aufgetreten.

err_rsakey **-204**

Beschreibung: Der an die Funktion übergebene RSA-Schlüssel ist nicht korrekt.

err_rsalength **-205**

Beschreibung: Die der RSA-Funktion übergebene Datenlänge ist nicht korrekt.

err_modulus **-206**

Beschreibung: Der Modulus des übergebenen RSA-Schlüssels ist nicht korrekt.

err_random **-207**

Beschreibung: Die Random-Struktur wurde nicht initialisiert.

err_privkey **-208**

Beschreibung: Der der Funktion übergebene private RSA-Schlüssel (privateKey) ist nicht korrekt.

err_pubkey **-209**

Beschreibung: Der der Funktion übergebene öffentliche RSA-Schlüssel (publicKey) ist nicht korrekt.

err_signature **-210**

Beschreibung: Die zur Verifizierung übergebene Signatur stimmt nicht mit dem Original überein.

err_encralgo **-211**

Beschreibung: Der in der RSA-Funktion verwendete Encryption-Algorithm ist nicht bekannt.

err_certparm **-300**

Beschreibung: Bei der Funktion *ImportCertificate* wurde ein ungültiger Parameter angegeben.

err_certimport **-301**

Beschreibung: Bei der Funktion *ImportCertificate* wurde ein nicht unterstütztes X.509 Zertifikat übergeben.

err_certlength **-302**

Beschreibung: Der der Zertifikats-Extrahier-Funktion übergebene Speicherbereich ist zu klein.

err_certalgo **-303**

Beschreibung: Bei X.509 Zertifikaten wird nur der Encryption-Algorithm RSA unterstützt.

err_certhash **-304**

Beschreibung: Bei X.509 Zertifikaten werden nur die Hashtypen MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160 unterstützt.

err_certstart **-305**

Beschreibung: Beim Verifizieren des X.509 Zertifikats wurde festgestellt, dass das aktuelle Datum kleiner als das Zertifikats-Beginndatum ist. Die Signatur des Zertifikats ist korrekt.

err_certend **-306**

Beschreibung: Beim Verifizieren des X.509 Zertifikats wurde festgestellt, dass das Ablaufdatum des Zertifkats überschritten ist. Die Signatur des Zertifikats ist korrekt.

err_certoid **-307**

Beschreibung: Die, mit der Funktion *GetExtensionByOID* gesuchte, Erweiterung ist nicht vorhanden.

err_asn1 **-400**

Beschreibung: Das zu importierende Objekt hat eine fehlerhafte oder nicht unterstützte ASN.1 Struktur.

err_asn1table **-401**

Beschreibung: Das importierte ASN.1 Objekt ist nicht mit der aufgerufenen Funktion kompatibel.

err_hashOID **-402**

Beschreibung: Das importierte PKCS-Objekt enthält einen nicht unterstützten Hashtype.

err_contentinf **-403**

Beschreibung: Das importierte PKCS-Objekt enthält eine nicht unterstützte Content-Information.

err_hmac **-404**

Beschreibung: Der HMAC des importierten PKCS#12-Objekts ist nicht gültig. Eventuell wurde ein falsches Passwort übergeben.

err_authsafe **-405**

Beschreibung: Das importierte PKCS#12-Objekt enthält einen nicht unterstützten Authenticated Safe.

err_pbeType **-406**

Beschreibung: Das importierte PKCS#12-Objekt enthält einen nicht unterstützten Pbe-Typ (PBE=PasswordBasedEncryption). Unterstützt werden folgende Algorithmen: pbeWithSHAAnd128BitRC4, pbeWithSHAAnd40BitRc4, pbeWithSHAAnd3KeyTripleDES-CBC, pbeWithSHAAnd2KeyTripleDES-CBC, pbeWithSHAAnd128BitRC2-CBC und pbeWithSHAAnd40BitRC2-CBC.

err_certbag **-407**

Beschreibung: Das importierte PKCS#12-Objekt enthält einen nicht unterstützten Certification Bag.

err_certbagType **-408**

Beschreibung: Der im importierten PKCS#12-Objekt enthaltene Certification Bag enthält ein nicht unterstütztes Zertifikatsformat. Die Formate x509Certificate und sdsiCertificate werden unterstützt.

err_noauthsafe **-409**

Beschreibung: Das importierte PKCS#12-Objekt enthält keinen Authenticated Safe.

err_safeBag **-410**

Beschreibung: Das importierte PKCS#12-Objekt enthält keinen Safe Bag Type 'pkcs8-shroudedKeybag'.

err_privKey **-411**

Beschreibung: Das importierte PKCS#12-Objekt enthält keinen geheimen Schlüssel (PrivateKey).

err_RSAEnc **-412**

Beschreibung: Bei PKCS#12-Objekten wird nur der Encryption-Algorithm RSA unterstützt.

err_nokeyBag **-413**

Beschreibung: Das importierte PKCS#12-Objekt enthält keinen Key Bag.

err_hmacAlgo **-414**

Beschreibung: Das importierte PKCS#12-Objekt enthält einen nicht unterstützten HMAC Algorithmus. Unterstützt werden MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160.

err_Algo **-415**

Beschreibung: Das importierte PKCS#7-Objekt enthält einen nicht unterstützten Hash Type. Unterstützt werden MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 und RipeMD160.

err_EAlgo **-416**

Beschreibung: Das importierte PKCS#7-Objekt enthält einen nicht unterstützten Encryption Algorithm. Unterstützt wird nur RSA.

err_noSigner **-417**

Beschreibung: Für den im importierten PKCS#7-Sign-data Objekt enthaltenen Unterzeichner (Signer) wurde kein Aussteller-Zertifikat (Trusted Signer) gefunden.

err_noSignerCert **-418**

Beschreibung: Für den im importierten PKCS#7-Sign-data Objekt enthaltenen Unterzeichner (Signer) wurde kein Zertifikat gefunden.

err_messagedig **-419**

Beschreibung: Der Prüfwert (Message Digest) des im importierten PKCS#7-Sign-data Objekt enthaltenen Unterzeichners (Signer) ist nicht gültig.

err_verify **-420**

Beschreibung: Die Verifizierung der Signatur des im importierten PKCS#7-Sign-data Objekt enthaltenen Unterzeichners (Signer) ist nicht gültig.

err_unknownSigner **-421**

Beschreibung: Der bei der Funktion *VerifySigner* übergebene Unterzeichner ist im importierten PKCS#7-Sign-data Objekt nicht enthalten.

err_noData **-422**

Beschreibung: Das importierte PKCS#7-Objekt enthält keine Daten.

err_noCert **-423**

Beschreibung: Das importierte PKCS#7-Objekt enthält kein Zertifikat.

err_noRecipient **-424**

Beschreibung: Das importierte PKCS#7 Enveloped-data Objekt enthält keinen Empfänger (Recipient).

err_p12notvalid **-425**

Beschreibung: Das der Funktion *ImportEnvelopedData* übergebene PKCS#12 Objekt ist nicht gültig.

err_invopt **-426**

Beschreibung: Die der Funktion *Create…Data* übergebene Option ist nicht gültig.

err_noTrustedSign **-427**

Beschreibung: Für den der Funktion *VerifySigner* übergebenen Unterzeichner ist kein vertrauenswürdiger Aussteller (Trusted Signer) vorhanden.

err_maxData **-428**

Beschreibung: Die Funktion *AddPKCS7Data* hat die maximale zu verarbeitende Datenmenge überschritten.

err_ssl_inv_version **-500**

Beschreibung: Ein Client versucht mit einer nicht unterstützten SSL-Version eine Verbindung zu starten.

err_ssl_inv_side **-501**

Beschreibung: Beim SSL_Init wurde eine ungültige Protocolside angegeben. Gültig sind SSL_ClientSide und SSL_ServerSide.

err_ssl_cipher_invalid **-502**

Beschreibung: Bei der Funktion SSL_Set_Cipher wurde ein ungültiger oder nicht unterstützter Cipher angegeben.

err_ssl_cert_not_trusted **503**

Beschreibung: Beim verifizieren des X.509 Zertifikats ist ein Fehler aufgetreten.

err_ssl_send **-504**

Beschreibung: Beim Übertragen der Daten ist ein TCP/IP Fehler aufgetreten.

err_ssl_select **-505**

Beschreibung: Beim TCP/IP Befehl *select* ist ein Fehler aufgetreten.

err_ssl_read **-506**

Beschreibung: Beim Lesen der Daten ist ein TCP/IP Fehler aufgetreten.

err_ssl_unsupportedType **-507**

Beschreibung: Beim Lesen der Daten wurde ein unbekannter Typ erkannt.

err_ssl_unsupportedVersion **-508**

Beschreibung: Beim Lesen der Daten wurde eine unbekannte oder nicht unterstützte Version erkannt.

err_ssl_unsupportedCipher **-509**

Beschreibung: Bei einer Client-Session sendet der Server einen nicht unterstützten Cipher. Bei einer Server-Session hat ein Client keine vom Server unterstützte Cipher beantragt.

err_ssl_toomanydata **-510**

Beschreibung: Es wurden Daten gelesen die die beim SSL Protokoll maximale Datenlänge von 32767 Byte überschreiten.

err_ssl_protocolviolation **-511**

Beschreibung: Beim SSL Handshake ist eine Protokollverletzung aufgetreten.

err_ssl_messageHash **-512**

Beschreibung: Beim SSL Handshake ist ein ungültiger Message-Hash erkannt worden.

err_ssl_alertRead **-513**

Beschreibung: Beim Lesen wurde eine Alert-Message vom Partner erkannt.

err_ssl_reqCertInv **-514**

Beschreibung: Es wurde ein nicht unterstütztes X.509 Zertifikat gelesen.

err_ssl_noCertsAvailable -515

Beschreibung: Beim Handshake wurde kein X.509 Zertifikat gefunden. Eventuell ist kein SSL_Set_PrivateKey durchgeführt worden.

err_ssl_unsupported_RSAkey -516

Beschreibung: Der übergebene PrivateKey wird nicht unterstützt.

err_ssl_hash_RSAkey_inv -517

Beschreibung: Beim ServerKeyExchange wurde ein Fehler festgestellt.

err_ssl_premasterkey_inv -518

Beschreibung: Es wurde ein ungültiger PreMasterKey festgestellt.

err_ssl_certverify_inv -519

Beschreibung: Das eingelesene Client Zertifikat ist ungültig.

err_ssl_peer_not_trusted -520

Beschreibung: Bei einer Client-Verbindung wurde ein Server-Zertifikat gelesen, das nicht denen, die mit der Funktion SSL_Add_DN geladen wurden, entspricht.
Bei einer Server-Verbindung wurde ein Client-Zertifikat gelesen, das nicht denen, die mit der Funktion SSL_Add_DN geladen wurden, entspricht.