Technical
Documentation

Secure Store & Forward (SSF)

API Specifications

Version 1.0

Dr. J. Schneider
Michael Friedrich

Version 1.0
11.06.99

SAP AG
Neurottstraße 16
D-69190 Walldorf
Germany

Copyright

Contents

Figures

Tables

Preface

This document contains the high-level specifications of the SSF interface for "Secure Store & Forward" mechanisms with SAP R/3 (SSF API). This mainly includes the use of **digital signatures** and **encryption** at the R/3 application level.

After introducing the goals of SSF and describing the SSF architecture, the individual SSF functions will be described in a language-neutral way. The high-level specifications are mapped to ABAP function modules, and to "C" functions as low-level specifications (separate documents).

Disclaimer

This is the final version of the SSF API specifications, version 1.0.

1. SSF Goals and Requirements

The support provided by "Secure Store & Forward (SSF)" enables the protection of R/3 data and documents when saved on data carriers and when transmitted via insecure communication paths. To facilitate this, digital signatures and encryption are utilized at the application level. In the process, the data is saved regardless of the type of its contents, and regardless of the selected transport procedure. The creation, sending, and receipt of the data can occur asynchronously.

R/3 Security

Various components currently contribute to the security of the SAP R/3 System. *Authentication* of users of the R/3 System is carried out by the logon procedure (**R/3 logon**). The *authorization* of R/3 users to carry out specific SAP transactions and access R/3 datasets occurs via the **R/3 authorization concept** and the associated access controls ("authority check").

With SNC (**"Secure Network Communications"**) support, security is increased at the level of network communications. By authenticating end systems and system processes in a distributed R/3 System, secure network connections are guaranteed. Applying hash procedures and encrypting data streams sent over network connections - such as between the SAPgui and the application server - guarantees the *integrity* and *confidentiality* of the data during an online session. Furthermore, securing RFC ("Remote Function Call") and SAProuter with SNC prevents unauthorized external access to R/3 Systems. At the network level, these security mechanisms are transparent to the R/3 applications. In addition, card readers and smartcards can be used to *authenticate* R/3 users with SNC.

SSF Functions

With the SSF (**"Secure Store & Forward"**) functions, security is achieved at the R/3 application level. Independent data units can be secured even outside the context of an existing communication link. For example, this includes documents and datasets saved in the R/3 System (such as financial data, planning data, personnel data, etc.), even outside of running transactions. Moreover, the use of SSF is particularly advantageous when data leaves the R/3 System for a time, in order to exchange it between people and institutions - as is often the case for processing electronic orders, purchase orders, deliveries, or payments.

With the SSF functions, R/3 data and documents are "wrapped" in secure formats - the so-called "security wrapper" - before they are saved on data carriers or transmitted via insecure communication links. A **digital signature** ensures that the data is not falsified, that the sender (signatory) can be clearly determined, and that proof of award of contract exists (see Figure 1). The subsequently assigned **digital envelope** ensures that the contents of the data are only visible to the intended recipient(s) (see Figure 2). As a result, no security gaps arise, even if the data is temporarily stored during transport or at its destination. These two principle mechanisms are using **public-key technology**.

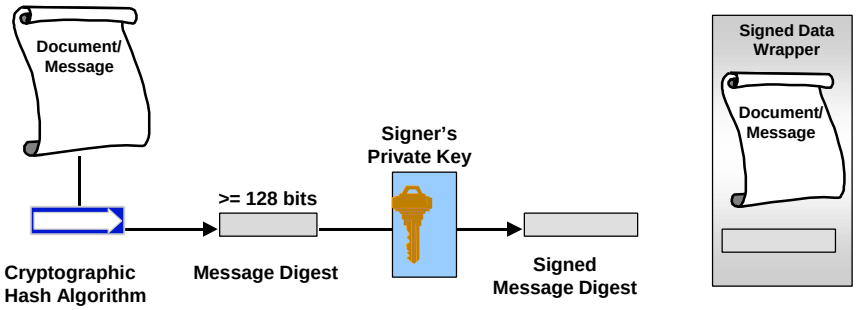

Figure 1. Digital Signature

Protecting R/3 data and documents with SSF fulfills the following basic security requirements:

- *Integrity* of data (protection from falsification)
- *Confidentiality* of data (protection from unauthorized viewing)
- *Authenticity* of the sender (protection from counterfeiters)
- Proof of award of contract (*Non-repudiation*)

SSF Properties

In addition, the following SSF properties are also extremely relevant for electronic transactions:

- SSF is asynchronous - i.e. creation, transmission, receipt, processing, and confirmation of business transactions can occur as several independent steps
- Independence from the transport method - various transport media and procedures are possible (such as a public network, provider, online service, online service, Internet, tapes, disks), as well as various communication services and protocols (such as HTTP, FTP, e-mail, EDI).

These properties are retained even after the data transmission is complete, as long as the data is saved in the secure format.

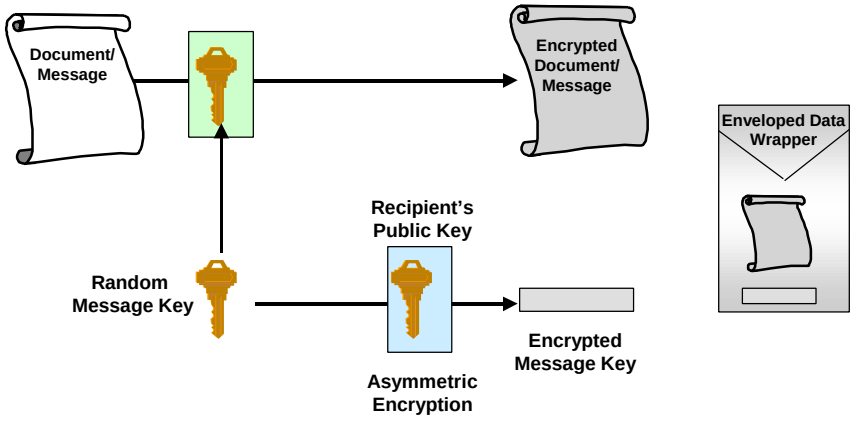

Figure 2. Digital Envelope

2. SSF Architecture and Application Scenarios

2.1 “Pre-Phase”

Before the actual use of the SSF functions, the infrastructure required to use the public key technology must be set up in a preparatory phase (“pre-phase”). This includes determining which organization acts as the “Certification Authority (CA)”, which certifies the authenticity of Public Keys for the involved parties. This role can initially be assumed by one or more of the affected customers, banks, etc., or can be assumed by SAP. In the future, there will be inspected, public certification points. The preparatory phase also includes installing the security products, generating keys, and distributing certificates to the involved partners. For a complete public key infrastructure, the revocation of certificates by the CA via publishing and distributing so-called „certificate revocation lists (CRLs)“ should also be defined.

2.2 “In-Phase”

The architecture and workflow during actual use of the SSF functions (“in-phase”) are illustrated in Figure 3. The R/3 applications (FI, SD, etc.) access the SSF functions using various ABAP function modules provided for the “SSF Call Interface” by the Basis software.

The security aspects (such as digital signature, encryption) of the data are passed on to the corresponding SSF ABAP function module, which in turn calls the appropriate “C” functions (for the front end via RFC (“Remote Function Call”)). The SSF RFC server program or the R/3 kernel calls the appropriate “C” functions using the SSF API.

To enable use of the SSF API functions, a function library that provides the “C” functions integrated through the SSF API is dynamically loaded at runtime. The implementation through the “C” functions specified by the SSF API establish the connection to the security product (“security toolkit”). In the process, the specific API functions of the respective security product are called. After the return from the RFC or the kernel, the secure data is passed from the SSF ABAP function module back to the application.

This entire process is indicated with “Sign/Envelope” in Figure 3. To verify secure data and make it readable again, the reverse process is applied (“Develope/Verify” in Figure 3).

Frontend and Application Server

The SSF operations can be carried out at the application server and at the front end. Whenever possible, the application server’s security toolkit is used, e.g. to verify signatures or digitally sign data with the system’s private

key. Frontend operations are required, when a user has to sign a document and to decrypt documents encrypted with his public key.

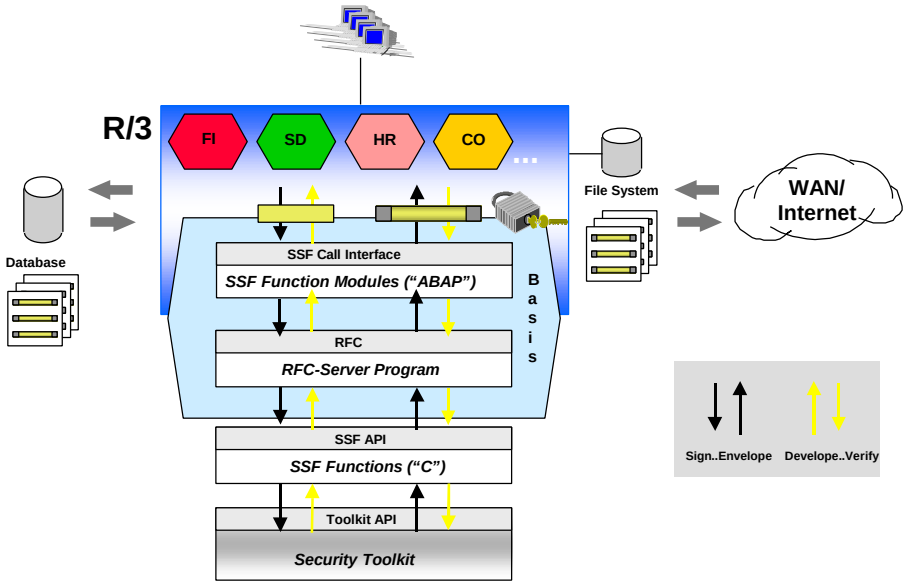

Figure 3. SSF Architecture

2.3 SSF API Certification

The function library which includes the implementation of the SSF API can come from various security product vendors, which means that many different security products can be used.

To guarantee proper interaction with the R/3 System, however, the security product must be certified by SAP for use with SSF.

For complete information about the BC-SSF certification, see http://www.sap.com/csp/scenarios (→ Business Technology). More information, e.g. the SSF Test Plan, header files and a test tool, are available there on request.

SAP Security Library

As of Release 4.5, R/3 is shipped with SAPSECULIB (SAP Security Library) as the default SSF service provider. SAPSECULIB is a software solution with functionality that is limited to digital signatures at the application server. For support of crypto hardware (for example, smart cards, crypto boxes, etc.) or digital envelopes, you need a SAP-certified external security product.

2.4 SSF Standards

The format used for signed and/or encrypted data is PKCS#7 [1]. The use of SSF functions applies X.509 [2] as the standard for "Public Key" certificates. These standards form a foundation that is currently the most widespread

worldwide, while still maintaining maximum flexibility for future enhancements.

2.5 Application Scenarios

The SSF functions can be applied in various scenarios for protecting data and documents:

1. Clear-text data entered in the SAPgui is immediately transposed to the secure format by the application, and is then saved in the R/3 database in that secure format. When the data is required again, it is read from the R/3 database and then verified and/or decrypted by the SSF functions before the actual use.
2. If necessary, the application can also save the data entered in the SAPgui directly in the file system in the secure format. When the data is required again, it is read from the file system and then verified and/or decrypted by the SSF functions before the actual use.
3. The data from an R/3 transaction is initially stored in the various tables of the R/3 System. In further transactions in the SAP workflow, the data is then read from the R/3 database, a digital signature and/or encryption is applied, and the data is saved again in the R/3 database in the secure format.
4. Data is read from the R/3 database and prepared for external storage and/or transport/transmission. To do this, the data is initially transformed into the required external format and then secured with the SSF functions. Once the data is available in the secure format, it can be safely saved to disk or transmitted through insecure communication links, like the Internet. The intended recipient can be another R/3 System, or a different system that supports the secured format used.
5. Data is received in secure format from the Internet (or through another method) and imported into the R/3 System. Note that the secure data does not necessarily must have been generated with R/3 - it only needs to be available in the secure format used. After decryption with the SSF functions, the data is available in clear text. In addition, the digital signature is verified as necessary.

3. SSF Functions

3.1 Overview

In its first version, the SSF interface includes 17 functions, 11 functions described in this chapter and six auxiliary functions as described in the next chapter. A listing of the function names can be found in Table 1. The functions are generally used in pairs: one function for "wrapping" the data and/or documents, and a corresponding function for "unwrapping" them from the secure format. The format itself is variable, and can be specified as a parameter in any of the functions. Initially, however, only the PKCS#7 [1] format is supported. Use of a different, unsupported format will result in the issuing of an appropriate return code.

SsfEncode	**SsfDecode**
SsfSign **SsfAddSign**	**SsfVerify**
SsfEnvelope	**SsfDevelope**
SsfDigest	
SsfVersion	**SsfQueryProperties**
SsfDELSsfOctetstring	

Table 1. SSF Functions (Version 1)

The **SsfVersion** function returns information on the current version of the SSF API and of the underlying security product. With **SsfQueryProperties**, further properties of the security toolkit can be queried.

The **SsfEncode** function transforms clear-text entries into a uniform, encoded format. Under PKCS#7, this is an ASN.1 encoding that corresponds to the ASN.1 data types defined in [1] and encoding in accordance with ASN.1 encoding rules. This transforms the data into a binary format that is suitable for transmission between computers (Octetstring). In PKCS#7, the "Data" format is used. Please note that the data is not yet secured by digital signatures or encryption/decryption. The Octetstring is decoded again into the local representation on a given computer by the **SsfDecode** function. For more information see paragraph 3.4.

The **SsfSign** function generates one or more digital signatures under the input data. With PKCS#7, the "SignedData" format is used for this. The **SsfAddSign** function is used to add several digital signatures in several steps and at different times. It adds an additional signature to data that has already been signed. **SsfVerify** is used to verify the digital signatures.

The **SsfEnvelope** and **SsfDevelope** functions encrypt data for one or more recipients and decrypt it for each recipient, respectively. With PKCS#7, the "EnvelopedData" format is used for this.

Function **SsfDigest** is used to compute a hash value from the input data. In PKCS#7, the hash value is returned in "DigestedData" format. **SsfDigest** is a one-way function, and does not have a corresponding reverse function.

The **SsfDELSsfOctetstring** function deletes output data returned from calls to the other SSF functions and frees the associated memory.

Note: The auxiliary functions described in chapter 4 are part of the SSF API.

3.2 Utilized Types and Codes

3.2.1 Types

The following data types are used for parameters and results of the SSF functions. To simplify the description of the formal parameters in this document, we use the informal name instead of their C type. The correct type can be found in the C prototypes.

Charstring Character string

```
typedef char *   SsfCharstring;
```

Octet String Binary string

```
typedef char *   SsfOctetstring;
```

Integer Whole number (always greater than or equal to 0 in this document)

```
typedef int      SAP_INT;
typedef unsigned int SAP_UINT;
```

Note: It is assumed here, that the size of `int` is four bytes.

Boolean TRUE/FALSE

```
typedef enum { FALSE = 0, TRUE = 1 } SAP_BOOL;
```

SigRcpSsf Information Composite type with the components:

`strSigRcpId`	Charstring
`strSigRcpL`	Integer
`strSigRcpReserved`	Charstring
`strSigRcpReservedL`	Integer
`strSigRcpProfile`	Charstring
`strSigRcpProfileL`	Integer

strSigRcpPassword	Charstring
strSigRcpPasswordL	Integer
uResult	Integer

This structure is always passed as pointer:

```
typedef SigRcpSsfInformation *PtrSigRcpSsfInformation;
```

List of ... Finite, non-recursive list with elements of the respective composite type

```
typedef struct SigRcpSsfInformationListStruct *
        SigRcpSsfInformationList;
```

Note: Output parameters are passed as a pointer to the respective element.

The following listing is an extract of the C include file that defines these types:

```
/* SSF type for function return codes: SAPRETURN */
typedef int      SAPRETURN;

/* SSF type for Characterstring names */
typedef char *   SsfCharstring;

/* SSF type for Octetstring data */
typedef char *   SsfOctetstring;

/* SSF type for Integer: SAP_INT */
typedef int      SAP_INT;
typedef unsigned int SAP_UINT;

/* SSF type for Boolean */
typedef enum { FALSE = 0, TRUE = 1 } SAP_BOOL;

/*** Signer/Recipient Security Information types ***/

/* Security info of one signer/recipient */
typedef struct /* SigRcpSsfInformation */
{
    SsfCharstring  strSigRcpId;           /* name of signer/recipient */
    SAP_INT        strSigRcpIdL;          /* length of above          */
    SsfCharstring  strSigRcpReserved;     /* reserved for future use  */
    SAP_INT        strSigRcpReservedL;    /* length of above          */
    SsfCharstring  strSigRcpProfile;      /* signer/recip sec profile */
    SAP_INT        strSigRcpProfileL;     /* length of above          */
    SsfCharstring  strSigRcpPassword;     /* password signer/recipient*/
    SAP_INT        strSigRcpPasswordL;    /* length of above          */
    SAP_UINT       uResult;               /* result of SSF operation  */
}   SigRcpSsfInformation;

/* Pointer to security info of one signer/recipient */
typedef SigRcpSsfInformation *      PtrSigRcpSsfInformation;

/* List of signer security info */
typedef struct SigRcpSsfInformationListStruct *
    SigRcpSsfInformationList;
```



```
/* List element of signer security info */
typedef struct SigRcpSsfInformationListStruct
{
    SigRcpSsfInformationList   ptrNextSigRcp;   /*next element*/
    PtrSigRcpSsfInformation    ptrSigRcp;       /*this element*/
}   SigRcpSsfInformationListElement;
```

3.2.2 Return Codes

Following is the complete list of possible return codes of the SSF functions. These return codes are an indication of the overall result of the SSF function.

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0 or input data does not match signed data
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_SIGNER_ERRORS	(5)	Error detected for one or more signatories
SSF_API_NORESULTLISTMEMORY	(6)	Not enough dynamic memory available for results list
SSF_API_UNKNOWN_PAB	(7)	Private address book not found
SSF_API_INVALID_PAB_PASSWORD	(8)	Password for private address book is not valid
SSF_API_RECIPIENT_ERRORS	(9)	Errors were detected for one or more recipients
SSF_API_INVALID_MDALG	(10)	Unknown or unsupported hash algorithm
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product
SSF_API_INVALID_SYMALG	(14)	Unknown or unsupported symmetric encryption algorithm

Table 2. SSF Return Codes

3.2.3 Character String Codes

Following are summary tables for valid character string codes used with the SSF API.

Note: These parameters have the type string so that new entries (e.g. algorithms) can be added without the need to change this specification. Thus, new strings will be added in the future.

Furthermore, the supported codes depend on the used security product: Not every security product must support all valid strings and new strings might be added by security products. Nevertheless, a common subset of valid strings should be supplied, see the recommendations.

Format

The following are the security wrapper formats that are supported at the SSF API. This may vary depending on the used security toolkit.

FORMAT	
PKCS#7 [1] (required)	"PKCS7"

Table 3. Character string codes for formats

Hash Algorithms

The following are the hash algorithms that are supported at the SSF API. This may vary depending on the used security toolkit.

HASH ALGORITHM	
MD5 (required)	"MD5"
SHA-1 (required)	"SHA1"
MD2	"MD2"
MD4	"MD4"
RIPEMD-160	"RIPEMED160"

Table 4. Character string codes for hash algorithms

Symmetric Encryption Algorithms

The following are the symmetric encryption algorithms that are supported at the SSF API. This may vary depending on the used security toolkit.

SYMMETRIC ENCRYPTION ALGORITHM	
DES CBC (required)	"DES-CBC"
Triple DES	"TRIPLE-DES"
IDEA	"IDEA"

Table 5. Character string codes for symmetric encryption algorithms

3.2.4 Signer/Recipient Information and Results Codes

The composite data type `SigRcpSsfInformation` contains fields that are used for input and output. The first four fields provide information to identify a signatory or recipient, respectively. For a signing operation, they are input fields. When verifying digital signatures, they are output fields.

For the first four fields of the data type `SigRcpSsfInformation` the character string type has been used. String length information is provided in the associated length fields of type Integer. Please note that, although the strings need not be zero-terminated and may contain any character, they are customizable within the R/3 system and thus should be in human readable form.

ID

The `strSigRcpId` field contains the name of the signatory/recipient. Usually, the `strSigRcpId` field will hold a name that is globally unique for the signatory/recipient (e.g. X.500 Distinguished Name).

Profile

The `strSigRcpProfile` field contains the specification where to find the security profile of the signatory/recipient. For example, this can be a directory path or file name, database index or smartcard reader specification. See section 3.2.5.

Password

The `strSigRcpPassword` field provides a means of passing password information for a signatory/recipient's security profile with the SSF call. This may be needed to open the security profile and access private key information, in particular. The usage of this field is optional.

uResult

The fifth field `uResult` of the composite data type `SigRcpSsfInformation` is used for output. It contains the result of the respective SSF function for that signatory/recipient. Applications using SSF functions check the `uResult` field in case the overall return code of an SSF call does not equal SSF_API_OK (see Table 2). Possible values for the `uResult` field are listed in the following table.

uResult	(in composite type SigRcpSsfInformation)	
SSF_API_SIGNER_OK	(0)	Successful, no error for signatory
SSF_API_RECIPIENT_OK	(0)	Successful, no error for recipient
SSF_API_UNKNOWN_SIGNER	(22)	Signatory unknown
SSF_API_UNKNOWN_RECIPIENT	(22)	Recipient unknown
SSF_API_UNKNOWN_PROFILE	(23)	Security profile for signatory not found
SSF_API_INVALID_PROFILE	(24)	Security profile for signatory not applicable
SSF_API_INVALID_PASSWORD	(25)	Invalid password for signatory
SSF_API_NOCERTIFICATE	(26)	Certificate for signatory not available
SSF_API_SIGNER_NOT_OK	(27)	Digital signature could not be provided/verified
SSF_API_RECIPIENT_NOT_OK	(27)	Encryption/decryption not possible for recipient
SSF_API_UNDEFINED_RESULT	(28)	Result undefined

Table 6. Integer codes for signer/recipient results

3.2.5 SSF Profile and Private Address Book

To access the user's private key or any public key, the SSF API uses *SSF Profiles* and *Private Address Books*.

They are character strings that are customized within the R/3 system and passed to the security toolkit. With this information (together with the respective password), the security toolkit must be able to find the respective private or public key. For example, this can be a directory path or file name, database index or smartcard reader specification. The usage of this field is optional and may vary between different security toolkits[1].

SSF Profile Information for the security toolkit to determine how to find and use the private key of a signatory/recipient. In addition, it is assumed that the (default) cryptographic algorithm for signing and any other required information should be derived from the security profile specification. This provides the desired level of abstraction from details in the security infrastructure as needed by the applications using SSF.

Private Address Book Information for the private address book in the system, consisting of a name and password for opening, as well as the corresponding length specifications. The private address book is used to verify signatures and to find the recipient's public keys when enveloping data. The private address book typically contains information (and possibly certificates as well) on possible

[1] We don't make assumptions concerning the contents of this field; it is simply passed to the security toolkit. However, note that this string should be in human readable form, as it has to be entered into the R/3 database.

recipients and senders of digitally signed and/or encrypted data. The information contained therein enables direct or indirect access to public keys for verifying digital signatures or encrypting data for specific recipients. Private address books are generally used when no public directory infrastructure (such as X.500) is available, or as a supplement to such an infrastructure.

Note: Private address book and profile might be identical, although this is not required. (Profile is used to access the private key, whereas the private address book is used to retrieve or verify the public component of others.)

Note: If the security product has to distinguish private address book and profile, different names should be chosen (e.g. by adding a prefix)

3.2.5.1 SSF profile and private address books for application servers

When installing profiles and private address books for R/3 systems (e.g. to verify the users' signatures or to create system signatures) one should keep in mind that most R/3 systems consist of multiple application servers:

- Every application server must have access to the private and public keys (see next section 'Passwords' as well).
- As the profile and PAB information is identical for all application servers, each application server's security toolkit must be able to handle the same information. Thus, (absolute) filenames are very dangerous as this can lead to problems when using different platforms (e.g. Windows and Unix). (Note that the same problem arises at the front end when users use multiple front ends to create digital signatures.)

3.2.5.2 Passwords

There are two possible locations where the security toolkit can be used: Either at the front end or at the application server. For those two situations, there are different requirements to protect the profile and PAB.

Front end

On the front end, the security toolkit is used interactively. Thus, the password protecting the profile or PAB can be entered by the user.

Note: Currently, the password is prompted by R/3. Depending on the security toolkit used, there's the possibility that the password can be prompted by the toolkit itself (optionally showing the to be signed data), in future versions. See next section.

Application Server

On the application server, the security toolkit is used as a service. Therefore, it isn't possible to ask for a password. Thus, the password passed to the security toolkit is left blank and should be ignored.[2] Instead, it is up to the security toolkit to limit the access to the profile and PAB. This can be realized with credentials. They can be created either when the security toolkit

[2] Otherwise it would be necessary to store the password in the database

is installed or each time the application server is started, depending on the customer's needs and security policy.

3.2.5.3 Password Policies

Password aging and user locking

The security toolkit is responsible for assuring the security of profiles and PABs. Security policies for password limitations (e.g. minimum length), password aging, account locking and logging of the successful and/or false attempts to open the profile, should be defined and controlled by the security toolkit. If necessary, the security toolkit can interact with the user (e.g. when the user has to change his password), see next section.

Password caching and SNC

Some security toolkits have a single sign-on, where you only have to enter the password the first time you access the private key. While this feature might be useful for encryption (depending on the chosen security policy), we do not recommend it for digital signatures. Digital signatures should only be created if the user explicitly authorizes the security toolkit to create a signature by entering the password. Especially when the security toolkits supports SNC (Secure Network Communication) as well, these two shouldn't be mixed.

3.2.5.4 Password prompt by the security toolkit

Password popups

By default, the password is prompted by R/3. Depending on the security toolkit used, there's the possibility that the password can be prompted by the toolkit itself (optionally showing the to be signed data).

Note: Currently, this feature isn't available with all R/3 applications; some of them will always prompt and supply a password.

This can be enabled by returning a value of '`1`' or '`X`' for `SsfQueryProperties("SSF_POPUPS",10,NULL,0)`. If this property isn't available, this feature is automatically disabled.

A value of '`X`' indicates that the security toolkit supports popups (with the capability to show the to be signed text). If '`1`' is returned, only the password popup is available.

In these cases, the security toolkit must be prepared to receive the password '`!`' (either for the profile or for the PAB) indicating that it has to prompt for the password. (This exclamation-mark may be followed by up to two characters to specify the kind of popup. Currently, no special popup types are defined.)

Note: This limits the range of valid passwords. However, passwords starting with '!' aren't valid passwords within R/3 systems and the maximum length (3) of these 'special' passwords is very short.

Note: For empty passwords, no popup should be shown. Return an INVALID_PASSWORD error, if the profile or PAB cannot be accessed without password.

3.3 Utilized Input/Output Information Blocks

Here's an overview of the parameters used by the SSF API. For an exact description, see the individual function description.

INPUT Parameters:

Format

strFormat (Charstring)

A character string with the ID of the required source/target security wrapper format of the SSF function (see Table 3).

Contained Certificates

bIncCerts (Boolean)

bUseCerts (Boolean)

Indicates whether certificates (containing the signer's public key) are to be included in the security wrapper data when digital signatures are created or whether the certificates included in the security wrapper data are to be used when verifying digital signatures. A value of "TRUE" means "include/use", a value of "FALSE" means "do not include/do not use".

Signature and Hash Only

bDetached (Boolean)

Specifies whether the digital signature or the hash is to be generated "detached", that is without the data (TRUE), or with the data (FALSE) included in the signed data (digested data) wrapper.

Input data

ostrInputData, ostrInputDataL (Octet string, Integer)

A binary string with the data for handling and a length specification for the binary string itself. For most of the functions, this data has to be encoded fist. See SsfEncode and SsfDecode.

Comparison Data

ostrInputData, ostrInputDataL (Octet string, Integer)

A binary string with the data to compare and a length specification for the binary string itself. This data is needed for verification of "detached" digital signatures.

Hash Algorithm

(Charstring, Integer)

Name of the hash algorithm to be applied and the corresponding length specification (see Table 4). This is used both for `SsfDigest` (where the hash is calculated) and `SsfSign` (`SsfAddSign`).

Symmetric Encryption Algorithm (Charstring, Integer)

Name of the symmetric encryption algorithm to be applied for message encryption and the corresponding length specification (see Table 6). Symmetric encryption is applied to the contents of enveloped data using a newly generated message key. The message key is encrypted using the public keys of the intended recipients. The private/public key mechanisms and actual key pairs needed for the SSF operations are located by the security product using the profile information passed for signatories and recipients.

IN/OUT Parameters:

Signer List (List of `SigRcpSsfInformation`, used in `SsfSign`)

A list of signers. See next item Signer.

Signer (`SigRcpSsfInformation`, used in `SsfAddSign`)

Identification of the signatory, the security profile to be applied for that signatory, and a password for opening the security profile for accessing the signatory's security information. In addition, the `uResult` field returns the result of the SSF function for the signatory (see Table 6). For more information, see section 3.2.4.

Signer Result List (List of `SigRcpSsfInformation`, used in `SsfVerify`)

Identification of the signatory. For the verification of digital signatures, the security profile field contains additional attributes, such as signing time. The `uResult` field returns the result of the SSF function for the respective signatory (see Table 6).

Recipient List (List of `SigRcpSsfInformation`, used in `SsfEnvelope`)

A list of recipients. For each recipient: information to identify the recipient. The security product uses this information to search for the public key (in the given private address book). The security profile field and the password field must be empty. In addition, the `uResult` field returns the result of the SSF function for the respective recipient (see Table 6).

Recipient (`SigRcpSsfInformation`, used in `SsfDevelope`)

Information for a recipient - e.g. information to identify the recipient, the security profile to be used for this recipient and the password to be used for opening the security profile. The respective security product uses this information to access the private key for decryption. In addition, the `uResult` field returns the result of the SSF function for the recipient (see Table **6**).

Private Address Book (Charstring, Integer, Charstring, Integer, used in SsfVerify and SsfEnvelope)

Information for the private address book in the system, consisting of a name and password for opening, as well as the corresponding length specifications.

OUT Parameters:

Return Code (Integer)

Specification of the general result of an SSF function (in addition to the output parameters). See Table 2.

Output Data (Octetstring, Integer)

A binary string with the processed data (actual result) and a length specification for the binary string. This string must be freed with SsfDELSsfOctetstring.

3.4 Encoding and Decoding

All binary input data that is processed by `SsfSign` and `SsfEnvelope` has to be converted to an internal format with `SsfEncode`, first. Correspondingly, the output data returned by `SsfVerify` and `SsfDevelope` has to be converted back to the original data with `SsfDecode`.

Therefore, data processing looks like this:

1. `SsfEncode` the original data
2. `SsfSign`, `SsfAddSign`, `SsfEnvelope` (you can use the output of these functions as new input for the next functions as their output is encoded).
3. …
4. `SsfDevelope` and `SsfVerify` (according to step 2, reverse order)
5. `SsfDecode` to get back the initial data from step 1.

The encoded data returned by `SsfEncode` isn't used directly by any caller of the SSF API. Thus, there's no predefined format: Any toolkit provider is free to choose an appropriate format. (It is even possible that `SsfEncode` and `SsfDecode` don't modify the data.)

Note: For PKCS7, we recommend using PKCS7 plain data.

Remark: This has been introduced so that the sequence `SsfSign` followed by `SsfEnvelope` can distinguish the data returned by `SsfSign` from plain input data.

	Ssf Encode	**Ssf Decode**	**Ssf Sign**	**Ssf AddSign**	**Ssf Verify**	**Ssf Envelope**	**Ssf Develope**	**Ssf Digest**	**Ssf Version**	**Ssf QueryProps**
IN										
Format	X	X	X	X	X	X	X	X		
include/use certificates			X	X	X					
detached			X					X		
Input data	X	X	X	X	X	X	X	X		
Compare data				X	X					
Private address book					X	X				
Hash alg.			X	X				X		
Symmetric algorithm						X				
INOUT										
Signer				X						
Signer list			X							
Recipient							X			
Recipient list						X				
OUT										
Signer result list					X					
Return code	X	X	X	X	X	X	X	X	X	X
Output data	X	X	X	X	X	X	X	X	X	X

Table 7. SSF functions - input/output information

3.5 Functions

The following sections describe the SSF functions.

3.5.1 SsfEncode

SsfEncode	With this function, input data is transformed from the local representation into a uniformly encoded format. The data is not secured by digital signatures and/or encryption. In PKCS#7, this is an ASN.1 encoding that corresponds to the ASN.1 data types defined in [1] and encoding in accordance with ASN.1 Basic Encoding Rules. This transforms the data into a binary format that is suitable for transmission between computers (Octetstring). In PKCS#7, the “Data” format is used. The SsfDecode function is used to decode the string.

IN		
strFormat	(Charstring)	Target format
strFormatL	(Integer)	Length of strFormat
ostrInputData	(Octet string)	Input data in uncoded format
ostrInputDataL	(Integer)	Length of ostrInputData

OUT		
OstrEncodedData	(Octet string)	Output data (input data in encoded format). This string must be freed with SsfDelSsfOctetstring.
ostrEncodedDataL	(Integer)	Length of ostrEncodedData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.2 SsfDecode

SsfDecode	This function is used to decode the encoded data - that is, to transform it back to the local representation. In PKCS#7, this is an ASN.1 decoding that corresponds to the ASN.1 data types defined in [1] and in accordance with ASN.1 Basic Encoding Rules. The input data must be available in encoded PKCS#7 “Data” format. The SsfEncode function is used to encode the data.

IN		
strFormat	(Charstring)	Source format
strFormatL	(Integer)	Length of strFormat
ostrEncodedData	(Octet string)	Input data in encoded format
ostrEncodedDataL	(Integer)	Length of ostrEncodedData

OUT		
ostrOutputData	(Octet string)	Output data (input data in uncoded format). This string must be freed with SsfDelSsfOctetString.
ostrOutputDataL	(Integer)	Length of ostrOutputData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.3 SsfSign

SsfSign	This function is used to digitally sign the encoded input data. In PKCS#7, the encoded “SignedData” format is returned. The `SsfVerify` function is used to verify the digital signatures. Comments: • Multiple signatories are supported. The result is only generated when all signatories are known and the security information of all signatories can be located and used. In this case (success case), ostrSignedData refers to the encoded result string with all digital signatures. SignerList contains all signatories with the result “OK”. • In an error case, `ostrSignedData` has value `NULL` and `ostrSignedDataL` the value `0`. SignerList contains information on both the valid signatories and the invalid signatories. • Sign must check that the given StrSigRcpId is correct (i.e. matches the ID specified with StrSigRcpProfile); return uResult `SSF_API_UNKNOWN_SIGNER` otherwise. Don’t simply use the key specified by StrSigRcpProfile. • Each digital signature includes a time stamp indicating when the digital signature was created, which is also protected by the digital signature. With the PKCS#7 format the „SigningTime“ attribute defined in PKCS#9 is to be added as an authenticated attribute.

IN		
strFormat	(Charstring)	Target format
strFormatL	(Integer)	Length of strFormat
strHashalg	(Charstring)	Target hash algorithm to be used
strHashalgL	(Integer)	Length of strHashalg
bIncCerts	(Boolean)	Include certificates with the signed data (TRUE) or do not include certificates in the signed data
bDetached	(Boolean)	Generate signature without input data contained in the signed data (TRUE) or with input data contained in the signed data (FALSE)
ostrInputData	(Octetstring)	Input data (encoded with SsfEncode)
ostrInputDataL	(Integer)	Length of ostrInputData

INOUT	(All parameters are input, `uresult` is set as output)	
SignerList	(List of SigRcpSsfInformation)	
StrSigRcpId	(Charstring)	Identification (name) of signatory
StrSigRcpIdL	(Integer)	Length of strSignerId
StrSigRcpReserved	(Charstring)	Not used, must be NULL
StrSigRcpReservedL	(Integer)	0
StrSigRcpProfile	(Charstring)	Identification of security profile for strSignerId
StrSigRcpProfileL	(Integer)	Length of strSignerProfile
StrSigRcpPassword	(Charstring)	Password of signatory
StrSigRcpPasswordL	(Integer)	Length of strSignerPassword
Uresult	(Integer)	Signer result code
	SSF_API_SIGNER_OK (0)	Valid signature
	SSF_API_UNKNOWN_SIGNER (22)	Signatory not found
	SSF_API_UNKNOWN_PROFILE (23)	Security profile unknown
	SSF_API_INVALID_PROFILE (24)	Security profile not applicable
	SSF_API_INVALID_PASSWORD (25)	Password not valid
	SSF_API_NOCERTIFICATE (26)	Certificate not found
	SSF_API_SIGNER_NOT_OK (27)	Signature not valid
	SSF_API_UNDEFINED_RESULT (28)	Result not defined
DERIVED (from Profile)	• Signatories' private keys • Signatories' certificates • Signatories' Public Key algorithms • Signatories' Message Digest algorithm	

OUT		
ostrSignedData	(Octet string)	Output data (signed input data in encoded format). This string must be freed with SsfDelSsfOctetString.
ostrSignedDataL	(Integer)	Length of ostrSignedData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_SIGNER_ERRORS	(5)	Error detected for one or more signatories
SSF_API_INVALID_MDALG	(10)	Unknown or unsupported hash algorithm
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.4 SsfAddSign

SsfAddSign	This function is used to add an additional digital signature to input data that has already been digitally signed with SsfSign. In PKCS#7, the encoded “SignedData” format is expected and returned. The verification of digital signatures is performed using the SsfVerify function. Comments: • One additional signatory is supported. The result is only generated when the signatory is known and the security information of this signatory can be located and used. In this case (success case), ostrUpdatedSignedData refers to the encoded result string with all digital signatures. SignerResult contains the result “OK” for the specified signatory. • In an error case, ostrUpdatedSignedData has value NULL, and ostrUpdatedSignedDataL has value 0. SignerResult contains error information for the specified signatory. • Sign must check that the given StrSigRcpId is correct (i.e. matches the ID specified with StrSigRcpProfile); return uResult `SSF_API_UNKNOWN_SIGNER` otherwise. Don’t simply use the key specified by StrSigRcpProfile. • The digital signature includes a time stamp indicating when the digital signature was created, which is also protected by the digital signature. With the PKCS#7 format the „SigningTime“ attribute defined in PKCS#9 is to be added as an authenticated attribute. • In case the input signed data was generated without the signed data included (detached digital signature) the data that was signed has to be supplied in the ostrInputData parameter. If the input signed data contains the data and ostrInputData also contains data, the two datasets have to be equal. If not, the function returns an error (SSF_API_NODATA). • Most of the parameters have to be identical to a former call to SsfSign or SsfAddSign.

IN		
StrFormat	(Charstring)	Source/target format
StrFormatL	(Integer)	Length of strFormat
StrHashalg	(Charstring)	Target hash algorithm to be applied
StrHashalgL	(Integer)	Length of strHashalg
BincCerts	(Boolean)	Include certificate with the updated signed data (TRUE) or do not include certificate with the updated signed data (FALSE)
OstrSignedData	(Octet string)	Input data in encoded format as returned by a former call to SsfSign or SsfAddSign
OstrSignedDataL	(Integer)	Length of ostrSignedData
OstrInputData	(Octet string)	Data to sign (encoded with SsfEncode). May be omitted if the signed data is included in OstrSignedData (i.e. no detached signature).
OstrInputDataL	(Integer)	Length of data to sign

INOUT	(All parameters are input, `uresult` is set as output)	
Signer	(SigRcpSsfInformation)	
strSigRcpId	(Charstring)	Identification (name) of signatory
strSigRcpIdL	(Integer)	Length of strSignerId
StrSigRcpReserved	(Charstring)	Not used, must be NULL
StrSigRcpReservedL	(Integer)	0
strSigRcpProfile	(Charstring)	Identification of security profile for strSignerId
strSigRcpProfileL	(Integer)	Length of strSignerProfile
strSigRcpPassword	(Charstring)	Password of signatory
strSigRcpPasswordL	(Integer)	Length of strSignerPassword
uResult	(Integer)	Signer result code
	SSF_API_SIGNER_OK (0)	Valid signature
	SSF_API_UNKNOWN_SIGNER (22)	Signatory not found
	SSF_API_UNKNOWN_PROFILE (23)	Security profile unknown
	SSF_API_INVALID_PROFILE (24)	Security profile not applicable
	SSF_API_INVALID_PASSWORD (25)	Password not valid
	SSF_API_NOCERTIFICATE (26)	Certificate not found
	SSF_API_SIGNER_NOT_OK (27)	Signature not valid.
	SSF_API_UNDEFINED_RESULT (28)	Result not defined
DERIVED (from Profile)	• Signatory's private key • Signatory's certificates • Signatory's Public Key algorithm • Signatory's Message Digest algorithm	

OUT		
ostrUpdatedSignedData	(Octet string)	Output data (input data - with an added signature - in encoded format) This string must be freed with SsfDelSsfOctetString.
ostrUpdatedSignedDataL	(Integer)	Length of ostrUpdatedSignedData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0 or input data does not match signed data
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_SIGNER_ERRORS	(5)	Error detected for signatory
SSF_API_INVALID_MDALG	(10)	Unknown or unsupported hash algorithm
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.5 SsfVerify

SsfVerify	This function is used to verify the digital signatures of the input data. Because multiple signatories are possible, a list is returned with the results as output parameters. In addition, the signed data is returned. In PKCS#7, the encoded "SignedData" format is the required input format. The digital signature is generated using the `SsfSign` function. Comments: • In case digital signatures were generated as detached signatures the input data parameter has to point to the data that was signed. If the input data parameter is not `NULL` and the digital signature is not a detached signature the two datasets are compared for equality and `SSF_API_NODATA` is returned if they are different. • The output data (i.e. the data that has been signed) must be returned even if an error occurred (especially if the signature couldn't be verified). • Please note the special return code `SSF_API_UNKNOWN_SIGNER` that should be used if the signature was successfully verified with an included certificate but the validity of that certificate couldn't been verified (either because it is self-signed or issued by an unknown CA) • The included (or retrieved) certificate should be returned in the field `strSigRcpPassword` of the `SignerResultList`. If the certificate isn't available as much information as possible (e.g. of the ToBeSigned part of the certificate) should be returned.

IN		
StrFormat	(Charstring)	Source format
StrFormatL	(Integer)	Length of strFormat
bUseCerts	(Boolean)	Use included certificates (TRUE) or not (FALSE). In most cases, TRUE should be passed.
OstrSignedData	(Octet string)	Input data in encoded format
OstrSignedDataL	(Integer)	Length of ostrSignedData
OstrInputData	(Octet string)	Comparison data (encoded with SsfEncode) - only required if OstrSignedData has been created detached.
OstrInputDataL	(Integer)	Length of comparison data
strPab	(Charstring)	Identification of private address book
strPabL	(Integer)	Length of strPab
strPabPassword	(Charstring)	Password for private address book
strPabPasswordL	(Integer)	Length of strPabPasswordL

DERIVED	• Signatories' certificates (from PAB) • Message Digest algorithms (from input data) • Public Key algorithms (from input data)

OUT			
SignerResultList		(List of SigRcpSsfInformation)	
strSigRcpId		(Charstring)	Identification (X.500 distinguished name) of signatory If the name is not available (e.g. if the certificate isn't available) return "`issuer-name/No-serialnumber`": `CN=CA, OU=SSF-TEST, O=SAP-AG, C=DE/No-2`). uResult mustn't be `SSF_API_OK`, then, use `SSF_API_NO-CERTIFICATE.`
strSigRcpIdL		(Integer)	Length of `strSignerId`
StrSigRcpReserved		(Charstring)	Not used, set to NULL and ignore
StrSigRcpReservedL		(Integer)	0
strSigRcpProfile		(Charstring)	Timestamp when the digital signature was created. The format is: "SigningTime= *human readable form* (UTCTime: *utctime*)" Example: "`SigningTime= Mon Jul 13 10:14:11 1998 (UTCTime: 980713081411Z)`". **Note:** Additionally, multiple signed attributes might be returned. Separate them by a 0-character (\0), prefix them with "`attributename=` " and assure that the first attribute is the signing time.
strSigRcpProfileL		(Integer)	Length of `strSignerProfile`
strSigRcpPassword		(Charstring)	Certificate of Signer (ASN1 encoded), if available. If the certificate isn't available, provide as much information as possible (of the ToBeSigned part) and encode it with a dummy signature of length 0.
strSigRcpPasswordL		(Integer)	Length of `strSignerPassword`
uResult	(Integer)		Signer result code SSF_API_SIGNER_OK (0) Valid signature SSF_API_UNKNOWN_SIGNER (22) Signatory not found Signature has been verified with the user supplied certificate, but it is either self-signed or not issued by a trusted CA. SSF_API_NOCERTIFICATE (26) Certificate not found. Certificate couldn't be retrieved and the signature hasn't been checked SSF_API_SIGNER_NOT_OK (27) Signature not valid SSF_API_UNDEFINED_RESULT (28) Result not defined
ostrOutputData		(Octet string)	Output data (the signed data). This string must be freed with SsfDelSsfOctetString.
ostrOutputDataL		(Integer)	Length of ostrOutputData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error (i.e. all signatures were successfully verified; otherwise return `SSF_API_-SIGNER_ERRORS`)
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0 or does not match signed data
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_SIGNER_ERRORS	(5)	Error detected for one or more signatories
SSF_API_NORESULTLISTMEMORY	(6)	Not enough dynamic memory available for results list
SSF_API_UNKNOWN_PAB	(7)	Private address book not found
SSF_API_INVALID_PAB_PASSWORD	(8)	Password for private address book is not valid
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.6 SsfEnvelope

SsfEnvelope	This function is used to encrypt (“wrap”) input data for one or more recipients. In PKCS#7, the encoded “EnvelopedData” format is used. Decryption (“unwrapping”) of the data is performed using the SsfDevelope function. Comments: • Multiple recipients are supported. The result is only generated when all the recipients are known, and the security information of all recipients can be located and used. In this case (success case), ostrEnvelopedData refers to the encoded result string with all recipient information. RecipientList contains all the recipients with the result OK. • In an error case, ostrEnvelopedData has value NULL, and ostrEnvelopedDataL has value 0. RecipientList contains information about both the valid and invalid recipients.

IN		
strFormat	(Charstring)	Target format
strFormatL	(Integer)	Length of strFormat
strSymEncr	(Charstring)	Target symmetric encryption algorithm
strSymEncrL	(Integer)	Length of strSymEncr
ostrInputData	(Octet string)	Input data (encoded with SsfEncode)
ostrInputDataL	(Integer)	Length of ostrInputData
strPab	(Charstring)	Identification of private address book
strPabL	(Integer)	Length of strPab
strPabPassword	(Charstring)	Password for private address book
strPabPasswordL	(Integer)	Length of strPabPasswordL

INOUT	(All parameters are input, `uResult` is set as output)	
RecipientList	(List of SigRcpSsfInformation)	
strSigRcpId	(Charstring)	Identification (name) of recipient.
strSigRcpIdL	(Integer)	Length of strSigRcpId
StrSigRcpReserved	(Charstring)	Not used yet, NULL
StrSigRcpReservedL	(Integer)	0
StrSigRcpProfile	(Charstring)	Not used yet, NULL
StrSigRcpProfileL	(Integer)	0
StrSigRcpPassword	(Charstring)	Not used yet, NULL
StrSigRcpPasswordL	(Integer)	0

uResult	(Integer)		Recipient result code
	SSF_API_RECIPIENT_OK	(0)	Recipient OK
	SSF_API_UNKNOWN_RECIPIENT	(22)	Recipient not found invalid RcpId (cf. error code 26)
	SSF_API_NOCERTIFICATE	(26)	Certificate (private key) not found
	SSF_API_RECIPIENT_NOT_OK	(27)	Encryption not possible for recipient
	SSF_API_UNDEFINED_RESULT	(28)	Result not defined

DERIVED (from PAB)	• Recipients' public keys • Recipients' Public Key algorithms

OUT		
ostrEnvelopedData	(Octet string)	Output data (input data encrypted for the given recipients in encoded format). This string must be freed with SsfDelSsfOctetString.
ostrEnvelopedDataL	(Integer)	Length of ostrEnvelopedData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_UNKNOWN_PAB	(7)	Private address book not found
SSF_API_RECIPIENT_ERRORS	(9)	Errors were detected for one or more recipients
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product
SSF_API_INVALID_SYMALG	(14)	Unknown or unsupported symmetric encryption algorithm

3.5.7 SsfDevelope

SsfDevelope	This function is used to decrypt (“unwrap”) input data for one recipient. In PKCS#7, the encoded “EnvelopedData” format is expected. The decrypted contents are returned. The data is encrypted (“wrapped”) using the SsfEnvelope function. Comments: • A recipient is specified for which decryption is to be performed. The result is only generated when the recipient is known, and the security information for that user can be located and used. In this case (success case), ostrOutputData refers to the decrypted result string. Recipient contains the result “OK” for the specified recipient. • In an error case, ostrOutputData has value NULL, and ostrOutputDataL has value 0. Recipient contains error information for the specified recipient.

IN		
strFormat	(Charstring)	Source format
strFormatL	(Integer)	Length of strFormat
ostrEnvelopedData	(Octet string)	Input data in encoded format
ostrEnvelopedDataL	(Integer)	Length of ostrEnvelopedData

INOUT	(All parameters are input, `uResult` is set as output)	
Recipient	(SigRcpSsfInformation)	
strSigRcpId	(Charstring)	Identification (name) of recipient
strSigRcpIdL	(Integer)	Length of strRecipientId
StrSigRcpReserved	(Charstring)	Not used yet, NULL
StrSigRcpReservedL	(Integer)	0
strSigRcpProfile	(Charstring)	Identification of security profile for strRecipientId
strSigRcpProfileL	(Integer)	Length of strRecipientProfile
strSigRcpPassword	(Charstring)	Password for recipient’s security information
strSigRcpPasswordL	(Integer)	Length of strSigRcpPassword
uResult	(Integer)	Recipient result code

See next page…

uResult	(Integer)	Recipient result code
	SSF_API_RECIPIENT_OK (0)	Recipient OK
	SSF_API_UNKNOWN_RECIPIENT(22)	Recipient not found (Id doesn't match profile)
	SSF_API_UNKNOWN_PROFILE (23)	Security profile unknown
	SSF_API_INVALID_PROFILE (24)	Security profile not applicable
	SSF_API_INVALID_PASSWORD (25)	Password not valid
	SSF_API_NOCERTIFICATE (26)	Certificate not found
	SSF_API_RECIPIENT_NOT_OK (27)	Decryption failed for recipient
	SSF_API_UNDEFINED_RESULT (28)	Result not defined

DERIVED	• Recipient's private key (from Profile) • Recipient's Public Key algorithm (from input data)

OUT		
ostrOutputData	(Octet string)	Output data (input data decrypted for the recipient). This string must be freed with SsfDelSsfOctetString.
ostrOutputDataL	(Integer)	Length of ostrOutputData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_RECIPIENT_ERRORS	(9)	Errors were detected for recipient
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product
SSF_API_INVALID_SYMALG	(14)	Unknown or unsupported symmetric encryption algorithm

3.5.8 SsfDigest

SsfDigest	This function is used to calculate a hash value for the input data. In PKCS#7, the encoded “DigestedData” format is returned.

IN		
strFormat	(Charstring)	Target format
strFormatL	(Integer)	Length of strFormat
bDetached	(Boolean)	Return hash value without the input data included (TRUE) or with the input data included (FALSE)
ostrInputData	(Octet string)	Input data (encoded with SsfEncode)
ostrInputDataL	(Integer)	Length of ostrDigestedData
strHashalg	(Charstring)	Target hash algorithm
strHashalgL	(Integer)	Length of strHashalg

OUT		
ostrDigestedData	(Octet string)	Output data (hash value in requested format). This string must be freed with SsfDelSsfOctetString.
ostrDigestedDataL	(Integer)	Length of ostrDigestedData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_INVALID_FORMAT	(2)	Unknown or unsupported security wrapper format
SSF_API_NODATA	(3)	Length of input data is 0
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_INVALID_MDALG	(10)	Unknown or unsupported hash algorithm
SSF_API_ENCODE_FAILED	(11)	ASN.1 encoding not possible
SSF_API_DECODE_FAILED	(12)	ASN.1 decoding not possible
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.9 SsfVersion

SsfVersion	This function returns information on the version of the SSF API and the security product used. The information should be in human readable form and look like: `SSFLIB Version 1.46 ; more information` Comments: • Replace `SSFLIB` by the name of your security toolkit. • The major version number must be 1 for this version of the SSF API, the minor number(s) can be chosen freely.

OUT		
OstrOutputData	(Octet string)	Output data (contains version information). This string must be freed with SsfDelSsfOctetString.
OstrOutputDataL	(Integer)	Length of ostrOutputData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.10 SsfQueryProperties

SsfQueryProperties	This function returns information on various properties of the SSF API and the security product used. Multiple entries are separated by a semicolon ';'. Currently, the following properties are defined: `PROPERTIES` List of supported properties (don't include PROPERTIES itself!) Example: `FORMATS;HASHALGS;ENCRALGS` `FORMATS` List of supported formats `HASHALGS` List of supported hash algorithms `ENCRALGS` List of supported symmetric encryption algorithms `SSF_POPUPS` (optional) Security toolkit supports (password) popups, see section 3.2.5.3

IN		
strProperty	(Charstring)	Name of the property
strPropertyL	(Integer)	Length of strProperty
strArgument	(Charstring)	(optional) parameter used with some properties.
strArgumentL	(Integer)	Length of strArgument

OUT		
OstrOutputData	(Octet string)	Output data (property information). This string must be freed with SsfDelSsfOctetString.
OstrOutputDataL	(Integer)	Length of ostrOutputData

ERRORS		
SSF_API_OK	(0)	API call ended successfully, no error
SSF_API_NOSECTK	(1)	Could not find the security product library (dynamic loading failed)
SSF_API_NODATA	(3)	strProperty is no legal property for this security product.
SSF_API_NOMEMORY	(4)	Not enough dynamic memory available for this operation
SSF_API_UNKNOWN_SECTK_ERROR	(13)	Unknown or unspecified error in security product

3.5.11 SsfDELSsfOctetstring

SsfDELSsfOctetstring	With this function, output data from a previous call to an SSF function (e.g. SsfSign, SsfVerify, …) is deleted and the associated memory is freed. Comments: • The parameters are pointer to the respective data, see the C-prototype. • The parameter ostrToBeDeleted is set to NULL and ostrToBeDeletedL is set to 0. • It is save to call this function multiple times, i.e. when ostrToBeDeleted is NULL nothing is done.

INOUT	
OstrToBeDeleted	(SsfCharstring) data to be deleted
OstrToBeDeletedL	(Integer) Length of ostrToBeDeleted

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error

4. Auxiliary Functions

4.1 Overview

In order to provide the required information for parameters at the SSF API for the SSF functions described in chapter 3 several auxiliary functions are helpful. These functions deal with the creation and deletion of lists of signer and recipient information and external representation of signer and recipient information.[3]

The auxiliary functions required in conjunction with the SSF functions are described in this chapter. Their realization is part of the SSF API.

The **SsfNEWSigRcpSsfInfo** function creates a new and initialized signer/recipient information data structure and returns a reference to the new structure.

The **SsfDELSigRcpSsfInfo** function deletes a signer/recipient information data structure and frees the associated memory.

The **SsfINSSigRcpSsfInfo** function inserts a signer/recipient information data structure into a list of signer/recipient information data structures and returns a reference to the updated list.

The **SsfDELSigRcpSsfInfoList** function deletes a list of signer/recipient information data structures and frees the associated memory.

The **SsfPRISigRcpSsfInfo** function prints the formatted information from a signer/recipient information data structure into a string.

The **SsfPRISigRcpSsfInfoList** function prints the formatted information from a list of signer/recipient information data structures into a string.

SsfNEWSigRcpSsfInfo	**SsfDELSigRcpSsfInfo**
SsfINSSigRcpInfo	**SsfDELSigRcpSsfInfoList**
SsfPRISigRcpSsfInfo	**SsfPRISigRcpSsfInfoList**

Table 8. SSF Auxiliary Functions (Version 1)

[3] They are required as allocation and freeing of memory has to be done both by the security library. To simplify work, we have provided a default implementation.

4.2 Utilized Types and Codes

4.2.1 Types

The data types used by the SSF auxiliary functions are identical to those of the SSF functions, see chapter 3.2.1.

4.2.2 Return Codes

Following is the complete list of possible return codes of the SSF auxiliary functions.

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error
SSF_AUX_NOMEMORY	(51)	Not enough dynamic memory available for this operation
SSF_AUX_NOSIGRCPID	(52)	Signer/recipient id is empty.
SSF_AUX_NOSIGRCPINFO	(53)	Signer/recipient information is empty.
SSF_AUX_NOSIGRCPINFOLIST	(54)	Signer/recipient information list is empty.

Table 9. SSF Auxiliary Function Return Codes

4.3 Auxiliary Functions

The following sections describe the SSF auxiliary functions.

4.3.1 SsfNEWSigRcpSsfInfo

SsfNEWSigRcpSsfInfo	With this function, a new signer/recipient information data structure is created and initialized with the given input values. The input values are copied into the new structure and thus can be freed or reused after a successful call of this function.

IN		
strSigRcpId	(Charstring)	signer/recipient id
strSigRcpIdL	(Integer)	Length of strSigRcpId
StrSigRcpReserved	(Charstring)	Not used, must be NULL (be prepared to handle a string, here)
StrSigRcpReservedL	(Integer)	0
strSigRcpProfile	(Charstring)	signer/recipient security profile
strSigRcpProfileL	(Integer)	Length of strSigRcpProfile
strSigRcpPassword	(Charstring)	signer/recipient profile password
strSigRcpPasswordL	(Integer)	Length of strSigRcpPasword
uResult	(Integer)	signer/recipient result code

OUT		
ptrSigRcpSsfInfo	(SigRcpSsfInformation)	The created data structure. Free this with either SsfDELSigRcpSsfInfo or SsfDELSigRcpSsf-InfoList (if it has been inserted in a list).

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error
SSF_AUX_NOMEMORY	(51)	Not enough dynamic memory available for this operation
SSF_AUX_NOSIGRCPID	(52)	Signer/recipient id is empty.

4.3.2 SsfINSSigRcpSsfInfo

SsfINSSigRcpSsfInfo	With this function, a signer/recipient information data structure is inserted at the end of a signer/recipient information list (may be empty).

IN	
PtrSigRcpSsfInfo	(SigRcpSsfInformation)

INOUT	
SigRcpSsfInfoList	(List of SigRcpSsfInformation) A pointer to the list. Free the inserted signer/recipient information data structures and the list with SsfDELSigRcpSsfInfoList.

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error
SSF_AUX_NOMEMORY	(51)	Not enough dynamic memory available for this operation
SSF_AUX_NOSIGRCPINFO	(53)	Signer/recipient information is empty.

4.3.3 SsfDELSigRcpSsfInfo

SsfDELSigRcpSsfInfo	With this function, a signer/recipient information data structure that has been created with SsfNEWSigRcpSsfInfo is deleted and the associated memory is freed. • The parameter ptrSigRcpSsfInfo is set to NULL.

INOUT	
ptrSigRcpSsfInfo	(SigRcpSsfInformation)

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error
SSF_AUX_NOSIGRCPINFO	(53)	Signer/recipient information is empty.

4.3.4 SsfDELSigRcpSsfInfoList

SsfDELSigRcpSsfInfoList	With this function, a list of signer/recipient information data structure is deleted and associated memory is freed. • The parameter `sigRcpSsfInfoList` is set to NULL. • The elements of the list are freed with `SsfDELSigRcpSsfInfo` by this routine, so this mustn't be done explicitly.

INOUT	
SigRcpSsfInfoList	(List of SigRcpSsfInformation) A pointer to the list.

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error
SSF_AUX_NOSIGRCPINFOLIST	(54)	Signer/recipient information list is empty.

4.3.5 SsfPRISigRcpSsfInfo

SsfPRISigRcpSsfInfo	With this function, a signer/recipient information data structure is printed formatted into a string.

IN	
ptrSigRcpSsfInfo	(SigRcpSsfInformation)

OUT		
OstrOutputData	(Octet string)	Output data (contains signer/recipient information). This string must be freed with SsfDelSsfOctetString.
OstrOutputDataL	(Integer)	Length of ostrOutputData

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error
SSF_AUX_NOMEMORY	(51)	Not enough dynamic memory available for this operation
SSF_AUX_NOSIGRCPINFO	(53)	Signer/recipient information is empty.

Note: This function isn't used any more (as of R/3 release 4.6). For compatibility reasons, this function should be defined and exported. It is ok to always return error code 51. (This function was used when writing trace files.)

4.3.6 SsfPRISigRcpSsfInfoList

SsfPRISigRcpSsfInfoList	With this function, a list of signer/recipient information data structure is printed formatted into a string.

IN	
SigRcpSsfInfoList	(List of SigRcpSsfInformation)

OUT		
OstrOutputData	(Octet string)	Output data (contains signer/recipient information). This string must be freed with SsfDelSsfOctetString.
OstrOutputDataL	(Integer)	Length of ostrOutputData

ERRORS		
SSF_AUX_OK	(0)	API call ended successfully, no error
SSF_AUX_NOMEMORY	(51)	Not enough dynamic memory available for this operation
SSF_AUX_NOSIGRCPINFOLIST	(54)	Signer/recipient information list is empty.

Note: This function isn't used any more (as of R/3 release 4.6). For compatibility reasons, this function should be defined and exported. It is ok to always return error code 51. (This function was used when writing trace files.)

5. Bibliography

[1] RSA Laboratories, “PKCS #7: Cryptographic Message Syntax Standard”, November 1993

[2] ITU Recommendation X.509, „The Directory - Authentication Framework“