Specification

A Cryptographic Library for Dyalog APL

Peter-Michael Hager
Release: May 2011

Glossary

Item	Description
AAD	Additional Authenticated Data
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard by the NIST [FIPS-197]
ASN.1	Abstract Syntax Notation One [ITU-T X.680]
BER	Basic Encoding Rules [ITU-T X.690]
Camellia	Symmetric block cipher, recommended by NESSIE and CRYTREC [NTT]
CAST	Carlisle Adams and Stafford Tavares [RFC2144]
CBC	Cipher Block Chaining mode by the NIST [SP800-38A]
CCM	Counter mode with CBC-MAC by the NIST [SP800-38C]
CFB	Cipher Feedback mode by the NIST [SP800-38A]
CMAC	Cipher-based Message Authentication Code
CTR	Counter mode by the NIST [SP800-38A]
DER	Distinguished Encoding Rules [ITU-T X.690]
DES	Data Encryption Standard
DES-X	With XORs enhanced version of the DES algorithm
DH	Diffie-Hellman key exchange
DSA	Digital Signature Algorithm
EC	Elliptic Curve cryptography
ECDH	Elliptic Curve based Diffie-Hellman key exchange
ECDSA	Elliptic Curve Digital Signature Standard
ECB	Electronic Codebook mode by the NIST [SP800-38A]
EDI	Electronic data interchange
EMSA	Encoding Method for Signatures with Appendix
GCM	Galois/Counter-Mode by the NIST [SP800-38D]
HMAC	Keyed Hash Message Authentication Code
IDEA	International Data Encryption Algorithm
IV	Initialization Vector
MAC	Message Authentication Code
MD2	Message Digest Algorithm 2 (for backward compatibility only) [RSA]
MD4	Message Digest Algorithm 4 [RSA]
MD5	Message Digest Algorithm 5 [RSA]
MDC-2	Modification Detection Code 2 [IBM]
MGF1	Mask Generation function 1 [PKCS#1v2.1]
MSB	Most Significant Bit first
OAEP	Optimal Asymmetric Encryption Padding [PKCS#1v2.1]
OFB	Output Feedback mode by the NIST [SP800-38A]
OID	Object Identifier [ITU-T X.680]
OSI	Open Systems Interconnection [ITU-T X.200]
PKCS	Public Key Cryptographic Standard [RSA]
PKCS#1	RSA Cryptography Standard
PKCS#5	Password-Based Encryption Standard
PKCS#8	Private-Key Information Syntax Standard
PKCS#12	Personal Information Exchange Syntax
PKCS1-v1_5	Encryption and Signature Schemes based on [PKCS#1v1.5]
PSS	Probabilistic Signature Scheme [PKCS#1v2.1]
RC2	Rivest Cipher Algorithm 2 [RSA]
RC4	Rivest Cipher Algorithm 4 [RSA]
RC5	Rivest Cipher Algorithm 5 [RSA]
RIPEMD-160	RACE Integrity Primitives Evaluation Message Digest
RSA	Rivest Shamir Adleman [RSA]

RSAES	RSA Encryption Scheme
RSASSA	RSA Signature Scheme with Appendix
SEED	Symmetric block cipher by the Korean Information Security Agency [KISA]
SHA-1	Secure Hash Algorithm 1 [NIST]
SHA-224	Secure Hash Algorithm 224 [NIST]
SHA-256	Secure Hash Algorithm 256 [NIST]
SHA-384	Secure Hash Algorithm 384 [NIST]
SHA-512	Secure Hash Algorithm 512 [NIST]
X.501	OSI - The Directory: Models (distinguished name) [ITU-T X.501]
X.509	OSI - The Directory: Authentication Framework [ITU-T X.509]

Function Overview

The library functions are divided into these categories:

- Cryptographic base functions
- ASN.1 base functions
- Base64 helper functions
- Derived function samples

Cryptographic base functions

These are functions to perform cryptographic algorithms:

Function	Description
#.Crypt.Init	Loads and initializes the access to the external library.
#.Crypt.Encrypt	Performs symmetric data encryption and authentication.
#.Crypt.Decrypt	Performs symmetric data decryption and verification.
#.Crypt.PKey	Asymmetric computation and key pair/parameter generation
#.Crypt.Random	Generates a sequence of true random bytes.
#.Crypt.Hash	Calculates a message digest.
#.Crypt.Exit	Unloads the external library.

The execution mode of these functions is manged by these variables:

Variables and Constants	Description
Digest algorithms	Constants to select a digest or authentication algorithm.
Cipher algorithms	Constants to select a symmetric cipher algorithm.
Public Key algorithms	Constants to select an asymmetric public key algorithm.
Padding schemes	Constants to select a padding scheme.
Modes of operation	Constants to select a mode of operation.
Cipher suites	Useful combinations of ciphers, modes and schemes.

These are helper functions and functions composed from the algorithmic functions:

Function	Description
#.Crypt.OidToAlgid	Transforms an OID into algorithm identifier(s).
#.Crypt.AlgidToOid	Transforms algorithm identifier(s) into an OID.
#.Crypt.Sign	Generates a signature over a sequence of bytes.
#.Crypt.VerifySignature	Verifies a signature.

ASN.1 base functions

These are functions to encode or decode ASN.1 content:

Funktion	Description
#.ASN1.Init	Loads and initializes the access to the external library.
#.ASN1.Code	Encodes and decodes ASN.1 structures into nested arrays.
#.ASN1.Adjust	Adjusts the length of an ASN.1 structure with trailing bytes.
#.ASN1.Exit	Unloads the external library.

These variables are useful to manage tags and OIDs:

Variables and Constants	Description
Class	Enumerating constants for the ASN.1 classes.
Form	Enumerating constants for the ASN.1 forms.
Tag	Enumerating constants for the ASN.1 tags.
Universal tag	Enumerating constants for the ASN.1 universal tags.
Universal tag options	Coding options for the ASN.1 universal tags.
OidTab	Table of well known OIDs and its mnemonics.

These are helper functions to encode and decode ASN.1 content:

Function	Description
#.ASN1.InitOidTab	Helper function to rebuild the OidTab.

Base64 helper functions

These helper functions are useful fore accessing base64 content:

Function	Description
#.ASN1.Base64.Encode	Encodes a binary vector into a base64 text vector.
#.ASN1.Base64.Decode	Decodes a base64 text vector into a binary vector.
#.ASN1.Base64.FileEncode	Base64 encodes and saves binary vectors to a file.
#.ASN1.Base64.FileDecode	Loads and decodes a base64 file.

Derived function samples

These functions give an overview on how the cryptographic and ASN.1 functions can be used to handle more complex structures:

Function	Description
#.ASN1.X501.FormatName	Format a distinguished name structure into a string.
#.ASN1.X501.ResolveName	Resolve a formatted name string into an ASN.1 structure.
#.ASN1.X509.ResolveCertificate	Resolve an X.509 certificate into a nested structure.
#.ASN1.X509.ResolveExtensions	Resolve the certificate extensions into a matrix.
".GetCertificateSerialNumber	Return a certificate's serial number.
".GetCertificateSubject	Return a certificate's subject.
".GetCertificateSubjectPublicKey	Return the PKCS#1 encoded SubjectPublicKey.
".GetCertificateValidity	Return the notBefore and notAfter validity fields.
".GetCertificateIssuer	Return a certificate's issuer.

".QueryCertificateType	Check the type of a certificate.
".VerifyCertificateChain	Verify the signature chain of certificates.
#.ASN1.PKCS1.GetKeyLength	Return the key length in bits from an RSAKey.
#.ASN1.PKCS1.KeyDecode	Return the 2 or 8 key elements fron an encoded key or cert.
#.ASN1.PKCS1.KeyEncode	Encode the 2 or 8 key elements into an encoded key.
#.ASN1.PKCS1.GetIdentifier	Extract the algorithm from an EMSA signature structure.

#.Crypt.Init

The function #.Crypt.Init initializes the access to the further functions in this section.

```
#.Crypt.Init
```

Annotations

Multiple execution of this function is ignored.

Requirements

Library	DyaCrypt.dll must exist in the executable directory.

References

#.Crypt.Exit #.Crypt.Encrypt #.Crypt.Decrypt #.Crypt.PKey #.Crypt.Random #.Crypt.Hash

#.Crypt.Encrypt

The function #.Crypt.Encrypt performs a symmetric data encryption and/or authentication.

```
(CipherData Digest) ←
      (Algid CipherKey IV DigestKey) #.Crypt.Encrypt (AuthData PlainData)
```

Parameter values

Algid
: [required] This is a combination (+) of a cipher algorithm, an operating mode, a padding scheme (e.g. PKCS#5 or none), and a digest algorithm. Optionally a flag (HMOD_CIPH) can be set in Algid, to define that the digest algorithm shall be performed on CipherData instead of on PlainData.

CipherKey
: [required] The symmetric encryption key.

IV
: [optional] The initialization vector. Not required in ECB mode.

DigestKey
: [optional] When using a keyed digest algorithm, this is the key for the CMAC or HMAC. If omitted though a keyed digest algorithm has been given, the CipherKey will be used instead.

AuthData
: [optional] The data for an authenticated encryption function that is to be authenticated but not encrypted (AAD). In case of using a hash algorithm, this data is prepended to the plain data when computing the hash value. In case of a CMAC or HMAC, it is prepended to the encrypted data.

PlainData
: [required] The data to be encrypted.

Return values

CipherData
: If a cipher algorithm was given, the encrypted data, an empty vector otherwise.

Digest
: If a digest algorithm was given, the message digest, an empty vector otherwise.

Annotations

The CipherKey and the DigestKey should be kept secret. In case of using a combined AEAD mode, like GCM, the padding scheme and the digest algorithm are implicitly set and need not be specified.

Requirements

Loader	#.Crypt.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.Crypt.Decrypt

#.Crypt.Decrypt

The function #.Crypt.Decrypt performs a symmetric data decryption and/or verification.

```
(PlainData Digest) ←
      (Algid CipherKey IV DigestKey) #.Crypt.Decrypt (AuthData CipherData)
```

Parameter values

Algid
: [required] This is a combination (+) of a cipher algorithm, an operation mode, a padding scheme (e.g. PKCS#5 or none), and a digest algorithm. Optionally a flag (HMOD_CIPH) can be set in Algid, to define that the digest algorithm shall be performed on CipherData instead of on PlainData.

CipherKey
: [required] The symmetric decryption key.

IV
: [optional] The initialization vector. Not required in ECB mode.

DigestKey
: [optional] When using a keyed digest algorithm, this is the key for the CMAC or HMAC. If omitted though a keyed digest algorithm has been given, the CipherKey will be used instead.

AuthData
: [optional] The data for an authenticated decryption function that is to be authenticated unencrypted (AAD). In case of using a hash algorithm, this data is prepended to the plain data when computing the hash value. In case of a CMAC or HMAC, it is prepended to the encrypted data.

CipherData
: [required] The data to be decrypted.

Return values

PlainData
: If a cipher algorithm was given, the decrypted data, an empty vector otherwise.

Digest
: If a digest algorithm was given, the message digest, an empty vector otherwise.

Annotations

For a successful decryption the Algid, CipherKey, IV, DigestKey, and the AuthData should be identical to those given in the encryption. In case of using a combined AEAD mode, like GCM, the padding scheme and the digest algorithm are implicitly set and need not to be specified. The function does not compare the digest with the one calculated at encryption time, so a compare (≡) of both is required after decryption. If the compare fails, the resulting plain data should be discarded.

Requirements

Loader	#.Crypt.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.Crypt.Encrypt

#.Crypt.PKey

The function #.Crypt.PKey computes an asymmetric transformation, which can be a public encryption, a private decryption, the agreement for a common secret, the generation of a signature, or a signature verification. In a special form with a single left argument, the function generates a key pair, or, where suited, key parameters.

```
(OutputData)           ← (Algid Key) #.Crypt.PKey (InputData)

(ValidFlag)            ← (Algid Key) #.Crypt.PKey (InputData ToBeSigned)

(PrivateKey PublicKey) ←  PKEY_RSA   #.Crypt.PKey (KeyLenth PublicExponent)

(DsaParamset)          ←  PKEY_DSA   #.Crypt.PKey (KeyLenth DigestLength)
(PrivateKey PublicKey) ←  PKEY_DSA   #.Crypt.PKey (DsaParamsetOrName)

(PrivateKey PublicKey) ←  PKEY_EC    #.Crypt.PKey (EcParamsetOrName)

(DhParamset)           ←  PKEY_DH    #.Crypt.PKey (KeyLenth Generator)
(PrivateKey PublicKey) ←  PKEY_DH    #.Crypt.PKey (DhParamsetOrName)
```

Parameter values

Algid

[required] This is a possible combination (+) of a public-key algorithm, an optional key-padding scheme (RSAES-PKCS1v1_5/-OAEP, RSASSA-PKCS1v1_5/-PSS) and, where aproprate, an optional digest algorithm (with EC/DSA, or for the MGF1 in case of OAEP/PSS or with RSASSA-PKCS1v1_5 for the hash OID). Possible combinations are already assigned as constants beginning with PKEY. For crypting operations, when having a keyas a second left argument, the funtion determines the algorithm selection from the internal structure of that key, (wether RSA, DSA, EC, or DH, and wether private or public) in precedece over the public-key algorithm part of the Algid. So, this mechanism may override a given public-key algorithm, though, where applicable, a given padding scheme or digest algorithm still remain recognized.

Key

[public encrypt, private decrypt, key agreement, sign, verify] A private or a public key, represented in its encoded ASN.1 structure. If omitted, the function generates a key pair or key parameters.

InputData

[public encrypt, private decrypt, key agreement, sign, verify] Data value to transform, represented as character or bit vector in MSB (big-endian) notation. Usually a secret key, a public-encrypted secret key, a key agreement key, a document hash, or a signature.

OutputData

[public encrypt, private decrypt, key agreement, sign, verify] Transformed data value, represented as character vector in MSB (big-endian) notation. Usually a public-encrypted secret key, a private-decrypted secret key, an agreed secret, a signature, or public-decrypted document hash.

ToBeSigned

[verify] Data value to verify, MSB (big-endian). In signature modes without message recovery like RSASSA-PSS, the to-be-signed (InputData for signing) cannot get recovered from the signature (OutputData from signing), though it can get verified. Consequently, at least for those modes, it is essential to pass the to-be-signed as a parameter. The result is then a ValidFlag.

ValidFlag
[verify] False (0) if the signature verification fails, true (1) upon success.

KeyLength
[keygen, required] Bit count of the key pair to generate.

PublicExponent
[keygen, optional] The public exponent for the key pair, preferably a Fermat prime (F0=3, F1=5, F2=17, F3=257, F4=65537) or, preferably in cases where the same InputData needs to be encrypted by 65537 or more recipient keys, an anti-elite prime (e.g. 13, 97, 193, 641, 769, 12289, 40961, 786433, 167772161, 1107296257, 2281701377, 3221225473, 77309411329), default F4=65537.

DigestLength
[parameter set generation, optional] Bit count the signable digest length which can be verified when using a key pair generated from the resulting DSA paramter set.

Generator
[parameter set generation, optional] Mantissa in the DH function. This value should be 2 or 3 for self generated DH parameters. Though pre-defined parameter sets may have much higher generator values, within an acceptable generation time a trusted security cannot be given with other values. The default value is 2.

DsaParamsetOrName
[keygen, required] Either a DSA parameter set, as an encoded ASN.1 structure, or a shortname for a well known DSA parameter set. Possible shortnames are already assigned to constants beginning with DSA_.

EcParamsetOrName
[keygen, required] Either a EC/ECDSA parameter set, as an encoded ASN.1 structure, or a shortname for a well known EC parameter set. Possible shortnames are already assigned to constants beginning with EC_..

DhParamsetOrName
[keygen, required] Either a DH parameter set, as an encoded ASN.1 structure, or a shortname for a well known DH parameter set. Possible shortnames are already assigned to constants beginning with DH_..

Return values

OutputData
[compute] Transformed data value, MSB (big-endian).

ValidFlag
[verify] 0 if the signature verification fails, 1 otherwise.

PrivateKey
[keygen] Generated private key of a key-pair as an encoded ASN.1 structure.

PublicKey
[keygen] Generated public key of a key-pair as an encoded ASN.1 structure.

DsaParamset
[parameter set generation] Generated DSA parameter set as an encoded ASN.1 structure.

DhParamset
[parameter set generation] Generated DH parameter set as an encoded ASN.1 structure.

Annotations

As the public key associated with a private key is in the RSA, DSA, and EC modes completely contained in the private key, it can be obtained from it with the #.ASN1.Code function by extracting the public components.

Requirements

Loader	#.Crypt.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.Crypt.Init

#.Crypt.Random

The function #.Crypt.Random generates a sequence of true random bytes.

```
(Sequence) ← #.Crypt.Random (Count)
```

Parameter values

Count
[required] Number of bytes to generate.

Return values

Sequence
[optional] Sequence of random bytes generated.

Annotations

The implementation depends on the platform. Under Windows this function incorporates the CryptGenRandom() call from the MS CryptoAPI.

Requirements

Loader	#.Crypt.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.Crypt.Init, #.Crypt.Encrypt, #.Crypt.Decrypt

#.Crypt.Hash

The function #.Crypt.Hash calculates a message digest.

```
(Digest) ← (Algid Count DigestLength IdByte) #.Crypt.Hash (Data)

(Digest) ← (Algid DigestKey Count DigestLength) #.Crypt.Hash (Data)
```

Parameter values

Algid

[required] a digest algorithm. Only when a hash algorithm is used, the optional IdByte parameter can be used. And only for HMAC and CMAC algorithms the DigestKey parameter can be used.

DigestKey

[optional, default: empty text vector] This parameter defines the digest key in the keyed HMAC and CMAC modes. In CMAC mode its length must match the key length of the underlying cipher algorithm. In HMAC modes it can be of arbitrary length.

Count

[optional, default: 0] Iteration count. This is the number of times to repeatedly calculate the digest's digest. The default value indicates that a single digest shall be executed only. In case of a HMAC or CMAC algorithm, the result is composed from the XOR of all iterative results, according to the PBKDF2 definition in [PKCS#5].

DigestLength

[optional, default: algorithm dependent] Number of digest bytes to return. If in case of a hash algorithm the length is greater than the algorithm's digest size, additional blocks will be generated according to PKCS#12 addendum B.2 (omitting the first digest block which contains ID bytes). The default length depends on the chosen algorithm (MD2/4/5 = 16 bytes, SHA-1 = 20 bytes, SHA-xxx = xxx/8 bytes). If in case of a HMAC or CMAC algorithm this parameter is used, a block counter is appended to Data, according to the PBKDF2 definition in [PKCS#5].

IdByte

[optional] According to PKCS#12 addendum B.3 there are three defined values for this parameter, (1 for keys, 2 for IVs, and 3 for MACs). However, any value between 0 and 255 is valid. If this value is omitted or set to -1, the function works in a PEM mode. When using a HMAC or CMAC algorithm this parameter must be omitted.

Data

[required] Data to calculate the hash or MAC from.

Return values

Digest

[required] Calculated hash or MAC value.

Annotations

The iteration count serves the purpose to increase the serialized computation performance when deriving keys from a password, as required in PKCS#5 and PKCS#12. Its functions can be found at #.Crypt.PKCS5.PBKDF1, #.Crypt.PKCS5.PBKDF2, and #.Crypt.PKCS12.PBKDF.

Requirements

Loader	#.Crypt.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.Crypt.Init

#.Crypt.Exit

The function #.Crypt.Exit unloads the external library.

```
#.Crypt.Exit
```

Annotations

After execution the functions from this section are no longer available. Multiple executions are ignored.

Requirements

Loader	#.Crypt.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.Crypt.Init

Digest algorithms

The digest algorithms are identified by constant values. These are divided into hash algorithms, HMACs, and CMACs.

```
⍝                        hex        algorithm             hash-length block-le
⍝                        ---------- ---------------       ----------- --------

HASH_NONE    ←   0×16*2 ⍝ 0x00000000 no digest

HASH_WP      ←   1×16*2 ⍝ 0x00000100 Whirlpool                 512 bit  512 bit
HASH_MD2     ←   2×16*2 ⍝ 0x00000200 MD2                       128 bit  512 bit
HASH_MD4     ←   4×16*2 ⍝ 0x00000400 MD4                       128 bit  512 bit
HASH_MD5     ←   5×16*2 ⍝ 0x00000500 MD5                       128 bit  512 bit
HASH_MDC2    ←   6×16*2 ⍝ 0x00000600 MDC-2                     128 bit  512 bit
HASH_RMD160  ←   8×16*2 ⍝ 0x00000800 RIPEMD-160                160 bit  512 bit
HASH_SHA     ←  11×16*2 ⍝ 0x00000B00 SHA                       128 bit  512 bit
HASH_SHA1    ←  12×16*2 ⍝ 0x00000C00 SHA-1                     160 bit  512 bit
HASH_SHA224  ←  13×16*2 ⍝ 0x00000D00 SHA-224                   224 bit  512 bit
HASH_SHA256  ←  14×16*2 ⍝ 0x00000E00 SHA-256                   256 bit  512 bit
HASH_SHA384  ←  15×16*2 ⍝ 0x00000F00 SHA-384                   384 bit 1024 bit
HASH_SHA512  ←  16×16*2 ⍝ 0x00001000 SHA-512                   512 bit 1024 bit

HMAC_WP      ←  65×16*2 ⍝ 0x00004100 HMAC-Whirlpool            512 bit  512 bit
HMAC_MD2     ←  66×16*2 ⍝ 0x00004200 HMAC-MD2                  128 bit  512 bit
HMAC_MD4     ←  68×16*2 ⍝ 0x00004400 HMAC-MD4                  128 bit  512 bit
HMAC_MD5     ←  69×16*2 ⍝ 0x00004500 HMAC-MD5                  128 bit  512 bit
HMAC_MDC2    ←  70×16*2 ⍝ 0x00004600 HMAC-MDC-2                128 bit  512 bit
HMAC_RMD160  ←  72×16*2 ⍝ 0x00004800 HMAC-RIPEMD-160           160 bit  512 bit
HMAC_SHA     ←  75×16*2 ⍝ 0x00004B00 HMAC-SHA                  128 bit  512 bit
HMAC_SHA1    ←  76×16*2 ⍝ 0x00004C00 HMAC-SHA-1                160 bit  512 bit
HMAC_SHA224  ←  77×16*2 ⍝ 0x00004D00 HMAC-SHA-224              224 bit  512 bit
HMAC_SHA256  ←  78×16*2 ⍝ 0x00004E00 HMAC-SHA-256              256 bit  512 bit
HMAC_SHA384  ←  79×16*2 ⍝ 0x00004F00 HMAC-SHA-384              384 bit 1024 bit
HMAC_SHA512  ←  80×16*2 ⍝ 0x00005000 HMAC-SHA-512              512 bit 1024 bit

CMAC_DES     ← 129×16*2 ⍝ 0x00008100 CMAC-DES      (7/8)64,128,192 bit   64 bit
CMAC_DESX    ← 132×16*2 ⍝ 0x00008400 CMAC-DES-X           184/192 bit   64 bit
CMAC_IDEA    ← 133×16*2 ⍝ 0x00008500 CMAC-IDEA                128 bit   64 bit
CMAC_CAST    ← 134×16*2 ⍝ 0x00008600 CMAC-CAST-128       40...128 bit   64 bit
CMAC_BF      ← 135×16*2 ⍝ 0x00008700 CMAC-Blowfish       32...448 bit   64 bit
CMAC_RC2     ← 139×16*2 ⍝ 0x00008B00 CMAC-RC2                 128 bit   64 bit
CMAC_RC5     ← 141×16*2 ⍝ 0x00008D00 CMAC-RC5                 128 bit   64 bit
CMAC_AES     ← 142×16*2 ⍝ 0x00008E00 CMAC-AES         128,192,256 bit  128 bit
CMAC_CM      ← 145×16*2 ⍝ 0x00009100 CMAC-Camellia    128,192,256 bit  128 bit
CMAC_SEED    ← 148×16*2 ⍝ 0x00009400 CMAC-SEED                128 bit  128 bit
```

Annotations

As being hashes, the algorithms from 2 to 16 do not require a key. So, these can be used in the function #.Crypt-Hash. Additionally, these algorithms can be combined with the OAEP or a signature padding scheme used by the #.Crypt.PKey function. HMAC (from 65 to 80) and CMAC (from 129 to 148) require a DigestKey and are intended to be used in #.Crypt.Encrypt and #.Crypt.Decrypt.

As shown in the hexadecimal comment, the algorithm is encoded in the 2nd byte and can be added to operation modes, padding modes and Cipher algorithms to compose a complete Algid. For currently valid complete Algids refer to the Cipher suites section.

References

#.Crypt.Encrypt #.Crypt.Decrypt #.Crypt.PKey #.Crypt.Hash

Cipher algorithms

The cryptographic algorithms are identified by constant values.

```
⍝                      hex        algorithm              key-length  block-l
⍝                      ---------- ------------           ---------- --------

CIPH_NONE ← 0×16*4 ⍝ 0x00000000 no cipher

CIPH_DES  ← 129×16*4 ⍝ 0x00810000 DES     56/64,112/128,168/192 bit   64 bit
CIPH_DESX ← 132×16*4 ⍝ 0x00840000 DES-X                    192 bit   64 bit
CIPH_IDEA ← 133×16*4 ⍝ 0x00850000 IDEA                     128 bit   64 bit
CIPH_CAST ← 134×16*4 ⍝ 0x00860000 CAST5-128              40-128 bit   64 bit
CIPH_BF   ← 134×16*4 ⍝ 0x00870000 Blowfish                32-448 bit   64 bit
CIPH_RC2  ← 139×16*4 ⍝ 0x008B0000 RC2                      128 bit   64 bit
CIPH_RC4  ← 140×16*4 ⍝ 0x008C0000 RC4 (stream)            40-128 bit    8 bit
CIPH_RC5  ← 141×16*4 ⍝ 0x008D0000 RC5-32/12/16             128 bit   64 bit
CIPH_AES  ← 142×16*4 ⍝ 0x008E0000 AES                128,192,256 bit  128 bit
CIPH_CM   ← 145×16*4 ⍝ 0x00910000 Camellia           128,192,256 bit  128 bit
CIPH_SEED ← 148×16*4 ⍝ 0x00940000 SEED                     128 bit  128 bit
```

Annotations

The symmetric algorithms can be used through the functions #.Crypt.Encrypt and #.Crypt.Decrypt.

As shown in the comment in hexadecimal, the algorithm is encoded in the 3rd byte and can be added to Modes of operation, Padding schemes, and Digest algorithms, encoded in other bytes or nibbles, to compose a complete Algid.

For algorithms in the table with multiple or a range of key lengths, the length (⍴) of the key given to the crypto-function defines the specific sub-algorithm. (For example, with a ⍴CipherKey equal 24, the key length will be 24 × 8 bit per byte = 192 bit. If the chosen algorithm is CALG_DES, a 3-key DES-EDE will be used. In case it is CALG_AES, an AES-192 will be used.) For the DES algorithms, the key length values left from the slashes show the effectively used number of key bits.

For currently implemented and valid Algids, refer to the Cipher suites section. In the current state of a very first release the enumeration is subject to change.

References

#.Crypt.Encrypt #.Crypt.Decrypt #.Crypt.PKey

Public Key algorithms

The public key algorithms are identified by constant values.

```
⍝                      hex        algorithm                          key-length
⍝                      ---------- ---------------------------------- ------------

PKEY_AUTO ← 0×16*0   ⍝ 0x00000000 auto cipher

PKEY_RSA  ← 1×16*0   ⍝ 0x00000001 RSA   (sign & key exchange)     128-16384 bit
PKEY_DSA  ← 2×16*0   ⍝ 0x00000002 DSA   (sign)                    128-16384 bit
PKEY_EC   ← 4×16*0   ⍝ 0x00000004 ECDSA (sign)                      112-571 bit
PKEY_DH   ← 8×16*0   ⍝ 0x00000008 DH    (key agreement)           128-16384 bit
```

Annotations

These identifiers are intended for to use with #.Crypt.PKey to define whether its asymmetric algorithm shall be based on primes (RSA), discrete logarithms (DSA and DH), or elliptical curves (EC/ECDSA).

RSA is suited for both, signing and the encryption of session keys, whereas DSA and EC, due to their non recoverable signature schemes, are intended for signing only.

Diffie-Hellman (DH), though like DSA based on the discrete logarithm problem, can be used to establish a common secret between two persons. By getting the public key from each ones counterpart and processing it with the own private key, both parties will generate the same shared secret.

The different modes and implied hash parameters which can be used in combination with these base algorithms can be looked up in the Cipher suites section. In the current state of a very first release the enumeration is subject to change.

References

#.Crypt.Encrypt #.Crypt.Decrypt #.Crypt.PKey

Padding schemes

The padding schemes are identified by constant values.

```
⍝                          hex                      scheme
⍝                       ---------- ---------------------------------------

PPAD_NONE ←  0×16*1   ⍝ 0x00000000 no scheme

PPAD_PK1E ←  1×16*1   ⍝ 0x00000010 RSAES-PKCS1-V1_5
PPAD_SSL3 ←  2×16*1   ⍝ 0x00000020 SSL3/TLS1
PPAD_OAEP ←  3×16*1   ⍝ 0x00000030 RSAES-OAEP

PPAD_PK1S ←  8×16*1   ⍝ 0x00000080 RSASSA-EMSA-PKCS1-V1_5
PPAD_X931 ←  9×16*1   ⍝ 0x00000090 ANSI-X9.31
PPAD_PSS  ← 10×16*1   ⍝ 0x000000A0 RSASSA-EMSA-PSS non-recoverable
PPAD_PSSR ← 11×16*1   ⍝ 0x000000A0 RSASSA-EMSA-PSS-R recoverable

CPAD_NONE ←  0×16*7   ⍝ 0x00000000 no scheme

CPAD_PKCS ←  1×16*7   ⍝ 0x10000000 PKCS (repeated padding-length bytes)
CPAD_NIST ←  2×16*7   ⍝ 0x20000000 NIST SP800-38A (0x80,0x00...0x00)
CPAD_LENG ←  3×16*7   ⍝ 0x30000000 LENG (0x00...0x00,padding-length)
CPAD_ZERO ←  4×16*7   ⍝ 0x40000000 ZERO (null-bytes)
CPAD_EDI  ←  5×16*7   ⍝ 0x50000000 EDI (spaces)
```

Annotations

The RSA schemes (0 and 1 to 6) are for #.Crypt.PKey. Some of these, which are intended for signing (CPAD_SS15, CPAD_PSS) need to be combined with a digest algorithm, either to generate the object identifier sequence within the padding, or as an internal argument to the MGF1 of the probabilistic signature scheme. The key encryption scheme CPAD_OAEP is usually combined with the HASH_SHA1 as internal hash function of the MGF1.

The schemes for symmetric block ciphers (0 and 17 to 21) are intended for #.Crypt.Encrypt and #.Crypt.Decrypt.

As shown in the comment in hexadecimal, the padding is encoded in the 3rd byte and can be added to Cipher algorithms, Modes of operation, and Digest algorithms, encoded in other bytes, to compose a complete Algid.

For currently implemented and valid Algid-s refer to the Cipher suites section. In the current state of a very first release the enumeration is subject to change.

References

#.Crypt.Encrypt #.Crypt.Decrypt #.Crypt.PKey

Modes of operation

The cryptographic modes of operation are identified by constant values.

```
⍝                        hex                 mode
⍝                    ---------- -------------------------------

CMOD_ECB ← 0×256*3 ⍝ 0x00000000 ECB Electronic Code Book
CMOD_CBC ← 1×256*3 ⍝ 0x01000000 CBC Cipher Block Chaining
CMOD_CFB ← 2×256*3 ⍝ 0x02000000 CFB Cipher Feedback Mode
CMOD_OFB ← 3×256*3 ⍝ 0x03000000 OFB Output Feedback Mode
CMOD_CTR ← 4×256*3 ⍝ 0x04000000 CTR Counter Mode
CMOD_CTS ← 5×256*3 ⍝ 0x05000000 CTS Ciphertext Stealing Mode
CMOD_CCM ← 9×256*3 ⍝ 0x09000000 CCM Counter with CBC-MAC
CMOD_GCM ← 10×256*3 ⍝ 0x0A000000 GCM Galois/Counter Mode
CMOD_EAX ← 11×256*3 ⍝ 0x0B000000 EAX Authenticated Encryption Scheme
CMOD_CWC ← 12×256*3 ⍝ 0x0C000000 CWC Carter-Wegman and Counter Mode
CMOD_OCB ← 13×256*3 ⍝ 0x0D000000 OCB Offset Codebook Mode
```

Annotations

The modes from 0 to 5 do not inherit a data authentication. So these can be combined with (keyed) Digest algorithms. The CBC and ECB modes also require a Padding schemes unless the data size is a multiple of the cipher's block size. The ones from 9 to 13 are so called AEAD modes, performing an implicit authentication and padding. So, these cannot be combined with extra Digest algorithms or Padding schemes.

As shown in the comment in hexadecimal, the algorithm is encoded in the 4^{th} byte and can be added to Cipher algorithms, Padding schemes, and Digest algorithms, encoded in other bytes, to compose a complete Algid.

For currently implemented and valid Algids refer to the Cipher suites section. In the current state of a very first release the enumeration is subject to change.

References

#.Crypt.Encrypt #.Crypt.Decrypt

Cipher suites

The cipher suites are identified by combinations of ciphers, modes, schemes and digests.

```
PKEY_RSA_PK1E←PKEY_RSA+PPAD_PK1E                ⍝ RSA crypt RSAES-PKCS#1 V1.5
PKEY_RSA_OAEP←PKEY_RSA+PPAD_OAEP+HASH_SHA1      ⍝ RSA crypt RSAES-OAEP
PKEY_RSA_SSL3←PKEY_RSA+PPAD_SSL3                ⍝ RSA crypt SSL/TLS padding
PKEY_RSA_PK1S←PKEY_RSA+PPAD_PK1S                ⍝ RSASSA-PKCS#1 sign
PKEY_RSA_PK1S_WP←PKEY_RSA_PK1S+HASH_WP          ⍝ RSASSA-PKCS#1 sign Whirlpool
PKEY_RSA_PK1S_MD2←PKEY_RSA_PK1S+HASH_MD2        ⍝ RSASSA-PKCS#1 sign MD2
PKEY_RSA_PK1S_MD4←PKEY_RSA_PK1S+HASH_MD4        ⍝ RSASSA-PKCS#1 sign MD4
PKEY_RSA_PK1S_MD5←PKEY_RSA_PK1S+HASH_MD5        ⍝ RSASSA-PKCS#1 sign MD5
PKEY_RSA_PK1S_MDC2←PKEY_RSA_PK1S+HASH_MDC2      ⍝ RSASSA-PKCS#1 sign MDC-2
PKEY_RSA_PK1S_RMD160←PKEY_RSA_PK1S+HASH_RMD160  ⍝ RSASSA-PKCS#1 sign RIPEMD-160
PKEY_RSA_PK1S_SHA←PKEY_RSA_PK1S+HASH_SHA        ⍝ RSASSA-PKCS#1 sign SHA
PKEY_RSA_PK1S_SHA1←PKEY_RSA_PK1S+HASH_SHA1      ⍝ RSASSA-PKCS#1 sign SHA-1
PKEY_RSA_PK1S_SHA224←PKEY_RSA_PK1S+HASH_SHA224  ⍝ RSASSA-PKCS#1 sign SHA-224
PKEY_RSA_PK1S_SHA256←PKEY_RSA_PK1S+HASH_SHA256  ⍝ RSASSA-PKCS#1 sign SHA-256
PKEY_RSA_PK1S_SHA384←PKEY_RSA_PK1S+HASH_SHA384  ⍝ RSASSA-PKCS#1 sign SHA-384
PKEY_RSA_PK1S_SHA512←PKEY_RSA_PK1S+HASH_SHA512  ⍝ RSASSA-PKCS#1 sign SHA-512
PKEY_RSA_PSS←PKEY_RSA+PPAD_PSS                  ⍝ RSASSA/EMSA-PSS sign
PKEY_RSA_PSS_WP←PKEY_RSA_PSS+HASH_WP            ⍝ RSASSA/EMSA-PSS sign Whirlpool
PKEY_RSA_PSS_MD2←PKEY_RSA_PSS+HASH_MD2          ⍝ RSASSA/EMSA-PSS sign MD2
PKEY_RSA_PSS_MD4←PKEY_RSA_PSS+HASH_MD4          ⍝ RSASSA/EMSA-PSS sign MD4
PKEY_RSA_PSS_MD5←PKEY_RSA_PSS+HASH_MD5          ⍝ RSASSA/EMSA-PSS sign MD5
PKEY_RSA_PSS_MDC2←PKEY_RSA_PSS+HASH_MDC2        ⍝ RSASSA/EMSA-PSS sign MDC-2
PKEY_RSA_PSS_RMD160←PKEY_RSA_PSS+HASH_RMD160    ⍝ RSASSA/EMSA-PSS sign RIPEMD-160
PKEY_RSA_PSS_SHA←PKEY_RSA_PSS+HASH_SHA          ⍝ RSASSA/EMSA-PSS sign SHA
PKEY_RSA_PSS_SHA1←PKEY_RSA_PSS+HASH_SHA1        ⍝ RSASSA/EMSA-PSS sign SHA-1
PKEY_RSA_PSS_SHA224←PKEY_RSA_PSS+HASH_SHA224    ⍝ RSASSA/EMSA-PSS sign SHA-224
PKEY_RSA_PSS_SHA256←PKEY_RSA_PSS+HASH_SHA256    ⍝ RSASSA/EMSA-PSS sign SHA-256
PKEY_RSA_PSS_SHA384←PKEY_RSA_PSS+HASH_SHA384    ⍝ RSASSA/EMSA-PSS sign SHA-384
PKEY_RSA_PSSN_SHA512←PKEY_RSA_PSSN+HASH_SHA512  ⍝ RSASSA/EMSA-PSS sign SHA-512
PKEY_RSA_X931←PKEY_RSA+PPAD_X931                ⍝ RSA/ANSI-X9.31 sign
PKEY_RSA_X931_WP←PKEY_RSA_X931+HASH_WP          ⍝ RSA/ANSI-X9.31 sign Whirlpool
PKEY_RSA_X931_RMD160←PKEY_RSA_X931+HASH_RMD160  ⍝ RSA/ANSI-X9.31 sign RIPEMD-160
PKEY_RSA_X931_SHA1←PKEY_RSA_X931+HASH_SHA1      ⍝ RSA/ANSI-X9.31 sign SHA-1
PKEY_RSA_X931_SHA224←PKEY_RSA_X931+HASH_SHA224  ⍝ RSA/ANSI-X9.31 sign SHA-224
PKEY_RSA_X931_SHA256←PKEY_RSA_X931+HASH_SHA256  ⍝ RSA/ANSI-X9.31 sign SHA-256
PKEY_RSA_X931_SHA384←PKEY_RSA_X931+HASH_SHA384  ⍝ RSA/ANSI-X9.31 sign SHA-384
PKEY_RSA_X931_SHA512←PKEY_RSA_X931+HASH_SHA512  ⍝ RSA/ANSI-X9.31 sign SHA-512
PKEY_DSA_SHA1←PKEY_DSA+HASH_SHA1                ⍝ DSA sign SHA-1
PKEY_DSA_SHA224←PKEY_DSA+HASH_SHA224            ⍝ DSA sign SHA-224
PKEY_DSA_SHA256←PKEY_DSA+HASH_SHA256            ⍝ DSA sign SHA-256
PKEY_EC_SHA1←PKEY_EC+HASH_SHA1                  ⍝ ECDSA sign SHA-1
PKEY_EC_SHA224←PKEY_EC+HASH_SHA224              ⍝ ECDSA sign SHA-224
PKEY_EC_SHA256←PKEY_EC+HASH_SHA256              ⍝ ECDSA sign SHA-256
PKEY_EC_SHA384←PKEY_EC+HASH_SHA384              ⍝ ECDSA sign SHA-384
PKEY_EC_SHA512←PKEY_EC+HASH_SHA512              ⍝ ECDSA sign SHA-512

CIPH_DES_CBC←CIPH_DES+CMOD_CBC+CPAD_PKCS        ⍝ DES in CBC mode
CIPH_DES_ECB←CIPH_DES+CMOD_ECB+CPAD_PKCS        ⍝ DES in ECB mode
CIPH_DES_CFB←CIPH_DES+CMOD_CFB+CPAD_NONE        ⍝ DES in CFB mode
CIPH_DES_OFB←CIPH_DES+CMOD_OFB+CPAD_NONE        ⍝ DES in OFB mode
CIPH_DESX_CBC←CIPH_DESX+CMOD_CBC+CPAD_PKCS      ⍝ DES-X in CBC mode
CIPH_IDEA_CBC←CIPH_IDEA+CMOD_CBC+CPAD_PKCS      ⍝ IDEA in CBC mode
CIPH_IDEA_ECB←CIPH_IDEA+CMOD_ECB+CPAD_PKCS      ⍝ IDEA in ECB mode
CIPH_IDEA_CFB←CIPH_IDEA+CMOD_CFB+CPAD_NONE      ⍝ IDEA in CFB mode
CIPH_IDEA_OFB←CIPH_IDEA+CMOD_OFB+CPAD_NONE      ⍝ IDEA in OFB mode
CIPH_CAST_CBC←CIPH_CAST+CMOD_CBC+CPAD_PKCS      ⍝ CAST5 in CBC mode
CIPH_CAST_ECB←CIPH_CAST+CMOD_ECB+CPAD_PKCS      ⍝ CAST5 in ECB mode
CIPH_CAST_CFB←CIPH_CAST+CMOD_CFB+CPAD_NONE      ⍝ CAST5 in CFB mode
CIPH_CAST_OFB←CIPH_CAST+CMOD_OFB+CPAD_NONE      ⍝ CAST5 in OFB mode
CIPH_BF_CBC←CIPH_BF+CMOD_CBC+CPAD_PKCS          ⍝ Blowfish in CBC mode
CIPH_BF_ECB←CIPH_BF+CMOD_ECB+CPAD_PKCS          ⍝ Blowfish in EBC mode
```

```
CIPH_BF_CFB←CIPH_BF+CMOD_CFB+CPAD_NONE          ⍝ Blowfish in CFB mode
CIPH_BF_OFB←CIPH_BF+CMOD_OFB+CPAD_NONE          ⍝ Blowfish in OFB mode
CIPH_RC2_CBC←CIPH_RC2+CMOD_CBC+CPAD_PKCS        ⍝ RC2 in CBC mode
CIPH_RC2_ECB←CIPH_RC2+CMOD_ECB+CPAD_PKCS        ⍝ RC2 in ECB mode
CIPH_RC2_CFB←CIPH_RC2+CMOD_CFB+CPAD_NONE        ⍝ RC2 in CFB mode
CIPH_RC2_OFB←CIPH_RC2+CMOD_OFB+CPAD_NONE        ⍝ RC2 in OFB mode
CIPH_RC5_CBC←CIPH_RC5+CMOD_CBC+CPAD_PKCS        ⍝ RC5 in CBC mode
CIPH_RC5_ECB←CIPH_RC5+CMOD_ECB+CPAD_PKCS        ⍝ RC5 in ECB mode
CIPH_RC5_CFB←CIPH_RC5+CMOD_CFB+CPAD_NONE        ⍝ RC5 in CFB mode
CIPH_RC5_OFB←CIPH_RC5+CMOD_OFB+CPAD_NONE        ⍝ RC5 in OFB mode
CIPH_AES_CBC←CIPH_AES+CMOD_CBC+CPAD_PKCS        ⍝ AES in CBC mode
CIPH_AES_ECB←CIPH_AES+CMOD_ECB+CPAD_PKCS        ⍝ AES in ECB mode
CIPH_AES_CFB←CIPH_AES+CMOD_CFB+CPAD_NONE        ⍝ AES in CFB mode
CIPH_AES_OFB←CIPH_AES+CMOD_OFB+CPAD_NONE        ⍝ AES in OFB mode
CIPH_AES_CTR←CIPH_AES+CMOD_CTR+CPAD_NONE        ⍝ AES in CTR mode
CIPH_AES_CCM←CIPH_AES+CMOD_CCM                  ⍝ AES in CCM mode with auth
CIPH_AES_GCM←CIPH_AES+CMOD_GCM                  ⍝ AES in GCM mode with auth
CIPH_CM_CBC←CIPH_CM+CMOD_CBC+CPAD_PKCS          ⍝ Camellia in CBC mode
CIPH_CM_ECB←CIPH_CM+CMOD_ECB+CPAD_PKCS          ⍝ Camellia in ECB mode
CIPH_CM_CFB←CIPH_CM+CMOD_CFB+CPAD_NONE          ⍝ Camellia in CFB mode
CIPH_CM_OFB←CIPH_CM+CMOD_OFB+CPAD_NONE          ⍝ Camellia in OFB mode
CIPH_CM_CTR←CIPH_CM+CMOD_CTR+CPAD_NONE          ⍝ Camellia in CTR mode
CIPH_CM_GCM←CIPH_CM+CMOD_GCM                    ⍝ Camellia in GCM mode with auth
CIPH_SEED_CBC←CIPH_SEED+CMOD_CBC+CPAD_PKCS      ⍝ SEED in CBC mode
CIPH_SEED_ECB←CIPH_SEED+CMOD_ECB+CPAD_PKCS      ⍝ SEED in ECB mode
CIPH_SEED_CFB←CIPH_SEED+CMOD_CFB+CPAD_NONE      ⍝ SEED in CFB mode
CIPH_SEED_OFB←CIPH_SEED+CMOD_OFB+CPAD_NONE      ⍝ SEED in OFB mode
CIPH_SEED_GCM←CIPH_SEED+CMOD_GCM                ⍝ SEED in GCM mode with auth
```

Annotations

The cipher suites in the upper part are for the public-key signing and key encryption function #.Crypt.PKey.

The ones in the lower part are for data encryption with #.Crypt.Encrypt or data decryption with #.Crypt.Decrypt and can be combined with any of the Digest algorithms or even some of the public-key cipher suites.

References

#.Crypt.PKey #.Crypt.Encrypt #.Crypt.Decrypt

#.ASN1.Init

The function #.ASN1.Init initializes the access to the further functions in this section.

```
#.ASN1.Init
```

Annotations

Multiple executions of this function are ignored.

Requirements

Library	DyaCrypt.dll must exist in the executable directory.

References

#.ASN1.Exit ASN1.Code #.ASN1.Adjust

#.ASN1.Code

The function #.ASN1.Code encodes and decodes ASN.1 structures into nested arrays of a given depth.

```
(OutputStructure) ← #.ASN1.Code (InputStructure)

(OutputStructure) ← (Depth) #.ASN1.Code (InputStructure)

(OutputStructure) ← (Depth UniversalTagOptions) #.ASN1.Code (InputStructure)
```

Parameter values

Depth

[optional] This is the maximum depth (≡) of the output requested. If this value is 1, the output will be a text vector containing a completely encoded ASN.1 structure. If this value is 2, the output will be a vector with the three header values (class, form, tag) in the first element, and the content in successive simple elements. With each increase of Depth these successive elements get decoded one step deeper. If Depth is 0, the function will decode the content to maximum depth the structure has. The default value is 0.

UniversalTagOptions

[optional] This is an up to 30 elements numeric vector defining the decoding behavior for the universal tag data types. If the vector has less than 30 elements, the default behavior is chosen for the not given elements.

InputStructure

[required] This is an input vector containing a completely encoded ASN.1 structure or an arbitrarily decoded nested ASN.1 structure. The result does not depend on whether this is decoded or decoded.

Return values

OutputStructure

This is the resulting vector containing a completely encoded ASN.1 structure or an arbitrarily decoded nested ASN.1 structure. The grade of decoding is given in the Depth parameter.

Annotations

In cases of a syntactically incorrect input the result is an empty vector. The function #.ASN1.Adjust can then be used then to determine the correct length of an input vector having some trailing spare bytes.

Requirements

Loader	#.ASN1.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.ASN1.Init #.ASN.Adjust

#.ASN1.Adjust

The function #.ASN1.Adjust corrects the length of an ASN.1 structure.

```
(OutputStructure) ← #.ASN1.Adjust (InputStructure)
```

Parameter values

InputStructure

[required] A text vector containing an encoded ASN.1 structure, which might have some trailing spare bytes.

Return values

OutputStructure

A vector containing an encoded ASN.1 structure with the length as denoted inside the structure. If the implicit length is longer than the length of the input, the result is an empty vector.

Requirements

Loader	#.ASN1.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.ASN1.Init

#.ASN1.Exit

The function #.ASN1.Exit unloads the external library.

```
#.ASN1.Exit
```

Annotations

After execution the functions from this section are no longer available. Multiple executions are ignored.

Requirements

Loader	#.ASN1.Init must have been executed previously.
Library	DyaCrypt.dll must exist in the executable directory.

References

#.ASN1.Init

Class

These are the possible values for the first header element of an ASN.1 structure.

```
CLASS_UNIVERSAL   ← 0 ⍝ Universal        (defined by ITU-T X.680)
CLASS_APPLICATION ← 1 ⍝ Application      (defined by ITU-T X.nnn)
CLASS_CONTEXT     ← 2 ⍝ Context-specific (defined by its predecessor)
CLASS_PRIVATE     ← 3 ⍝ Private          (defined by the application)
```

References

#.ASN1.Code

Form

These are the possible values for the second header element of an ASN.1 structure, defining the encoding type of the following content.

```
FORM_PRIMITIVE   ← 0 ⍝  primitive   (content is a direct datum)
FORM_CONSTRUCTED ← 1 ⍝  constructed (content is a vector of ASN.1 elements)
```

References

#.ASN1.Code

Tag

These are the possible values for the third header element of an ASN.1 structure when being in the universal class. In the other three classes any non-negative number is allowed.

```
TAG_EOC             ←  0 ⍝ End-of-contents octets (indefinite length case)
TAG_BOOLEAN         ←  1 ⍝ TRUE or FALSE
TAG_INTEGER         ←  2 ⍝ Arbitrary precision integer
TAG_BITSTRING       ←  3 ⍝ Sequence of bits
TAG_OCTETSTRING     ←  4 ⍝ Sequence of bytes
TAG_NULLTAG         ←  5 ⍝ NULL (No Data will follow)
TAG_OID             ←  6 ⍝ Object Identifier (numeric sequence)
TAG_OBJDESCRIPTOR   ←  7 ⍝ Object Descriptor (human readable)
TAG_EXTERNAL        ←  8 ⍝ External / Instance Of
TAG_REAL            ←  9 ⍝ Real (Mantissa * Base^Exponent)
TAG_ENUMERATED      ← 10 ⍝ Enumerated
TAG_EMBEDDED_PDV    ← 11 ⍝ Embedded Presentation Data Value
TAG_UTF8STR         ← 12 ⍝ UTF-8 String (RFC2044)
TAG_RES_13          ← 13 ⍝ reserved
TAG_RES_14          ← 14 ⍝ reserved
TAG_RES_15          ← 15 ⍝ reserved
TAG_SEQUENCE        ← 16 ⍝ Constructed Sequence / Sequence Of
TAG_SET             ← 17 ⍝ Constructed Set / Set Of
TAG_NUMERICSTR      ← 18 ⍝ Numeric String (digits only)
TAG_PRINTABLESTR    ← 19 ⍝ Printable String
TAG_T61STR          ← 20 ⍝ T61 String (Teletex)
TAG_VIDEOTEXSTR     ← 21 ⍝ Videotex String
TAG_IA5STR          ← 22 ⍝ IA5 String
TAG_UTCTIME         ← 23 ⍝ UTC Time
TAG_GENERALIZEDTIME ← 24 ⍝ Generalized Time
TAG_GRAPHICSTR      ← 25 ⍝ Graphic String
TAG_VISIBLESTR      ← 26 ⍝ Visible String (ISO 646)
TAG_GENERALSTR      ← 27 ⍝ General String
TAG_UNIVERSALSTR    ← 28 ⍝ Universal String
TAG_RES_29          ← 29 ⍝ reserved
TAG_BMPSTR          ← 30 ⍝ Basic Multilingual Plane String
TAG_SUBSEQ          ← 31 ⍝ Subsequent (ASN1_ID2_Octets will follow)
```

References

#.ASN1.Code

Universal tag

These are the possible composed values for the headers of universal tag elements and some more syntax elements of the ASN.1 notation.

```
EOC             ←,⊂CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_EOC
BOOLEAN          ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_BOOLEAN
INTEGER          ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_INTEGER
BITSTRING        ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_BITSTRING
OCTETSTRING      ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_OCTETSTRING
NULLTAG         ←,⊂CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_NULLTAG
OID              ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_OID
OBJDESCRIPTOR    ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_OBJDESCRIPTOR
EXTERNAL         ← CLASS_UNIVERSAL FORM_CONSTRUCTED TAG_EXTERNAL
REAL             ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_REAL
ENUMERATED       ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_ENUMERATED
EMBEDDED_PDV     ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_EMBEDDED_PDV
UTF8STR          ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_UTF8STR
SEQUENCE         ← CLASS_UNIVERSAL FORM_CONSTRUCTED TAG_SEQUENCE
SET              ← CLASS_UNIVERSAL FORM_CONSTRUCTED TAG_SET
NUMERICSTR       ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_NUMERICSTR
PRINTABLESTR     ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_PRINTABLESTR
T61STR           ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_T61STR
VIDEOTEXSTR      ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_VIDEOTEXSTR
IA5STR           ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_IA5STR
UTCTIME          ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_UTCTIME
GENERALIZEDTIME  ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_GENERALIZEDTIME
GRAPHICSTR       ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_GRAPHICSTR
VISIBLESTR       ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_VISIBLESTR
GENERALSTR       ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_GENERALSTR
UNIVERSALSTR     ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_UNIVERSALSTR
BMPSTR           ← CLASS_UNIVERSAL FORM_PRIMITIVE   TAG_BMPSTR

CONTEXT          ← {CLASS_CONTEXT FORM_CONSTRUCTED ⍵}
IMPLICIT         ← {CLASS_CONTEXT FORM_PRIMITIVE ⍵}
OPTIONAL         ← {⍵:⍺ ⋄ ''}
DEFAULT          ← {⍺{(≡⍺)>≡⍵:((⊂⍴⍺)⊃⍺)∇ ⍵ ⋄ ⍺≡⍵}⍵:'' ⋄ ⍺}
```

References

#.ASN1.Code

Universal tag options

These are the possible values for the #.ASN1.Code parameter UniversalTagOptions.

```
UTO_STR    ← 0 ⍝ Code tag as string
UTO_NUM    ← 1 ⍝ Code TAG_BOOLEAN TAG_BITSTRING TAG_OID numerical (default)
UTO_SPCSEQ ← 2 ⍝ Code TAG_BITSTRING TAG_OCTETSTRING speculative if possible
UTO_SPCALL ← 3 ⍝ Code TAG_BITSTRING TAG_OCTETSTRING speculative if possible
UTO_I32    ← 1 ⍝ Code TAG_INTEGER TAG_ENUMERATED within 32 bit numeric
UTO_I48    ← 2 ⍝ Code TAG_INTEGER TAG_ENUMERATED within 48 bit numeric (default)
UTO_I53    ← 3 ⍝ Code TAG_INTEGER TAG_ENUMERATED within 53 bit numeric
UTO_FMT    ← 4 ⍝ Code TAG_INTEGER TAG_ENUMERATED as formatted squence
UTO_HEX    ← 8 ⍝ Code TAG_INTEGER TAG_ENUMERATED as hexadecimal string
UTO_ANSI   ← 1 ⍝ Code character string types as ANSI string (default)
UTO_WIDE   ← 2 ⍝ Code character string types as Unicode string
UTO_ZULU   ← 1 ⍝ Code TAG_UTCTIME TAG_GENERALIZEDTIME as Zulu time
UTO_LOCAL  ← 2 ⍝ Code TAG_UTCTIME TAG_GENERALIZEDTIME as local time (default)
UTO_AUTO   ← 4 ⍝ Code TAG_UTCTIME as Generalized Time if necessary

⍝ Defaults for class universal tag options

UnivTagOptions                      ← 30⍴⊂⍬
UnivTagOptions[TAG_BOOLEAN]         ← UTO_NUM
UnivTagOptions[TAG_INTEGER]         ← UTO_I48 ⍝ might be combined with UTO_FMT
UnivTagOptions[TAG_BITSTRING]       ← UTO_SPCSEQ
UnivTagOptions[TAG_OCTETSTRING]     ← UTO_SPCALL
UnivTagOptions[TAG_OID]             ← UTO_NUM
UnivTagOptions[TAG_ENUMERATED]      ← UTO_I48 ⍝ might be combined with UTO_FMT
UnivTagOptions[TAG_UTF8STR]         ← UTO_ANSI
UnivTagOptions[TAG_NUMERICSTR]      ← UTO_ANSI
UnivTagOptions[TAG_PRINTABLESTR]    ← UTO_ANSI
UnivTagOptions[TAG_T61STR]          ← UTO_ANSI
UnivTagOptions[TAG_VIDEOTEXSTR]     ← UTO_STR
UnivTagOptions[TAG_IA5STR]          ← UTO_ANSI
UnivTagOptions[TAG_UTCTIME]         ← UTO_LOCAL+UTO_AUTO
UnivTagOptions[TAG_GENERALIZEDTIME] ← UTO_LOCAL
UnivTagOptions[TAG_GRAPHICSTR]      ← UTO_STR
UnivTagOptions[TAG_VISIBLESTR]      ← UTO_STR
UnivTagOptions[TAG_GENERALSTR]      ← UTO_STR
UnivTagOptions[TAG_UNIVERSALSTR]    ← UTO_ANSI
UnivTagOptions[TAG_BMPSTR]          ← UTO_ANSI
```

References

#.ASN1.Code

OidTab

The OidTab is a matrix consisting of two columns and several hundred rows. In each row there is the OID number on the first element and the name of the OID on the second element. Below there is a short extract of the table for a better understanding this structure. This table can be used for an verbose presentation of an ASN.1 structure.

```
OidTab ← 0 2 ⍴ ⍬ 'ObjectIdentifier'
. . .
OidTab ⍪← (1 2 840 113549 1 1 1) 'pkcs-1-rsaEncryption'
OidTab ⍪← (1 2 840 113549 1 1 2) 'pkcs-1-md2WithRSAEncryption'
OidTab ⍪← (1 2 840 113549 1 1 3) 'pkcs-1-md4WithRSAEncryption'
OidTab ⍪← (1 2 840 113549 1 1 4) 'pkcs-1-md5WithRSAEncryption'
OidTab ⍪← (1 2 840 113549 1 1 5) 'pkcs-1-sha1WithRSAEncryption'
OidTab ⍪← (1 2 840 113549 1 1 6) 'pkcs-1-rsaOAEPEncryptionSET'
OidTab ⍪← (1 2 840 113549 1 1 7) 'pkcs-1-id-RSAES-OAEP'
OidTab ⍪← (1 2 840 113549 1 1 8) 'pkcs-1-id-mgf1'
OidTab ⍪← (1 2 840 113549 1 1 9) 'pkcs-1-id-id-pSpecified'
OidTab ⍪← (1 2 840 113549 1 5 1) 'pkcs-5-pbeWithMD2AndDES-CBC'
OidTab ⍪← (1 2 840 113549 1 5 3) 'pkcs-5-pbeWithMD5AndDES-CBC'
OidTab ⍪← (1 2 840 113549 1 5 4) 'pkcs-5-pbeWithMD2AndRC2-CBC'
OidTab ⍪← (1 2 840 113549 1 5 6) 'pkcs-5-pbeWithMD5AndRC2-CBC'
OidTab ⍪← (1 2 840 113549 1 5 9) 'pkcs-5-pbeWithMD5AndXOR'
OidTab ⍪← (1 2 840 113549 1 5 10)'pkcs-5-pbeWithSHA1AndDES-CBC'
OidTab ⍪← (1 2 840 113549 1 5 11)'pkcs-5-pbeWithSHA1AndRC2-CBC'
OidTab ⍪← (1 2 840 113549 1 7 1) 'pkcs-7-data'
OidTab ⍪← (1 2 840 113549 1 7 2) 'pkcs-7-signedData'
OidTab ⍪← (1 2 840 113549 1 7 3) 'pkcs-7-envelopedData'
OidTab ⍪← (1 2 840 113549 1 7 4) 'pkcs-7-signedAndEnvelopedData'
OidTab ⍪← (1 2 840 113549 1 7 5) 'pkcs-7-digestData'
OidTab ⍪← (1 2 840 113549 1 7 6) 'pkcs-7-encryptedData'
. . .
```

References

#.ASN1.Code