A High Level Public-Key Based Cryptographic Services API

Introduction

This document specifies a high-level Application Programming Interface (API) in the C language for public-key based cryptographic services. Currently, PKI-enabled applications must use proprietary, vendor-provided APIs to interface with their PKI, thus making support across multiple PKI products difficult. To facilitate the development and wide-deployment of PKI-enabled applications, NIST is working with several federal agencies to make this interface to a PKI consistent, regardless of the PKI product being used. If each PKI product and each application can meet at a common interface, more applications will become PKI enabled for all PKIs. Figure 1 illustrates the components of a PKI-enabled application and the specific interface that this document attempts to address.

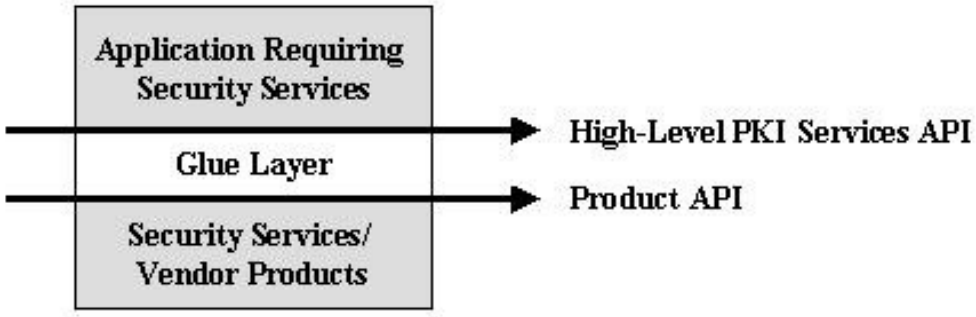

Figure 1. High-Level PKI Services API

The **application requiring security services** is any application that needs digital signature and/or encryption services. The **security services/vendor products** are the existing vendor products that provide the signing and encryption functionality. The **product API** is the vendor-specific interface provided by the product for calling the signing and encryption services. The **high-level PKI services API** is the common API that this document specifies to provide a consistent interface to signing and encryption services irrespective of the product being used. The high-level API is designed to hide the complexity of the underlying security mechanisms but facilitate service requests through simple service calls. The **glue layer** is the code necessary to translate the high-level PKI services API into a product-specific API.

In this specification, the term "PKI" is loosely used to refer to all the components below the high-level PKI Services API. The API is specified in the C language.

Overview

The high-level PKI services API defines five functions for operating on buffers: signBuffer, verifyBuffer, encryptBuffer, decryptBuffer, and CMSBufferParser. The signBuffer function signs the contents of a buffer and returns the signature, which may or may not include a copy of the buffer that was signed. The verifyBuffer function determines whether a message was properly signed, and if so, may return a copy of the buffer that was signed along with singer's identity and the time that the signature was generated. CMSBufferParser extracts the same information from a signed message as verifyBuffer does, but CMSBufferParser does not attempt to verify the signature. The encryptBuffer command encrypts the contents of a buffer in such a

way that only the specified set of recipients can decrypt the message. These recipients may use the decryptBuffer command to extract the original buffer contents.

The high-level PKI services API also defines five functions for operating on files: signFile, verifyFile, encryptFile, decryptFile, and CMSFileParser. These functions are identical to the corresponding buffer operations except that they take their input from files instead of buffers. Depending on the function, the application may be able to specify whether the output is to be placed in a file or in a buffer. Functions that do not provide this option always place the output in a file.

This API was not designed to support the full range of applications and protocols that require cryptographic support. The functions defined in this API are not appropriate for implementing such low-level protocols as SSL/TLS and IPsec. The API was designed to present a simple interface to application developers who need access to the basic cryptographic services of signing and encrypting information. Most of the complexities of PKI are hidden beneath the level of the API and must be addressed by the implementer of the PKI (i.e., the product API or the glue layer).

It is, for example, the PKI's responsibility to maintain user login information, including the user's identity. This information may either be stored in a PKI-specific configuration file or may be requested from the user. Since the user's identity is not provided by the application to the glue layer, it is assumed that each instantiation of the glue layer can only support a single user at a time. The PKI may provide a single login capability that can be shared across multiple applications. If the user is not logged in when a call to a function that requires user authentication is made, the PKI will need to prompt the user to login. The PKI may provide a mechanism for the user to log in and out of the PKI independent from the application. When an application calls one of the functions that may require use of the user's private key, the API provides the application with the capability to specify that user authentication is required. In this case, the PKI must authenticate the user at the time of the function call even if the user was already logged in.

It is the responsibility of the application to allocate memory for all data that is output to buffers by the API functions. In cases in which the maximum length of the output can be predicted, the amount of memory that needs to be allocated by the application is specified in this document. For example, in each function, if the function does not complete successfully, the PKI may output a string that provides information about the nature of the error. This string may be up to 256 bytes long (including the end-of-string character), so the application must allocate at least 256 bytes for this parameter. In cases where the maximum length of the output cannot be predicted, the application must specify the size of the buffer that it is providing and the PKI will specify the actual length of the data. If the application does not provide a sufficiently large buffer, the PKI will return the error code INSUFFICIENT_BUFFER_SIZE and will indicate the amount of buffer space required, thus allowing the application to make a subsequent call in which sufficient memory has been allocated.

The API makes use of a few data types throughout the specification. Parameters that specify Boolean information use the C data type int, with the Boolean value TRUE being represented by the integer 1 and FALSE by the integer 0. Times are encoded using the Distinguished Encoding Rules (DER) [1] for the ASN.1 [2] type GeneralizedTime, with the restriction that fractional-seconds must be omitted and the resulting string must be terminated by an end-of-string character. That is, the time must be represented by the 15-byte string "YYYYMMDDHHMMSSZ", in which the time is specified in Greenwich Mean Time.

Applications must allocate at least 16 bytes for any output parameter that is to hold time information.

Signing and Verifying

The output specified by this API for the signBuffer and signFile functions is based on the **SignedData** type of RFC 2630 (Cryptographic Message Syntax) [3]. The **SignedData** type is DER encoded. The data that was signed may be included within the DER encoded **SignedData**, but this inclusion is optional. If the application chooses not to have the signed data encapsulated within the signature, then it is the application's responsibility to maintain the association between the signature and the signed data so that they may be presented together for signature verification. Both opaque signing (inclusion of the signed data in the signature) and clear signing (exclusion of the signed data from the signature) have merits for certain applications. It is the responsibility of application designers to determine which option to choose for their particular application.

The **SignerInfos** field of the **SignedData** structure generated by signBuffer and signFile must include exactly one **SignerInfo**. This field must include the signing-time attribute, specifying the time at which the signer performed the signing process, as a signed attribute.

The **SignedData** structure can also, optionally, include certificates and CRLs that may be useful in validating the signature. It is recommended that the **SignedData** structure include at least the certificate that contains the public key corresponding to the private key used to generate the signature. The PKI's implementations of verifyBuffer and verifyFile are responsible for obtaining any certificates and CRLs not included in the signature that are needed for path building and validation. A configuration file for the PKI may specify the location of a repository to facilitate searches for needed certificates and CRLs.

In addition to providing the signature and signed data to the verifyBuffer and verifyFile functions, the application must specify the level of assurance it requires from the signature. This assurance level is specified as a non-negative integer, and it is assumed that a higher number indicates a higher level of assurance. An application that is willing to accept any signature in which the certificate corresponding to the public key used for verification passes certification path validation may specify that it requires an assurance level of 0. Assurance levels higher than 0 indicate that more assurance is required. The specific meaning of each assurance level (e.g., 1, 2, 3) is PKI specific.

Typically, assurance levels will correspond to certificate policies. For example, a PKI may specify three certificate policies, high-assurance, medium-assurance, and low-assurance, and assign an object identifier (OID) to each policy. This policy OID will be inserted into any certificate that meets the requirements of that policy. If the certification path validates under the high-assurance policy, then the PKI will assert that the signature meets assurance level 3. If the certification path validates under the medium-assurance policy, but not the high-assurance policy, then signatures only meet assurance level 2. Signatures corresponding to certification paths that only validate under the low-assurance policy meet assurance level 1. Signatures that correspond to certification paths that validate, but not under any of the previously listed policies, are assigned assurance level 0.

When the verifyBuffer or verifyFile function is called, the PKI must determine whether the signature verifies, and if so, at what assurance level. If the assurance level is at least as high as the minimum acceptable assurance level specified by the application, then the verification function may succeed.

The certificate path validation algorithm specified in X.509 [4] accepts several inputs in addition to the list of certificates in the certification path. It is up to the PKI (i.e., the glue layer and/or the product API) to specify the values for these inputs. The values to be used may be dependent on the required level of assurance specified by the application.

Encryption and Decryption

The output specified by this API for the encryptBuffer and encryptFile functions is based on the **EnvelopedData** type of RFC 2630. The type is DER encoded. While RFC 2630 allows the encrypted data to be either included in or omitted from the encoding of **EnvelopedData**, this specification requires that the encrypted data always be included.

A single call to encryptBuffer or encryptFile may result in the provided data being encrypted for one or more recipients. If the data is to be encrypted for more than one recipient, then the output should contain the DER encoding of a single **EnvelopedData** in which the **recipientInfos** field consists of a set of one **RecipientInfo** for each intended recipient. The data must be encrypted in such a way that it can only be decrypted by those recipients explicitly listed by the application. If the application wishes for the data to be encrypted in such a way that the sender can decrypt the data, the application must explicitly include the sender in the list of recipients. It is the responsibility of the PKI (i.e., the glue layer and/or the product API) to obtain the necessary keying material (e.g., certificates) for the recipients based on the information provided by the application.

Implementation Guidance

Glue Layer Implementers

In order to simplify application development, much of the complexity involved in working with a PKI is hidden underneath the level of the high-level PKI services API. It is the responsibility of the glue layer (or the product API on which it is to be built) to locate certificates and CRLs, build and validate certification paths, understand the meaning of assurance levels of signatures and determine whether a given signature meets a given assurance level, maintain user identity and authenticate users when necessary, and determine how to encrypt messages to a group of recipients based on those recipients' identities.

In order to accomplish these tasks, the glue layer will need access to certain information that will not be provided by the application. This information may be obtained either by obtaining it from the user or by reading it in from a configuration file. Whenever possible, it is considered preferable for the glue layer to obtain this information from configuration files. Information in a configuration file can be generated once by a knowledgeable system administrator, thus saving the application user from needing to know or understand this information. User authentication information and other information that needs to be kept secret for security reasons should be obtained directly from the user rather than storing it in a potentially insecure configuration file.

Examples of information that may be placed in a configuration file are:

- the location of a repository that contains certificates and CRLs;
- the default signature, encryption, and key management algorithms to use;
- the mappings between assurance levels and certificate policy OIDs;
- the input values to be used for path validation (e.g., whether policy mapping is inhibited); and

- the location of a configuration file required by the implementation of the product API.

In addition to obtaining information from a configuration file, the glue layer may request information directly from the user. For example, if the signBuffer function is called, and either the user is not currently logged in or caller specifies that authentication must be performed, the glue layer may present the user with an interactive dialog box requesting that user provide authentication information. Similarly, when the encryptBuffer command is called, the glue layer may request input from the user as necessary to locate the key management certificates of all of the intended recipients.

The high-level PKI services API defines a fixed set of error codes that may be returned from any of the functions defined in the API. Appendix A specifies the complete list of error codes and each function specifies the set of error codes that may be returned by that function. A function that completes successfully must return the code NO_ERR, whereas a function that does not complete successfully must return one of the other error codes. If a function completes with a warning from the underneath PKI services, a glue layer implementation can return ERR_WARNING if it is deemed desirable for the glue layer to reflect this warning to the calling application. In this case, all the output parameters of this function should be populated with data as if the function had returned successfully. If a function fails as a result of a failed call to a function in the underlying product API, the vendor-specific error code should be mapped to one of the error codes defined in this API. If none of the specified error codes is appropriate for the error that resulted in the failure, the generic ERR_OPERATION_FAILED error code may be used. In any case, more specific information about the error may be provided in the output parameter *error_data*. As the contents of the string in *error_data* does not need to be machine readable, it may simply be copied from an error string provided by the underlying product API. However, the glue layer must ensure that the string placed in *error_data* is at most 256 bytes long, including the end-of-string character.

For output parameters in which the length of the output is variable in length, the glue layer must ensure that no more data is written to the output parameter than the amount of space that the application indicated was allocated for the buffer. If the application indicates that the no space was allocated for the output buffer, then it is possible that the output parameter will have a value of NULL. If the output parameter was not allocated enough space to hold the output value, the glue layer must indicate to the application the amount of space that would be required to hold the output value in addition to returning an error code of INSUFFICIENT_BUFFER_SIZE. Other than *error_data* and the output length parameter in cases where a function fails due to ERR_INSUFFICIENT_BUFFER_SIZE, the output parameters from a failed operation shall not be populated with data and their contents should be ignored by the calling application.

Application Implementers

The high-level PKI services API was designed to hide most of the implementation complexities from the application's developer. There are, however, a few issues that are left as a responsibility for the application's programmer. For example, the application is responsible for allocating memory for all output parameters. In some cases, the maximum length of an output parameter is set by this specification. For example, the application needs to allocate at least 256 bytes for the *error_data* parameter in each function call and 16 bytes for the *time_data_signed* parameter. In cases where it was not possible to determine a maximum output length, this API requires that the application indicate in the function call the amount of space that was allocated for a given output parameter. On return, the function will indicate whether a sufficient amount of memory was allocated, and if not, how much memory is needed.

If an application is unable to guess the amount of space that needs to be allocated for an output parameter, the application may adopt a strategy of calling the function twice. In the first call to the function, no memory is allocated. Then the amount of memory that is specified as required can be allocated and then the function can be called a second time with that amount of memory. This will simplify the implementation of the application. However, depending on the implementation of the glue layer this may be expensive. The cost, in time, of calling a function to determine the amount of space required for the output parameter may be the same as the amount of time required to perform the requested operation. Thus, this strategy may double the amount of time required to perform an operation, even in cases where the operation is relatively time consuming.

High Level PKI Services API

```
int signBuffer(
    IN          uint32          data_length,
    IN          char*           data_to_sign,
    IN          int             authent_required,
    IN          int             encap_data_flag,
    IN/OUT      uint32*         signed_data_length,
    IN/OUT      uchar*          signed_data,
    OUT         char*           error_data
);
```

This function will cause the underlying components to

- Reauthenticate the user to the PKI if the *authent_required* flag is set to TRUE.
- Locate the signer's key and generate the digital signature over the data to be signed.
- DER-encode the generated signature and other relevant information (such as signerInfo) in the SignedData type of the Cryptographic Message Syntax (CMS) defined in the RFC2630.
- Return error code or success to the application.

Parameters

data_length: the length of *data_to_sign*.
data_to_sign: the buffer for the data to be signed.
authent_required: Specifies TRUE or FALSE to indicate whether a user should be reauthenticated to the PKI before a digital signature can be generated.
encap_data_flag: Specifies TRUE or FALSE to indicate whether the signed content should be included in EncapsulatedContentInfo.
signed_data_length: A pointer to the size of the *signed_data* buffer. As input to the API, it points to the memory size allocated for *signed_data*. As output, it points to the actual length of *signed_data*.
signed_data: the buffer to hold the signature octet string that corresponds to the SignedData type of RFC2630.
error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
Function completed successfully.

ERR_WARNING
Function completed with a warning.

ERR_ INSUFFICIENT_BUFFER_SIZE
The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_CERT_NOT_FOUND

The signer's signing certificate or key could not be found. The output parameter *error_data* may contain the Distinguished Name (DN) of the certificate owner.

ERR_CERT_EXPIRED

The signer's certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID

The signer's certificate is not valid for reasons not covered by any other code in Appendix A. This can be returned when an implementation actually checks the validity of the signer's certificate before a signature is generated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND

A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL

The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE

The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED

The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION

The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE

The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED

The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED

The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER

The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED

The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int signFile(

IN	**char***	infile,
IN	**int**	authent_required,
IN	**int**	encap_data_flag,
IN	**int**	output_to_file,
IN	**char***	outfile,
IN/OUT	**uint32***	signed_data_length,
IN/OUT	**uchar***	signed_data,
OUT	**char***	error_data

);

This function will cause the underlying components to

- Reauthenticate the user to the PKI if the *authent_required* flag is set to TRUE.
- Open and read the file to be signed; return error if it cannot be opened or read.
- Locate the signer's key and generate a digital signature over the file to be signed.
- DER-encode the signature and other relevant information (such as signerInfo) in the SignedData type. Write the signature output to the *signed_data* buffer or to the *outfile* file, depending on the choice specified in *output_to_file*.
- Return error code or success to the application.

Parameters

infile: the name of the file to be signed.

authent_required: Specifies TRUE or FALSE to indicate whether a user should be reauthenticated to the PKI before a digital signature can be generated.

encap_data_flag: Specifies TRUE or FALSE to indicate whether the file to be signed should be included in EncapsulatedContentInfo of SignedData. The default is FALSE. It is generally undesirable to include the signed file, unless small in size, in EncapsulatedContentInfo.

output_to_file: Specifies TRUE or FALSE to indicate whether the signature output should be sent to a file or a buffer. If the output is sent to a buffer, *signed_data_length* and *signed_data* shall be used; and *verifyBuffer* rather than *verifyFile* should be called to verify the signature contained in the buffer. If the signature output is to be written to *outfile*, then *verifyFile* should be called later on for signature verification.

outfile: the name of the file to receive the signature output.

signed_data_length: A pointer to the size of the *signed_data* buffer. As input to the API, it points to the memory size allocated for *signed_data*. As output, it points to the actual length of *signed_data*. *Signed_data_length* and *signed_data* shall be used only when *output_to_file* is FALSE.

signed_data: the buffer to hold the signature octet string that corresponds to the SignedData type of RFC2630. *Signed_data_length* and *signed_data* shall be used only when *output_to_file* is FALSE.

error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR

Function completed successfully.

ERR_WARNING

Function completed with a warning.

ERR_FILE_OPERATION
: An error occurred in a file operation. The output parameter *error_data* may contain the name of the file in error.

ERR_ INSUFFICIENT_BUFFER_SIZE
: The allocated memory is not sufficient for an output parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
: An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_CERT_NOT_FOUND
: The signer's signing certificate or key could not be found. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXPIRED
: The signer's certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID
: The signer's certificate is not valid for reasons not covered by any other code in Appendix A. This can be returned when an implementation actually checks the validity of the signer's certificate before a signature is generated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND
: A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL
: The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE
: The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED
: The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION
: The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE
: The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED
: The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED
: The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER
: The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED
: The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int verifyBuffer(

IN	**uint32**	signed_data_length,
IN	**uchar***	signed_data,
IN	**ushort**	policy,
OUT	**char***	signer,
OUT	**char***	time_data_signed,
IN/OUT	**uint32***	data_verified_length**,**
IN/OUT	**char***	data_verified,
OUT	**char***	error_data

);

This function is used to verify the signature contained in the *signed_data* buffer. Upon a successful verification, the signer identity, timestamp, and the original signed content are returned.

Parameters

signed_data_length: the length of *signed_data*.
signed_data: the buffer that holds the signature octet string that corresponds to the SignedData type of RFC2630.
policy: the required policy or assurance level under which the verification certificate must be issued. Applications that do not require policy checking can set this to zero as input.
signer: the buffer to receive the signer identity.
time_data_signed: the buffer to receive the signing date and time in the GeneralizedTime format.
data_verified_length: Specifies the length of *data_verified,* which is also the data signed. Depending on whether the signed content was included in EncapsulatedContentInfo, this parameter has different settings as input to the API. If the signed content was included in EncapsulatedContentInfo, this parameter, as input to the API, points to the memory size allocated for *data_verified*. As output, it points to the actual length of *data_verified*. However, if the original signed content was not included in EncapsulatedContentInfo, then it should be specified through *data_verified* and *data_verified_length* for signature verification. In this case, as input and output to the API, this parameter always points to the length of the signed content held in the *data_verified* buffer.
data_verified: the buffer to hold or holding the signed content depending on whether the content was included in EncapsulatedContentInfo. If the signed content was included in EncapsulatedContentInfo, this parameter, as input to the API, specifies an empty buffer. As output, it holds the signed content whose signature is just verified. On the other hand, if the signed content was not included in EncapsulatedContentInfo, this parameter, as input and output to the API, buffers the data that was signed.
error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
Function completed successfully.

ERR_WARNING
Function completed with a warning.

ERR_ASN1_PARSE_FAILURE
The ASN.1 object specified by *signed_data* cannot be decoded.

ERR_ INSUFFICIENT_BUFFER_SIZE
: The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
: An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_UNSUPPORTED_ALGORITHM
: The requested algorithm is not supported. The output parameter *error_data* may contain the algorithm identifier.

ERR_CERT_NOT_FOUND
: The signer's certificate could not be found. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXPIRED
: The signer's certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_PATH_ERROR
: A complete certification path could not be built or validated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXT_UNKNOWN_CRITICAL
: This certificate contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_ASSURANCE_LEVEL_NOT_MET
: The signer's certificate policy does not meet the required assurance level. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID
: The signer's certificate is not valid for reasons not covered by any other code in Appendix A. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND
: A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL
: The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE
: The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED

The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION
: The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE
: The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED
: The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED
: The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER
: The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED
: The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int verifyFile(

IN/OUT	**char***	file_signed,
IN	**char***	signature_file,
IN	**ushort**	policy,
OUT	**char***	signer,
OUT	**char***	time_data_signed,
OUT	**char***	error_data

);

This function is used to verify a digital signature contained in the *signature_file*. Upon a successful verification, the signer identity, timestamp, and the original signed content are returned.

Parameters

file_signed: names the file to receive or holding the signed data depending on whether the original signed content was included in EncapsulatedContentInfo. If the signed file content was included in EncapsulatedContentInfo, this parameter should specify a file name to receive the content signed upon a successful verification. If the file content was not included in EncapsulatedContentInfo, this parameter, as input and output to the API, names the file that was signed.
signature_file: names the file that contains the signature octet string.
policy: the required policy or assurance level under which the verification certificate must be issued. Applications that do not require policy checking can set this to zero as input.
signer: the buffer to receive the signer identity.
time_data_signed: the buffer to receive the signing date and time in the GeneralizedTime format.
error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
Function completed successfully.

ERR_WARNING
Function completed with a warning.

ERR_ASN1_PARSE_FAILURE
The ASN.1 object contained in the signature file cannot be decoded.

ERR_FILE_OPERATION
An error occurred in a file operation. The output parameter *error_data* may contain the name of the file in error.

ERR_ INSUFFICIENT_BUFFER_SIZE
The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_UNSUPPORTED_ALGORITHM
: The requested algorithm is not supported. The output parameter *error_data* may contain the algorithm identifier.

ERR_CERT_NOT_FOUND
: The signer's certificate could not be found. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXPIRED
: The signer's certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_PATH_ERROR
: A complete certification path could not be built or validated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXT_UNKNOWN_CRITICAL
: This certificate contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_ASSURANCE_LEVEL_NOT_MET
: The signer's certificate policy does not meet the required assurance level. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID
: The signer's certificate is not valid for reasons not covered by any other code in Appendix A. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND
: A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL
: The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE
: The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED
: The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION
: The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE

The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED
: The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED
: The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER
: The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED
: The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int encryptBuffer(

IN	**char****	recipientlist,
IN	**uint32**	data_length,
IN	**char***	data_to_encrypt,
IN	**ushort**	encryption_algorithm,
IN	**int**	authent_required,
IN/OUT	**uint32***	enveloped_data_length,
IN/OUT	**uchar***	enveloped_data,
OUT	**char***	error_data

);

This function will cause the underlying components to

- Locate each recipient's encryption certificate or key agreement public key certificate depending on the choice of key management techniques. Check the validity of the recipient's certificate. Locate the sender's private key if key agreement is selected.
- Generate a random session or one-time symmetric key. If key transport is the choice for deriving the same encrypting key between the sender and the recipient, then the session or one-time key should be encrypted under the recipient's public encryption key. If key agreement is the choice, then the sender's private key and the recipient's key agreement public key are used to generate a pair wise symmetric key, which is then used to encrypt the session or one-time key. If the key management choice is neither key transport nor key agreement, but rather to use a previously distributed symmetric key encryption key, then the session or one-time key should be encrypted under this key encryption key. In order for this key management scheme to work, the sender must have a previously distributed key encryption key with each recipient that the sender may communicate with. Since this key management scheme does not scale well to a large user community, we recommend that only key transport or key agreement be used.
- Encrypt the data buffer under the session or one-time key using the encryption algorithm specified in *encrytion_algorithm.*
- DER-encode the recipient-specific information and cipher text in the envelopedData type of RFC2630. Note that the application will send the same DER-encoded envelopedData to all the recipients. It is the responsibility of each recipient's PKI to decode the envelopedData and parse the recipient-specific information in order to derive the encryption key that is to decrypt the cipher text.

Parameters

recipientlist: the list of recipients for the encrypted information.
data_length: the length of the *data_to_encrypt* buffer.
data_to_encrypt: the data buffer to be encrypted.
encryption_algorithm: the encryption algorithm to be used.
authent_required: Relevant only if key agreement is used during the encryption process, this flag specifies TRUE or FALSE to indicate whether the user needs to be reauthenticated to the PKI before the private key can be used to perform key agreement. If key agreement is not used, this flag should be set to FALSE.
enveloped_data_length: A pointer to the size of the *enveloped_data* buffer. As input to the API, it points to the memory size allocated for *enveloped_data*. As output, it points to the actual size of *enveloped_data*.
enveloped_data: the buffer to hold the encrypted octet string that corresponds to the EnvelopedData type of RFC2630.

error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
: Function completed successfully.

ERR_WARNING
: Function completed with a warning.

ERR_ INSUFFICIENT_BUFFER_SIZE
: The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
: An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_UNSUPPORTED_ALGORITHM
: The requested algorithm is not supported. The output parameter *error_data* may contain the algorithm identifier.

ERR_CERT_NOT_FOUND
: A recipient's encryption certificate could not be found. The output parameter *error_data* may contain the DN of the recipient.

ERR_CERT_EXPIRED
: A recipient's encryption certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_PATH_ERROR
: A complete certification path could not be built or validated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXT_UNKNOWN_CRITICAL
: This certificate contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID
: A recipient's certificate is not valid for reasons not covered by any other code in Appendix A. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND
: A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL
: The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE

The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED

The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION

The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE

The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED

The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED

The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER

The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED

The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int encryptFile(

IN	**char****	recipientlist,
IN	**char***	file_to_encrypt,
IN	**ushort**	encryption_algorithm,
IN	**int**	authent_required,
IN	**int**	output_to_file,
IN	**char***	encrypted_file,
IN/OUT	**uint32***	enveloped_data_length,
IN/OUT	**uchar***	enveloped_data,
OUT	**char***	error_data

);

This function will cause the underlying components to

- Open the specific files for reading/writing. Return error code if it encounters a problem.
- Locate each recipient's encryption certificate or key agreement public key certificate depending on the choice of key management techniques. Check the validity of the recipient's certificate. Locate the sender's private key if key agreement is selected; the sender may need to be reauthenticated if *authent_required* is set to TRUE.
- Generate a random session or one-time symmetric key. If key transport is the choice for deriving the same encrypting key between the sender and the recipient, then the session or one-time key should be encrypted under the recipient's public encryption key. If key agreement is the choice, then the sender's private key and the recipient's key agreement public key are used to generate a pair wise symmetric key, which is then used to encrypt the session or one-time key. If the key management choice is neither key transport nor key agreement, but rather to use a previously distributed symmetric key encryption key, then the session or one-time key should be encrypted under this key encryption key. In order for this key management scheme to work, the sender must have a previously distributed key encryption key with each recipient that the sender may communicate with. Since this key management scheme does not scale well to a large user community, we recommend that only key transport or key agreement be used.
- Read the file content into a data buffer; encrypt the buffer under the session or one-time key using the specified encryption algorithm.
- DER-encode the recipient-specific information and cipher text in the envelopedData type of RFC2630. Depending on the choice specified in *output_to_file*, the encrypted output should be written to a buffer, *enveloped_data,* or to a file named by *encrypted_file*. Note that the application will send the same encrypted file or buffer to all the recipients. It is the responsibility of each recipient's PKI to read the file or buffer, decode the envelopedData, and parse the recipient-specific information in order to derive the encryption key that is to decrypt the cipher text.

Parameters

recipientlist: the list of recipients for the encrypted information.
file_to_encrypt: the name of the file to be encrypted.
encryption_algorithm: the encryption algorithm to be used.
authent_required: Relevant only if key agreement is used during the encryption process, this flag specifies TRUE or FALSE to indicate whether the user needs to be reauthenticated to the PKI before the private key can be used to perform key agreement. If key agreement is not used, this flag should be set to FALSE.

output_to_file: Specifies TRUE or FALSE to indicate whether the encrypted output should be sent to a file or a buffer. If the output is sent to a buffer, *enveloped_data_length* and *enveloped_data* shall be used; and *decryptBuffer* rather than *decryptFile* should be called to decrypt the encrypted buffer. If the output is sent to the *encrypted_file* file, then *decryptFile* should be called to decrypt the encrypted file.

encrypted_file: names the file to receive the encrypted output.

enveloped_data_length: A pointer to the size of the *enveloped_data* buffer. As input to the API, it points to the memory size allocated for *enveloped_data*. As output, it points to the actual length of *enveloped_data*. This parameter and *enveloped_data* shall be used only when *output_to_file* is set to FALSE.

enveloped_data: the buffer to hold the encrypted octet string that corresponds to the EnvelopedData type of RFC2630. This parameter and *enveloped_data_length* shall be used only when *output_to_file* is set to FALSE.

error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
Function completed successfully.

ERR_WARNING
Function completed with a warning.

ERR_FILE_OPERATION
An error occurred in a file operation. The output parameter *error_data* may contain the name of the file in error.

ERR_ INSUFFICIENT_BUFFER_SIZE
The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_UNSUPPORTED_ALGORITHM
The requested algorithm is not supported. The output parameter *error_data* may contain the algorithm identifier.

ERR_CERT_NOT_FOUND
A recipient's encryption certificate could not be found. The output parameter *error_data* may contain the DN of the recipient.

ERR_CERT_EXPIRED
A recipient's encryption certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_PATH_ERROR
A complete certification path could not be built or validated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXT_UNKNOWN_CRITICAL
: A recipient's encryption certificate contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID
: A recipient's certificate is not valid for reasons not covered by any other code in Appendix A. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND
: A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL
: The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE
: The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED
: The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION
: The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE
: The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED
: The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED
: The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER
: The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED
: The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int decryptBuffer(

IN	**uint32**	enveloped_data_length,
IN	**uchar***	enveloped_data,
IN	**int**	authent_required,
IN/OUT	**uint32***	plain_text_length,
IN/OUT	**char***	plain_text,
OUT	**ushort***	encryption_algorithm,
OUT	**char***	error_data

);

This function is used to decrypt an encrypted *enveloped_data* buffer. Upon successful decryption, the plain text and the encryption algorithm used are returned.

Parameters

enveloped_data_length: the length of the *enveloped_data* buffer.
enveloped_data: the buffer that holds the encrypted octet string that corresponds to the EnvelopedData type of RFC2630.
authent_required: Relevant only if key transport or key agreement is used during the encryption process, this flag specifies TRUE or FALSE to indicate whether the user needs to be reauthenticated to the PKI before his private key can be used to perform key transport or key agreement. If key transport or key agreement is not used, this flag should be set to FALSE.
plain_text_length: a pointer to the size of the *plain_text* buffer. As input to the API, it points to the memory size allocated for *plain_text*. As output, it points to the actual length of *plain_text*.
plain_text: the buffer to receive the decrypted text.
encryption_algorithm: a pointer to the encryption algorithm used.
error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
Function completed successfully.

ERR_WARNING
Function completed with a warning.

ERR_ASN1_PARSE_FAILURE
The ASN.1 object specified by *enveloped_data* cannot be decoded.

ERR_ INSUFFICIENT_BUFFER_SIZE
The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_UNSUPPORTED_ALGORITHM

The requested algorithm is not supported. The output parameter *error_data* may contain the algorithm identifier.

ERR_CERT_NOT_FOUND
: The decryptor's encryption certificate or key could not be found. The output parameter *error_data* may contain the DN of the key owner.

ERR_CERT_EXPIRED
: The decryptor's encryption certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_PATH_ERROR
: A complete certification path could not be built or validated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXT_UNKNOWN_CRITICAL
: The decryptor's encryption certificate contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID
: The decryptor's encryption certificate is not valid for reasons not covered by any other code in Appendix A. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND
: A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL
: The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE
: The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED
: The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION
: The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE
: The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED
: The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED

The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER

The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED

The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int decryptFile(

IN	**char***	encrypted_file,
IN	**int**	authent_required,
IN	**char***	plain_text_file,
OUT	**ushort***	encryption_algorithm,
OUT	**char***	error_data

);

This function is used to decrypt an encrypted file. Upon successful decryption, the plain text and the encryption algorithm used are returned.

Parameters

encrypted_file: the name of the file that contains the encrypted octet string.
authent_required: Relevant only if key transport or key agreement is used during the encryption process, this flag specifies TRUE or FALSE to indicate whether the user needs to be reauthenticated to the PKI before his private key can be used to perform key transport or key agreement. If key transport or key agreement is not used, this flag should be set to FALSE.
plain_text_file: the name of the file that is to receive the decrypted text.
encryption _algorithm: a pointer to the encryption algorithm used.
error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
Function completed successfully.

ERR_WARNING
Function completed with a warning.

ERR_ASN1_PARSE_FAILURE
The ASN.1 object contained in the *encrypted_file* cannot be decoded.

ERR_FILE_OPERATION
An error occurred in a file operation. The output parameter *error_data* may contain the name of the file in error.

ERR_ INSUFFICIENT_BUFFER_SIZE
The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_UNSUPPORTED_ALGORITHM
The requested algorithm is not supported. The output parameter *error_data* may contain the algorithm identifier.

ERR_CERT_NOT_FOUND

The decryptor's encryption certificate or key could not be found. The output parameter *error_data* may contain the DN of the key owner.

ERR_CERT_EXPIRED
: The decryptor's encryption certificate has expired. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_PATH_ERROR
: A complete certification path could not be built or validated. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_EXT_UNKNOWN_CRITICAL
: The decryptor's encryption certificate contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CERT_INVALID
: The decryptor's encryption certificate is not valid for reasons not covered by any other code in Appendix A. The output parameter *error_data* may contain the DN of the certificate owner.

ERR_CRL_NOT_FOUND
: A CRL from this issuer could not be found. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_EXT_UNKNOWN_CRITICAL
: The CRL from this issuer contains an unrecognized extension that is marked critical. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_CRL_SIGNATURE_FAILURE
: The signature on the CRL failed to verify. The output parameter *error_data* may contain the DN of the CRL issuer.

ERR_RR_AFFILIATION_CHANGED
: The certificate has been revoked due to affiliation changes. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_CESSATION_OF_OPERATION
: The certificate has been revoked due to a cessation of operation. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_KEY_COMPROMISE
: The certificate has been revoked because the key has been compromised. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_SUPERSEDED
: The certificate has been superseded. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_UNSPECIFIED
: The certificate has been revoked for unspecified reasons. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_RR_OTHER

The certificate has been revoked for other reason not listed above. The output parameter *error_data* may contain the DN of the revoked certificate.

ERR_OPERATION_FAILED

The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int CMSBufferParser (

IN	**uint32**	signed_data_length,
IN	**uchar***	signed_data,
OUT	**char***	signer,
OUT	**char***	time_data_signed,
IN/OUT	**uint32***	content_length,
IN/OUT	**char***	content_signed,
OUT	**char***	error_data

);

*** Note this is a non-cryptographic function call.** It allows an application that is unaware of the complex CMS structure to be able to obtain information about the signer without having to verify the signature. The idea came from the S/MIME (Secure/Multipurpose Internet Mail Extensions) client implementation where one may receive a signed message but does not feel the need to verify the signature, and yet wants to know what was signed and who signed it. Note that if the content that was signed had not been included in the EncapsulatedContentInfo of SignedData, then the output *content_length* should return zero and *content_signed* shall be an empty string.

Parameters

signed_data_length: the length of the *signed_data* buffer.
signed_data: the buffer that holds the signature octet string that corresponds to the SignedData type of RFC2630.
signer: the buffer to receive the signer identity.
time_data_signed: the buffer to receive the signing date and time in the GeneralizedTime format.
content_length: a pointer to the size of *content_signed*. As input to the API, it points to the memory size allocated for *content_signed*. As output, it points to the actual length of *content_signed*.
content_signed: the buffer to receive the original signed content. If that content was not included in EncapsulatedContentInfo, then *content_signed* should be an empty string.
error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
Function completed successfully.

ERR_WARNING
Function completed with a warning.

ERR_ASN1_PARSE_FAILURE
The ASN.1 object specified by *signed_data* cannot be decoded.

ERR_ INSUFFICIENT_BUFFER_SIZE
The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_OPERATION_FAILED

The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

int CMSFileParser (

IN	**char***	signature_file,
IN/OUT	**char***	file_signed,
OUT	**char***	signer,
OUT	**char***	time_data_signed,
OUT	**char***	error_data

);

- **Note this is a non-cryptographic function call.** It allows an application that is unaware of the complex CMS structure to be able to obtain information about the signer without having to verify the signature. The idea came from the S/MIME client implementation where one may receive a signed message but does not feel the need to verify the signature, and yet wants to know what was signed and who signed it. In the case of a signed file, the file content by default will not be included in the EncapsulatedContentInfo, therefore, the signed content will not be written to *file_signed*. However, if the signed content was included in EncapsulatedContentInfo, the signed content will be written to the *file_signed* file.

Parameters

signature_file: the name of the signature file.
file_signed: the name of the file to receive the original signed content.
signer: the buffer to receive the signer identity.
time_data_signed: the buffer to receive the signing date and time in the GeneralizedTime format.
error_data: the buffer to hold pertinent information about the error condition.

Return values

NO_ERR
: Function completed successfully.

ERR_WARNING
: Function completed with a warning.

ERR_ASN1_PARSE_FAILURE
: The ASN.1 object contained in the signature file cannot be decoded.

ERR_FILE_OPERATION
: An error occurred in a file operation. The output parameter *error_data* may contain the name of the file in error.

ERR_ INSUFFICIENT_BUFFER_SIZE
: The allocated memory is not sufficient for a parameter. The output parameter *error_data* may contain the name of the parameter in error.

ERR_ INVALID_PARAMETER
: An invalid parameter was passed to this function. The output parameter *error_data* may contain the name of the parameter in error.

ERR_OPERATION_FAILED
: The requested service failed or resource was unavailable. The output parameter *error_data* should contain a specific error description.

References

[1] ITU-T Recommendation X.690: Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). December 1997.

[2] ITU-T Recommendation X.680: Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation. December 1997.

[3] Housley, R. Cryptographic Message Syntax. RFC2630. June 1999.

[4] ITU-T Recommendation X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. March 2000.

APPENDIX A – RETURN CODES

Value	Error Code	Meaning	If present, data is
0	NO_ERR	Function completed successfully	
100	ERR_WARNING	Function completed with a warning, but not a fatal error	Warning description
101	ERR_ASN1_PARSE_FAILURE	Could not parse an ASN.1 object	
102	ERR_FILE_OPERATION	Error occurred in a file operation	Filename
103	ERR_INSUFFICIENT_BUFFER_SIZE	Allocated memory is not sufficient for a parameter	Parameter in error
104	ERR_INVALID_PARAMETER	An invalid parameter was passed to this function	Parameter in error
105	ERR_UNSUPPORTED_ALGORITHM	The requested algorithm is not supported	Algorithm ID
106	ERR_CERT_NOT_FOUND	The required certificate or key could not be found	DN of key owner
107	ERR_CERT_EXPIRED	The certificate has expired	DN of certificate
108	ERR_CERT_PATH_ERROR	No valid path can be found to validate a certificate	DN of certificate
109	ERR_CERT_EXT_UNKNOWN_CRITICAL	The certificate contains an unknown extension marked critical	DN of certificate
110	ERR_CERT_ASSURANCE_LEVEL_NOT_MET	Certificate policy does not meet the required assurance level	DN of certificate
111	ERR_CERT_INVALID	The certificate is invalid for unspecified reasons	DN of certificate
112	ERR_CRL_NOT_FOUND	The required CRL could not be found	DN of CRL issuer
113	ERR_CRL_EXT_UNKNOWN_CRITICAL	The CRL contains an unknown extension marked critical	DN of CRL issuer
114	ERR_CRL_SIGNATURE_FAILURE	The signature on the CRL could not be verified	DN of CRL issuer
115	ERR_RR_AFFILIATION_CHANGED	The certificate has been revoked due to affiliation changes	DN of revoked certificate
116	ERR_RR_CESSATION_OF_OPERATION	The certificate has been revoked due to a cessation of operation	DN of revoked certificate
117	ERR_RR_KEY_COMPROMISE	The certificate has been revoked due to key compromise	DN of revoked certificate
118	ERR_RR_SUPERSEDED	The certificate has been revoked for being superseded	DN of revoked certificate
119	ERR_RR_UNSPECIFIED	The certificate has been revoked for an unspecified reason	DN of revoked certificate
120	ERR_RR_OTHER	The certificate has been revoked for other reason not listed above	DN of revoked certificate
121	ERR_OPERATION_FAILED	General failure, requested service failed or resource unavailable	Error description received from underneath PKI service calls