

## Simple Cryptographic Program Interface (Crypto API)

### Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

### Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

### Abstract

This document describes a simple Application Program Interface to cryptographic functions. The main purpose of such an interface is to separate cryptographic libraries from internet applications, thus allowing an independent development of both. It can be used in various internet applications such as [IPsec], [ISAKMP], [IKE], [TLS].

### Table of Contents

1. Introduction. . . . .	2
1.1. Summary . . . . .	2
1.2. Terminology . . . . .	2
1.3. Objectives of Development . . . . .	3
2. Cryptoplugin Structure. . . . .	3
3. Program Interface . . . . .	4
3.1. Cryptoplugin Initialization Function. . . . .	4
3.1.1. Description of CryptoPluginInfo structure . . . . .	6
3.1.2. Description of CryptoAlgInfo structure. . . . .	6
3.2. Cryptoplugin Deinitialization Function. . . . .	9
3.3. Cryptographic Context Opening Function. . . . .	10
3.4. Cryptographic Context Reopening Function. . . . .	11
3.5. Cryptographic Context Closing Function. . . . .	12
3.6. Key Verification Function . . . . .	12
3.7. Data Transformation Function. . . . .	13
3.7.1. For CRYPTO_TYPE_ENCRYPT Algorithm Type. . . . .	13
3.7.2. For CRYPTO_TYPE_DECRYPT Algorithm Type. . . . .	14
3.7.3. For CRYPTO_TYPE_SIGN Algorithm Type . . . . .	15
3.7.4. For CRYPTO_TYPE_VERIFY Algorithm Type . . . . .	17
3.7.5. For CRYPTO_TYPE_COMPRESS Algorithm Type . . . . .	18

3.7.6. For CRYPTO_TYPE_UNCOMPRESS Algorithm Type . . . . .	18
3.7.7. For CRYPTO_TYPE_HASH Algorithm Type . . . . .	19
3.7.8. For CRYPTO_TYPE_RANDOM Algorithm Type. . . . .	21
3.8. Cryptographic Context Control Function. . . . .	22
4. Cryptoplugin Registration Procedure . . . . .	23
5. Security Considerations . . . . .	23
6. References. . . . .	23
7. Author's Address . . . . .	24
Appendix A. The interface specification as a C header file . . . .	25
Full Copyright Statement . . . . .	30

## 1. Introduction

### 1.1. Summary

Nowadays internet applications that require cryptographic functions at the level of operating system kernel, use the method that assumes the libraries must be compiled/linked together with the module (driver) which provides product functionality. For the sake of possibility of independent development of the cryptographic modules and in order to provide a simple, effective and universal (suitable for application and as well kernel level of operating system) solution this specification offers the method to extract encrypting algorithms to the separate cryptographic modules.

This document describes simple open interface (Crypto API) to external cryptographic libraries optimized both for the application and kernel level of the operating system.

### 1.2. Terminology

#### Cryptoplugin

Operation system unit (driver, shared library, module) that provides cryptographic functions via well-defined (but OS-specific) interface.

#### Cryptolibrary

Part of cryptoplugin that provides its cryptographic functionality via Crypto API.

#### Wrapper

Part of cryptoplugin that provides interfaces translation between Crypto API and OS-specific interface.

Definition of all cryptography related terms can be found in [Schneier].

### 1.3. Objectives of Development

The objectives of Simple CryptoAPI development are as follows:

- 1) To extract program implementations of encryption, one-way hash function, digital signature and random numbers generation algorithms to separate, independently developed modules.
- 2) To provide version independence between using encryption modules and external cryptoplugin.
- 3) To ensure platform independent developments of encrypting algorithm modules with portable source code.
- 4) To enable independent development of modules and compatibility of modules developed independently.

## 2. Cryptoplugin Structure

In order to provide fast exchange between the cryptoplugin and its client the cryptoplugin is implemented as a separate driver (or module) of the particular operating system (Fig.1). Cryptoplugin consists of two parts (Fig.2):

- 1) cryptolibrary itself (1)
- 2) system-dependent module (wrapper) for interaction between cryptolibrary and its client (2)

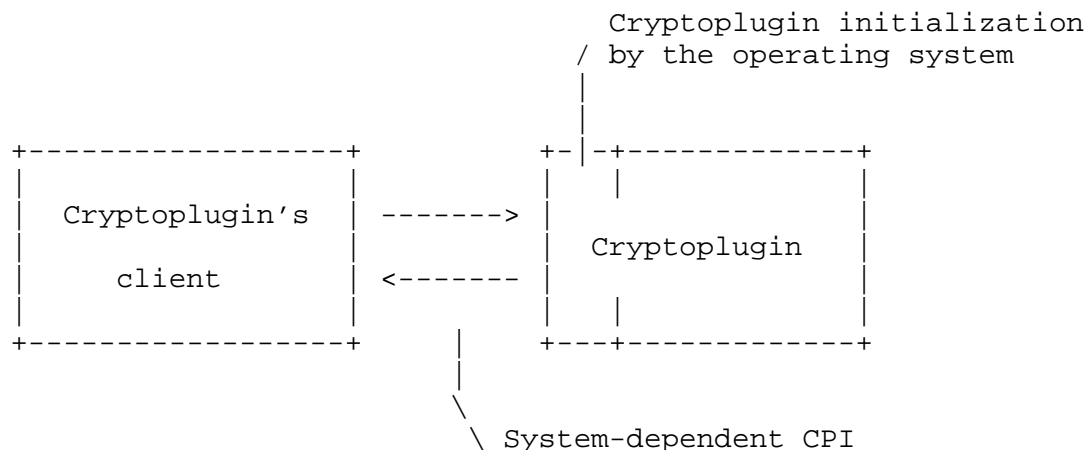


Fig. 1 Interaction between cryptoplugin and its client

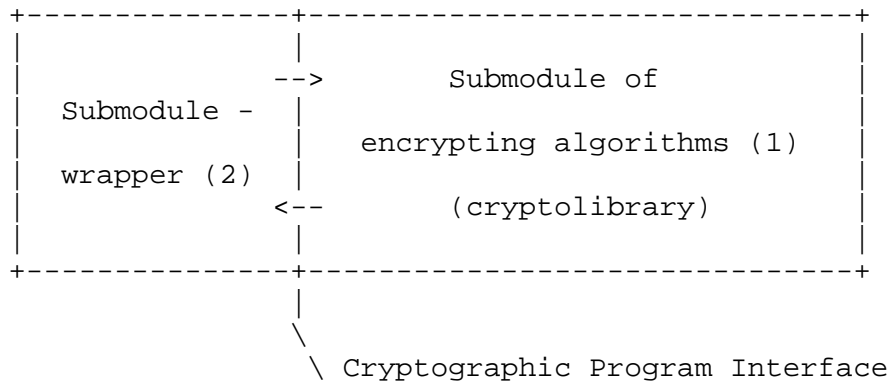


Fig. 2 Cryptoplugin structure

The system-dependent module (wrapper) is delivered by the driver-client developer in the form of source code or in the form of libraries (for example, in the form of object files) for particular operating system. The wrapper is intended for translation of system-independent application interface to the particular system-dependent interface with the cryptoplugin's client. The wrapper context does not include components specific to cryptoplugin's client functionality or to some cryptographic algorithm. The interface described in section 3 is the standard for interaction between the submodules (1) and (2).

A cryptoplugin can contain a number of different algorithms. Moreover, it can contain some different implementations of one particular algorithm.

### 3. Program Interface

The CPI (Cryptographic Program Interface) consists of a set of functions exported by encrypting algorithm submodule (cryptolibrary). The interface functions are described below (see also Appendix A).

#### 3.1. Cryptoplugin Initialization Function

The function is intended for cryptoplugin initialization and obtaining information about algorithms contained in cryptoplugin. The function is called once before the beginning of cryptoplugin operation.

```

/* CryptoPlugin initialization. Returns pointer to CryptoPluginInfo
structure on success or NULL on fatal error. */
CryptoPluginInfo *CryptoPluginInit(
    void                *param); /* Ptr to OS parameters
                                (platform-specific) */
  
```

#### Description of parameters:

param - pointer to system-dependent parameters transmitted to cryptoplugin by the operating system. Intention and format of parameters are specific to each operating system and should be described in documentation on the cryptoplugin wrapper.

The function is called at the moment of cryptoplugin initialization. If succeeded it returns the pointer to CryptoPluginInfo structure that describes the module and algorithms implemented in the cryptolibrary. If function call did not succeed, function will return NULL or appropriate error code in CryptoPluginInfo structure status field. If the initialization is partially succeeded then the cryptoplugin either returns CryptoPluginInfo structure transformed so that it contains only successfully initialized algorithms or returns appropriate error code in status field of CryptoAlgInfo structures that describes the reason for the failure.

#### Error codes for the function:

NULL - fatal unsuccessful cryptoplugin initialization. The module is unable even to indicate the reason of failure.

The pointer to cryptoplugin description structure in the case of full or partial success. The status fields in CryptoPluginInfo structure and in comprised CryptoAlgInfo structures can be set to the following values:

CRYPTO\_OK - cryptoplugin (algorithm) is initialized successfully.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_NOT\_SUPPORTED - (only for algorithm) - the algorithm is not supported by the module at the moment.

CRYPTO\_ERR\_HARDWARE - error of hardware initialization.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

### 3.1.1. Description of CryptoPluginInfo structure

The CryptoPluginInfo structure consists of header of fixed size that generally describes cryptoplugin and array of CryptoAlgInfo structures following the header. Each structure describes particular algorithm implemented in the cryptolibrary (see Appendix A)

Structure fields description:

*api\_version* - CPI version (should be CRYPTO\_VER (1,0)). CPI version determines both functions set and fields layout in CryptoPluginInfo/CryptoAlgInfo structures.

*status* - returns the error code if cryptoplugin initialization failed (otherwise should be CRYPTO\_OK)

*name* - text cryptoplugin description (ASCII-7 characters only; all unused bytes must be set to 0).

*version* - cryptoplugin version (CRYPTO\_VER(maj,min)).

*flags* - various flags that characterize the cryptoplugin.

*number\_of\_algs* - number of algorithms the cryptolibrary comprises of (i.e. the number of consequent CryptoAlgInfo structures).

### 3.1.2. Description of CryptoAlgInfo structure

Structure fields description

*status* - returns the error code if particular algorithm initialization failed (otherwise should be CRYPTO\_OK).

*id* - algorithm identifier (CRYPTO\_A\_XXX). Values in the range of 0..249 are reserved; Values in the range of 250..32767 indicate algorithms not enrolled in standard list. It should be emphasized that algorithm IDs are independent for each algorithm type. But it is considered that pairs of types CRYPTO\_TYPE\_ENCRYPT and CRYPTO\_TYPE\_DECRYPT, CRYPTO\_TYPE\_SIGN and CRYPTO\_TYPE\_VERIFY, CRYPTO\_TYPE\_COMPRESS and CRYPTO\_TYPE\_UNCOMPRESS are equivalent because they define reverse actions of the same nature.

group - algorithm implementation group (variants algorithm implementations with various parameters not covered by CryptoAlgInfo structure). Values in the range of 0..32767 are well-known numbers defined in Appendix A; vendors may arbitrarily use values in the range of 32768..65535.

type - algorithm type (CRYPTO\_TYPE\_XXX). Unambiguously determines algorithm application.

version - version of algorithm implementation (CRYPTO\_VER (maj,min)).

flags - flags that characterize the algorithm and its implementation. All bits, that are not defined in Appendix A, must be zeroed.

maxcontexts - maximum cryptographic contexts number that are simultaneously supported by the algorithm implementation (0 if the number is unlimited or is limited only by environmental conditions like memory size).

name - text algorithm name (ASCII characters use only; all unused bytes must be set to 0).

The next information depends on algorithm type:

For encryption algorithms (CRYPTO\_TYPE\_ENCRYPT and CRYPTO\_TYPE\_DECRYPT):

blocklen - data block length in bytes (value 1 must be used for stream cipher algorithms).

keylen - encrypting (or decrypting) key length in bytes.

outlen - output data size for conversion of one input data block in bytes. Usually it is equal to blocklen. When prediction of this value is impossible zero value must be indicated.

milen - size of initialization vector (for block algorithms) or message indicator (for stream algorithms) in bytes. For block algorithms zero value of the parameter means that the algorithm implements ECB encoding. Non-zero milen parameter means that the algorithm implements CBC encoding. For stream algorithms zero value of the parameter means that the message indicator is not required.

For signature algorithms (CRYPTO\_TYPE\_SIGN):

blocklen - block size in bytes. The length of input signature data will be padded up to this value. When there is no need in padding the value of 1 must be set.

keylen - private key length in bytes.

outlen - signature length in bytes. When prediction of this value is impossible 0 value must be indicated. If the signature consists of several values then the total length is indicated.

milen - non-zero value specifies signature parameter length (random number), zero value indicates that the parameter is not required.

For signature verification algorithms (CRYPTO\_TYPE\_VERIFY):

blocklen - is not used.

keylen - length of public key in bytes.

outlen - signature length in bytes. When prediction of this value is impossible 0 value must be indicated. If the signature consists of several values then the total length is indicated.

milen - is not used.

For data compression algorithms (CRYPTO\_TYPE\_COMPRESS):

blocklen - see outlen.

keylen - is not used.

outlen - if the algorithm provides the fixed compression with known value then it is indicated as blocklen/outlen ratio. The values can be arbitrary. If the compression value is not known then outlen is set to 0 and blocklen is not used.

milen - is not used.

For data uncompressing algorithms (CRYPTO\_TYPE\_UNCOMPRESS):

blocklen - see outlen.

keylen - is not used.

outlen - if the algorithm provides the fixed compression with known value then it is indicated as blocklen/outlen ratio. The values can be arbitrary. It is natural that the ratio will be reverse to the similar value for the same algorithm but of CRYPTO\_TYPE\_COMPRESS type. If the compression value is not known then outlen is set to 0 and blocklen is not used.

milen - is not used.

For one-way hash function algorithms (CRYPTO\_TYPE\_HASH):

blocklen - block size in bytes. The length of input data will be padded up to this value. When there is no need in padding value 1 should be used.

keylen - is not used.

outlen - resulting hash value length in bytes.

milen - is not used.

For random number generation algorithms (CRYPTO\_TYPE\_RANDOM):

blocklen - is not used.

keylen - initial seed length (0 - if not required, for example in a physical effects based generators).

outlen - resulting random number length in bytes (0 - arbitrary)

milen - is not used.

### 3.2. Cryptoplugin Deinitialization Function

```
/* Plugin deinitialization */  
CRYPTO_STATUS  CryptoPluginFini(void);
```

The function is called before the cryptoplugin operation is to be terminated. Function execution causes closing of all open cryptographic contexts, system resources deallocation and hardware deinitialization. The value returned is informational only.

Return codes for the function:

CRYPTO\_OK - cryptoplugin is deinitialized successfully.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_UNCLOSED\_HANDLES - warning that there were open cryptographic contexts during cryptoplugin deinitialization. The warning is informational only. The open contexts are destroyed anyway.

### 3.3. Cryptographic Context Opening Function

```
New algorithm instance (cipher state) */
CRYPTO_STATUS  CryptoOpen(

    CRYPTO_HANDLE  *state, /* Pointer to cipher state
                           handle (filled on exit) */
    long           alnum, /* Algorithm number in
                           CryptoPluginInfo structure */
    const char     *key); /* key (in plain) */
```

The function creates cryptographic context copy inside cryptoplugin and initializes it with the provided key. Later the handle of the context is used in calls of other algorithm functions.

Description of parameters:

state - pointer to the variable that will be set to the handle of the context created if succeeded. NULL parameter value should result in the CRYPTO\_ERR\_BAD\_PARAMS code returned by the function.

alnum - algorithm number in the cryptoplugin. It is equal to the number of CryptoAlgInfo structure (that describes the algorithm) in CryptoPluginInfo structure. The number begins with zero value. It should be taken into account that it is not an algorithm identifier but its number in the cryptoplugin.

key - pointer to the key (if it is required) or to the seed (for random number generation algorithm).

Notes.

1. Generated cryptographic context is stored inside the cryptoplugin until it will be destroyed by the CryptoAlgClose function call. The maximum number of cryptographic contexts supported by cryptoplugin can be indicated in algorithm parameters description. If maximum number of cryptographic contexts equals to zero then the cryptographic contexts number is either unlimited (for example, for stateless algorithms like random number generators and one-way hash functions) or it is limited by external factors only (like memory size).

Return codes for the function:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters (invalid algorithm number, zero pointer to the handle or to key (seed) if it is required).

### 3.4. Cryptographic Context Reopening Function

```
/* Reinitialize algorithm instance */
CRYPTO_STATUS  CryptoReOpen(
    CRYPTO_HANDLE  state, /* current cipher state handle */
    const char     *key); /* key (in plain) */
```

The function reinitializes an existing context. This function is used for key change without new system resources allocation. The function parameters are handle of opened earlier context and pointer to a new key.

Return codes for the function:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general

CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module may release system memory and try function call once more.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters (invalid algorithm number, zero pointer to the handle or to key (seed) if it is required).

### 3.5. Cryptographic Context Closing Function

```
/* Destroy algorithm instance */
CRYPTO_STATUS  CryptoClose(
    CRYPTO_HANDLE  state); /* Handle of cipher state */
```

The function provides cryptographic context destruction. The cryptographic context handle is its parameter. The value returned is informational only.

Return codes for the function:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

### 3.6. Key Verification Function

```
/* Check key for possible weakness */
CRYPTO_STATUS  CryptoCheckForWeakKey(
    long                alnum, /* Algorithm number in
                                CryptoPluginInfo structure */
    const char         *key); /* Proposed key */
```

The function verifies key material whether it is weak (from the algorithm's point of view). The function is actual for encryption/decryption or signing/verification algorithms only. Algorithm number (similar to CryptoAlgOpen) and pointer to the key to be verified are the parameters.

Return codes for the function:

CRYPTO\_O - the key has passed the test.

CRYPTO\_ERR\_WEAK\_KEY - the key has not passed the test (being weak or possibly weak).

CRYPTO\_ERR\_NOT\_SUPPORTED - is not supported.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

### 3.7. Data Transformation Function

```

/* Perform CryptoTransform (depends on cipher state type) */
CRYPTO_STATUS  CryptoTransform(
    CRYPTO_HANDLE  state, /* Cipher state */
    const char     *inbuff, /* input data */
    long           inlen, /* input data length */
    char           *outbuff, /* output buffer */
    long           *outlen, /* On entry - output buffer
                           length, on exit - number of
                           bytes written to outbuff */
    char           *mi); /* Message indicator */

```

This is a cryptographic data transformation function. Function call results and function parameters are dependent on algorithm type. For algorithm types CRYPTO\_TYPE\_ENCRYPT, CRYPTO\_TYPE\_DECRYPT, CRYPTO\_TYPE\_SIGN and CRYPTO\_TYPE\_VERIFY (items 3.7.1 - 3.7.4) function call results are history independent.

Note. Stream encryption algorithms may seem an "exception". However the same cryptoalgorithm handle must hide its history dependence. For algorithm types CRYPTO\_TYPE\_COMPRESS, CRYPTO\_TYPE\_UNCOMPRESS and CRYPTO\_TYPE\_HASH (items 3.7.5 - 3.7.7) function calls are history dependent. For the CRYPTO\_TYPE\_RANDOM algorithm function call may be for different implementations either dependent or independent on the history.

#### 3.7.1. For CRYPTO\_TYPE\_ENCRYPT Algorithm Type:

The function encrypts input data. Its parameters are intended for:

inbuff - pointer to the input data. If this parameter is equal to NULL then the function should return the CRYPTO\_ERR\_BAD\_PARAMS error code.

inlen - input data size (in bytes). If the size indicated in algorithm description is divisible by blocklen then padding is not carried out. Otherwise the algorithm either carries out padding according to the algorithm standard or returns appropriate error code (CRYPTO\_ERR\_BAD\_PARAMS). The zero parameter is allowed so that the function quits at once and returns CRYPTO\_OK code.

outbuff - output data buffer. NULL parameter value results in the outlen parameter setting to output buffer size required to encrypt the input buffer represented. In this case the CRYPTO\_ERR\_SMALL\_BUFFER error should not be returned.

outlen - Output buffer size is an input function parameter while the number of bytes written in the output buffer is the output parameter. Both the NULL parameter value and the zero value addressed result in CRYPTO\_ERR\_BAD\_PARAMS code returned by the function.

mi - message indicator. Its content depends on whether the block or stream algorithm is applied. In the block algorithm case it is set to the last block encrypted. When the first block is encrypted mi parameter specifies initial initialization vector. In the stream algorithm case it is set to the offset of the first byte encrypted in the stream. If the algorithm uses the message indicator and the mi parameter value is set to NULL then function should return CRYPTO\_ERR\_BAD\_PARAMS. If the algorithm (ECB Mode encrypting as an example) does not apply the message indicator then NULL value of mi is acceptable while non-NULL value should be ignored.

Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

3.7.2. For CRYPTO\_TYPE\_DECRYPT Algorithm Type:

The function decrypts the input data. Its parameters are intended for:

inbuff - pointer to the input data. If the parameter is equal to NULL then the function should return the CRYPTO\_ERR\_BAD\_PARAMS error code.

inlen - input data size (in bytes). When the parameter is set to zero the function quits at once and CRYPTO\_OK code is returned.

outbuff - output data buffer. NULL parameter value results in the outlen parameter setting to output buffer size required to decrypt the input buffer represented. In this case the CRYPTO\_ERR\_SMALL\_BUFFER error should not be returned.

outlen - Output buffer size is an input function parameter while the number of bytes written in the output buffer is the output parameter. Both the NULL parameter value and the zero value addressed result in CRYPTO\_ERR\_BAD\_PARAMS code returned by the function.

mi - message indicator. The content depends on whether the block or stream algorithm is applied. In the block algorithm case it is set to the last block encrypted. When the first block is decrypted mi specifies initial initialization vector. In the stream algorithm case it is set to the offset of the first byte decrypted in the stream. If the algorithm uses the message indicator and the mi parameter is set to NULL then function should return CRYPTO\_ERR\_BAD\_PARAMS. If the algorithm (ECB Mode as an example) does not apply the message indicator then NULL value of mi is acceptable while non-NULL value should be ignored.

#### Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

#### 3.7.3. For CRYPTO\_TYPE\_SIGN Type Algorithm:

The function signs the input data. Its parameters are intended for:

`inbuff` - pointer to the input data. If the parameter is equal to `NULL` then the function should return the `CRYPTO_ERR_BAD_PARAMS` code error.

`inlen` - input data size (in bytes). If the size indicated in algorithm description is divisible by `blocklen` then padding is not carried out. Otherwise the algorithm either carries out padding according to the algorithm standard or returns appropriate error code (`CRYPTO_ERR_BAD_PARAMS`). The zero parameter is allowed so that the function quits at once and returns `CRYPTO_OK` code.

`outbuff` - output data buffer. `NULL` parameter value results in the `outlen` parameter setting to output buffer size required to sign the input buffer represented. In this case the `CRYPTO_ERR_SMALL_BUFFER` error should not be returned.

`outlen` - Output buffer size is an input function parameter while the number of bytes written in the output buffer is the output parameter. Both the `NULL` parameter value and the zero value addressed result in `CRYPTO_ERR_BAD_PARAMS` code returned by the function.

`mi` - pointer to signature parameter (random number usually) if `milenn` parameter in algorithm description is non-zero. In this case zero `mi` parameter indicates that the parameter should be chosen (generated) inside the algorithm. If `milenn` parameter in algorithm description is set to zero then `mi` parameter is ignored.

Returned values:

`CRYPTO_OK` - successful completion.

`CRYPTO_ERR_GENERAL` - internal error.

`CRYPTO_ERR_BAD_HANDLE` - invalid cryptographic context handle.

`CRYPTO_ERR_NO_RESOURCES` - insufficient internal resources.

`CRYPTO_ERR_NO_MEMORY` - not enough memory. Contrary to general `CRYPTO_ERR_NO_RESOURCES` error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

`CRYPTO_ERR_SMALL_BUFFER` - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

#### 3.7.4. For CRYPTO\_TYPE\_VERIFY Algorithm Type:

The function verifies input data signature. Its parameters are intended for:

inbuff - pointer to the input data. If the parameter is equal to NULL then the function should return the CRYPTO\_ERR\_BAD\_PARAMS code error.

inlen - input data size (in bytes). The zero parameter is allowed so that the function quits at once and returns CRYPTO\_OK code.

outbuff - pointer to the signature. If the parameter is set to NULL then the function returns CRYPTO\_ERR\_BAD\_PARAMS error code. If the signature consists of several parts then they are combined to one array.

outlen - specifies the signature length if the signature length is set to zero in algorithm description structure. If non-zero value is specified in algorithm description structure then the parameter is ignored. If the signature consists of several parts then the maximum part length multiplied by the number of parts is specified.

mi - is not used.

Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_INVALID\_SIGNATURE - invalid signature.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

### 3.7.5. For CRYPTO\_TYPE\_COMPRESS Algorithm Type:

The function compresses the input data. Its parameters are intended for:

inbuff - pointer to the input data.

inlen - input data size (in bytes). The zero parameter is allowed so that the function quits at once and returns CRYPTO\_OK code.

outbuff - output data buffer. NULL parameter value results in the outlen parameter setting to output buffer size required to compress the input buffer represented. In this case the CRYPTO\_ERR\_SMALL\_BUFFER error should not be returned.

outlen - Output buffer size is an input function parameter while the number of bytes written in the output buffer is the output parameter. Both the NULL parameter value and the zero value addressed result in CRYPTO\_ERR\_BAD\_PARAMS code returned by the function.

mi - is not used.

Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources  
CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

### 3.7.6. For CRYPTO\_TYPE\_UNCOMPRESS Algorithm Type:

The function decompresses the input data. Its parameters are intended for:

inbuff - pointer to the input data.

inlen - input data size (in bytes). The zero parameter is allowed so that the function quits at once and returns CRYPTO\_OK code.

outbuff - output data buffer. NULL parameter value results in the outlen parameter setting to output buffer size required to decompress the input buffer represented. In this case the CRYPTO\_ERR\_SMALL\_BUFFER error should not be returned.

outlen - Output buffer size is an input function parameter while the number of bytes written in the output buffer is the output parameter. Both the NULL parameter value and the zero value addressed result in CRYPTO\_ERR\_BAD\_PARAMS code returned by the function.

mi - is not used.

Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

### 3.7.7. For CRYPTO\_TYPE\_HASH Algorithm Type:

The function calculates the hash value of the input data. Its parameters are intended for:

inbuff - pointer to the input data. If the parameter is of NULL value then the function calculates cumulative hash value for the data represented (taking into account all previous data represented). If total length of all the data represented by the moment is divisible by blocklen and outbuff is non-NULL then it is returned to outbuff. Nothing is written in outbuff when the length is not divisible by blocklen. NULL inbuff indicates the last conversion when the input data is padded up

to the blocklen size and the result is written to outbuff address. The padding procedure is defined for the algorithm.

inlen - input data size (in bytes). The zero parameter is allowed when the function quits at once and returns CRYPTO\_OK code.

outbuff - output data buffer.

outlen - Output buffer size is an input function parameter while the number of bytes written in the output buffer is the output parameter. If intermediate conversion value (inbuff is not NULL) and total length of data represented by the moment are not divisible by blocklen then outlen is set to zero and the hash value is not written in outbuff. Both the NULL parameter value and the zero value addressed result in CRYPTO\_ERR\_BAD\_PARAMS code returned by the function.

mi - is not used.

#### Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

### 3.7.8. For CRYPTO\_TYPE\_RANDOM Algorithm Type:

The function generates a random number. Its parameters are intended for:

inbuff - pointer to the input data used for generation (when one of the pseudorandom algorithms is implemented). NULL parameter indicates absence of the input data.

inlen - input data size (in bytes).

outbuff - output data

outlen - Output buffer size is an input function parameter while the number of bytes written in the output buffer is the output parameter. If zero (i.e. arbitrary) generated number size is set in the algorithm description then the outlen value determines the number of random bytes required by the calling procedure.

mi - is not used.

Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

### 3.8. Cryptographic Context Control Function

```

/* Algorithm control */
CRYPTO_STATUS  CryptoControl(
    CRYPTO_HANDLE  state, /* Cipher state handle */
    long           cmd,   /* Control command */
    long           param, /* Parameter id */
    char           val,   /* Parameter value */
    long           *len); /* For CRYPTO_GET: on entry -
                           val buffer length, on exit -
                           number of bytes written to
                           val; for CRYPTO_SET: length
                           of value to set */

```

The function provides cryptographic context internal parameters management. It may be used to check context parameters or to change the context state, for example it may return information about cryptoalgorithm (is given context uses hardware encryption facilities), or it may "scroll" stream algorithms context if necessary, etc.

Description of parameters:

state - cryptographic context handle.

cmd - command (CRYPTO\_GET or CRYPTO\_SET).

param - identifier of parameter. Values in the range of 0..32767 are assigned well-known numbers for all algorithms. Values in the range of 32768..65535 mean various variables for various algorithms (may be arbitrarily used by cryptolibrary developer).

val - pointer to the data buffer.

len - data size (in bytes).

Returned values:

CRYPTO\_OK - successful completion.

CRYPTO\_ERR\_GENERAL - internal error.

CRYPTO\_ERR\_BAD\_HANDLE - invalid cryptographic context handle.

CRYPTO\_ERR\_NO\_RESOURCES - insufficient internal resources.

CRYPTO\_ERR\_NO\_MEMORY - not enough memory. Contrary to general

CRYPTO\_ERR\_NO\_RESOURCES error this code assumes that the calling module can release system memory (if it is in position to) and try to call the function once again.

CRYPTO\_ERR\_SMALL\_BUFFER - insufficient output buffer size.

CRYPTO\_ERR\_BAD\_PARAMS - invalid parameters.

#### 4. Cryptoplugin Registration Procedure

Cryptoplugin should be linked together with the cryptoplugin wrapper library delivered by the cryptoplugin's client developer according to the rules specified by the module-client developer for each platform. It should result in a driver (module) of appropriate operating system that implements the cryptolibrary functions. The driver should be one of the drivers loaded during operating system boot. The procedure of cryptoplugin driver installation should be defined, documented, and automated when necessary, by the cryptoplugin developer. At the beginning of operation the driver-client determines cryptoplugin driver availability and establishes interconnection with it. Both module-client configuration and current security policy determine data conversion algorithms to be chosen.

#### 5. Security Considerations

Security issues are addressed throughout this memo.

#### 6. References

- [Schneier] Bruce Schneier, Applied Cryptography - Protocols, Algorithms, and Source Code in C (Second Edition), John Wiley & Sons, Inc., 1996.
- [IPsec] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.
- [ISAKMP] Maughan, D., Schertler, M. Schneider, M. and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.
- [IKE] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.
- [TLS] Dierks, T. and C. Allen, "The TLS protocol Version 1.0", RFC 2246, January 1999.

## 7. Author's Address

Valery Smyslov  
TWS  
Centralny prospekt, 11,  
Moscow, Russia

Phone: +7 (095) 531 4633  
Fax: +7 (095) 531 2403  
EMail: [svan@trustworks.com](mailto:svan@trustworks.com)

## Appendix A. The interface specification as a C header file

```

#ifndef __CRYPTPI_H
#define __CRYPTPI_H

#define CRYPTO_VER(maj,min)      (((maj & 0xff) << 8) | (min & 0xff))
#define CRYPTO_MAJ_VER(ver)      ((ver >> 8) & 0xff)
#define CRYPTO_MIN_VER(ver)      (ver & 0xff)

#define CRYPTO_PLUGIN_NAME_LEN 64      /* Must be multiple of 4 to */
#define CRYPTO_ALG_NAME_LEN 32        /* avoid alignment problems */

#ifndef CRYPTO_HANDLE
#define CRYPTO_HANDLE            void*   /* cipher state handle */
#endif

typedef enum tag_CRYPT_STATUS {
    CRYPTO_OK = 1,                  /* success */
    CRYPTO_ERR_GENERAL,              /* undefined (internal) error */
    CRYPTO_ERR_NOT_SUPPORTED,        /* unsupported */
    CRYPTO_ERR_BAD_HANDLE,           /* invalid handle */
    CRYPTO_ERR_SMALL_BUFFER,         /* insufficient output buffer
                                     size */
    CRYPTO_ERR_WEAK_KEY,              /* key is considered to be weak
                                     (semiweak, pseudoweak) */
    CRYPTO_ERR_NO_RESOURCES,         /* insufficient resources to
                                     perform operation */
    CRYPTO_ERR_NO_MEMORY,            /* insufficient memory to
                                     perform operation */
    CRYPTO_ERR_BAD_PARAMS,           /* invalid parameters */
    CRYPTO_ERR_HARDWARE,             /* hardware error */
    CRYPTO_ERR_INVALID_SIGNATURE,    /* invalid signature */
    CRYPTO_ERR_UNCLOSED_HANDLES      /* unclosed handles exist while
                                     plugin deinitializes */
} CRYPTO_STATUS;

/* CryptoControl commands */
#define CRYPTO_GET                  1      /* get parameter */
#define CRYPTO_SET                  2      /* set parameter */

/* Currently defined algorithm types */
#define CRYPTO_TYPE_ENCRYPT          1
#define CRYPTO_TYPE_DECRYPT          2
#define CRYPTO_TYPE_SIGN             3
#define CRYPTO_TYPE_VERIFY          4
#define CRYPTO_TYPE_COMPRESS        5
#define CRYPTO_TYPE_UNCOMPRESS      6
#define CRYPTO_TYPE_HASH             7

```

```
#define CRYPTO_TYPE_RANDOM                8

/* Currently defined algorithm IDs (for types
   CRYPTO_TYPE_ENCRYPT & CRYPTO_TYPE_DECRYPT) */
#define CRYPTO_AE_DUMMY                    1      /* no encryption */
#define CRYPTO_AE_DES                      2      /* DES-CBC */
#define CRYPTO_AE_3DES_EDE                 3      /* Triple DES-EDE-CBC */
#define CRYPTO_AE_IDEA                     4      /* IDEA-CBC */
#define CRYPTO_AE_RC2                      5      /* RC2 */
#define CRYPTO_AE_RC4                      6      /* RC4 */
#define CRYPTO_AE_RC5                      7      /* RC5 */
#define CRYPTO_AE_SAFER                    8      /* SAFER */
#define CRYPTO_AE_CAST                     9      /* CAST */
#define CRYPTO_AE_BLOWFISH                 10     /* Blowfish */
#define CRYPTO_AE_RSA                      11     /* RSA */
#define CRYPTO_AE_GOST                     12     /* GOST */

/* Currently defined algorithm IDs (for types
   CRYPTO_TYPE_SIGN & CRYPTO_TYPE_VERIFY) */
#define CRYPTO_AS_RSA                      2      /* RSA */
#define CRYPTO_AS_DSA                      3      /* DSA */
#define CRYPTO_AS_GOST                     4      /* GOST */

/* Currently defined algorithm IDs (for types
   CRYPTO_TYPE_COMPRESS & CRYPTO_TYPE_UNCOMPRESS) */
#define CRYPTO_AC_DUMMY                    1      /* no compression */
#define CRYPTO_AC_DEFLATE                  2      /* Deflate */
#define CRYPTO_AC_LZS                      3      /* LZS */

/* Currently defined algorithm IDs (for type CRYPTO_TYPE_HASH) */
#define CRYPTO_AH_MD5                      1      /* MD5 */
#define CRYPTO_AH_SHA                      2      /* SHA-1 */
#define CRYPTO_AH_GOST                     3      /* GOST */

/* Currently defined algorithm IDs (for type CRYPTO_TYPE_RANDOM) */
#define CRYPTO_AR_UNKNOWN                  1

/* Currently defined plugin flags */
#define CRYPTO_PLUGIN_HARDWARE              1      /* plugin uses hdw */
/* TBD more */

/* Currently defined algorithm flags */
#define CRYPTO_ALG_HARDWARE                1      /* algorithm implemented
                                                    in hardware */
#define CRYPTO_ALG_MULTITHREADED           2      /* implementation allows
                                                    multithreading */
/* TBD more */
```

```

/* Currently defined parameters identifiers for CryptoControl */
#define CRYPTO_PARAM_KEY 1 /* Only for CRYPTO_GET -
                             get current key */

/* TBD more */

typedef struct tag_CryptoAlgInfo {
    long    status; /* Algorithm status */
    long    type; /* algorithm type (One of
                  CRYPTO_TYPE_XXX) */

    long    id; /* algorithm ID */
    long    group; /* algorithm group */
    long    version; /* algorithm version
                    (CRYPTO_VER) */

    long    flags; /* algorithm flags
                  (CRYPTO_ALG_XXX) */

    long    maxcontexts; /* max number of cipher states
                        supported (0 - any) */

    char    name[CRYPTO_ALG_NAME_LEN]; /* algorithm name */
    /* CRYPT SIGN COMPRESS HASH RANDOM */
    /* DECRYPT VERIFY */
    long    blocklen; /* blklen (blklen) inlen blklen - */
    long    keylen; /* keylen keylen - seedlen */
    long    outlen; /* outlen (signlen) outlen hashlen randlen */
    long    milen; /* milen (param) - - - */
} CryptoAlgInfo;

typedef struct tag_CryptoPluginInfo {
    long    cpi_version; /* Crypto PI version (currently
                        CRYPTO_VER(1,0)) */

    long    status; /* Plugin status */
    char    name[CRYPTO_PLUGIN_NAME_LEN]; /* plugin text
                                          description */

    long    version; /* plugin version
                    (CRYPTO_VER) */

    long    flags; /* plugin flags
                  (CRYPTO_PLUGIN_XXX) */

    long    number_of_algs; /* number of AlgInfo structures
                          followed (min 1) */

    CryptoAlgInfo algs[1]; /* array of AlgInfo structures
                          (min 1) */
} CryptoPluginInfo;

#ifdef __cplusplus
extern "C" {
#endif

/* CryptoPlugin initialization. Returns pointer to CryptoPluginInfo
structure on success or NULL on fatal error. */

```

```

CryptoPluginInfo *CryptoPluginInit(
    void                *param); /* Ptr to OS parameters
                                (platform-specific) */

/* Plugin deinitialization */
CRYPTO_STATUS   CryptoPluginFini(void);

/* Get new algorithm instance (cipher state) */
CRYPTO_STATUS   CryptoOpen(
    CRYPTO_HANDLE    *state, /* Pointer to cipher state
                              handle (filled on exit) */
    long             alnum, /* Algorithm number in
                              CryptoPluginInfo structure */
    const char       *key); /* key (in plain) */

/* Reinitialize algorithm instance */
CRYPTO_STATUS   CryptoReOpen(
    CRYPTO_HANDLE    state, /* current cipher state handle */
    const char       *key); /* key (in plain) */

/* Destroy algorithm instance */
CRYPTO_STATUS   CryptoClose(
    CRYPTO_HANDLE    state); /* Handle of cipher state */

/* Check key for possible weakness */
CRYPTO_STATUS   CryptoCheckForWeakKey(
    long             alnum, /* Algorithm number in
                              CryptoPluginInfo structure */
    const char       *key); /* Proposed key */

/* Perform CryptoTransform (depends on cipher state type) */
CRYPTO_STATUS   CryptoTransform(
    CRYPTO_HANDLE    state, /* Cipher state handle */
    const char       *inbuff, /* input data */
    long             inlen, /* input data length */
    char             *outbuff, /* output buffer */
    long             *outlen, /* On entry - output buffer
                              length, on exit - number of
                              bytes written to outbuff */
    char             *mi); /* Message indicator */

/* Algorithm control */
CRYPTO_STATUS   CryptoControl(
    CRYPTO_HANDLE    state, /* Cipher state handle */
    long             cmd, /* Control command */
    long             param, /* Parameter id */
    char             val, /* Parameter value */
    long             *len); /* For CRYPTO_GET: on entry -

```

val buffer length, on exit -  
number of bytes written to  
val; for CRYPTO\_SET: length  
of value to set \*/

```
#ifdef __cplusplus  
{  
#endif
```

```
#endif /* __CRYPTPI_H */
```

## Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

