**Dyalog**

**Cryptographic**

**Library**

**(DCL)**

**Reference**

**Release: May 2011**

This product includes software developed by Peter-Michael Hager for Dyalog Ltd.

Copyright © 2011 Dyalog Ltd.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)

Copyright © 1998-2007 The OpenSSL Project.

This product includes cryptographic software written by Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com))

Copyright © 1995-1998 Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com))

# 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 4th 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

# Licence for use

The Dyalog Cryptographic Library includes components released under the following three licences. Use of the library is subject to all three.

## MIT Licence

Copyright © 2020 Dyalog Ltd.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## OpenSSL License

Copyright © 1998-2011 The OpenSSL Project.  All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"

1. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact [openssl-core@openssl.org](mailto:openssl-core@openssl.org).
2. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
3. Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com)).  This product includes software written by Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com))

## Original SSLeay License

Copyright © 1995-1998 Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com))

All rights reserved.

This package is an SSL implementation written by Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com)).

The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to.  The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code.  The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com)).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com))"

The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

1. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:

"This product includes software written by Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com))"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publicly available version or derivative of this code cannot be changed.  i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]