Specification

SmartTrust WIB™ Plug-in Specification for Application Developers

Document number: 60077036

Revision: C. 2003-12-02.

© 2003 SmartTrust AB. All rights reserved.

SmartTrust endeavors to ensure that the information in this document is correct and fairly stated, but does not accept liability for any error or omission. The development of SmartTrust products and services is continuous and published information may not be up to date. It is important to check the current position with SmartTrust. This document is not part of a contract or license save insofar as may be expressly agreed.

Unless otherwise noted, all names of companies, products, street addresses and persons contained herein are part of a completely fictitious scenario and are designed solely to document the use of the described product or service.

SmartTrust and SmartTrust WIB are trademarks of SmartTrust AB.

All the other trademarks are the property of their respective owners.

Document number: 60077036
Revision: C. 2003-12-02.

Contents

Document number: 60077036
Revision: C. 2003-12-02.

1 Document profile

1.1 Purpose of the document

This document specifies the interfaces and functionality of the standard SmartTrust WIB™ plug-ins. The documentation directs towards WIG application development with examples given in WIG WML, [WIGWML]. The content of this documentation requires knowledge about the SmartTrust Delivery Platform and WIB.

1.2 Target audience

WIG application developers.

1.3 Terms, acronyms and abbreviations

Term	Definition
DF	Dedicated File
EF	Elementary File
GSM	Global System for Mobile communications
IV	Initialization Vector
LSB	Least Significant Byte
MAC	Message Authentication Code
MSB	Most Significant Byte
PIN1	Personal Identification Number 1 (CHV1 on SIM)
PKCS#7	Public-Key Cryptography Standard, specification number 7
RSA	Rivest-Shamir-Adleman, asymmetric encryption algorithm
SAT	SIM Application Toolkit
TAR	Toolkit Application Reference, part of GSM 03.48 specification
TLV	Tag-Length-Value, coding scheme
WIB	SmartTrust WIB™
WIG	Wireless Internet Gateway
WML	Wireless Markup Language

Document number: 60077036
Revision: C. 2003-12-02.

1.4 Symbols

Symbol	Description
K1, K2, K, K'	DES keys.
X || Y	Concatenation of byte-strings X and Y (in that order).
SHA1	SHA-1 hash function. See [SHA1] for further reference.
ISO_9797_ALG3	ISO9797 MAC algorithm 3. See [ISO9797] section 7.3 for further reference.
ISO_9797_PAD2	ISO9797 padding method 2. See [ISO9797] section 6.1.2 for further reference.
PKCS5_PAD	PKCS#5 padding function. See [PKCS5] section 6.1.1.
PKCS5_UNPAD	Inverse of PKCS5_PAD. See [PKCS5] section 6.1.1.
TDEA_ENCR	Triple DES encryption algorithm. See Appendix A.1 for details regarding the algorithm.
TDEA_DECR	Triple DES decryption algorithm. See Appendix A.2 for details regarding the algorithm.
<i..j>	Sub-string extraction operator. Extracts bytes i through j. $1 \le i \le j$.

1.5 References

Ref.	Title
[WIGWML]	SmartTrust. Specification – WIG WML Version 4. Doc.no. 50316002.
[WMLCLIB]	WAP WMLScript Crypto library,Version 05-Nov-1999, with specification changes as of 27-Dec-2000. Wireless Application Forum
[GSM03.38]	ETSI. GSM 03.38. Alphabets and language specific information. Version 7.2.0. Release 1998.
[GSM11.14]	ETSI. GSM 11.14. Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface. Version 8.7.0. Release 1999.
[PKCS1]	"PKCS #1 v2.0: RSA Cryptography Standard", RSA Laboratories
[PKCS5]	RSA Laboratories , "PKCS #5 v2.0: Password-Based Cryptography Standard", http://www.rsalabs.com/pkcs/
[PKCS7]	"PKCS #7 v1.5: Cryptographic Message Syntax", RSA Laboratories
[CMS]	"Cryptographic Message Syntax", RFC 2630, R. Housley
[WTLS]	Wireless Application Forum , "Wireless Transport Layer Security", http://www.wapforum.org/

Ref.	Title
[DEA]	ISO 8731-1, "Banking – Approved algorithms for message authentication – Part 1: DEA"
[ISO9797]	ISO/IEC 9797-1:1999(E) – Information technology – Security techniques – Message Authentication Codes (MACs)
[SHA1]	FIPS PUB 180-1, "Secure Hash Standard (SHS)"
[MODES]	ISO/IEC 10116 – Security Techniques – Modes of Operation for an n-bit Block Cipher Algorithm"

1.6 Revision history

Rev.	Comments
A	First release.
B	Changed the title and document number for [WIGWML] in reference list.
C	Updated for Delivery Platform version 6.1. Changed syntax in WIG WML examples.

Document number: 60077036
Revision: C. 2003-12-02.

2 Introduction

The plug-ins included in this specification are specified by SmartTrust. The plug-ins are implemented by SIM vendors on a WIB card. Due to space limitations on the SIM card the Card Issuer (e.g. Mobile Operator) often selects only the plug-ins for their needs to be included on a specific SIM card. It is also open for anyone to specify and implement proprietary plug-ins on SIM cards. Therefore it is necessary to find out the current availability of plug-ins on a specific target SIM before building applications.

The WIG WML used in the examples is supported by SmartTrust Delivery Platform version 6.1.

2.1 What is a WIB™ plug-In?

A WIB plug-in is a SIM located component providing additional features to the SmartTrust WIB™. Such features may be e.g. cryptographic functionality or file and data management. Plug-ins may be used in web applications and be called from within WIG WML document located on Internet Web servers.

The general syntax for calling a plug-in from within a WIG WML document is according to [WIGWML][1]:

```
<plugin name="NAME" destvar="DESTVAR" params="PARAMS" class="CLASS"/>
```

The arguments have the following meaning[2]:

Name	Value	Explanation	Var
`class` optional	`SMS-DEFAULT` | `UCS2` | `binary`	The WIB encoding of the plug-in output. Default is binary.	No
`destvar` mandatory	variable-name	Name of a variable that will contain the output data from the plug-in.	No
`name` mandatory	string	The name of the plug-in to call.	No
`params` mandatory	string	The input parameters to the plug-in.	Yes

[1] For backward compatibility, earlier versions of the plug-in calling syntax is supported, but the use of the syntax specified in this document is encouraged.
[2] For further details, refer to [WIGWML].

Document number: 60077036
Revision: C. 2003-12-02.

Each plug-in also has a specific interface shared between the plug-in and the application using the plug-in. The specific interface is the interface provided by the content of the params and destvar arguments found in the generic interface. The specific interface is transparent to the WIG and therefore it is the responsibility of the application to comply with.

2.2 Organization of this document

The plug-ins are categorized into the following functional areas:

- SIM card management: Plug-ins enabling a server with OTA management via WIB for updating subscriber information in a SIM card.
- Security: Plug-ins providing specific security functions, such as encryption and decryption of the submitted information, to WIB.

Each plug-in is described using the following template:

- Name – the name of the plug-in used when calling the plug-in
- Description – states the main functionality of the plug-in in question
- Input Parameters – plug-in specific input parameters in calling order. Parameters identified with description and length of bytes. In case of dynamic variable length, a notation *N* is used. Input parameters are passed to the plug-in through the params argument.
- Output Parameters – plug-in specific output parameters. The output parameters are returned by the plug-in in the variable named by the destvar argument.
- WIG WML Example – calling syntax for the plug-in with explanations.

Document number: 60077036
Revision: C. 2003-12-02.

3 SIM Card Management Plug-Ins

This section describes the management plug-ins for WIB. These are:

- Display User Data, DUDA
- Encrypted User Data, EUDA
- Retrieve Terminal Profile, RTPROF
- Remote File Management, RRFMS
- Change PIN, CP
- Reset PIN, RP
- Event Manager, EM

Document number: 60077036
Revision: C. 2003-12-02.

3.1 DUDA - Display User Data

Name

DUDA

Description

The *Display User Data* plug-in is called from WIG WML to let the user display and/or update the value of user specific data.

When the plug-in is called, the requested data on the SIM is displayed to the user together with a descriptive text. The user will then have the opportunity to update the value through the normal editing functions in the terminal.

The displaying of the user data is protected with a PIN1 that needs to be entered before the data is shown. The PIN1 will be blocked in case it has been entered incorrectly too many times. If the PIN1 is disabled the WIB execution is aborted.

User data is contained within user data objects stored in a file on the SIM. An object contains a tag, length, data coding scheme, value and descriptive text. The object and its usage is defined by the mobile operator and the user may only change the value part through the DUDA plug-in.

The data managed by the DUDA plug-in may be retrieved by an application by means of the EUDA plug-in.

Input Parameters

Parameter	Description	Length
`User Data Object Tag`	This field identifies the user data to be requested. The mobile operator defines the content of the objects on SIM and their usage. Object Tags may take the values within the interval: **'0x01'** to **'0xFE'**.	1

Output Parameters

None.

Document number: 60077036
Revision: C. 2003-12-02.

WIG WML Example

In this example, the DUDA plug-in is called to display/update the Credit Card Number. The user data object tag for the Credit Card Number is here '01'. The name of the plug-in is written with capital letters. The output variable “DUMMY” is just a dummy value and is not used.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- set the input parameters -->
    <setvar name="IN" value="\x01"/>
    <!-- call the plug-in -->
    <plugin name="DUDA" params="$(IN)" destvar="DUMMY"/>
  </p>
  </card>
</wml>
```

3.2 EUDA - Encrypted User Data

Name

EUDA

Description

The Encrypted User Data plug-in is called to fetch Encrypted User Data from the SIM.

The plug-in fetches user data objects from the user data object file. To secure the data for transportation, the list of objects is padded and encrypted before it is stored in the output variable. The outcome of the plug-in is always an encrypted user data object value string. The maximum length for the output of the plug-in is 255 bytes. The input parameters include a key id to be used for fetching an encryption key.

User data objects are objects stored in a file on the SIM. An object contains a tag, length, data coding scheme, value and descriptive text. The object and its usage is defined by the mobile operator and its data may be retrieved with the EUDA plug-in.

Input Parameters

Parameter	Description	Length
`Encryptionkey`	The key index to be used. Specified by **\xHH** syntax, where HH shall be replaced by the desired hexadecimal value.	1
`User Data Object Tags`	This field is a list (byte sequence) of tags for the user data objects to be requested. The mobile operator defines the values of the object tags and their usage. Object Tags may take the values within the interval: **'0x01'** to **'0xFE'**.	N

Output Parameters

The output data is an encrypted string containing a list of requested User Data Object values. The data objects are stored in the same order as they are defined in the input parameter of the plug-in call. The encrypted output is formatted according to following table:

Document number: 60077036
Revision: C. 2003-12-02.

Parameter	Description	Length
Padding Information	The padding information gives the number of bytes added as padding to the end of the plaintext before encryption. The following values are valid: **'0x00' – '0x07'**	1
Ciphertext	This is the encrypted plaintext. It is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

The decrypted ciphertext contains the list of user data objects and is formatted as:

Parameter	Description	Length
Length	The length of the following list (including padding bytes).	1
User Data Objects	This field is a list (byte sequence) of user data objects.	N
Padding Bytes	This is the bytes used for padding the plaintext before encryption. All bytes have the value '0x00'.	N

Every each user data object is formatted according to:

Parameter	Description	Length
User Data Object Tag	This field identifies the user data. The mobile operator defines the objects tags and their usage. Object Tags may take the values within the interval: **'0x01'** to **'0xFE'**	1
Length	The length of the following fields.	1
Character encoding scheme	Character encoding scheme used in the User Data field. It shall contain one of the following values: **"0x00"** = SMS default alphabet, 7-bit unpacked **"0x08"** = UCS2	1
User Data	User specific data.	N

Document number: 60077036
Revision: C. 2003-12-02.

If the length of the list of objects is longer than the maximum output length allowed, only the first objects that can be fully added into the output list are included. The succeeding objects are ignored.

WIG WML Example

In this example, the EUDA plug-in is called to retrieve two User Data Object values: “Credit card number” and “Home address”. The user data object tag for the Credit Card Number is '01' and for the Home Address '03'.

The input value is stored into the variable “IN”. By a call to the EUDA plug-in, the data is fetched, formatted to output format, encrypted with the 3DES key with index 2 (indicated by using “\x02” first in the params argument), and then stored in the variable CCD. The output of the plug-in is then sent to the Content Provider.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- set the input parameters -->
    <setvar name="IN" value="\x01\x03"/>
    <!-- call the plug-in -->
    <plugin name="EUDA" params="\x02$(IN)" destvar="CCD"/>
    <go href="http://www.shop.com?CREDITCARDDATA=$(CCD)"/>
  </p>
  </card>
</wml>
```

3.3 RTPROF - Retrieve Terminal Profile

Name

RTPROF

Description

The terminal profile indicates the SIM Toolkit capabilities of the ME. It is downloaded to the SIM as part of the SIM initialization procedure if the terminal supports SIM Toolkit.

The terminal profile could be very useful for the WIG WML application to for example customize the user dialog. The Retrieve Terminal Profile plug-in enables the applications to retrieve this information from the SIM.

Input Parameters

None.

Output Parameters

Parameter	Description	Length
`Terminal Profile`	This field holds the profile as a sequence of bytes where each bit indicates the presence of a certain SIM Application Toolkit facility. The Terminal Profile is detailed in [GSM 11.14].	N

WIG WML Example

This example illustrates the use of the RTPROF plug-in. The plug-in is used to fill the Profile variable that is then posted to the host. The field for input value is left blank, as there are no input parameters for this plug-in.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- call the plug-in to retrieve the terminal profile-->
    <plugin name="RTPROF" params="" destvar="Profile"/>
    <!-- send the profile to the host -->
    <go href="http://www.someapplication.com/set?Prof=$(Profile)"/>
  </p>
  </card>
</wml>
```

3.4 RRFMS - Retrieve Remote File Management Status

Name

RRFMS

Description

The *Retrieve Remote File Management Status* plug-in is used for retrieving content of a special dedicated file on the SIM. The information can be used to report to the user, the outcome of the GSM 03.48 Remote File Operation.

The plug-in gets three bytes as an input value and according to those seeks the file EF 6F09 in the directory DF 2900 and reads the required information and returns it in the output variable.

Input Parameters

Parameter	Description	Length
`Offset high`	This field identifies the high-order byte of offset. Legal values are: **'0x00'** to **'0xFF'**	1
`Offset low`	This field identifies the low-order byte of offset. Legal values are: **'0x00'** to **'0xFF'**	1
`Length`	This field sets the length of data to be retrieved	1

The following content and their plug-in input values are defined:

Retrieve status code:
Offset high:'0x00'
Offset low: '0x00'
Length: '0x02'

Retrieve status description:
Offset high:'0x00'
Offset low: '0x02'
Length: '0x29'

Document number: 60077036
Revision: C. 2003-12-02.

Output Parameters

Parameter	Description	Length
Content	This field give the content requested from file EF 6F09.	N

When using the input values defined for retrieving the status code or status description the output will be 1 byte or 40 bytes respectively. The maximum size of the status description, i.e. the space that is reserved for it in EF (6F09), is 40 bytes.

WIG WML Example

This is an example of an application retrieving the status description after having performed an OTA activity. The description is stored in the variable OUT. Here the offset is '0002h', and the number of bytes to fetch is '29h' (41 bytes). The plug-in will then read a length-value pair, starting at the third byte in EF (6F09). It will store the value part in the variable OUT.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- set the input parameters to read -->
    <!-- the third byte and the following 40 bytes -->
    <setvar name="IN" value="\x00\x02\x29"/>
    <!-- call the plug-in -->
    <plugin name="RRFMS" params="$(IN)" destvar="OUT"/>
   <!-- Show the status to the user -->
   $(OUT)
  </p>
  </card>
</wml>
```

3.5 CP – Change PIN

Name

CP

Description

The *Change PIN* plug-in is used for requesting change of a PIN. The new PIN value is specified by the user through the ME keypad. The user is requested to enter the new PIN twice.

Input Parameters

Parameter	Description	Length
`ID Type`	This field identifies the type of the ID supplied in the next parameter. Legal values are: **"U"** – Asymmetric key usage **"S"** – Symmetric key ID Type of the parameter is SMS default character.	1
`Private object ID`	This field identifies the ID of the private object who's PIN shall be changed. Legal values for ID type = "U" are: **"N"** – 'non repudiation' key. **"S"** – 'sign' key **"D"** – 'decrypt' key **"U"** – 'unwrap' key Legal values for ID type = "S" are: **'0x01'** to **'0xFE'** – key ID of the selected symmetric key.	1

Output Parameters

None.

Document number: 60077036
Revision: C. 2003-12-02.

WIG WML Example

This is an example of an application requesting the user to change a PIN. The *ID Type* is set to ‘Asymmetric key usage’ and *Private ID object* is a ‘sign’ key. The prompts are held on the SIM card and used by the plug-in.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- set the input parameters -->
    <!-- use value="US" or value="\x55\x53" -->
    <setvar name="IN" value="US"/>
    <!-- call the plug-in to request change pin -->
    <plugin name="CP" params="$(IN)" destvar="OUT1"/>
  </p>
  </card>
</wml>
```

Document number: 60077036
Revision: C. 2003-12-02.

3.6 RP – Reset PIN

Name

RP

Description

This plug in offers means whereby a trusted party may reset a PIN in the SIM over-the-air and set a new value.

Input Parameters

Parameter	Description	Length
`ID Type`	This field identifies the type of the ID supplied in the next parameter. Legal values are: **"U"** – Asymmetric key usage **"S"** – Symmetric key ID Type of the parameter is SMS default character.	1
`Private object ID`	This field identifies the ID of the private object who's PIN shall be changed. Legal values for ID type = "U" are: **"N"** – 'non repudiation' key. **"S"** – 'sign' key **"D"** – 'decrypt' key **"U"** – 'unwrap' key Legal values for ID type = "S" are: **'0x01'** to **'0xFE'** – key ID of the selected symmetric key.	1
`Encrypted PIN block header`	Header specifying the algorithm used for wrapping. The following values are legal: **'0x01'** = 3DES + SHA-1 MDC **'0x02'** = 3DES + ISO9797 MAC	1
`Encrypted PIN block payload`	PIN value, encrypted and integrity protected. Detailed format according to section below. The algorithm specified in the header (the first byte) of this field determines the length.	N

Output Parameters

None.

Encrypted PIN block payload calculation

Creating the Encrypted PIN Block data to be sent to the Plug-in includes formatting, integrity protection and encryption of data. The procedure for the two algorithms is described below.

Algorithm 1 is the preferred algorithm and may be the only one supported by the SIM unless specific reasons exist, e.g. like non-existing SHA-1 support on a SIM card.

Algorithm 1: 3DES with SHA-1 MDC

1 Create nonce and PIN value.

2 Let *PB* be the PIN Block and formatted according to:

Bytes	Description	M/O	Length
1 – 8	Nonce. 8 bytes of random data.	M	8
9 – 16	PIN value. Each digit in the PIN shall be encoded with it's corresponding GSM default alphabet value. All unused digits at the end shall be encoded as '0xFF'.	M	8
17 – 36	SHA-1 MDC	M	20

Table 1: PIN Block for algorithm 1

3 Calculate a message digest $MD = SHA1(\text{'0x01'} \,||\, PB\langle 1..16\rangle)$.

4 Encrypt the PIN Block data $EPB = TDEA\_ENCR(PB)$ using the following cipher parameterization:

Keys K_1, K_2

Mode Triple encryption in outer CBC mode using two keys in DED operation. See Appendix A for more details.

IV 0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).

5 *EPB* is the Encrypted PIN Block data to be sent to the plug-in.

Note: Due to the DES block-size, the encrypted PIN payload will be 40 bytes (5 DES blocks).

Algorithm 2: 3DES with ISO9797 MAC

1 Create nonce and PIN value.

2 Let *PB* be the PIN Block and formatted according to:

Bytes	Description	M/O	Length
1 – 8	Nonce. 8 bytes of random data.	M	8
9 – 16	PIN value. Each digit in the PIN shall be encoded with it's corresponding GSM default alphabet value. All unused digits at the end shall be encoded as '0xFF'.	M	8
17 – 24	ISO 9797 MAC	M	8

Table 2: PIN Block data for algorithm 2

3 Create a padded PIN Block *PPB = ISO_9797_PAD2('0x02' || PB<1..16>)*.

4 Calculate *MAC = ISO_9797_ALG3(PPB)*.
Note: 8 bytes of output from the MAC calculation shall be used (i.e. *m=64* using ISO9797 terminology).

5 Encrypt the PIN Block data *EPB = TDEA_ENCR(PB)* using the following cipher parameterization:

Keys K_1, K_2

Mode Triple encryption in outer CBC mode using two keys in DED operation. See Appendix A for more details.

IV 0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).

Note: Using terminology from [ISO9797], keys *K* and *K'* shall be derived by complementing alternate sub-strings of four bits of K_1 and K_2 respectively, commencing with the first four bits.

WIG WML Example

An example of a call to the RP plug-in looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- set the input parameters -->
    <setvar name="ID" value="US"/>
    <setvar name="Algorithm" value="\x01"/>
    <setvar name="EncrPINBlock"
value="\xc0\x42\x7e\x1a\x52\xcf\xf6\x75\xec\xe5\x4f\x52\xb2\x07\x41\x6f\x
13\xde\x88\xaa\xb5\x37\xd0\x34\x7e\x61\xbd\x25\x03\xa7\x4f\x55\x6f\xf1\x8
8\xc3\x32\x69\x69\xe0"/>
    <!-- call the plug-in to reset the pin to a new value -->
    <plugin name="RP" params="$(ID)$(Algorithm)$(EncrPINBlock)"
     destvar=" OUT1"/>
  </p>
  </card>
</wml>
```

In above shown example, the *ID Type* is ‘Asymmetric key usage’, *Private object ID* is ‘sign’ key and *Encrypted PIN Block header* is set to value 1.

Document number: 60077036
Revision: C. 2003-12-02.

3.7 EM – Event Manager

Name

EM

Description

The *Event Manager* plug-in allows an application to enable/(refresh)/disable the event mechanisms supported in WIB 1.2 (and later), both handset events and internal SIM events.

When the plug-in is called with the DISABLE flag, WIB is set to ignore all incoming events, both internal and handset events. This is valid either until the plug-in is called again with the ENABLE flag, or until next SIM Initialization, whichever occurs first.

When the plug-in is called with the ENABLE/REFRESH flag, the plug-in examines the EF (EventConfig), 6F0B. For all internal events, a coupling from those events to the corresponding event-activated scripts in EF (6F03) is established. If handset events occur in the file, and the handset supports events, the SAT command SET UP EVENT LIST is issued (if needed).

Input Parameters

Parameter	Description	Length
`Mode of operation flag`	This field indicates the type of operation mode to be set for the *EventManager*. Legal values are: **'0x00'** – Disable event mechanism **'0x01'** – Enable/Refresh event mechanism	1

Output Parameters

None.

WIG WML Example

This example illustrates the use of the EM plug-in.

Document number: 60077036
Revision: C. 2003-12-02.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- Disable event mechanism -->
    <setvar name="IN" value="\x00"/>
    <plugin name="EM" params="$(IN)" destvar="OUT"/>
  </p>
  </card>
</wml>
```

4 1st Generation 3DES Security Plug-Ins

This chapter introduces the 1st generation 3DES security plug-ins for WIB. These plug-ins provide basic features of 3DES cryptography. They have the advantage of being implemented on most WIB cards. However, there is also a 2nd generation 3DES plug-ins that provide more flexibility and if present on the SIM card they should preferably be used, see chapter 2nd Generation 3DES Security Plug-Ins. The 1st generation plug-ins are:

- ENCR - 3DES Encryption
- DECR - 3DES Decryption
- SIGN - 3DES Signing

Document number: 60077036
Revision: C. 2003-12-02.

4.1 ENCR - 3DES Encrypt Plug-In

Name

ENCR

Description

The *3DES encrypt* plug-in is used to encrypt arbitrary application-level data. It is typically called from a WIG WML document to privacy-protect data before it is transmitted to a network application. The cryptographic algorithm used is triple DES as described in [DEA], see also Appendix A: Triple DES modes.

Input Parameters

Parameter	Description	Length
`Encryptionkey`	The key index to be used. Specified by **\xHH** syntax, where HH shall be replaced by the desired hexadecimal value.	1
`Plaintext`	This is the data to be encrypted. It is a byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Output Parameters

The output from the plug-in is a sequence of bytes with the following meaning and in the given order:

Parameter	Description	Length
`Padding Information`	The padding information gives the number of bytes added as padding to the end of the plaintext before encryption. The following values are valid: **'0x00' – '0x07'**	1
`Ciphertext`	This is the ciphertext, that is, the encrypted plaintext. It is a byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Note. For a network application the plaintext is received after decrypting bytes 2 to N and stripping the number of padding characters according to byte 1 of the output. Thus, the length of the output is always 8n+1.

Document number: 60077036
Revision: C. 2003-12-02.

WIG WML Example

This example illustrates the use of the ENCR plug-in. The plug-in encrypts the text entered by the user and the ciphertext is posted to the host. In the plug-in call the key index is set to 2.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    To pay the ordered goods we need you to supply your Credit Card No
(will be sent encrypted)
    <!-- Get credit card as input from the user -->
    <input title="Enter Credit Card No:" name="PlainText" format="*N"/>
    <!-- call the encryption plug-in -->
    <plugin name="ENCR" params="\x02$(PlainText)" destvar="CipherText"/>
    <!-- send the ciphertext to the host -->
    <go href="http://www.smarttrust.com/pay?CCNo=$(CipherText)"/>
  </p>
  </card>
</wml>
```

Document number: 60077036
Revision: C. 2003-12-02.

4.2 DECR - 3DES Decrypt Plug-In

Name

DECR

Description

The *3DES decrypt* plug-in is used to decrypt arbitrary application-level data. It is typically called from a WIB script to recover the data that has been privacy protected by a network application. The cryptographic algorithm used is triple DES as described in [DEA], see also Appendix A: Triple DES modes.

Input Parameters

Parameter	Description	Length
`Encryptionkey`	The key index to be used. Specified by **\xHH** syntax, where HH shall be replaced by the desired hexadecimal value.	1
`Padding Information`	The padding information gives the number of bytes added as padding to the end of the plaintext before encryption. The following values are valid: **'0x00' – '0x07'**	1
`Ciphertext`	This is the data to be decrypted. It is a byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Output Parameters

The output from the plug-in is a sequence of bytes with the following meaning and in the given order:

Parameter	Description	Length
`Plaintext`	This is the plaintext received from decrypting the ciphertext. It is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

WIG WML Example

This example illustrates the use of the DECR plug-in. Information is sent to the user in encrypted form. The plug-in decrypts the information. The information is then displayed for the user. In the plug-in call the key index is set to 1.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- Set the encrypted data -->
    <setvar name="CipherText"
value="\x04\x2b\x85\x36\x05\x25\x67\x48\xe5"
        class="Binary"/>
    <!-- call the decryption plug-in -->
    <plugin name="DECR" params="\x01$(CipherText)" destvar="PlainText" />
    <!-- Display the plaintext to the user -->
    Your new PIN code is: $(PlainText)
  </p>
  </card>
</wml>
```

Document number: 60077036
Revision: C. 2003-12-02.

4.3 SIGN - 3DES Sign Plug-In

Name

SIGN

Description

The 3DES sign plug-in is used to calculate a message authentication code (MAC) for arbitrary application-level data. The MAC can be used as a data integrity mechanism to verify that data has not been altered in an unauthorized manner. It can also be used as a message authentication mechanism to provide assurance that a message has been originated by an entity in possession of the secret key.

The plug-in displays the text to be signed to the user and prompts for a PIN before calculating the MAC.

The cryptographic algorithm used is triple DES with two keys (EDE2) in outer CBC mode. The first 4 bytes (32 bits) of the MAC calculation are used as output of this plug-in.

Input Parameters

Parameter	Description	Length
`Encryptionkey`	The key index to be used. Specified by **\xHH** syntax, where HH shall be replaced by the desired hexadecimal value.	1
`Text To Be Signed (TTBS)`	This field contains the text used as input for the MAC calculation.	1-160

Output Parameters

Parameter	Description	Length
`Signature`	The output from the plug-in is the signature (or more correctly, the MAC) on the text to be signed (TTBS). The length of the output is 4 bytes and is the 4 MSB of the MAC calculation. It is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Document number: 60077036
Revision: C. 2003-12-02.

WIG WML Example

This example illustrates the use of the SIGN plug-in. The following WIG WML document is received from the host and contains the text to be signed. The plug-in signs the text and the signature is posted to the host. In the plug-in call the key index is set to 3.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    Signature request from The Bank...
    <!-- set input parameters -->
    <setvar name="TTBS" value="Amount: 44 USD; Debit acc.no: 123456-7;
To: Credit acc.no: 9876-543210; Ref: The Insurance company"
class="Binary"/>
    <!-- call the signature plug-in -->
    <plugin name="SIGN" params="\x03$(TTBS)" destvar="MAC"/>
    <!-- send the signature to the host -->
    <go
href="http://www.smarttrust.com/pay?STEXT=$(MAC)&amp;TTBS=$(TTBS)"/>
  </p>
  </card>
</wml>
```

Document number: 60077036
Revision: C. 2003-12-02.

5 2nd Generation 3DES Security Plug-Ins

This chapter introduces the 2nd generation 3DES security plug-ins for WIB. The plug-ins offers the same basic functionality as the 1st generation plug-ins as well as significant improvements:

- Support for session keys. Enables a party to create keys dynamically, i.e. whenever they are needed and not only when the card is personalized.
- Key diversification. Enable the possibility to use different keys for different purposes. For example different keys can be used for digital signatures, data encryption/decryption and key transport.
- Small changes in the plug-in interface to improve flexibility.
- Possibility to associate secret keys and PINs.

The plug-ins are:

- *DE - 3DES Encryption
- *DD - 3DES Decryption
- *DS - 3DES Signing
- *DU - 3DES Unwrap

Document number: 60077036
Revision: C. 2003-12-02.

5.1 *DE- 3DES Encrypt Plug-In

Name

*DE

Description

The *3DES encrypt* plug-in is used to encrypt arbitrary application-level data. It is typically called from a WIG WML document to privacy-protect data before it is transmitted to a network application.

Input Parameters

Parameter	Description	Length
`Key identity`	The key identity identifies the key to be used for the operation. Its value is given in hexadecimal value. The following values are valid: **'0x01' – '0xFE'**	1
`Options`	Options bit-field with the following bit-values defined: **b1**: IV flag: 0 – IV=0 1 – Plaintext starts with IV (8 bytes) **b2**: Cipher spec: 0 – 3DES ECB EDE with two keys 1 – 3DES CBC EDE with two keys **b3 – b8**: RFU The combination 'Plaintext starts with IV' and '3DES ECB EDE with two keys' is illegal.	1
`Plaintext`	This is the data to be encrypted. It is a byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Output Parameters

The output from the plug-in is a sequence of bytes with the following meaning and in the given order:

Parameter	Description	Length
Ciphertext	This is the encrypted plaintext. It is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

WIG WML Example

This example illustrates the use of the *3DES encrypt* plug-in. The plug-in encrypts the text entered by the user and the ciphertext is posted to the host.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    To pay the ordered goods we need you to supply your Credit Card No
(will be sent encrypted)
    <!-- Get credit card as input from the user -->
    <input title="Enter Credit Card No:" name="PlainText" format="*N"/>
    <!-- set input parameters -->
    <setvar name="KeyId" value="\x02"/>
    <setvar name="Opts" value="\x00"/>
    <!-- call the encryption plug-in -->
    <plugin name="*DE" params="$(KeyId)$(Opts)$(PlainText)"
     destvar="CipherText"/>
    <!-- send the ciphertext to the host -->
    <go href="http://www.smarttrust.com/pay?CCNo=$(CipherText)"/>
  </p>
  </card>
</wml>
```

Document number: 60077036
Revision: C. 2003-12-02.

Decryption procedure

To decrypt the ciphertext received from the plug-in, do the following:

1 Select the appropriate Key (K_1, K_2).

2 Calculate the padded message *PM = TDEA_DECR(Ciphertext)* using the appropriate cipher parameterization for

K_1, K_2	Key(s)
Cipher mode	ECB or CBC according to cipherspec.
IV	IV flag.

3 Calculate the plaintext *M = PKCS5_UNPAD(PM)*.

Where:

PKCS5_UNPAD	See [PKCS5] section 6.1.1.
TDEA_DECR	Triple DES decryption algorithm. See Appendix A.2 for details regarding the algorithm.

Document number: 60077036
Revision: C. 2003-12-02.

5.2 *DD - 3DES Decrypt Plug-In

Name

*DD

Description

The *3DES decrypt* plug-in is used to decrypt arbitrary application-level data. It is typically called from a WIB script to recover the data that has been privacy protected by a network application.

Input Parameters

Parameter	Description	Length
`Key identity`	The key identity identifies the key to be used for the operation. Its value is given in hexadecimal value. The following values are valid: **'0x01' – '0xFE'**	1
`Options`	Options bit-field with the following bit-values defined: **b1**: IV flag: 0 – IV=0 1 – Ciphertext starts with IV (8 bytes) **b2**: Cipher spec: 0 – 3DES EBC EDE with two keys 1 – 3DES CBC EDE with two keys **b3 – b8**: RFU The combination 'Ciphertext starts with IV' and '3DES ECB EDE with two keys' is illegal.	1
`Ciphertext`	This is the data to be decrypted. It is a byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

WIG WML Example

This example illustrates the use of the *3DES decrypt* plug-in. Information is sent to the user in encrypted form. The plug-in decrypts the information. The information is then displayed for the user.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- Set the encrypted data -->
    <setvar name="CipherText" value="\x2b\x85\x36\x05\x25\x67\x48\xe5"
        class="Binary"/>
    <!-- set input parameters -->
    <setvar name="KeyId" value="\x01"/>
    <setvar name="Opts" value="\x00"/>
    <!-- call the decryption plug-in -->
    <plugin name="*DD" params="$(KeyID)$(Opts)$(CipherText)"
     destvar="PlainText" />
    <!-- Display the plaintext to the user -->
    Your new PIN code is: $(PlainText)
  </p>
  </card>
</wml>
```

Encryption procedure

To encrypt a plaintext to be sent to the plug-in, do the following:

1 Select the appropriate Key (K_1, K_2).

2 Calculate the padded message *PM = PKCS5_PAD(Plaintext)*.

3 Calculate encrypted message *EM = TDEA_ENCR(PM)* using the following cipher parameterization:

K_1, K_2	Key(s)
Cipher mode	ECB or CBC
IV	IV flag.

4 *EM* is the ciphertext that may be decrypted by the plug-in.

Where:

PKCS5_PAD	PKCS#5 padding function. See [PKCS5] section 6.1.1.
TDEA_ENCR	Triple DES encryption algorithm. See Appendix A.1 for details regarding the algorithm.

5.3 *DS - 3DES Sign Plug-In

Name

*DS

Description

The *3DES sign* plug-in is used to calculate a message authentication code (MAC) for arbitrary application-level data. The MAC can be used as a data integrity mechanism to verify that data has not been altered in an unauthorized manner. It can also be used as a message authentication mechanism to provide assurance that a message has been originated by an entity in possession of the secret key.

The plug-in displays the text to be signed to the user and prompts for a PIN before calculating the MAC.

The cryptographic algorithm used is ISO9797 MAC algorithm 3, padding method 2. There is an option of using first 4 bytes (32 bits) or 8 bytes (64 bits) of the MAC calculation as output of this plug-in.

Input Parameters

Parameter	Description	Length
`Key identity`	The key identity identifies the key to be used for the operation. Its value is given in hexadecimal value. The following values are valid: **'0x01' – '0xFE'**	1
`Options`	Options bit-field with the following bit-values: **b1:** Truncation flag: 1 – 8 byte output 0 – 4 byte output **b2 – b8:** RFU	1
`Character encoding scheme`	Character encoding scheme used in the TTBS. It shall contain one of the following values: **"U"** = UCS2 **"S"** = SMS default alphabet, 7-bit unpacked	1
`Text To Be Signed (TTBS)`	This field contains the text used as input for the MAC calculation.	N

Output Parameters

Parameter	Description	Length
`Signature`	The output from the plug-in is the signature (or more correctly, the MAC) on the text to be signed (TTBS). The length of the output is 4 or 8 bytes as indicated by the Truncation flag MSB of the MAC calculation. The output is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

WIG WML Example

This example illustrates the use of the *3DES sign* plug-in. The following WIG WML document is received from the host and contains the text to be signed. The plug-in signs the text and the signature is posted to the host.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    Signature request from The Bank…
    <!-- set input parameters -->
    <setvar name="TTBS" value="Amount: 44 USD; Debit acc.no: 123456-7;
To: Credit acc.no: 9876-543210; Ref: The Insurance company"
class="Binary" />
    <setvar name="KeyID" value="\x03" class="Binary"/>
    <setvar name="OPTS" value="\x01" class="Binary"/>
    <setvar name="CES" value="S" class="Binary"/>
    <!-- call the signature plug-in -->
    <plugin name="*DS" params="$(KeyID)$(OPTS)$(CES)$(TTBS)"
     destvar="MAC"/>
    <!-- send the signature to the host -->
    <go
href="http://www.smarttrust.com/pay?STEXT=$(MAC)&amp;TTBS=$(TTBS)"/>
  </p>
  </card>
</wml>
```

Document number: 60077036
Revision: C. 2003-12-02.

MAC calculation procedure

To calculate the MAC for verification do the following:

1 Select the Key (K, K')

2 Calculate the padded message *PM = ISO_9797_PAD2(TTBS)*.

3 Calculate *MAC = ISO_9797_ALG3(PM)* using the following cipher parameterization:

K, K'	Key(s)
Truncation	Truncation to either 4 MSB or 8.

4 For verification purposes the *MAC* is verified to be identical to the output from the plug-in.

Where:

ISO_9797_ALG3	ISO9797 MAC algorithm 3. See [ISO9797] section 7.3 for further reference.
ISO_9797_PAD2	ISO9797 padding method 2. See [ISO9797] section 6.1.2 for further reference.

Document number: 60077036
Revision: C. 2003-12-02.

5.4 *DU - 3DES Unwrap Key Plug-in

Name

*DU

Description

The *3DES Unwrap key* plug-in is a key-management plug-in that enables a party in possession of a certain secret key, called a *key encryption key*, to replace a key in the SIM based key file, $EF_{SKEY,}$ at its own desire, under a set of well-defined security conditions.

Input Parameters

Parameter	Description	Length
`Key identity`	The key identity identifies the secret key to be replaced/updated. Its value is given in hexadecimal value. The following values are valid: **'0x01' – '0xFE'**	1
`Algorithm identifier`	Algorithm identifier. The following values are legal: **'0x01'** = 3DES + SHA-1 MDC **'0x02'** = 3DES + ISO 9797 MAC All other values are RFU.	1
`Encrypted key data`	Key data, encrypted and integrity protected. The selected algorithm determines the length of the field.	N

Output Parameters

None.

WIG WML Example

This example illustrates the use of the *3DES unwrap key* plug-in. The following WIG WML document is received from the host and contains the key to be unwrapped. The plug-in unwraps and stores the key on.

Document number: 60077036
Revision: C. 2003-12-02.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    <!-- set input parameters -->
    <setvar name="KeyID" value="\x02" class="Binary"/>
    <setvar name="AlgID" value="\x01" class="Binary"/>
    <setvar name="KeyData"
value="\xaf\x4a\xbe\xd9\xbf\x5d\x55\x99\x2f\xa6\xc3\x18\x35\xf8\xa9\xbb\x
f1\xf3\x0d\xfb\x77\xcd\xb9\x86\x18\x8b\xb9\xb3\x03\x32\xde\x9e"
class="Binary"/>
    <!-- call the unwrap key plug-in -->
    <plugin name="*DU" params="$(KeyID)$(AlgID)$(KeyData)"
     destvar="dummy"/>
  </p>
  </card>
</wml>
```

Encrypted Key Data Calculation Procedure

Creating the Encrypted Key Data to be sent to the Plug-in includes formatting, integrity protection and encryption of key data. The procedure for the two algorithms is described below.

Algorithm 1 is the preferred algorithm and may be the only one supported by the SIM unless specific reasons exist, e.g. like non-existing SHA-1 support on a SIM card.

Algorithm 1: 3DES with SHA-1 MDC

1 Create nonce and new DES key

2 Let *KD* be the Key Data and formatted according to:

Bytes	Description	M/O	Length
1 – 8	Random nonce.	M	8
9 – 24	Double length DES key	M	16
25 – 32	Key checksum (truncated SHA-1 MDC)	M	8

3 Let *Key ID* be the key identifier of the key to be replaced/updated on the SIM.

4 Calculate a message digest

$$MD = SHA1(Key\ ID \,||\, '0x01' \,||\, KD\langle 1..24\rangle)$$

5 Calculate the Key Checksum

$$KC = MD\langle 1..8\rangle$$

Document number: 60077036
Revision: C. 2003-12-02.

6 Encrypt the key data *EKD = TDEA_ENCR(KD)* using the following cipher parameterization:

K_1, K_2	Key Encryption Keys.
Cipher mode	Outer CBC. See Appendix A.1 for details.
IV	0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).

7 *EKD* is the Encrypted Key Data to be sent to the plug-in.

Document number: 60077036
Revision: C. 2003-12-02.

Algorithm 2: 3DES with ISO 9797 MAC

1 Create nonce and new DES key

2 Let *KD* be the Key Data and formatted according to:

Bytes	Description	M/O	Length
1 – 8	Random nonce.	M	8
9-24	Double length DES key	M	16
25 – 32	Key checksum (ISO 9797 MAC)	M	8

3 Let *Key ID* be the key identifier of the key to be replaced/updated on the SIM.

4 Calculate the padded message
PM = ISO_9797_PAD2(Key ID || '02h || KD<1..24>)

5 Calculate the key checksum
KC = ISO_9797_ALG3(PM)

6 Encrypt the key data *EKD = TDEA_ENCR(KD)* using the following cipher parameterization:

K_1, K_2	Key Encryption Keys.
Cipher mode	Outer CBC. See Appendix A.1 for more details.
IV	0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).

7 *EKD* is the Encrypted Key Data to be sent to the plug-in.

Note: Using terminology from [ISO9797], keys *K* and *K'* shall be derived by complementing alternate sub-strings of four bits of K_1 and K_2 respectively, commencing with the first four bits.

Note: 8 bytes of output from the MAC calculation shall be used (i.e. *m=64* using ISO9797 terminology).

6 RSA Based Security Plug-Ins

This chapter introduces RSA based security plug-ins for WIB. The plug-ins are:

- P7 - PKCS#7 Signing
- FP - Fingerprint
- AD - Asymmetric Decryption

Note. The administration plug-ins, *Change PIN* and *Install PIN*, may be related to the same keys as the plug-ins in this chapter.

Document number: 60077036
Revision: C. 2003-12-02.

6.1 P7 - PKCS#7 Signature Plug-In

Name

P7

Description

The P7 plug-in is used to provide a digital signature based on a private RSA key stored on a SIM card. The output of this plug-in is compliant with the WMLScript Crypto Library SignText function, [WMLCLIB]. As such, P7 will also be compliant with other important (de-facto) standards: [PKCS1], [PKCS7], [CMS].

The plug-in first shows the text to be signed to the user and then prompts for a signature PIN. The plug-in implements true WYSIWYS (What-You-See-Is-What-You-Sign).

Input Parameters

Parameter	Description	Length
`Character Encoding Scheme`	This field indicates what character encoding scheme is used for the TTBS and hence shall be used when displaying the TTBS. The following values are valid: **"U"** – UCS2 **"S"** – SMS default alphabet, 7-bit unpacked	1

Document number: 60077036
Revision: C. 2003-12-02.

`Format Options`	This field specifies additional processing options to the plug-in. **Bit 1**: CTFLG – Content flag – If set, the plug-in must include the TTBS in the output. **Bit 2**: KHFLG – Key hash flag – If set, the plug-in must include the hash of the public key corresponding to the signature key in the output. **Bit 3**: CEFLG – Certificate flag – If set, the plug in must include a URL to the public key certificate in the output. **Bit 4**: ICCFLG – ICCID flag – If set, the plug-in must include the ICCID of the SIM in the output. **Bit 5**: MDFLG – Message digest flag – If set, the plug-in must include the message digest of the TTBS in the output. **Bits 6-8**: RFU.	1
`Text To Be Signed (TTBS)`	A valid TTBS must not exceed 160 bytes. The character encoding shall be as specified in the CES field.	1-160

Output Parameters

Parameter	Description	Length
`Signature or Error message`	The output from the P7 plug-in is one (and only one) of the following: - A byte string representing the SignedContent data structure as specified in [WMLCLIB]. - An error message. It is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Note: If the total length of the output from the P7 plug-in exceeds the maximum length of a variable, currently 255, this will lead to an error situation.

WIG WML Example

This example illustrates the use of the P7 plug-in. The following WIG WML document is received from the host and contains the text to be signed. The plug-in is called to sign the text and the signature is posted to the host.

Document number: 60077036
Revision: C. 2003-12-02.

The document to be signed is sent in SMS Default alphabet as given by the CES parameter. The OPTS parameter is set to request the document as well as the ICCID to be included in the signature message. This may be useful for an application verifying the signature.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    Signature request from The Bank
    <!-- set input parameters -->
    <setvar name="CES" value="S"/>
    <setvar name="OPTS" value="\x09"/>
    <setvar name="TTBS" value="Amount: 44 USD; Debit acc.no: 123456-7;
Credit acc.no: 9876-543210; Ref: The Insurance company" class="Binary"/>
    <!-- call the signature plug-in -->
    <plugin name="P7" params="$(CES)$(OPTS)$(TTBS)" destvar="STEXT"/>
    <!--send the signature to the host -->
    <go
href="http://www.smarttrust.com/sign?STEXT=$(STEXT)&amp;amp;RefID=54321"/>
  </p>
  </card>
</wml>
```

6.2 FP - Fingerprint Plug-In

Name

FP

Description

The FP plug-in is used to provide a digital signature based on a private RSA key stored on a SIM card. The output of this plug-in is a WrappedContent structure described below that includes a PKCS#1 compliant signature. As such, FP will also be compliant with other important (de-facto) standards: [PKCS1], [PKCS7], [CMS].

At first glance, FP may seem strikingly similar to the P7 plug-in. This is nevertheless not the case. As opposed to the P7 plug-in, FP will not operate strictly according to the WYSIWYS (What-You-See-Is-What-You-Sign) paradigm, but instead work more as an alternative to a smart card in a "fixed" PKI scenario. This ensures that FP can be utilized in cases where P7 is clearly unsuitable, i.e.

- Signing data larger than a few hundred bytes, e.g. an email message or word-processors document.
- Signing data that is not displayable on a mobile phone, e.g. a word-processor document or random nonce in a VPN set-up phase.

Other utilization is also easily imaginable.

Input Parameters

Parameter	Description	Length
`Key Usage identifier`	This field determines which type of key the plug-in shall use in the forthcoming signing operation. The following values are valid: **"S"** – sign **"N"** – nonRepudiation	1

`Format options`	This field specifies additional processing options to the plug-in. **Bit 1**: RFU. **Bit 2**: KHFLG – Key hash flag – If set, the plug-in must include the hash of the public key corresponding to the signature key in the output. **Bit 3**: CEFLG – Certificate flag – If set, the plug in must include a URL to the public key certificate in the output. **Bit 4**: ICCFLG – ICCID flag – If set, the plug-in must include the ICCID of the SIM in the output. **Bit 5-8**: RFU.	1
`Data To Be Signed (DTBS)`	This field represents the data to be signed. To be truly PKCS#1 compliant, this should be a DER encoded value of the DigestInfo ASN.1 type, as specified in [PKCS1].	16-255

Output Parameters

Parameter	Description	Length
`Signature or Error message`	The output from the FP plug-in is one (and only one) of the following: - A byte string representing the signature. The byte string follows the WrappedContent structure specified below. - An error message. It is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Note: If the total length of the output from the FP plug-in exceeds the maximum length of a variable, currently 255, this will lead to an error situation.

Format of WrappedContent

The output of the plug-in follows the WrappedContent format. The format includes both the signature and other related data. It is described using the same presentation language as used in [WTLS].

```
struct {
  opaque signature<0.. 2^16-1>;
} Signature;

enum { implicit(0), sha_key_hash(1), certificate_url(5), iccid
(128), key_usage_id(129), (255) } SignerInfoType;
```

Item	Description
Implicit	The signer is implied by the content.
sha_key_hash	The SHA-1 hash of the public key, encoded as specified in [WAPWTLS].
certificate_url	A URL where the certificate is located.
key_usage_id	An ID revealing the key usage flag (in the PKCS#15 sense) of the signature key.

```
struct {
  SignerInfoType signer_info_type;
  switch (signer_info_type) {
   case implicit: struct{};
   case sha_key_hash: opaque hash[20];
   case certificate_url: opaque url<0..255>;
   case iccid: opaque iccid[10];
   case key_usage_id: uint8;
  };
} SignerInfo;


struct {
  uint8 version;
  Signature signature;
  SignerInfo signer_infos<0..2^16-1>;
} WrappedContent;
```

Item	Description
Version	Version of the WrappedContent structure. For this specification the version is 1.
Signature	Signature
signer_infos	Information on the signer. This may contain zero items (in case the signer is implicit). Also, there may be multiple items of SignerInfo present (public key hash and a certificate).

Document number: 60077036
Revision: C. 2003-12-02.

WIG WML Example

This example illustrates the use of the FP plug-in. The following WIG WML document is received from a network application and contains the data to be signed. The data may for example origin from a word document. The plug-in signs the data and the signature is posted to the network application.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    Signature request from The Application, ref: 12345
    <!-- set input parameters -->
    <setvar name="KU" value="S"/>
    <setvar name="OPTS" value="\x00"/>
    <setvar name="DTBS"
value="\x02\x33\xF0\xA8\x89\x28\xF1\x80\x5A\x23\x33\xE3\x38\x61\x30\x11\x
19\x5A\x58\x12"/>
    <!-- call the signature plug-in -->
     <plugin name="FP" params="$(KU)$(OPTS)$(DTBS)"
      destvar="SDATA"/>
    <!-- send the signature to the host -->
    <go
href="http://www.smarttrust.com/sign?SDATA=$(SDATA)&amp;RefID=12345"/>
  </p>
  </card>
</wml>
```

Document number: 60077036
Revision: C. 2003-12-02.

6.3 AD - Asymmetric Decryption Plug-In

Name

AD

Description

This plug-in is used for application-level asymmetric decryption. The decryption is performed according to RSADP. See [PKCS1] for further reference.

Just as in the case with the FP plug-in, the motivation for this plug-in is to serve as a replacement for a smart card in a "fixed" PKI scenario. While the FP plug-in is focused on digital signatures, AD is focused on the remaining private key operation, namely decryption.

Together, FP and AD form a complete replacement to the "PC attached" smart card, and in addition offer other benefits like end-user mobility, cost effectiveness and easy deployment.

If the output of the plug-in shall be used in a network application it is crucial that the plaintext is protected by some means, e.g. using "blinding".

Input Parameters

Parameter	Description	Length
`Key Usage identifier`	This field determines which type of key the plug-in shall use in the forthcoming decryption operation. **"D"** = decrypt **"U"** = unwrap	1
`Ciphertext`	This field represents the ciphertext, a byte string of length k, where k is the length in bytes of the modulus n. Hence, for a 1024 bit key, it must be of length 128.	16-255

Document number: 60077036
Revision: C. 2003-12-02.

Output Parameters

Parameter	Description	Length
`Plaintext or Error message`	The output from the FP plug-in is one (and only one) of the following: - A byte string representing the decrypted data. - An error message. It is an byte string where the bytes may take any value in the range: **'0x00' – '0xFF'**	N

Note: If the total length of the output from the AD plug-in exceeds the maximum length of a variable, currently 255, this will lead to an error situation.

WIG WML Example

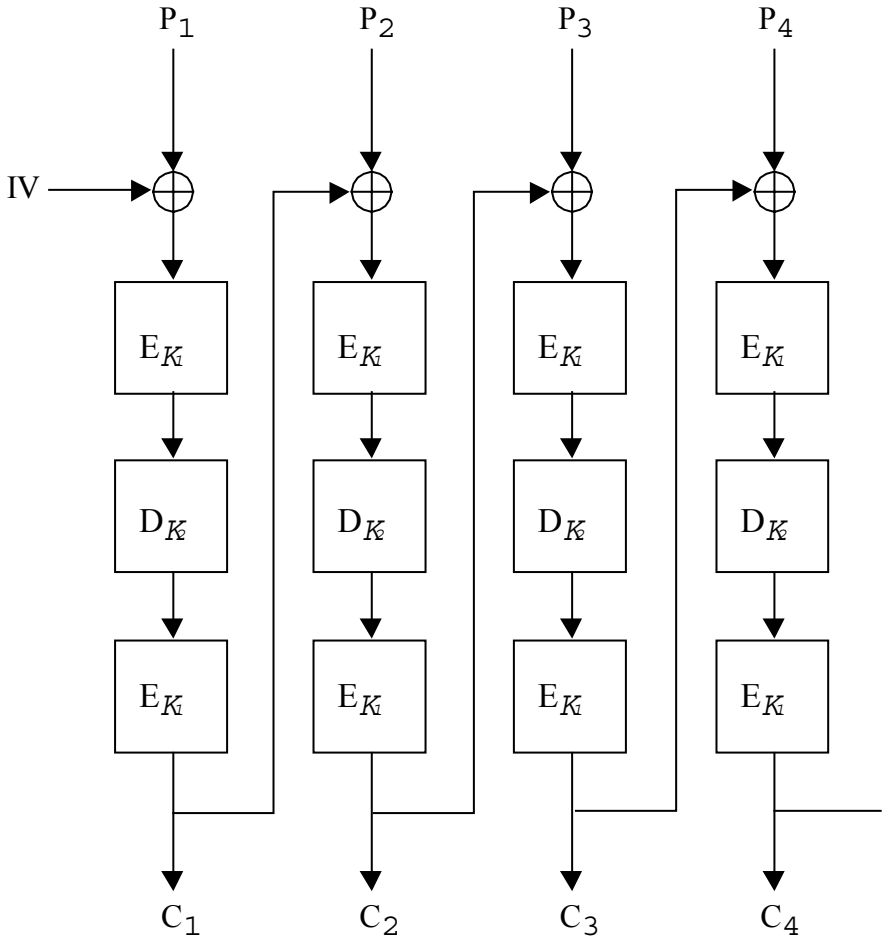
This example illustrates the use of the AD plug-in. The following WIG WML document is received from a network application and contains the data to be decrypted. The data may for example origin from a mail application. The plug-in decrypts the data and the data is posted to the network application. This example is using a modulus that is artificially small (since length of ciphertext is only 16 bytes). Nevertheless it suits its purpose as an example for the calling convention.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
  <p>
    Decryption request from The Application, ref: 135
    <!-- set input parameters -->
    <setvar name="KeyUsage" value="\x44"/>
    <setvar name="CipherText"
value="\xAC\x33\xF0\xA8\x28\xF1\x80\x23\x33\xE3\x61\x30\x11\x5A\x58\x13"
     />
    <!-- call the decryption plug-in -->
    <plugin name="AD" params="$(KeyUsage)$(CipherText)"
     destvar="DDATA"/>
    <!-- send the decrypted data to the application -->
    <go
href="http://www.smarttrust.com/decrypted?Data=$(DDATA)&amp;RefID=135"/>
  </p>
  </card>
</wml>
```

Appendix A: Triple DES modes

A.1 Triple encryption (TDEA_ENCR)

The figure below illustrates DES triple encryption in outer CBC mode using two keys in encrypt-decrypt-encrypt (EDE) operation.



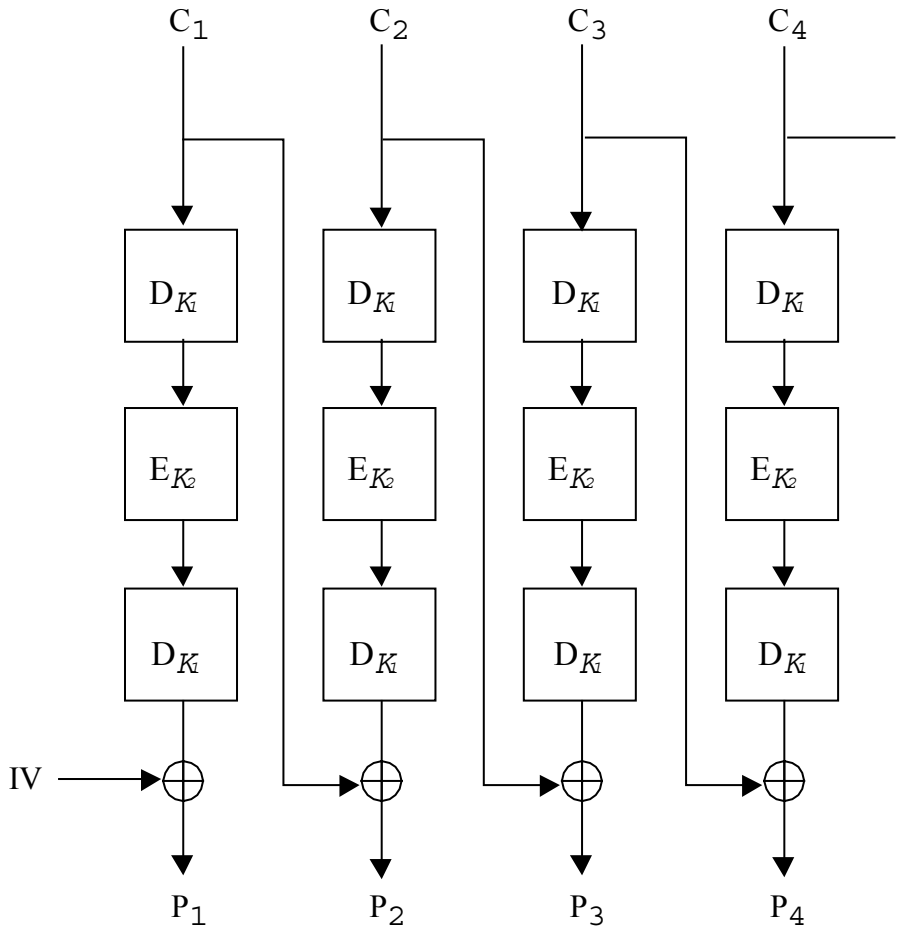
ECB operation is achieved by removing the ciphertext feedback and the IV.

See [DEA] for details regarding the DES algorithm and [MODES] for details regarding the CBC and ECB cipher mode.

Document number: 60077036
Revision: C. 2003-12-02.

A.2 Triple decryption (TDEA_DECR)

The figure below illustrates DES triple decryption in outer CBC mode using two keys in decrypt-encrypt-decrypt (DED) operation.



ECB operation is achieved by removing the ciphertext feedback and the IV.

See [DEA] for details regarding the DES algorithm and [MODES] for details regarding the CBC and ECB cipher mode.

Document number: 60077036
Revision: C. 2003-12-02.