Secure Store and Forward (SSF)
Programmers’ Guide
1.1.1 Copyright

© Copyright 2003 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice. Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.

IBM®, DB2®, DB2 Universal Database, OS/2®, Parallel Sysplex®, MVS/ESA, AIX®, S/390®, AS/400®, OS/390®, OS/400®, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere®, Netfinity®, Tivoli®, Informix and Informix® Dynamic Server® are trademarks of IBM Corporation in USA and/or other countries.

ORACLE® is a registered trademark of ORACLE Corporation.

UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.

Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc. HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA® is a registered trademark of Sun Microsystems, Inc.

JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MarketSet and Enterprise Buyer are jointly owned trademarks of SAP AG and Commerce One. SAP, SAP Logo, R/2, R/3, mySAP, mySAP.com and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are trademarks of their respective companies.
## Contents

1 Introduction ................................................................................................................. 5

1.1 General Information ................................................................................................. 5

1.2 Outline ........................................................................................................................ 5

2 Using SSF .................................................................................................................... 6

2.1 Principles .................................................................................................................... 6

2.1.1 Public keys ............................................................................................................ 6

2.1.2 Sign and Verify ..................................................................................................... 6

2.1.3 Envelope and Develope ....................................................................................... 7

2.1.4 Encoding and Decoding ....................................................................................... 7

2.2 Common Parameters ............................................................................................... 7

2.2.1 Signer and Recipient Information ......................................................................... 8

2.2.2 Include Certificates ............................................................................................ 8

2.2.3 Detached Signatures ......................................................................................... 9

2.3 Application Scenarios ........................................................................................... 9

3 SSF and C .................................................................................................................... 11

3.1 Initialization .............................................................................................................. 11

3.1.1 Load the library ................................................................................................. 11

3.1.2 Define Variables ............................................................................................... 11

3.2 Sign a Document ................................................................................................... 12

3.2.1 List of Signers ................................................................................................... 12

3.2.2 Encode and Sign .............................................................................................. 13

3.2.3 Clean up ............................................................................................................ 13

3.3 Verify a Signature .................................................................................................. 13

3.3.1 Verify ................................................................................................................. 14

3.3.2 Clean up ............................................................................................................ 14

3.4 Envelope a Document ......................................................................................... 14

3.4.1 Recipient List .................................................................................................... 14

3.4.2 Envelope ........................................................................................................... 15

3.4.3 Clean up ............................................................................................................ 15

3.5 Develope a Document ......................................................................................... 16

3.5.1 Develope .......................................................................................................... 16

3.5.2 Clean up ............................................................................................................ 16

4 SSF and ABAP ............................................................................................................ 17
4.1 Basics.........................................................................................................................................................17
  4.1.1 Passing Data ..........................................................................................................................................17
  4.1.2 Kernel or RFC ........................................................................................................................................18
  4.1.3 Signer and Recipient Information ...........................................................................................................18
  4.1.4 Choosing a Security Toolkit ...................................................................................................................19

4.2 Application’s default values ....................................................................................................................19

4.3 Sign a Document.......................................................................................................................................20

4.4 Verify a Signature .....................................................................................................................................22
2 Introduction

This document describes the basics of Secure Store & Forward (SSF) from the programmer’s point of view. It gives a brief overview on how to add security features to applications, especially to sign and encrypt arbitrary data, e.g. before storing or transmitting them.

It addresses both C and ABAP programmers.

Note that this document doesn’t cover all the details. These can be found in the technical documentation “Secure Store & Forward (SSF) – API Specifications”.

2.1 General Information

SSF (Secure Store & Forward) provides routines for both ANSI-C and ABAP that add authenticity and confidentiality to any given data or document. This is achieved by signing and enveloping the document using public-key technology.

In contrast to SNC (Secure Network Communications), SSF does not secure the communication but operates on a document basis. These documents can be stored or transmitted over an insecure network, then.

An overview and more information concerning SSF and security can be found in the technical documentation “Secure Store & Forward – Specifications”.

2.2 Outline

Chapter 2 explains the principles of SSF that are useful for both C and ABAP programming. You should read this chapter first.

Chapter 3 and 4 cover SSF programming from C and ABAP, respectively. These chapters are independent of each other, so you can skip chapter 3 if you are only interested in ABAP programming and vice-versa.
3 Using SSF

This chapter describes the principles of SSF, how you can use it, what is needed to work with it and what has to be done to use its functions.

3.1 Principles

The main purpose of SSF is to provide authenticity and confidentiality. This is realized by the function pairs SsfSign/SsfVerify and SsfEnvelope/SsfDevelope respectively. These functions can be combined to get both authenticity and confidentiality.

- If you want to assure that the document was created (or at least signed) by a specific person and hasn’t been altered by anyone (but the signer), use Sign and Verify.
- If you want to assure that the document can only be viewed by the valid recipients, use Envelope and Develope.
- If the data is confidential and you want to know and check its creator use both Sign and Envelope on the sender’s side and Develope and Verify on the receiver’s side.

This is true for both transmission and secure storing (where sender and receiver are probably identical).

3.1.1 Public keys

SSF uses public key technology for signing and enve loping. Each public key consists of a private and a public part.

- The private part (called private key) must be stored securely by the owner of the public key and is used for signing and developing.
- The public part (called public key) does not contain any secret information and thus can be freely distributed as long as authenticity is guaranteed (this is done by putting the key into a certificate where the key together with the owner’s name is signed by a trusted organization). It is used for verifying and enveloping.

However, one must realize that it is necessary

- to securely store the private key for signing and developing. Mainly, this is done by storing the data in either a PSE (personal security environment) or on a smart card. The (location of the) private key is specified in SSF using Signer and Recipient Information, see below.
- to have access to the public key to verify a signature and envelope a document. These are stored (locally) in a private address book (PAB) or in global address books.

3.1.2 Sign and Verify

These functions provide authenticity by adding a digital signature to the document that proves who has signed the document. This does not provide confidentiality in any way! Use Envelope for that reason.

---

1 So in this document public key is used for both the entire key and the public part thereof. When dealing with real objects always the public part is meant.
SsfSign This function signs any given input data using one’s own key (private part). The recipient can check the signature using SsfVerify.

SsfVerify This function verifies the signature generated with SsfSign and returns the original document (unless the signature has been created detached, see section 3.2.3 below). The recipient must know the sender’s public key to verify the signature. Using bIncCerts it is possible to append the certificate containing the sender’s public to the signed data (see section 3.2.2).

3.1.3 Envelope and Develope

This function pair is used to provide confidentiality. The data processed by SsfEnvelope can only be restored and read by the valid receiver(s) using SsfDevelope.

SsfEnvelope This function encrypts the given data so that it cannot be read by any third person. It can only be decrypted by the valid recipient(s) using SsfDevelope and the private part of their public key. To envelope (encrypt) the data the recipient’s public key must be known.

SsfDevelope This function decrypts the enveloped data and restores its original contents. This function can only be called by valid recipients of that document, as the private part of the recipient’s public key is required.

3.1.4 Encoding and Decoding

These two functions don’t provide any useful work (when viewed from the cryptographic side) but are needed to convert the document (which can be in any format) into a specific format so that it can be processed by the other functions.

Thus, you have to call SsfEncode as first step before calling any other function and SsfDecode as final function to get the “original” data.

SsfEncode Call this function once before you process the data with either SsfSign, SsfEnvelope or both.

SsfDecode Call this function after you have processed the data with either SsfVerify, SsfDevelope or both. This returns the data in the original format.

Note: Even if you use both Sign and Envelope you only have to encode the data once before calling the first function. Simply pass its result (which is already encoded) to the second function, then.

Note: There are no corresponding ABAP functions. In ABAP, this is always done implicitly using the flags B_INENC (input encoding) for SSF_SIGN and SSF_ENVELOPE and B_OUTDEC (output decoding) for SSF_VERIFY and SSF_DEVELOPE.

3.2 Common Parameters

The mentioned functions have most of their parameters in common. These are described here:

Format A text describing the format that is used for encoding the data. Currently only “PKCS7” is available.

HashAlg (SsfSign and SsfDigest only) A text that specifies the hash algorithm that should be used for signing, e.g. “MD5”.
SymEncr (SsfEnvelope only) A text that specifies the algorithm that should be used for encryption (enveloping), e.g. “DES–CBC”.

InputData This is the data that should be transformed by the function. Except for SsfEncode (or when auto-encoding is enabled) this must be encoded data.

Signatory/Recipient (List of) This structure (or list of structures) describes the “persons” (or better their public or private keys) who sign the document or for whom the document should be enveloped, see below.

Pab (Private Address Book, needed for SsfEnvelope and SsfVerify). A text together with its password which specifies the place where public keys are stored (for the current user).

<table>
<thead>
<tr>
<th></th>
<th>One’s own Private Key</th>
<th>Sender’s or Receiver’s (i.e. the other’s) Public Key (e.g. from one’s own Private Address Book)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SsfSign</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SsfVerify</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>SsfEnvelope</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>SsfDevelope</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Public Key Requirements for the SSF functions

3.2.1 Signer and Recipient Information

The composite data type SsfRcpSsfInformation is used by SSF to identify a “person” (i.e. its public or private key), to specify the place where the private key is located and for person specific result values.

- The person’s identity is given by strSigRcpId.
- strSigRcpProfile and its password strSigRcpPassword specify where the private key is located (for SsfSign and SsfDevelope). Only set these fields when specifying a private key; for a public key (e.g. with SsfEnvelope), these fields must be left empty.
- Having processed the data structure, all SSF functions set the field uResult. With this, it is possible to check the individual results for multiple signers or recipients.

Note: Most of the parameters (especially strSigRcpProfile) depend on the used security product. Use the values obtained from that security product.

Note: There’s a parameter strSigRcpIdNamespace that was used in previous versions of SSF to specify the type of profile. It isn’t used any more. Pass an empty string (for C: a NULL pointer).

3.2.2 Include Certificates

When signing a document you can specify whether the signer’s certificate should be stored together with the signature. With this, the verifier has the possibility to use this information to verify the digital signature. Otherwise, the public key must be made available to the verifier by other means (e.g. exchanged before or in a global address book).
Note: Using the included certificate is only possible, if the certificate is signed by a CA (Certification Authority) whose certificate is known by the verifier (or the CA’s certificate is signed by a known CA…). Thus, it is almost useless to include a self-signed certificate.

3.2.3 Detached Signatures

When signing a document you can choose whether the signature should include the data or should be detached, i.e. without the data.

The detached signature is shorter and can save a lot memory, especially for large documents. However, keep in mind:

- You need the original data to verify the signature. The data must be identical to the signed data.
- If the verification fails, there’s no way to determine what has been signed. Thus, when verifying data and the verification fails, you cannot say what part of the data has been modified. It is even possible that the signature has been modified.

3.3 Application Scenarios

This section shows typical applications of SSF. These examples are described abstractly only. More information on how they are implemented in C or ABAP can be found in the following chapters.

SSF can be used
- to securely store and retrieve data
- to securely send and receive data
- for digital signatures
- to authenticate a user

Note: The terms store/retrieve are used here to express that both actions are done by the same user. In contrast, send and receive will be used when the data is passed to another user. For our purpose, it doesn’t matter whether the data is transported to another location (as one might think of for the second terms).

Note: Securely is used for a combination of Confidentiality (if the data is enveloped).

<table>
<thead>
<tr>
<th>Activity</th>
<th>Sign/Verify</th>
<th>Envelope/Develope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td>S: one’s own private key</td>
<td>E: one’s own public key</td>
</tr>
<tr>
<td>Retrieve</td>
<td>V: one’s own public key</td>
<td>D: one’s own private key</td>
</tr>
<tr>
<td>Send</td>
<td>S: one’s own private key</td>
<td>E: recipient’s public key</td>
</tr>
<tr>
<td>Receive</td>
<td>V: recipient’s public key</td>
<td>D: one’s own private key</td>
</tr>
<tr>
<td>Digitally Sign</td>
<td>S: one’s own private key</td>
<td>-</td>
</tr>
<tr>
<td>Check Signature</td>
<td>V: signer’s public key</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2 Public Key Requirements for some Application Scenarios

The common scheme for these activities looks like this:

1. Encode the data (see section 3.1.4)
2. **Sign** the data using your own private key

3. **Envelope** the data with the recipient’s public key. (If the document should be readable by more persons there’s the possibility to specify multiple recipients.)

**Note:** Depending on your needs, perform either step 2, step 3 or both of them.

1. **Develope** the document using your own private key (reverses step 3 above)

2. Check the signature by calling `Verify` using the sender’s public key (corresponds to step 2 above)

3. **Decode** the data (see section 3.1.4)

**Note:** Depending on your needs, perform either step 1, step 2 or both of them.
4 SSF and C

This chapter describes how you can access the SSF functionality from within your C code. It shows what has to be done and gives some almost ready-to-use source code examples that can be included in your code.

4.1 Initialization

To use the SSF API it is necessary to load the security library. To simplify that task, ssfxxsup.c contains a routine that does most of the work for you.

Note: If you are using the SSF functions from with the R/3 kernel, have a look at ssfxxlib.h. This header file defines a wrapper with which you can use the kernel’s SSF functions. These functions start with SsfLib in contrast to Ssf.

4.1.1 Load the library

Include the header file ssfxxsup.h and define a SsfLib structure (in this example this is called mySsfLib):

```
#include "ssfxxsup.h"
SsfLib mySsfLib
```

To load the SSF library and initialize the structure call SsfSupInit:

```
int rc;
rc = SsfSupInit(mySsfLibName, &mySsfLib, NULL, NULL);
if ( rc != SSF_SUP_OK ) {
    printf("could not load SSF library %s .\n", mySsfLibName);
}
```

Note: mySsfLibName is the name of the library with fully qualified path.

If SsfSupInit returned SSF_SUP_OK you can use the SSF functions. To do that, use a syntax like

```
rc = (mySsfLib.fp_SsfSign) (...);
```

At the end of your program, free the allocated resources with a call to SsfSupEnd. It is save to call SsfSupEnd multiple times or if SsfSupInit failed.

```
SsfSupEnd(&mySsfLib);
```

Note: You must link your code with ssfxxsup.o (or library ssfsuplib$(LIBE) in newer releases). Additionally, you have to link with the following libraries: dlllib$(LIBE) dptrl2lib$(LIBE) $(SY)prtlib$(LIBE)

4.1.2 Define Variables

Now define some (global) variables that specify the format and kind of algorithms that should be used with SSF. These depend on the SSF library loaded and your needs. Although these values are
probably constant for the whole code, this is not necessarily true. There are circumstances where it is useful to use different values at different locations in your source code.

For the following examples, ‘default’ values that are valid throughout the entire source code are used.

```c
char myFormat = "PKCS7";
int myFormatL = 5;
char myHashAlg = "MD5";
int myHashAlgL = 3;
char mySymEncrAlg = "DES-CBC";
int mySymEncrAlgL = 7;
```

**Note:** In SSF, text parameters are always passed with their length as second parameter. Thus there’s no need to 0-terminate them, although this will be true (for practical reasons) in most situations. They are passed with their lengths as this is necessary for input and output data (the raw data can include the character 0x00).

### 4.2 Sign a Document

Signing a document (named `document`, with length `documentL`) consists of two steps:

1. First, create a list of who should sign the document. In most cases, this list will only contain one signer (you).
2. Now encode and sign the document

#### 4.2.1 List of Signers

This code fragment first creates a signer and builds, as second step, a one-element list containing that signer.

```c
/* build Signer */
if((rc = (mySsfLib.fp_SsfNEWSigRcpSsfInfo) (SignerName, SignerNameL, NULL, 0, /* reserved */ SignerProfile, SignerProfileL, SignerPassword, SignerPasswordL, SSF_API_UNDEFINED_RESULT, &Signer)) != SSF_AUX_OK)
    return FatalError(rc, "SsfNEWSigRcpSsfInfo() failed!");

/* build list containing that signer */
if(rc = (mySsfLib.fp_SsfINSSigRcpSsfInfo) (Signer, &SignerList) != SSF_AUX_OK)
    return FatalError(rc, "SsfINSSigRcpSsfInfo() failed!");
```
4.2.2 Encode and Sign

Now encode the input data (document, length documentL) to encoded_data and then sign it. As the encoded data isn’t needed after signing free it using mySsfLib.fp_SsfDELSsfOctetstring. Note that you have to free the signed data as well, after you have processed (e.g. stored, sent) it.

```c
char *encoded_data = NULL; /* stores the encoded data */
int encoded_dataL = 0;
char *signed_data = NULL; /* stores the signed data */
int signed_dataL = 0;

/* Encode before Sign */
rc = (mySsfLib.fp_SsfEncode) (myFormat, myFormatL, document, documentL, &encoded_data, &encoded_dataL);

if (rc == SSF_API_OK)
  rc = (mySsfLib.fp_SsfSign) (myFormat, myFormatL, myHashAlg, myHashAlgL, FALSE, /* don’t include the certificates */ FALSE, /* don’t create a detached signature */ encoded_data, encoded_dataL, SignerList, &signed_data, &signed_dataL);

if (encoded_data != NULL) /* free the encoded data */
  (mySsfLib.fp_SsfDELSsfOctetstring) (&encoded_data, &encoded_dataL);

Note: If Sign fails (i.e. it returns a value not equal to SSF_API_OK) you’ll find further error information in Signer->uResult. (When using multiple signers check the complete list)

4.2.3 Clean up

After processing of the signed document, free the allocated resources: the signer list (including the signer(s)), the signed and the encoded data (the last has already been freed before!):

```c
if (SignerList != NULL)
  (mySsfLib.fp_SsfDELSigRcpSsfInfoList) (&SignerList);
if (signed_data != NULL) /* free the signed data */
  (mySsfLib.fp_SsfDELSsfOctetstring) (&signed_data, &signed_dataL);
```

4.3 Verify a Signature

Having received a signed document (called signed data with a length of signed dataL) do the following steps to check the signature and get the signed document.
4.3.1 Verify

SigRcpSsfInformationList SignerList;

rc = (mySsfLib.fp_SsfVerify) (myFormat, myFormatL, TRUE, /* use included certificates */ signed_data, signed_dataL NULL, 0, /* no 'original' data to compare with */ PabName, PabNameL, PabPassword, PabPasswordL, &SignerList, &encoded_data, &encoded_dataL);

/* Decode after Verify */
if (rc == SSF_API_OK)
    rc = (mySsfLib.fp_SsfDecode) (myFormat, myFormatL, encoded_data, encoded_dataL, &document, &documentL);

if (encoded_data != NULL) /* free the encoded data */
    (mySsfLib.fp_SsfDELSsfOctetstring) (&encoded_data, &encoded_dataL);

If all is OK, i.e. rc has the value SSF_API_OK, you can check who has signed the document by viewing the SignerList. The document that has been signed correctly is available as document with a length of documentL.

Note: If you have a detached signature (see section 3.2.3), you must pass the data that has been signed (replacing the arguments NULL, 0). This data has to be encoded using SsfEncode.

4.3.2 Clean up

At the end, free the allocated resources with:

if (SignerList != NULL)
    (mySsfLib.fp_SsfDELSigRcpSsfInfoList) (&SignerList);
if (document != NULL) /* free the document */
    (mySsfLib.fp_SsfDELSsfOctetstring) (&document, &documentL);

4.4 Envelope a Document

To envelope a document, one first has to specify the recipients. Only those persons are able to develop the document. Note that the public keys of all recipients are needed to envelope the document.

4.4.1 Recipient List

In most cases, the document is only sent to one person as it is done in the following example. If you want to address multiple persons, simply repeat the steps shown below.
This code fragment is similar to that in the Sign section 4.2.1. However, please note that the fields for Profile and Password have been set to NULL, 0 as this information is neither needed nor available. Providing any other value will return an error when calling SsfEnvelope.

```c
SigRcpSsfInformation *Recipient = NULL;
SigRcpSsfInformationList RecipList = NULL;

/* build Recipient */
if((rc = (mySsfLib.fp_SsfNEWSigRcpSsfInfo) (RecipientName, RecipientNameL,
    NULL, 0, /* reserved */
    NULL, 0, /* no Profile */
    NULL, 0, /* no Password */
    SSF_API_UNDEFINED_RESULT,
    &Recipient)) != SSF_AUX_OK)
    return FatalError(rc, "SsfNEWSigRcpSsfInfo() failed!");

/* build list containing that recipient */
if(rc = (mySsfLib.fp_SsfINSSigRcpSsfInfo) (Recipient, &RecipList) != SSF_AUX_OK)
    return FatalError(rc, "SsfINSSigRcpSsfInfo() failed!");
```

### 4.4.2 Envelope

Now envelope the document for that recipient(s):

```c
/* Encode before Envelope */
rc = (mySsfLib.fp_SsfEncode) (myFormat, myFormatL,
    document, documentL,
    &encoded_data, &encoded_dataL);

if (rc == SSF_API_OK)
    rc = (mySsfLib.fp_SsfEnvelope) (myFormat, myFormatL,
    mySymEncrAlg, mySymEncrAlgL,
    encoded_data, encoded_dataL,
    PabName, PabNameL,
    PabPassword, PabPasswordL,
    RecipList,
    &enveloped_data, &enveloped_dataL);

if (encoded_data != NULL) /* free the encoded data */
    (mySsfLib.fp_SsfDELSsfOctetstring) (&encoded_data, &encoded_dataL);
```

### 4.4.3 Clean up

```c
if (RecipList != NULL)
    (mySsfLib.fp_SsfDELSigRcpSsfInfoList) (&RecipList);
if (enveloped_data != NULL) /* free the enveloped data */
```
4.5  Develope a Document

The following code fragment develops an enveloped document. This looks similar to the SsfSign example.

4.5.1  Develope

```
SigRcpSsfInformation *Recipient;
if((rc = (mySsfLib.fp_SsfNEWSigRcpSsfInfo) (RecipientName, RecipientNameL, NULL, 0, /* reserved */ RecipientProfile, RecipientProfileL, RecipientPassword, RecipientPasswordL, SSF_API_UNDEFINED_RESULT, &Recipient)) != SSF_AUX_OK)
    return FatalError(rc, "SsfNEWSigRcpSsfInfo() failed!");

/* Develope */
rc = (mySsfLib.fp_SsfDevelope) (myFormat, myFormatL, enveloped_data, enveloped_dataL, Recipient, &encoded_data, &encoded_dataL);

/* Decode after Develope */
if (rc == SSF_API_OK)
    rc = (mySsfLib.fp_SsfDecode) (myFormat, myFormatL, encoded_data, encoded_dataL, &document, &documentL);

if (encoded_data != NULL) /* free the encoded data */
    (mySsfLib.fp_SsfDELSsfOctetstring) (&encoded_data, &encoded_dataL);
```

If no error occurred, the original document is available as `document` with a size of `documentL`. Otherwise `Recipient->uResult` contains the appropriate error code.

4.5.2  Clean up

At the end, free the allocated resources with `SSfDelSigRcpSsfInfo`:

```
if (Recipient != NULL) /* free recipient */
    (mySsfLib.fp_SsfDELSigRcpSsfInfo) (&Recipient);
if (document != NULL) /* free the document */
    (mySsfLib.fp_SsfDELSsfOctetstring) (&document, &documentL);
```
5 SSF and ABAP

This chapter shows how you can use the SSF functionality from ABAP code. First, the ABAP specific basics are outlined. Then, the main procedures are described in detail.

5.1 Basics

The SSF functionality can be accessed from ABAP using the function group SSFG. You can call the security functions using a CALL FUNCTION statement.

5.1.1 Passing Data

There are two ways to pass the data that should be processed to the ABAP functions:

<table>
<thead>
<tr>
<th>ITab</th>
<th>The data is passed via an internal table (itab) and its length.</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The data is passed via a file, you only have to specify the filename.</td>
</tr>
</tbody>
</table>

You choose the method with the parameter IO_SPEC: The value ‘T’ says that the input argument is an internal table. When passing the value ‘F’ this internal table contains the filename of the data file.

**Note:** For security reasons the possibility to pass the data via a file has been disabled. Therefore, you currently must use internal tables. If you think that your application needs the file support, please contact us.

When passing an internal table please note the following:

- The structure of the internal tables is currently limited: You only can pass tables with similar columns (of TYPC). Don’t pass integer values, as their representation is system dependent! See SAP Documentation “Remote Communications” page 10-35 for more information.
- The size defines the number of bytes of the internal table. This is required since the last line of the table might not completely be filled with useful data. If the given size does not match the length of the internal table, an error is returned.
- The structure of the internal tables for Sign and the matching Verify (or Envelope and Develope) should be equal.
- Encoded data (as returned by Sign or Envelope) is passed and returned as an internal table with the structure SSFBIN.

When passing data via files please note the following:

- The filenames are passed via the internal tables. Set their sizes correctly.
- The files are loaded and saved by the security library. Thus, the file path depends on the location and home directory of the library. See the next section 5.1.2.
- The filename’s suffix is changed depending on the operation performed:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign</td>
<td>The suffix .sig is added.</td>
</tr>
<tr>
<td>Verify</td>
<td>If the signed file has the suffix .sig the suffix is removed, otherwise the...</td>
</tr>
<tr>
<td>Envelope</td>
<td>The suffix .env is added.</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Develope</td>
<td>If the enveloped file has the suffix .env the suffix is removed, otherwise the suffix .dev is added.</td>
</tr>
<tr>
<td>AddSign</td>
<td>The filename is not changed.</td>
</tr>
</tbody>
</table>

| Table 3 Suffix Conversion Rules |

**Note:** The ABAP routines provide an automatic encoding and decoding that can be activated using B_INENC and B_OUTDEC, see section 3.1.4. This should be activated when handling with normal data.

### 5.1.2 Kernel or RFC

Most of the functions are available in two forms: With the prefix SSF_ and with the prefix SSF_KRN. This specifies the way and the place how and where the functions are processed:

**SSF_**
The functions are started using an RFC (Remote Function Call). You have to specify the location (RFC destination), where the function should be executed. You can use SAP_SSFATGUI to call the function at the user’s front end. This is necessary, if a user wants to sign or decrypt a message using his locally installed smart card or his local PSE.

**SSF_KRN_**
These functions are carried out in the R/3 kernel (i.e. by the application server). This avoids communication and is thus faster. You should prefer this method if no external keys are necessary (e.g. when verifying a signature and the necessary information is located on the application server).

Although this specifies the method how the functions are called, you can use the following rule to determine which routine to use: If the action (Sign, Develope) has to be done by the user, use the first function set. Otherwise, you should prefer the latter.

**Note:** When using the RFC version of a cryptographic function, an RFC server (called ssfrfc) is started on the client side that handles that function. Thus, the server program and the external SSF API library must be installed properly.

**Note:** The SSF_KRN-functions have been introduced in release 4.5. For release 4.0A, please use the SSF-functions with destination ‘SAP_SSFATAS’.

### 5.1.3 Signer and Recipient Information

The signer and recipient are specified by the parameter SIGNER and RECIPIENT respectively. These parameters are of type SSFINFO:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>SSFID</td>
<td>CHAR 255</td>
</tr>
<tr>
<td>NAMESPACE</td>
<td>SSFNS</td>
<td>CHAR 10</td>
</tr>
<tr>
<td>PROFILE</td>
<td>SSFPF</td>
<td>CHAR 132</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>SSFPW</td>
<td>CHAR 132</td>
</tr>
</tbody>
</table>
This structure corresponds to the fields described in section 3.2.1 Signer and Recipient Information.

**Note:** Instead of specifying the information directly, you can (and should) use the functions with the suffix `_BY_USER` and `_BY_AS` to choose a signer or recipient by (R/3 user) name or to specify the application server, see next sections.

**Note:** `NAMESPACE` is obsolete. Please set it to initial.

### 5.1.3.1 _BY_USER

When using `SSF_SIGN`, `SSF_DEVELOPE` and `SSF_ADDSIGN` there’s the possibility to specify the signer or recipient in form of an R/3 user: Use the function names with an added `_BY_USER`.

The corresponding information (including name, profile and RFC destination) is stored in table `ADR11` and can be maintained using the general address maintenance (e.g. via `SU01`). (In release 4.0 and 4.5 it was in stored table `TC70` and was maintained with transaction `007C`.)

Whenever possible, you should use these functions. They are easier to call (less arguments) and you don’t need to handle (and store) the signer or recipient information. Additionally they (optionally) automatically ask the user for his password.

**Note:** `_BY_USER` is only available with the RFC functions as the users have to sign locally (i.e. at their front end).

### 5.1.3.2 _BY_AS

Similar to _BY_USER there’s a function called `SSF_KRN_SIGN_BY_AS` that signs the data via a kernel call at the application server. The signer is the application server itself. (Note that every R/3 system has its own public key.)

### 5.1.4 Choosing a Security Toolkit

Starting with release 4.5B, there’s the possibility to install multiple security toolkits at the application server (this is useful if you install an external security toolkit while other applications are using SAP’s security toolkit `SAPSECULIB`).

For that purpose, all the `SSF_KRN_` functions have an additional, optional parameter `SSFTOOLKIT` to choose the security toolkit; see next chapter for information how to obtain the toolkit’s name. If this parameter is omitted or set to `SPACE`, the default security toolkit is used.

### 5.2 Application’s default values

For the common SSF parameters, we provide a table (`SSFARGS`) to store these application specific SSF parameters. They can be customized with transaction `SSFA` and retrieved with function `SSF_GET_PARAMETER`.

You can add new applications to the table `SSFAPPLIC` (using transaction `SE16`) and specify which parameters are used by your application (only those fields are shown in transaction `SSFA`)

**Note:** This function is available as of release 4.5B.

We recommend that every application uses this function to obtain the required values. This way, it is possible to use multiple security toolkits (see previous section) and multiple profiles and private address books.
DATA: l_SSFTOOLKIT LIKE SSFPARMS-SSFTOOLKIT,
    l_SSFFORMAT LIKE SSFPARMS-SSFFORMAT.

CALL FUNCTION 'SSF_GET_PARAMETER'
    EXPORTING
        APPLICATION = 'TEST'
    IMPORTING
        SSFTOOLKIT  =  l_SSFTOOLKIT
        STR_FORMAT  =  l_SSFFORMAT
        *          STR_HASHALG =
        *          STR_ENCRALG =
        *          B_INCCERTS =
        *          B_DETACHED =
    EXCEPTIONS
        ssf_parameter_not_found = 1
        OTHERS                  = 2.

5.3 Sign a Document

Signing data is rather easy as long as you have the following information:
1. An internal table containing the data that should be signed.
2. The information specifying the signer.

Although the first part sounds very easy, please consider the notes given in section 5.1.1 above: You can’t use every internal table. Moreover, in some cases it might be necessary to convert the data stored in a database table or shown in a list into an internal table.

Please keep the following in mind when creating the internal table:
- If a user signs the data, he should know what he is signing. Thus, you should exactly show him the information that will be signed. (When using SSF_SIGN_BY_USER there’s the possibility to automatically show the information and ask for the password.)
- When storing the signature detached, i.e. without the signed data (see section 3.2.3), you only can verify the signature if the original document is available (e.g. it is stored in the database). In this case, you must assure that the internal table can be recreated.

To specify the signer you must provide either a signer itab as described in section 5.1.3 or use one of the functions SSF_SIGN_BY_USER or SSF_KRN_SIGN_BY_AS. As mentioned earlier the latter should be preferred when possible.

In the following example, we use SSF_SIGN_BY_USER to let a user sign an internal table. If you want the application server to sign the data, use SSF_KRN_SIGN_BY_AS. The necessary modifications to the example code are noted below.

Suppose that you already have an internal table INPUTDATA containing the data that the current user should sign. The length (in bytes) of the data is DATALEN and can be calculated by the following code fragment, if you want to sign the entire table:

* calculate total length of itab INPUTDATA
DATA: DATALEN LIKE SSFPARMS-INDATALEN,
     DATALEN2 TYPE I.
DESCRIBE TABLE INPUTDATA LINES DATALEN.
DESCRIBE FIELD INPUTDATA LENGTH DATALEN2.
DATALEN = DATALEN * DATALEN2.

Define variables SIGNEDDATA and SIGNEDDATALEN for the signed data and CRC and RESULT to store the return values from the sign function.

DATA: SIGNEDDATA LIKE SSFBIN OCCURS 0 WITH HEADER LINE,
     SIGNEDDATALEN LIKE SSFPARMS-OUTDATALEN.
DATA: CRC LIKE SSFPARMS-SSFCRC,
     RESULT LIKE SSFINFO-RESULT.

Finally, call the function. The optional parameters and the individual exceptions have been omitted to show the basics. (To be correct, SIGNER is an optional parameter, too, with the default SY-UNAME. Nevertheless, this line is shown to emphasize that the signature should be done by the current user – or, to be more precise, with the private key owned by that user).

CALL FUNCTION 'SSF_SIGN_BY_USER'
  EXPORTING
      SIGNER                       = SY-UNAME
      OSTR_INPUT_DATA_L           = DATALEN
  IMPORTING
      OSTR_SIGNED_DATA_L          = SIGNEDDATALEN
      CRC                         = CRC
      RESULT                      = RESULT
  TABLES
      OSTR_INPUT_DATA             = INPUTDATA
      OSTR_SIGNED_DATA            = SIGNEDDATA
  EXCEPTIONS
      OTHERS                      = 11.
IF SY-SUBRC = 0.
  IF CRC = 0 AND RESULT = 0.
    * signing was successful
  ENDIF.
ENDIF.

Notes:
- As described in section 5.1.2, the signer information is resolved with table ADR11 (table T07C in release 4.0 and 4.5). Therefore, an entry for the current user is necessary. The signature is created via RFC at the destination specified there. In most situations, this will be on the user’s computer (via SAP_SSFATGUI).
- The user is automatically prompted for his or her password. You can disable this by exporting the parameter PASSWORD or setting ASK_PWD = '. (The second parameter is useful if you want to pass an empty password.)
• As the source table is not encoded (cf. section 3.1.4), the default value for B_INENC isn’t changed.

• Depending on your application, you can change the default values for STR_FORMAT (default: PKCS7), B_INC_CERTS (default: no) and B_DETACHED (default: no), see section 3.2 and its sub-sections.

• You must check (see online manual for a complete description of all error codes)
  • that the function did not throw an exception (exceptions are thrown if a severe problem in conjunction with RFC occurred),
  • that CRC = 0 (any other value shows that the signing failed – the value 5 indicates that there was a problem with the signer information and that RESULT contains a more precise error information) and
  • that RESULT = 0.

• If you want to sign the data by the application server, modify the source code as follows:
  • Change the function name to SSF_KRN_SIGN_BY_AS
  • Remove the exported parameter SIGNER
  • Rename the imported parameter RESULT to SRRC (For historical reasons, they have the same purpose but different names!)

5.4 Verify a Signature

To verify the signature you need the following data (besides the signature itself)
1. The original data (i.e. the data that was signed), if the signature is detached. Otherwise (i.e. it’s no detached signature), the data is returned if the verification succeeded.
2. The signer’s public key

The problems with detached signatures have already been discussed in the previous section and in sections 3.2.3.

This code fragment verifies signed_data_tab with length signed_data_l. The output data is stored in output_data_tab with length output_data_l; signer information is available in signer_info_tab.

DATA:  l_SSFTOOLKIT LIKE SSFPARMS-SSFTOOLKIT,
       l_SSFFORMAT LIKE SSFPARMS-SSFFORMAT.
       l_pab        TYPE ssfparms-pab,
       l_pabpw      TYPE ssfparms-pabpw,
       signer_info_tab   TYPE ssfinfo OCCURS 0.

CALL FUNCTION 'SSF_GET_PARAMETER'
  EXPORTING
  APPLICATION = 'TEST'
  IMPORTING
  SSFTOOLKIT   = l_SSFTOOLKIT
  STR_FORMAT   = l_SSFFORMAT
str_pab = l_pab
str_pab_password = l_pabpw

EXCEPTIONS
ssi_parameter_not_found = 1
OTHERS = 2.

CALL FUNCTION 'SSF_KRN_VERIFY'
EXPORTING
SSFTOOLKIT = l_SSFTOOLKIT
STR_FORMAT = l_SSFFORMAT
b_inc_certs = 'X'
* b_inenc = 'X'
B_OUTDEC = 'X'
* IO_SPEC = 'T'
ostr_signed_data_l = signed_data_l
* ostr_input_data_l = "detached only"
str_pab = l_pab
str_pab_password = l_pabpw

IMPORTING
OSTR_OUTPUT_DATA_L = output_data_l
CRC = l_crc

TABLES
ostr_signed_data = signed_data_tab
* ostr_input_data = "detached only"
signer_result_list = signer_info_tab
ostr_output_data = output_data_tab

EXCEPTIONS
ssf_krn_error = 1
ssf_krn_noop = 2
ssf_krn_nomemory = 3
ssf_krn_opinv = 4
ssf_krn_nossflib = 5
ssf_krn_input_data_error = 6
ssf_krn_invalid_par = 7
ssf_krn_invalid_parlen = 8
ssf_fb_input_parameter_error = 9
OTHERS = 10.

Notes:

- The signed data isn’t detached, so input_data is omitted.
- You must check (see online manual for a complete description of all error codes)
  - that the function did not throw an exception (exceptions are thrown if a severe problem occurred),
  - that CRC = 0. Any other value shows that the verification failed – the value 5 indicates that there was a problem with a signature; signer_info_tab-result is a detailed return code for that each signer.

5.5 Envelope a Document
Note: LIBSASPECU, the SSF library that is shipped with every R/3 system does not support enveloping and developing. An external security product must be used for that purpose.

5.6 Develope a Document

5.7 Authenticate a User

We plan to add a SSF function that does user authentication. If you need this functionality, please contact us.