CEN

WORKSHOP

AGREEMENT

CWA 14050-6

October 2003

ICS 35.200; 35.240.15; 35.240.40

Supersedes CWA 14050-6:2000

English version

Extensions for Financial Services (XFS) interface specification - Release 3.0 - Part 6: Pin Keypad Device Class Interface - Programmer's Reference

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties, the constitution of which is indicated in the foreword of this Workshop Agreement.

The formal process followed by the Workshop in the development of this Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN Management Centre can be held accountable for the technical content of this CEN Workshop Agreement or possible conflicts with standards or legislation.

This CEN Workshop Agreement can in no way be held as being an official standard developed by CEN and its Members.

This CEN Workshop Agreement is publicly available as a reference document from the CEN Members National Standard Bodies.

CEN members are the national standards bodies of Austria, Belgium, Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Luxembourg, Malta, Netherlands, Norway, Portugal, Slovakia, Spain, Sweden, Switzerland and United Kingdom.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: rue de Stassart, 36 B-1050 Brussels

© 2003 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Ref. No. CWA 14050-6:2003 D/E/F

Table of Contents

Foreword

This CWA is revision 3.02 of the XFS interface specification.

The CEN/ISSS XFS Workshop gathers suppliers as well as banks and other financial service companies. A list of companies participating in this Workshop and in support of this CWA is available from the CEN/ISSS Secretariat.

This document supersedes CWA 14050-6:2000.

This CWA was formally approved by the XFS Workshop meeting on 2003-05-21. The specification is continuously reviewed and commented in the CEN/ISSS Workshop on XFS. It is therefore expected that an update of the specification will be published in due time as a CWA, superseding this revision 3.02.

The CWA is published as a multi-part document, consisting of:

Part 1: Application Programming Interface (API) - Service Provider Interface (SPI); Programmer's Reference

Part 2: Service Classes Definition; Programmer's Reference

Part 3: Printer Device Class Interface - Programmer's Reference

Part 4: Identification Card Device Class Interface - Programmer's Reference

Part 5: Cash Dispenser Device Class Interface - Programmer's Reference

Part 6: PIN Keypad Device Class Interface - Programmer's Reference

Part 7: Check Reader/Scanner Device Class Interface - Programmer's Reference

Part 8: Depository Device Class Interface - Programmer's Reference

Part 9: Text Terminal Unit Device Class Interface - Programmer's Reference

Part 10: Sensors and Indicators Unit Device Class Interface - Programmer's Reference

Part 11: Vendor Dependent Mode Device Class Interface - Programmer's Reference

Part 12: Camera Device Class Interface - Programmer's Reference

Part 13: Alarm Device Class Interface - Programmer's Reference

Part 14: Card Embossing Unit Class Interface - Programmer's Reference

Part 15: Cash In Module Device Class Interface- Programmer's Reference

Part 16: Application Programming Interface (API) - Service Provider Interface (SPI) - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 17: Printer Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 18: Identification Card Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (see CWA 14050-4:2000; superseded) - Programmer's Reference

Part 19: Cash Dispenser Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 20: PIN Keypad Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (see CWA 14050-6:2000; superseded) - Programmer's Reference

Part 21: Depository Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 22: Text Terminal Unit Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 23: Sensors and Indicators Unit Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.01 (this CWA) - Programmer's Reference

Part 24: Camera Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 25: Identification Card Device Class Interface - PC/SC Integration Guidelines

Part 26: Identification Card Device Class Interface - Migration from Version 3.00 (see CWA 14050-4:2000; superseded) to Version 3.02 (this CWA) - Programmer's Reference

Part 27: PIN Keypad Device Class Interface - Migration from Version 3.00 (see CWA 14050-6:2000; superseded) to Version 3.02 (this CWA) - Programmer's Reference

Part 28: Cash In Module Device Class Interface - Migration from Version 3.00 (see CWA 14050-15:2000; superseded) to Version 3.02 (this CWA) - Programmer's Reference

In addition to these Programmer's Reference specifications, the reader of this CWA is also referred to a complementary document, called Release Notes. The Release Notes contain clarifications and explanations on the CWA specifications, which are not requiring functional changes. The current version of the Release Notes is available online from http://www.cenorm.be/isss/Workshop/XFS.

The information in this document represents the Workshop's current views on the issues discussed as of the date of publication. It is furnished for informational purposes only and is subject to change without notice. CEN/ISSS makes no warranty, express or implied, with respect to this document.

Revision History:

1.0	24 May 1993	Initial release of API and SPI specification
1.11	3 February 1995	Separation of specification into separate documents for API/SPI and service class definitions
2.00	11 November 1996	Update release encompassing the self-service environment
3.00	18 October 2000	Update release encompassing: - new commands to support the German ZKA chip card standard – support of Banksys Security Control Module - Added clarification note for Pin format 3624 - Added WFS_CMD_PIN_ENC_IO, which is currently used for the Swiss proprietary protocol only. - Double and triple zero clarification in WFS_CMD_PIN_GET_DATA - key deletion in WFS_CMD_PIN_IMPORT_KEY inserted. For a detailed description see CWA 14050-20 PIN Migration from Version 2.00 to Version 3.00
3.02	21 May 2003	Update release encompassing: - New commands to support EMV, GIE-CB, Remote Key Loading (Signature and Certificate), OPT, MAA MAC, and Multiple-Part Key Loading. - Added clarification notes on WFS_PIN_CRYPTTRIDESMAC to the WFS_INF_CAPABILITES and WFS_CMD_PIN_CRYPT For a detailed description see CWA 14050-27 PIN Migration from Version 3.00 to Version 3.02

1. Introduction

1.1 Background to Release 3.02

The CEN XFS Workshop is a continuation of the Banking Solution Vendors Council workshop and maintains a technical commitment to the Win 32 API. However, the XFS Workshop has extended the franchise of multi vendor software by encouraging the participation of both banks and vendors to take part in the deliberations of the creation of an industry standard. This move towards opening the participation beyond the BSVC's original membership has been very successful with a current membership level of more than 20 companies.

The fundamental aims of the XFS Workshop are to promote a clear and unambiguous specification for both service providers and application developers. This has been achieved to date by sub groups working electronically and quarterly meetings.

The move from an XFS 3.00 specification to a 3.02 specification has been prompted by a series of factors. There has been pressure from the market to have this new Pin Pad functionality incorporated into the specification. The following functionality has been added to the specification.

The Pinpad Specification now includes the capability to load the initial master symmetric DES keys in an automated secure way from a remote location. This is done by two alternative schemes for providing the key-loading authentication: three-party authentication through certificates and two-party authentication through signatures.

The specification provides the capability to support EMV 4.0 in a branch or self-service environment.

It also supports Multiple Key Part loading where keys are loaded into the encryptor in parts. GIE-CB, which is the exchange of cryptographic secured data between hosts and ATMs is defined on a basis of „interchanges“ and „data items“ is also supported in this proposal.

The specification also supports OPT (Online Personalization of Terminals). The German ZKA committee has defined a way to send keys to the EPP in an online dialog.

Finally, this specification supports the ability to do a MAC computation with the MAA algorithm.

The clear direction of the XFS Workshop, therefore, is the delivery of a new Release 3.02 specification based on a C API. It will be delivered with the promise of the protection of technical investment for existing applications and the design to safeguard future developments. All XFS 3.00 clarifications should be used when integrating with the XFS 3.02 Pin Pad specification.

1.2 XFS Service-Specific Programming

The service classes are defined by their service-specific commands and the associated data structures, error codes, messages, etc. These commands are used to request functions that are specific to one or more classes of service providers, but not all of them, and therefore are not included in the common API for basic or administration functions.

When a service-specific command is common among two or more classes of service providers, the syntax of the command is as similar as possible across all services, since a major objective of the Extensions for Financial Services is to standardize command codes and structures for the broadest variety of services. For example, using the **WFSExecute** function, the commands to read data from various services are as similar as possible to each other in their syntax and data structures.

In general, the specific command set for a service class is defined as the union of the specific capabilities likely to be provided by the developers of the services of that class; thus any particular device will normally support only a subset of the defined command set.

There are three cases in which a service provider may receive a service-specific command that it does not support:

- The requested capability is defined for the class of service providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability is ***not*** considered to be fundamental to the service. In this case, the service provider returns a successful completion, but does no operation. An example would be a request from an application to turn on a control indicator on a passbook printer; the service provider recognizes the command, but since the passbook printer it is managing does not include that indicator, the service provider does no operation and returns a successful completion to the application.
- The requested capability is defined for the class of service providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability ***is*** considered to be fundamental to the service. In this case, a WFS_ERR_UNSUPP_COMMAND error is returned to the calling application. An example would be a request from an application to a cash dispenser to dispense coins; the service provider recognizes the command but, since the cash dispenser it is managing dispenses only notes, returns this error.
- The requested capability is ***not*** defined for the class of service providers by the XFS specification. In this case, a WFS_ERR_INVALID_COMMAND error is returned to the calling application .

This design allows implementation of applications that can be used with a range of services that provide differing subsets of the functionalities that are defined for their service class. Applications may use the **WFSGetInfo** and **WFSAsyncGetInfo** commands to inquire about the capabilities of the service they are about to use, and modify their behavior accordingly, or they may use functions and then deal with WFS_ERR_UNSUPP_COMMAND error returns to make decisions as to how to use the service.

2. Personal Identification Number (PIN) Keypads

This section describes the application program interface for personal identification number keypads (PIN pads) and other encryption/decryption devices. This description includes definitions of the service-specific commands that can be issued, using the **WFSAsyncExecute**, **WFSExecute**, **WFSGetInfo** and **WFSAsyncGetInfo** functions.

This section describes the general interface for the following functions:

- Administration of encryption devices
- Loading of encryption keys
- Encryption / decryption
- Entering Personal Identification Numbers (PINs)
- PIN verification
- PIN block generation (encrypted PIN)
- Clear text data handling
- Function key handling
- PIN presentation to chipcard
- Read and write safety critical Terminal Data from/to HSM
- HSM and Chipcard Authentication
- EMV 4.0 PIN blocks, EMV 4.0 public key loading, static and dynamic data verification

If the PIN Pad device has local display capability, display handling should be handled using the Text Terminal Unit (TTU) interface.

The adoption of this specification does not imply the adoption of a specific security standard.

Important Notes:

- This revision of this specification does not define key management procedures; key management is vendor-specific.
- Key space management is customer-specific, and is therefore handled by vendor-specific mechanisms.
- Only numeric PIN pads are handled in this specification.

This specification also supports the Hardware Security Module (HSM), which is necessary for the German ZKA Electronic Purse transactions. Furthermore the HSM stores terminal specific data.
This data will be compared against the message data fields (Sent and Received ISO8583 messages) prior to HSM-MAC generation/verification. HSM-MACs are generated/verified only if the message fields match the data stored.

Keys used for cryptographic HSM functions are stored separate from other keys. This must be considered when importing keys.

This version of PinPad complies to the current ZKA specification 3.0. It supports loading and unloading against card account for both card types (Type 0 and Type 1) of the ZKA electronic purse. It also covers the necessary functionality for ‘Loading against other legal tender’.

Key values are passed to the API as binary hexadecimal values, for example:

    0123456789ABCDEF = 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF

3. References

1	XFS Application Programming Interface (API)/Service Provider Interface (SPI), Programmer's Reference Revision 3.00, October 18, 2000
2	RSA Laboratories, PKCS #7: *Cryptographic Message Syntax Standard.* Version 1.5, November 1993
3	SHA-1 Hash algorithm ANSI 9:30:2-1993: *Public Key Cryptography for Financial Services Industry Part2*
4	EMVCo, EMV2000 Integrated Circuit Card Specification for Payment Systems, Book 2 – Security and Key Management, Version 4.0, December 2000
5	Europay International, EPI CA Module Technical – Interface specification Version 1.4
6	ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ec-Karte mit Chip, Online-Personalisierung von Terminal-HSMs, Version 3.0, 2. 4. 1998
7	ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ZKA-Chipkarte, Online-Vor-Initialisierung und Online-Anzeige einer Außerbetriebnahme von Terminal-HSMs, Version 1.0, 04.08.2000
8	473x Programmers Reference Volume 1 - TP-820399-001A
9	473x Programmers Reference Volume 2 - TP-820403-001A
10	473x Programmers Reference Volume 3 - TP-820400-001A
11	473x Programmers Reference Volume 4 - TP-820404-001A
12	473x P-Model Programmers Reference - TP-820397-001A
13	473x Log Reference Guide - TP-820398-001A
14	Diebold's Specification for support of Online Preinitialization and Personalization of Terminal HSMs (OPT) and support for the PAC/MAC standards for the 473x Protocol, Diebold USA, Revision 1.10, revised on May 2002
15	Groupement des Cartes Bancaires "CB", Description du format et du contenu des données cryprographiques échangées entre GAB et GDG, Version 1.3 / Octobre 2002
16	ITU-T Recommendation X.690 – ASN.1 encoding rules (also published as ISO/IEC International Standard 8825-1), 1997

4. Info Commands

4.1 WFS_INF_PIN_STATUS

Description The WFS_INF_PIN_STATUS command returns several kinds of status information.

Input Param None.

Output Param

```
LPWFSPINSTATUS      lpStatus;

typedef struct _wfs_pin_status
   {
   WORD             fwDevice;
   WORD             fwEncStat;
   LPSTR            lpszExtra;
   } WFSPINSTATUS, * LPWFSPINSTATUS;
```

fwDevice
Specifies the state of the PIN pad device as one of the following flags:

Value	Meaning
WFS_PIN_DEVONLINE	The device is online (i.e. powered on and operable).
WFS_PIN_DEVOFFLINE	The device is offline (e.g., the operator has taken the device offline by turning a switch or pulling out the device).
WFS_PIN_DEVPOWEROFF	The device is powered off or physically not connected.
WFS_PIN_DEVNODEVICE	There is no device intended to be there; e.g. this type of self service machine does not contain such a device or it is internally not configured.
WFS_PIN_DEVHWERROR	The device is inoperable due to a hardware error.
WFS_PIN_DEVUSERERROR	The device is present but a person is preventing proper device operation.
WFS_PIN_DEVBUSY	The device is busy and unable to process an execute command at this time.

fwEncStat
Specifies the state of the Encryption Module as one of the following flags:

Value	Meaning
WFS_PIN_ENCREADY	The encryption module is initialized and ready (at least one key is imported into the encryption module).
WFS_PIN_ENCNOTREADY	The encryption module is not ready.
WFS_PIN_ENCNOTINITIALIZED	The encryption module is not initialized (no master key loaded).
WFS_PIN_ENCBUSY	The encryption module is busy (implies that the device is busy).
WFS_PIN_ENCUNDEFINED	The encryption module state is undefined.
WFS_PIN_ENCINITIALIZED	The encryption module is initialized and master key (where required) and any other initial keys are loaded; ready to import other keys.

lpszExtra
Specifies a list of vendor-specific, or any other extended, information. The information is returned as a series of "*key=value*" strings so that it is easily extendable by service providers. Each string will be null-terminated, with the final string terminating with two null characters.

For Remote Key Loading using Certificates, the following key/value pairs indicate the level of support of the service provider. If these pairs are not returned then this indicates the SP does not support the corresponding feature:

-CERTIFICATESTATE=<0xnnnnnnnn>, where nnnnnnn is the ASCII representation of a hexadecimal value. This state determines which public verification or encryption key should be read out of the device. For example CERTIFICATESATE =0x00000001 indicates that the state of the Encryptor is Primary. The possible values are the following:

Value	Meaning

WFS_PIN_CERT_PRIMARY	The encryption module indicates that all pre-loaded certificates have been loaded and that primary verification certificates will be accepted for the commands WFS_CMD_PIN_LOAD_CERTIFICATE or WFS_CMD_PIN_REPLACE_CERTIFICATE
WFS_PIN_CERT_SECONDARY	The encryption module indicates that primary verification certificates will not be accepted and only secondary verification certificates will be accepted. If primary certificates have been compromised (which the certificate authority or the host detects), then secondary certificates should be used in any transaction. This is done by calling the WFS_CMD_PIN_LOAD_CERTIFICATE command or the WFS_CMD_PIN_REPLACE_CERTIFICATE.
WFS_PIN_CERT_NOTREADY	The certificate module is not ready. (The device is powered off or physically not present).

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Comments Applications which require or expect specific information to be present in the *lpszExtra* parameter may not be device or vendor-independent.

4.2 WFS_INF_PIN_CAPABILITIES

Description This command is used to retrieve the capabilities of the PIN pad.

Input Param None.

Output Param `LPWFSPINCAPS lpCaps;`

```
typedef struct _wfs_pin_caps
   {
   WORD         wClass;
   WORD         fwType;
   BOOL         bCompound;
   USHORT       usKeyNum;
   WORD         fwAlgorithms;
   WORD         fwPinFormats;
   WORD         fwDerivationAlgorithms;
   WORD         fwPresentationAlgorithms;
   WORD         fwDisplay;
   BOOL         bIDConnect;
   WORD         fwIDKey;
   WORD         fwValidationAlgorithms;
   WORD         fwKeyCheckModes;
   LPSTR        lpszExtra;
   } WFSPINCAPS, * LPWFSPINCAPS;
```

wClass
Specifies the logical service class, value is:
WFS_SERVICE_CLASS_PIN

fwType
Specifies the type of the PIN pad security module as a combination of the following flags. PIN entry is only possible when at least WFS_PIN_TYPEEPP and WFS_PIN_TYPEEDM are set. In order to use the ZKA-Electronic purse, all flags must be set.

Value	Meaning
WFS_PIN_TYPEEPP	electronic PIN pad (keyboard data entry device)
WFS_PIN_TYPEEDM	encryption/decryption module
WFS_PIN_TYPEHSM	hardware security module (electronic PIN pad and encryption module within the same physical unit)

bCompound
Specifies whether the logical device is part of a compound physical device and is either TRUE or FALSE.

usKeyNum
Number of the keys which can be stored in the encryption/decryption module.

fwAlgorithms
Supported encryption modes; a combination of the following flags:

Value	Meaning
WFS_PIN_CRYPTDESECB	Electronic Code Book
WFS_PIN_CRYPTDESCBC	Cipher Block Chaining
WFS_PIN_CRYPTDESCFB	Cipher Feed Back
WFS_PIN_CRYPTRSA	RSA Encryption
WFS_PIN_CRYPTECMA	ECMA Encryption
WFS_PIN_CRYPTDESMAC	MAC calculation using CBC
WFS_PIN_CRYPTTRIDESECB	Triple DES with Electronic Code Book
WFS_PIN_CRYPTTRIDESCBC	Triple DES with Cipher Block Chaining
WFS_PIN_CRYPTTRIDESCFB	Triple DES with Cipher Feed Back
WFS_PIN_CRYPTTRIDESMAC	Last Block Triple DES MAC as defined in ISO/IEC 9797-1:1999, using: block length n=64, Padding Method 1 (when *bPadding*=0), MAC Algorithm 3, MAC length m where 32<=m<=64
WFS_PIN_CRYPTMAAMAC	MAC calculation using the Message authenticator algorithm as defined in ISO 8731-2

fwPinFormats
Supported PIN formats; a combination of the following flags:

Value	Meaning
WFS_PIN_FORM3624	PIN left justified, filled with padding characters, PIN length 4-16 digits. The Padding Character is a Hexadecimal Digit in the range 0x00 to 0x0F.
WFS_PIN_FORMANSI	PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number)
WFS_PIN_FORMISO0	PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, no minimum length specified, missing digits are filled with 0x00)
WFS_PIN_FORMISO1	PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits).
WFS_PIN_FORMECI2	(similar to WFS_PIN_FORM3624), PIN only 4 digits
WFS_PIN_FORMECI3	PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from X’0’ through X’F’.
WFS_PIN_FORMVISA	PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with X’0’ to the length of six, the padding character can range from X ' 0 ' through X ' 9 ' (This format is also referred to as VISA2).
WFS_PIN_FORMDIEBOLD	PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted.
WFS_PIN_FORMDIEBOLDCO	PIN with the length of 4 to 12 digits, each one with a value of X’0’ to X’9’, is preceded by the one-digit coordination number with a value from X’0’ to X’F’, padded with the padding character with a value from X’0’ to X’F’ and may be not encrypted, single encrypted or double encrypted.
WFS_PIN_FORMVISA3	PIN with the length of 4 to 12 digits, each one with a value of X’0’ to X’9’, is followed by a delimiter with the value of X’F’ and then padded by the padding character with a value between X’0’ to X’F’.
WFS_PIN_FORMBANKSYS	PIN is encrypted and formatted according to the Banksys Pin Block specifications.
WFS_PIN_FORMEMV	The PIN block is constructed as follows: PIN is preceded by 0x02 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, formatted up to 248 bytes of other data as defined within the EMV 4.0 specifications and finally encrypted with an RSA key.
WFS_PIN_FORMISO3	PIN is preceded by 0x03 and the length of the PIN (0x04 to 0x0C), padding characters sequentially or randomly chosen, XORed with digits from PAN.

fwDerivationAlgorithms
Supported derivation algorithms; a combination of the following flags:

Value	Meaning
WFS_PIN_CHIP_ZKA	Algorithm for the derivation of a chip card individual key as described by the German ZKA.

fwPresentationAlgorithms
Supported presentation algorithms; a combination of the following flags:

Value	Meaning
WFS_PIN_PRESENT_CLEAR	Algorithm for the presentation of a clear text PIN to a chipcard.

fwDisplay
Specifies the type of the display used in the PIN pad module as one of the following flags:

Value	Meaning
WFS_PIN_DISPNONE	no display unit
WFS_PIN_DISPLEDTHROUGH	lights next to text guide user
WFS_PIN_DISPDISPLAY	a real display is available (this doesn't apply for self-service)

bIDConnect
Specifies whether the PIN pad is directly physically connected to the ID card unit. The value of this parameter is either TRUE or FALSE.

fwIDKey
Specifies whether an ID key is supported as a combination of the following flags:

Value	Meaning
WFS_PIN_IDKEYINITIALIZATION	ID key supported in the WFS_CMD_PIN_INITIALIZATION command.
WFS_PIN_IDKEYIMPORT	ID key supported in the WFS_CMD_PIN_IMPORT_KEY command.

fwValidationAlgorithms
Specifies the algorithms for PIN validation supported by the service; combination of the following flags:

Value	Meaning
WFS_PIN_DES	DES algorithm
WFS_PIN_EUROCHEQUE	EUROCHEQUE algorithm
WFS_PIN_VISA	VISA algorithm
WFS_PIN_DES_OFFSET	DES offset generation algorithm
WFS_PIN_BANKSYS	Banksys algorithm.

fwKeyCheckModes
Specifies the key check modes that are supported to check the correctness of an imported key value; can be a combination of the following flags:

Value	Meaning
WFS_PIN_KCVSELF	The key check value is created by an encryption of the key with itself.
WFS_PIN_KCVZERO	The key check value is created by an encryption of the key with a zero value.

lpszExtra
Points to a list of vendor-specific, or any other extended information. The information is returned as a series of "*key=value*" strings so that it is easily extendable by service providers. Each string is null-terminated, with the final string terminating with two null characters.

For German HSMs this parameter will contain the following information:

- HSM=<HSM vendor> (can contain the values KRONE,ASCOM,IBM or NCR)
- JOURNAL=<0/1> (0 means that the HSM does not support journaling by the WFS_CMD_PIN_GET_JOURNAL command, 1 means it supports journaling)

For Remote Key Loading the following key/value pairs indicate the level of support of the service provider. If these pairs are not returned then this indicates the SP does not support the corresponding feature:

- REMOTE_KEY_SCHEME=<0xnnnnnnnn>, where nnnnnnnn is the ASCII representation of a hexadecimal value. REMOTE_KEY_SCHEME will specify to the user which type(s) of Remote Key Loading/Authentication is supported. For example, "REMOTE_KEY_SCHEME=0x00000002" indicates that three-party certificates are supported. The support level is defined as a combination of the following flags:

Value	Meaning
WFS_PIN_RSA_AUTH_2PARTY_SIG	Two-party Signature based authentication
WFS_PIN_RSA_AUTH_3PARTY_CERT	Three-party Certificate based authentication

- RSA_SIGN_ALGORITHM=<0xnnnnnnnn>, where nnnnnnnn is the ASCII representation of a hexadecimal value. RSA_SIGN_ALGORITHM will specify what type(s) of RSA Signature Algorithms is supported. For example, "RSA_SIGN_ALGORITHM=0x00000001" indicates that RSASSA_PKCS1_V1_5 is supported. The support level is defined as a combination of the following flags:

Value	Meaning
WFS_PIN_SIGN_RSASSA_PKCS1_V1_5	SSA_PKCS_V1_5 Signatures supported
WFS_PIN_SIGN_RSASSA_PSS	SSA_PSS Signatures supported

- RSA_CRYPT_ALGORITHM=<0xnnnnnnnn>, where nnnnnnnn is the ASCII representation of a hexadecimal value. RSA_CRYPT_ALGORITHM will specify what type(s) of RSA encipherment algorithms is supported. For example, "RSA_CRYPT_ALGORITHM=0x00000002" indicates that RSAES_OAEP is supported. The support level is defined as a combination of the following flags:

Value	Meaning
WFS_PIN_CRYPT_RSAES_PKCS1_V1_5	AES_PKCS_V1_5 algorithm supported
WFS_PIN_CRYPT_RSAES_OAEP	AES_OAEP algorithm supported

- RSA_KEY_CHECK_MODE=<0xnnnnnnnn>, where nnnnnnnn is the ASCII representation of a hexadecimal value. RSA_KEY_CHECK_MODE will specify what type of key check value can be returned from a RSA key import function. For example, "RSA_KEY_CHECK_MODE=0x00000001" indicates that SHA1 is supported. The support level is defined as a combination of the following flags:

Value	Meaning
WFS_PIN_RSA_KCV_SHA1	The key check value contains a SHA 1 of the public key as defined in Ref. 3.

- SIGNATURE_CAPABILITIES=<0xnnnnnnnn>, where nnnnnnnn is the ASCII representation of a hexadecimal value. SIGNATURE_CAPABILITIES will specify which capabilities are supported by the Signature scheme. The signature capabilities are defined as a combination of the following flags:

Value	Meaning
WFS_PIN_SIG_GEN_RSA_KEY_PAIR	Specifies if the Service Provider supports the RSA Signature Scheme WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR and WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED commands.
WFS_PIN_SIG_RANDOM_NUMBER	Specifies if the Service Provider returns a random number from the WFS_CMD_PIN_START_KEY_EXCHANGE command within the RSA Signature Scheme.

WFS_PIN_SIG_EXPORT_EPP_ID	Specifies if the Service Provider supports exporting the EPP Security Item within the RSA Signature Scheme.

For EMV support the following key/value pairs indicate the level of support of the service provider. Note that a series of this key/value pairs may occur that lists all import schemes supported by the PIN SP. If these pairs are not returned then this indicates that the SP does not support the corresponding feature.

-EMV_IMPORT_SCHEME=<0xnnnn>, this field will specify to the user how the specified key will be imported. nnnn is the ASCII representation of a single hexadecimal value which defines the import scheme. A series of these pairs may be returned to support multiple import schemes.

The specific values that are used for nnnn are defined within the 'C' include file see section "C – Header File". The following descriptions use the 'C' constant name.

Value	Meaning
WFS_PIN_EMV_IMPORT_PLAIN_CA	A plain text CA public key is imported with no verification.
WFS_PIN_EMV_IMPORT_CHKSUM_CA	A plain text CA public key is imported using the EMV 2000 verification algorithm. See [Ref. 4].
WFS_PIN_EMV_IMPORT_EPI_CA	A CA public key is imported using the self-sign scheme defined in the Europay International, EPI CA Module Technical – Interface specification Version 1.4, [Ref. 5]
WFS_PIN_EMV_IMPORT_ISSUER	An Issuer public key is imported as defined in EMV 2000 Book II, [Ref. 4].
WFS_PIN_EMV_IMPORT_ICC	An ICC public key is imported as defined in EMV 2000 Book II, [Ref. 4].
WFS_PIN_EMV_IMPORT_ICC_PIN	An ICC PIN public key is imported as defined in EMV 2000 Book II, [Ref. 4].
WFS_PIN_EMV_IMPORT_PKCSV1_5_CA	A CA public key is imported and verified using a signature generated with a private key for which the public key is already loaded.

-EMV_HASH=<0xnnnn>, this field will specify to the user which type of Hash Algorithm is supported by the service provider. nnnn is the ASCII representation of the combination of hash algorithms supported by the service provider.

Value	Meaning
WFS_PIN_HASH_SHA1_DIGEST	The SHA 1 digest algorithm is supported by the WFS_CMD_PIN_DIGEST command.

The capabilities associated with key loading in multiple part are defined by the following:

PIN_IMPORT_KEY_PARTS=<0/1> (0 means the device does not support key import in multiple parts, 1 means the device supports key import in multiple parts)

A Service Provider that supports the WFS_CMD_PIN_ENCIO command, shall add information about what protocols it supports as:

- ENCIOPROTOCOLS=0xnnnn

where nnnn is the ASCII representation of the combination of the values supported for the *wProtocol* parameter.

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Comments Applications, which require or expect specific information to be present in the *lpszExtra* parameter may not be device or vendor-independent.

4.3 WFS_INF_PIN_KEY_DETAIL

Description This command returns detailed information about the keys in the encryption module. This command will also return information on symmetric keys loaded during manufacture that can be used by applications. If a public or private key name is specifed this command will return WFS_ERR_PIN_KEYNOTFOUND. If the application wants all keys returned, then all keys except the public or private keys are returned.

Input Param `LPSTR lpsKeyName;`

lpsKeyName
Name of the key for which detailed information is requested.
If NULL, detailed information about all the keys in the encryption module is returned.

Output Param `LPWFSPINKEYDETAIL * lppKeyDetail;`

Pointer to a null-terminated array of pointers to key detail structures.

```
typedef struct _wfs_pin_key_detail
   {
   LPSTR        lpsKeyName;
   WORD         fwUse;
   BOOL         bLoaded;
   } WFSPINKEYDETAIL, * LPWFSPINKEYDETAIL;
```

lpsKeyName
Specifies the name of the key.

fwUse
Specifies the type of access for which the key is used as a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key can be used for encryption/decryption
WFS_PIN_USEFUNCTION	key can be used for PIN functions
WFS_PIN_USEMACING	key can be used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key
WFS_PIN_USECONSTRUCT	key is under construction through the import of multiple parts.

bLoaded
Specifies whether the key has been loaded (imported from Application or locally from Operator) and is either TRUE or FALSE.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key name is not found.

Comments None.

4.4 WFS_INF_PIN_FUNCKEY_DETAIL

Description This command returns information about the names of the Function Keys supported by the device. Location information is also returned for the supported FDKs (Function Descriptor Keys). This includes screen overlay FDKs.

This command should be issued before the first call to WFS_CMD_PIN_GET_PIN or WFS_CMD_PIN_GET_DATA to determine which Function Keys (FKs) and Function Descriptor Keys (FDKs) are available and where the FDKs are located. Then, in these two commands, they can then be specified as Active and Terminate keys and options on the customer screen can be aligned with the active FDKs.

Input Param

```
LPULONG       lpulFDKMask;
```

lpulFDKMask
Mask for the FDKs for which additional information is requested.
If 0x00000000, only information about function keys is returned.
If 0xFFFFFFFF, information about all the supported FDKs is returned.

Output Param

```
LPWFSPINFUNCKEYDETAIL      lpFuncKeyDetail;
```

```
typedef struct _wfs_pin_func_key_detail
   {
   ULONG           ulFuncMask;
   USHORT          usNumberFDKs;
   LPWFSPINFDK     * lppFDKs;
   } WFSPINFUNCKEYDETAIL, * LPWFSPINFUNCKEYDETAIL;
```

ulFuncMask
Specifies the function keys available for this physical device as a combination of the following flags. The defines WFS_PIN_FK_0 through WFS_PIN_FK_9 correspond to numeric digits:

WFS_PIN_FK_0	(numeric digit 0)
WFS_PIN_FK_1	(numeric digit 1)
WFS_PIN_FK_2	(numeric digit 2)
WFS_PIN_FK_3	(numeric digit 3)
WFS_PIN_FK_4	(numeric digit 4)
WFS_PIN_FK_5	(numeric digit 5)
WFS_PIN_FK_6	(numeric digit 6)
WFS_PIN_FK_7	(numeric digit 7)
WFS_PIN_FK_8	(numeric digit 8)
WFS_PIN_FK_9	(numeric digit 9)
WFS_PIN_FK_ENTER	
WFS_PIN_FK_CANCEL	
WFS_PIN_FK_CLEAR	
WFS_PIN_FK_BACKSPACE	
WFS_PIN_FK_HELP	
WFS_PIN_FK_DECPOINT	
WFS_PIN_FK_00	
WFS_PIN_FK_000	
WFS_PIN_FK_RES1	(reserved for future use)
WFS_PIN_FK_RES2	(reserved for future use)
WFS_PIN_FK_RES3	(reserved for future use)
WFS_PIN_FK_RES4	(reserved for future use)
WFS_PIN_FK_RES5	(reserved for future use)
WFS_PIN_FK_RES6	(reserved for future use)
WFS_PIN_FK_RES7	(reserved for future use)
WFS_PIN_FK_RES8	(reserved for future use)

The remaining 6 bit masks may be used as vendor dependent keys.

WFS_PIN_FK_OEM1
WFS_PIN_FK_OEM2
WFS_PIN_FK_OEM3
WFS_PIN_FK_OEM4
WFS_PIN_FK_OEM5
WFS_PIN_FK_OEM6

usNumberFDKs
This value indicates the number of FDK structures returned. This number can be less than the number of keys requested, if any keys are not supported.

lppFDKs
Pointer to an array of pointers to FDK structures. It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

```
typedef struct _wfs_pin_fdk
    {
    ULONG     ulFDK;
    USHORT    usXPosition;
    USHORT    usYPosition;
    } WFSPINFDK, * LPWFSPINFDK;
```

ulFDK
Specifies the code returned by this FDK, defined as one of the following values:

WFS_PIN_FK_FDK01
WFS_PIN_FK_FDK02
WFS_PIN_FK_FDK03
WFS_PIN_FK_FDK04
WFS_PIN_FK_FDK05
WFS_PIN_FK_FDK06
WFS_PIN_FK_FDK07
WFS_PIN_FK_FDK08
WFS_PIN_FK_FDK09
WFS_PIN_FK_FDK10
WFS_PIN_FK_FDK11
WFS_PIN_FK_FDK12
WFS_PIN_FK_FDK13
WFS_PIN_FK_FDK14
WFS_PIN_FK_FDK15
WFS_PIN_FK_FDK16
WFS_PIN_FK_FDK17
WFS_PIN_FK_FDK18
WFS_PIN_FK_FDK19
WFS_PIN_FK_FDK20
WFS_PIN_FK_FDK21
WFS_PIN_FK_FDK22
WFS_PIN_FK_FDK23
WFS_PIN_FK_FDK24
WFS_PIN_FK_FDK25
WFS_PIN_FK_FDK26
WFS_PIN_FK_FDK27
WFS_PIN_FK_FDK28
WFS_PIN_FK_FDK29
WFS_PIN_FK_FDK30
WFS_PIN_FK_FDK31
WFS_PIN_FK_FDK32

usXPosition
For FDKs, specifies the FDK position relative to the Left Hand side of the screen expressed as a percentage of the width of the screen.

usYPosition
For FDKs, specifies the FDK position relative to the top of the screen expressed as a percentage of the height of the screen.

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Comments None.

4.5 WFS_INF_PIN_HSM_TDATA

Description This function returns the current HSM terminal data. The data is returned as a series of "tag/length/value" items.

Input Param None.

Ouput Param `LPWFSXDATA    lpxTData;`

lpxTData
Contains the parameter settings as a series of "tag/length/value" items with no separators. See command WFS_CMD_PIN_HSM_SET_TDATA for the tags supported.

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Comments None.

4.6 WFS_INF_PIN_KEY_DETAIL_EX

Description This command returns extended detailed information about the keys in the encryption module, including DES, private and public keys. Information like generation, version, activating and expiry date can be returned only for keys which are loaded via the WFS_CMD_PIN_SECURE_MSG_SEND command with WFS_PIN_PROTISOPS or a vendor dependant mechanism. This command will also return information on all keys loaded during manufacture that can be used by applications.

Input Param `LPSTR  lpsKeyName;`

lpsKeyName
Name of the key for which detailed information is requested.
If NULL, detailed information about all the keys in the encryption module is returned.

Output Param `LPWFSPINKEYDETAILEX *      lppKeyDetailEx;`

Pointer to a null-terminated array of pointers to key detail structures.

```
typedef struct _wfs_pin_key_detail_ex
   {
   LPSTR        lpsKeyName;
   DWORD        dwUse;
   BYTE         bGeneration;
   BYTE         bVersion;
   BYTE         bActivatingDate[4];
   BYTE         bExpiryDate[4];
   BOOL         bLoaded;
   } WFSPINKEYDETAILEX, * LPWFSPINKEYDETAILEX;
```

lpsKeyName
Specifies the name of the key.

dwUse
Specifies the type of access for which the key is used as a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key can be used for encryption/decryption
WFS_PIN_USEFUNCTION	key can be used for PIN functions
WFS_PIN_USEMACING	key can be used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key

WFS_PIN_USEPINLOCAL	key is used for local PIN check
WFS_PIN_USERSAPUBLIC	key is used as a public key for RSA encryption including EMV PIN block creation
WFS_PIN_USERSAPRIVATE	key is used as a private key for RSA decryption.
WFS_PIN_USERSAPRIVATESIGN	key is used as a private key for RSA Signature generation. Only data generated within the device can be signed.
WFS_PIN_USECHIPINFO	key is used as KGK_{INFO} key (only ZKA standard)
WFS_PIN_USECHIPPIN	key is used as KGK_{PIN} key (only ZKA standard)
WFS_PIN_USECHIPPS	key is used as K_{PS} key (only ZKA standard)
WFS_PIN_USECHIPMAC	key is used as K_{MAC} key (only ZKA standard)
WFS_PIN_USECHIPLT	key is used as KGK_{LT} key (only ZKA standard)
WFS_PIN_USECHIPMACLZ	key is used as K_{PACMAC} key (only ZKA standard)
WFS_PIN_USECHIPMACAZ	key is used as K_{MASTER} key (only ZKA standard)
WFS_PIN_USERSAPUBLICVERIFY	key is used as a public key for RSA signature verification and/or data decryption.
WFS_PIN_USECONSTRUCT	key is under construction through the import of multiple parts. This value can be returned in combination with any one of the other key usage flags

bGeneration
Specifies the generation of the key as BCD value. Will be 0xff if no such information is available for the key.

bVersion
Specifies the version of the key as BCD value. Will be 0xff if no such information is available for the key.

bActivatingDate
Specifies the date when the key is activated as BCD value in the format YYYYMMDD. Will be 0xffffffff if no such information is available for the key.

bExpiryDate
Specifies the date when the key expires as BCD value in the format YYYYMMDD. Will be 0xffffffff if no such information is available for the key.

bLoaded
Specifies whether the key has been loaded (imported from Application or locally from Operator) and is either TRUE or FALSE.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key name is not found.

Comments When the PIN contains a public/private key-pair, only the private part of the key will be reported. Every private key in the PIN will always have a corresponding public key with the same name. The public key can be exported with WFS_CMD_PIN_EXPORT_EPP_SIGNED_ITEM.

5. Execute Commands

5.1 Normal PIN Commands

The following commands are those commands that are used in a normal transaction with the encryptor.

5.1.1 WFS_CMD_PIN_CRYPT

Description The input data is either encrypted or decrypted using the specified or selected encryption mode. The available modes are defined in the WFS_INF_PIN_CAPABILITIES command.

This command can also be used for random number generation.

Furthermore it can be used for Message Authentication Code generation (i.e. MACing). The input data is padded to the necessary length mandated by the encryption algorithm using the *bPadding* parameter. Applications can generate a MAC using an alternative padding method by pre-formatting the data passed and combining this with the standard padding method.

The input data can be expanded with a fill-character to the necessary length (mandated by the encryption algorithm being used).

The Start Value (or Initialization Vector) should be able to be passed encrypted like the specified encryption/decryption key. It would therefore need to be decrypted with a loaded key so the name of this key must also be passed. However, both these parameters are optional.

Input Param LPWFSPINCRYPT lpCrypt;

```
typedef struct _wfs_pin_crypt
   {
   WORD             wMode;
   LPSTR            lpsKey;
   LPWFSXDATA          lpxKeyEncKey;
   WORD             wAlgorithm;
   LPSTR            lpsStartValueKey;
   LPWFSXDATA          lpxStartValue;
   BYTE             bPadding;
   BYTE             bCompression;
   LPWFSXDATA          lpxCryptData;
   } WFSPINCRYPT, * LPWFSPINCRYPT;
```

wMode
Specifies whether to encrypt or decrypt, values are one of the following:

Value	Meaning
WFS_PIN_MODEENCRYPT	encrypt with key
WFS_PIN_MODEDECRYPT	decrypt with key
WFS_PIN_MODERANDOM	an 8 byte random value shall be returned (in this case all the other input parameters are ignored)

This parameter does not apply to MACing.

lpsKey
Specifies the name of the stored key. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpxKeyEncKey
If NULL, *lpsKey* is used directly for encryption/decryption. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for encryption/decryption. Key is a double length key when used for Triple DES encryption/decryption. Users of this specification must adhere to local regulations when using Triple DES. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

wAlgorithm
Specifies the encryption algorithm. Possible values are those described in

WFS_INF_PIN_CAPABILITIES. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpsStartValueKey
Specifies the name of the stored key used to decrypt the *lpxStartValue* to obtain the Initialization Vector. If this parameter is NULL, *lpxStartValue* is used as the Initialization Vector. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpxStartValue
DES and Triple DES initialization vector for CBC / CFB encryption and MACing. If this parameter is NULL *lpsStartValueKey* is used as the Start Value. If *lpsStartValueKey* is also NULL, the default value for CBC / CFB / MAC is 16 hex digits 0x0. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

bPadding
Specifies the padding character for encryption. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

bCompression
Specifies whether data is to be compressed (blanks removed) before building the MAC. If *bCompression* is 0x00 no compression is selected, otherwise *bCompression* holds the representation of the blank character in the actual code table. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpxCryptData
Pointer to the data to be encrypted, decrypted, or MACed. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

Output Param `LPWFSXDATA    lpxCryptData;`

lpxCryptData
Pointer to the encrypted or decrypted data, MAC value or 8 byte random value.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_MODENOTSUPPORTED	The specified mode is not supported.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* or *lpxStartValue* is not supported.
WFS_ERR_PIN_NOCHIPTRANSACTIVE	A chipcard key is used as encryption key and there is no chip transaction active.
WFS_ERR_PIN_ALGORITHMNOTSUPP	The specified algorithm is not supported.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments The data type LPWFSXDATA is used to pass hexadecimal data and is defined as follows:

```
typedef struct _wfs_hex_data
    {
    USHORT    usLength;
    LPBYTE    lpbData;
    } WFSXDATA, *LPWFSXDATA;
```

usLength
Length of the byte stream pointed to by *lpbData.*

lpbData
Pointer to the binary data stream.

5.1.2 WFS_CMD_PIN_IMPORT_KEY

Description The key passed by the application is loaded in the encryption module. The key can be passed in clear text mode or encrypted with an accompanying “key encryption key”. A key can be loaded in multiple unencrypted parts by combining the WFS_PIN_USECONSTRUCT value with the final usage flags within the *fwUse* field.

Input Param

```
LPWFSPINIMPORT      lpImport;

typedef struct _wfs_pin_import
   {
   LPSTR            lpsKey;
   LPSTR            lpsEncKey;
   LPWFSXDATA       lpxIdent;
   LPWFSXDATA       lpxValue;
   WORD             fwUse;
   } WFSPINIMPORT, * LPWFSPINIMPORT;
```

lpsKey
Specifies the name of key being loaded.

lpsEncKey
lpsEncKey specifies a key name or a format name which were used to encrypt the key passed in *lpxValue*. If *lpsEncKey* is NULL the key is loaded directly into the encryption module. lpsEncKey must be NULL if *fwUse* contains WFS_PIN_USECONSTRUCT.

lpxIdent
Specifies the key owner identification. The use of this parameter is vendor dependent.

lpxValue
Specifies the value of key to be loaded.

fwUse
Specifies the type of access for which the key can be used as a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key can be used for encryption/decryption
WFS_PIN_USEFUNCTION	key can be used for PIN functions
WFS_PIN_USEMACING	key can be used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key
WFS_PIN_USECONSTRUCT	key is under construction through the import of multiple parts. This value is used in combination with the actual usage flags for the key.

If *fwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* are ignored.

Output Param

```
LPWFSXDATA    lpxKVC;
```

lpxKVC
pointer to the key verification code data that can be used for verification of the loaded key, NULL if device does not have that capability.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key encryption key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_INVALIDID	The ID passed was not valid.

WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOVALUE	The specified key encryption key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments When keys are loaded in multiple parts, all parts of the key loaded must set the WFS_PIN_USECONSTRUCT value in the *fwUse* field along with any usage's needed for the final key use. The usage flags must be consistent for all parts of the key. Activation of the key entered in multiple parts is indicated through an additional final call to this command, where WFS_USECONSTRUCT is removed from *fwUse* but those other usage's defined during the key part loading must still be used. No key data is passed during the final activation of the key.

The optional KCV is only returned during the final activation step. Applications wishing to verify the KCV for each key part will need to load each key part into a temporary location inside the encryptor. If the application determines the KCV of the key part is valid, then the application calls the WFS_CMD_PIN_IMPORT_KEY again to load the key part into the device. The application should delete the temporary key part as soon as the KCV for that key part has been verified.

When the first part of the key is received, it is stored directly in the device. All subsequent parts are combined with the existing value in the device through XOR. No sub-parts of the key are maintained separately. While a key still has a *fwUse* value that is combined with WFS_PIN_USECONSTRUCT (i.e. it is still being loaded), it cannot be used for cryptographic functions.

5.1.3 WFS_CMD_PIN_DERIVE_KEY

Description A key is derived from input data using a key generating key and an initialization vector. The input data can be expanded with a fill-character to the necessary length (mandated by the encryption algorithm being used). The derived key is imported into the encryption module and is used for encryption or decryption operations.

Input Param

```
LPWFSPINDERIVE      lpDerive;

typedef struct _wfs_pin_derive
   {
   WORD             wDerivationAlgorithm;
   LPSTR            lpsKey;
   LPSTR            lpsKeyGenKey;
   LPSTR            lpsStartValueKey;
   LPWFSXDATA          lpxStartValue;
   BYTE             bPadding;
   LPWFSXDATA          lpxInputData;
   LPWFSXDATA          lpxIdent;
   } WFSPINDERIVE, * LPWFSPINDERIVE;
```

wDerivationAlgorithm
Specifies the algorithm that is used for derivation. Possible values are:
(see command WFS_INF_PIN_CAPABILITIES)

lpsKey
Specifies the name where the derived key will be stored.

lpsKeyGenKey
Specifies the name of the key generating key that is used for the derivation.

lpsStartValueKey
Specifies the name of the stored key used to decrypt the *lpxStartValue* to obtain the

Initialization Vector. If this parameter is NULL, *lpxStartValue* is used as the Initialization Vector.

lpxStartValue
DES initialization vector for the encryption step within the derivation.

bPadding
Specifies the padding character for the encryption step within the derivation.

lpxInputData
Pointer to the data to be used for key derivation.

lpxIdent
Specifies the key owner identification. The use of this parameter is vendor dependent.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized (or not ready for some vendor specific reason).
WFS_ERR_PIN_INVALIDID	The ID passed was not valid.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxStartValue* is not supported.
WFS_ERR_PIN_ALGORITHMNOTSUPP	The specified algorithm is not supported.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.4 WFS_CMD_PIN_GET_PIN

Description This function stores the PIN entry via the PIN pad. From the point this function is invoked, PIN digit entries are ***not*** passed to the application. For each PIN digit, or any other active key entered, an execute notification event is sent in order to allow an application to perform the appropriate display action (i.e. when the PIN pad has no integrated display). The application is not informed of the value entered, the execute notification only informs that a key has been depressed.

Some PIN pad devices do not inform the application as each PIN digit is entered, but locally process the PIN entry based upon minimum PIN length and maximum PIN length input parameters. These PIN pad devices which provide local PIN entry management and optional display tracking may or may not notify the application of a minimum PIN length violation.

When the maximum number of PIN digits is entered, or a completion key is pressed after the minimum number of PIN digits is entered, a WFS_EXEC_COMPLETE event message is sent to the application. Once this notification is received, the output parameters are then returned to the application from this function call. The depression of the <Cancel> key is also passed to the application via the WFS_EXEC_COMPLETE event message.

If *usMaxLen* is zero, the service provider does not terminate the command unless the application sets *ulTerminateKeys* or *ulTerminateFDKs*. In the event that *ulTerminateKeys* or *ulTerminateFDKs* are not set and *usMaxLen* is zero, the command will not terminate and the application must issue a WFSCancel command.

Terminating keys have to be active keys to operate.

If this command is cancelled by a WFSCancelAsyncRequest or a WFSCancelBlockingCall the PIN buffer is not cleared.

It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

Input Param

```
LPWFSPINGETPIN      lpGetPin;
```

```
typedef struct _wfs_pin_getpin
   {
   USHORT    usMinLen;
   USHORT    usMaxLen;
   BOOL      bAutoEnd;
   CHAR      cEcho;
   ULONG     ulActiveFDKs;
   ULONG     ulActiveKeys;
   ULONG     ulTerminateFDKs;
   ULONG     ulTerminateKeys;
   } WFSPINGETPIN, * LPWFSPINGETPIN;
```

usMinLen
Specifies the minimum number of digits which must be entered for the PIN. A value of zero indicates no minimum PIN length verification.

usMaxLen
Specifies the maximum number of digits which can be entered for the PIN.

bAutoEnd
If *bAutoEnd* is set to true, the service provider terminates the command when the maximum number of digits are entered. Otherwise, the input is terminated by the user using one of the termination keys. When *usMaxLen* is reached, the service provider will disable all numeric keys. *bAutoEnd* is ignored when *usMaxLen* is set to 0.

cEcho
Specifies the replace character to be echoed on a local display for the PIN digit.

ulActiveFDKs
Specifies those FDKs which are active during the execution of the command.

ulActiveKeys
Specifies those (other) Function Keys which are active during the execution of the command.

ulTerminateFDKs
Specifies those FDKs which must terminate the execution of the command.

ulTerminateKeys
Specifies those (other) Function Keys which must terminate the execution of the command.

Output Param

```
LPWFSPINENTRY lpEntry;
```

```
typedef struct _wfs_pin_entry
   {
   USHORT    usDigits;
   WORD      wCompletion;
   } WFSPINENTRY, * LPWFSPINENTRY;
```

usDigits
Specifies the number of PIN digits entered.

wCompletion
Specifies the reason for completion of the entry. Possible values are:

Value	Meaning
WFS_PIN_COMPAUTO	The command terminated automatically, because maximum PIN length was reached.
WFS_PIN_COMPENTER	The ENTER Function Key was pressed as terminating key.
WFS_PIN_COMPCANCEL	The CANCEL Function Key was pressed as terminating key.
WFS_PIN_COMPCONTINUE	Input continues, function key was pressed (this value is only used in the execute event WFS_EXEE_PIN_KEY).
WFS_PIN_COMPCLEAR	The CLEAR Function Key was pressed as terminating key and the previous input is cleared.
WFS_PIN_COMPBACKSPACE	The last input digit was cleared and the key was pressed as terminating key.
WFS_PIN_COMPFDK	Indicates input is terminated only if the FDK pressed was set to be a terminating FDK.
WFS_PIN_COMPHELP	The HELP Function Key was pressed as terminating key.
WFS_PIN_COMPFK	A Function Key (FK) other than ENTER, CLEAR, CANCEL, BACKSPACE, HELP was pressed as terminating key.
WFS_PIN_COMPCONTFDK	Input continues, FDK was pressed (this value is only used in the execute event WFS_EXEE_PIN_KEY).

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYINVALID	At least one of the specified function keys or FDKs is invalid.
WFS_ERR_PIN_KEYNOTSUPPORTED	At least one of the specified function keys or FDKs is not supported by the service provider.
WFS_ERR_PIN_NOACTIVEKEYS	There are no active function keys specified.
WFS_ERR_PIN_NOTERMINATEKEYS	There are no terminate keys specified and *usMaxLen* is not set to 0 and *bAutoEnd* is FALSE.
WFS_ERR_PIN_MINIMUMLENGTH	The minimum PIN length field is invalid or greater than the maximum PIN length field.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_EXEE_PIN_KEY	A key has been pressed at the PIN pad.

Comments None.

5.1.5 WFS_CMD_PIN_LOCAL_DES

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the DES validation algorithm and locally verified for correctness. The local DES verification is based on the IBM 3624 standard. The result of the verification is returned to the application. This command will clear the PIN.

Input Param

```
LPWFSPINLOCALDES    lpLocalDES;

typedef struct _wfs_pin_local_des
   {
   LPSTR        lpsValidationData;
   LPSTR        lpsOffset;
   BYTE         bPadding;
   USHORT       usMaxPIN;
   USHORT       usValDigits;
   BOOL         bNoLeadingZero;
   LPSTR        lpsKey;
   LPWFSXDATA       lpxKeyEncKey;
   LPSTR        lpsDecTable;
   } WFSPINLOCALDES, * LPWFSPINLOCALDES;
```

lpsValidationData
Validation data

lpsOffset
Offset for the PIN block; if NULL then no offset is used.

bPadding
Specifies the padding character for validation data.

usMaxPIN
Maximum number of PIN digits to be used for validation.

usValDigits
Number of Validation digits to be used for validation.

bNoLeadingZero
If set to TRUE and the first digit of result of the modulo 10 addition is a X'0', it is replaced with X'1' before performing the verification against the entered PIN. If set to FALSE, a leading zero is allowed in entered PINs.

lpsKey
Name of the validation key

lpxKeyEncKey
If NULL, *lpsKey* is used directly for PIN validation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for PIN validation.

lpsDecTable
ASCII decimalization table (16 character string containing characters '0' to '9'). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Output Param

```
LPBOOL       lpbResult;
```

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* is not supported.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.6 WFS_CMD_PIN_CREATE_OFFSET

Description This function is used to generate a PIN Offset that is used to verify PINs using the WFS_CMD_PIN_LOCAL_DES execute command. The PIN offset is computed by combining validation data with the keypad entered PIN. This command will clear the PIN.

Input Param

```
LPWFSPINCREATEOFFSET       lpPINOffset;

typedef struct _wfs_pin_create_offset
   {
   LPSTR        lpsValidationData;
   BYTE         bPadding;
   USHORT       usMaxPIN;
   USHORT       usValDigits;
   LPSTR        lpsKey;
   LPWFSXDATA       lpxKeyEncKey;
   LPSTR        lpsDecTable;
   } WFSPINCREATEOFFSET, * LPWFSPINCREATEOFFSET;
```

lpsValidationData
Validation data

bPadding
Specifies the padding character for validation data.

usMaxPIN
Maximum number of PIN digits to be used for PIN Offset creation.

usValDigits
Number of Validation Data digits to be used for PIN Offset creation.

lpsKey
Name of the validation key

lpxKeyEncKey
If NULL, *lpsKey* is used directly in PIN Offset creation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used in PIN Offset creation.

lpsDecTable
ASCII decimalization table (16 character string containing characters '0' to '9'). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Output Param

```
LPSTR  lpsOffset;
```

lpsOffset
Computed PIN Offset.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_NOTALLOWED	PIN entered by the user is not allowed.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments The list of ‘forbidden’ PINs (values that cannot be chosen as a PIN, e.g. 1111) is configured in the device in a vendor dependent way during the configuration of the system.

5.1.7 WFS_CMD_PIN_LOCAL_EUROCHEQUE

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the Eurocheque validation algorithm and locally verified for correctness. The result of the verification is returned to the application. This command will clear the PIN.

Input Param

```
LPWFSPINLOCALEUROCHEQUE    lpLocalEurocheque;
```

```
typedef struct _wfs_pin_local_eurocheque
   {
   LPSTR        lpsEurochequeData;
   LPSTR        lpsPVV;
   WORD         wFirstEncDigits;
   WORD         wFirstEncOffset;
   WORD         wPVVDigits;
   WORD         wPVVOffset;
   LPSTR        lpsKey;
   LPWFSXDATA       lpxKeyEncKey;
   LPSTR        lpsDecTable;
   } WFSPINLOCALEUROCHEQUE, * LPWFSPINLOCALEUROCHEQUE;
```

lpsEurochequeData
Track-3 Eurocheque data

lpsPVV
PIN Validation Value from track data.

wFirstEncDigits
Number of digits to extract after first encryption.

wFirstEncOffset
Offset of digits to extract after first encryption.

wPVVDigits
Number of digits to extract for PVV.

wPVVOffset
Offset of digits to extract for PVV.

lpsKey
Name of the validation key.

lpxKeyEncKey
If NULL, *lpsKey* is used directly for PIN validation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for PIN validation.

lpsDecTable
ASCII decimalization table (16 character string containing characters ‘0’ to ‘9’). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Output Param

```
LPBOOL       lpbResult;
```

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* is not supported.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.8 WFS_CMD_PIN_LOCAL_VISA

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the VISA validation algorithm and locally verified for correctness. The result of the verification is returned to the application. This command will clear the PIN.

Input Param

```
LPWFSPINLOCALVISA   lpLocalVISA;

typedef struct _wfs_pin_local_visa
   {
   LPSTR        lpsPAN;
   LPSTR        lpsPVV;
   WORD         wPVVDigits;
   LPSTR        lpsKey;
   LPWFSXDATA       lpxKeyEncKey;
   } WFSPINLOCALVISA, * LPWFSPINLOCALVISA;
```

lpsPAN
Primary Account Number from track data.

lpsPVV
PIN Validation Value from track data.

wPVVDigits
Number of digits of PVV.

lpsKey
Name of the validation key.

lpxKeyEncKey
If NULL, *lpsKey* is used directly for PIN validation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for PIN validation.

Output Param

```
LPBOOL       lpbResult;
```

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* is not supported.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.9 WFS_CMD_PIN_PRESENT_IDC

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the IDC presentation algorithm and presented to the smartcard contained in the ID Card unit. The result of the presentation is returned to the application. This command will clear the PIN.

Input Param

```
LPWFSPINPRESENTIDC  lpPresentIDC;

typedef struct _wfs_pin_presentidc
   {
   WORD          wPresentAlgorithm;
   WORD          wChipProtocol;
   ULONG         ulChipDataLength;
   LPBYTE        lpbChipData;
   LPVOID        lpAlgorithmData;
   } WFSPINPRESENTIDC, * LPWFSPINPRESENTIDC;
```

wPresentAlgorithm
Specifies the algorithm that is used for presentation. Possible values are: (see command WFS_INF_PIN_CAPABILITIES).

wChipProtocol
Identifies the protocol that is used to communicate with the chip. Possible values are: (see command WFS_INF_IDC_CAPABILITIES in the Identification Card Device Class Interface).

ulChipDataLength
Specifies the length of the byte stream pointed to by *lpbChipData*.

lpbChipData
Points to the data to be sent to the chip.

lpAlgorithmData
Pointer to a structure that contains the data required for the specified presentation algorithm. For the WFS_PIN_PRESENT_CLEAR algorithm, this structure is defined as:

```
typedef struct _wfs_pin_presentclear
  {
  ULONG         ulPINPointer;
  USHORT        usPINOffset;
  } WFSPINPRESENTCLEAR, * LPWFSPINPRESENTCLEAR;
```

ulPINPointer
Describes the byte position where to insert the PIN in the *lpbChipData* buffer. The first byte of the *lpbChipData* buffer is numbered 0.

usPINOffset
Describes the bit position where to insert the PIN in the *lpbChipData* buffer. In each byte, the most-significant bit is numbered 0, the less significant bit is numbered 7.

Output Param

```
LPWFSPINPRESENTRESULT      lpPresentResult;

typedef struct _wfs_pin_present_result
   {
   WORD      wChipProtocol;
   ULONG     ulChipDataLength;
   LPBYTE    lpbChipData;
   } WFSPINPRESENTRESULT, * LPWFSPINPRESENTRESULT;
```

wChipProtocol
Identifies the protocol that was used to communicate with the chip. This field contains the same value as the corresponding field in the input structure.

ulChipDataLength
Specifies the length of the byte stream pointed to by *lpbChipData*.

lpbChipData
Points to the data responded from the chip.

Error Codes

In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The ID card unit is not ready for PIN presentation or for any vendor specific reason. The ID card service provider, if any, may have generated a service event that further describes the reason for that error code.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_PROTOCOLNOTSUPP	The specified protocol is not supported by the service provider.
WFS_ERR_PIN_INVALIDDATA	An error occurred while communicating with the chip.

Events

Only the generic events defined in [Ref. 1] can be generated by this command.

Comments

None.

5.1.10 WFS_CMD_PIN_GET_PINBLOCK

Description

This function takes the account information and a PIN entered by the user to build a formatted PIN. Encrypting this formatted PIN once or twice returns a PIN block which can be written on a magnetic card or sent to a host. The PIN block can be calculated using one of the formats specified in the WFS_INF_PIN_CAPABILITIES command. This command clears the PIN.

Input Param

```
LPWFSPINBLOCK lpPinBlock;

typedef struct _wfs_pin_block
   {
   LPSTR        lpsCustomerData;
   LPSTR        lpsXORData;
   BYTE         bPadding;
   WORD         wFormat;
   LPSTR        lpsKey;
   LPSTR        lpsKeyEncKey;
   } WFSPINBLOCK, * LPWFSPINBLOCK;
```

lpsCustomerData
Used for ANSI, ISO-0 and ISO-1 algorithm to build the formatted PIN. For ANSI and ISO-0 the PAN (Primary Account Number) is used, for ISO-1 a ten digit transaction field is required. If not used a NULL is required.
Used for DIEBOLD with coordination number, as a two digit coordination number.

Used for EMV with challenge number (8 bytes) coming from the chip card. This number is passed as unpacked string, for example: 0123456789ABCDEF = 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45 0x46

lpsXORData
If the formatted PIN is encrypted twice to build the resulting PIN block, this data can be used to modify the result of the first encryption by an XOR-operation.

bPadding
Specifies the padding character.

wFormat
Specifies the format of the PIN block. Possible values are:
(see command WFS_INF_PIN_CAPABILITIES)

lpsKey
Specifies the key used to encrypt the formatted pin for the first time, NULL if no encryption is required. If this specifies a double length key, triple DES encryption will be performed. If this specifies an RSA key, RSA encryption will be performed.

lpsKeyEncKey
Specifies the key used to format the once encrypted formatted PIN, NULL if no second encryption required.

Output Param `LPWFSXDATA    lpxPinBlock;`

lpxPinBlock
Pointer to the encrypted PIN Block.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has been cleared.
WFS_ERR_PIN_FORMATNOTSUPP	The specified format is not supported.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.11 WFS_CMD_PIN_GET_DATA

Description This function is used to return keystrokes entered by the user. It will automatically set the PIN pad to echo characters on the display if there is a display. For each keystroke an execute notification event is sent in order to allow an application to perform the appropriate display action (i.e. when the PIN pad has no integrated display).

If *usMaxLen* is zero, the service provider does not terminate the command unless the application sets *ulTerminateKeys* or *ulTerminateFDKs*. In the event that *ulTerminateKeys* or *ulTerminateFDKs* are not set and *usMaxLen* is zero, the command will not terminate and the application must issue a WFSCancel command.

Terminating keys have to be active keys to operate.

It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

The following keys may effect the contents of the WFSPINDATA output parameter but are not returned in it:

WFS_PIN_FK_ENTER
WFS_PIN_FK_CANCEL
WFS_PIN_FK_CLEAR
WFS_PIN_FK_BACKSPACE

The WFS_PIN_FK_CANCEL and WFS_PIN_FK_CLEAR keys will cause the output buffer to be cleared. The WFS_PIN_FK_BACKSPACE key will cause the last key in the buffer to be removed.

Input Param

```
LPWFSPINGETDATA     lpPinGetData;
```

```
typedef struct _wfs_pin_getdata
    {
    USHORT    usMaxLen;
    BOOL      bAutoEnd;
    ULONG     ulActiveFDKs;
    ULONG     ulActiveKeys;
    ULONG     ulTerminateFDKs;
    ULONG     ulTerminateKeys;
    } WFSPINGETDATA, * LPWFSPINGETDATA;
```

usMaxLen
Specifies the maximum number of digits which can be returned to the application in the output parameter.

bAutoEnd
If *bAutoEnd* is set to true, the service provider terminates the command when the maximum number of digits are entered. Otherwise, the input is terminated by the user using one of the termination keys. When *usMaxLen* is reached, the service provider will disable all numeric keys. *bAutoEnd* is ignored when *usMaxLen* is set to 0.

ulActiveFDKs
Specifies those FDKs which are active during the execution of the command.

ulActiveKeys
Specifies those (other) Function Keys which are active during the execution of the command.

ulTerminateFDKs
Specifies those FDKs which must terminate the execution of the command.

ulTerminateKeys
Specifies those (other) Function Keys which must terminate the execution of the command.

Output Param

```
LPWFSPINDATA  lpPinData;
```

```
typedef struct _wfs_pin_data
    {
    USHORT           usKeys;
    LPWFSPINKEY *    lpPinKeys;
    WORD             wCompletion;
    } WFSPINDATA, * LPWFSPINDATA;
```

usKeys
Number of keys entered by the user (i.e. number of following WFSPINKEY structures).

lpPinKeys
Pointer to an array of pointers to WFSPINKEY structures that contain the keys entered by the user (for a description of the WFSPINKEY structure see the definition of the WFS_EXEE_PIN_KEY event).

wCompletion
Specifies the reason for completion of the entry. Possible values are:
(see command WFS_CMD_PIN_GET_PIN)

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYINVALID	At least one of the specified function keys or FDKs is invalid.
WFS_ERR_PIN_KEYNOTSUPPORTED	At least one of the specified function keys or FDKs is not supported by the service provider.
WFS_ERR_PIN_NOACTIVEKEYS	There are no active function keys specified.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_EXEE_PIN_KEY	A key has been pressed at the PIN pad.

Comments If the triple zero key is pressed one WFS_EXEE_PIN_KEY event is sent that contains the WFS_PIN_FK_000 code.

If the triple zero key is pressed when 3 keys are already inserted and *usMaxLen* equals 4 the key is not accepted and no event is sent to the application.

If the backspace key is pressed after the triple zero key only one zero is deleted out of the buffer.

Double zero is handled similar to this.

5.1.12 WFS_CMD_PIN_INITIALIZATION

Description The encryption module must be initialized before any encryption function can be used. Every call to WFS_CMD_PIN_INITIALIZATION destroys all application keys that have been loaded or imported, it does not affect those keys loaded during manufacturing. Usually this command is called by an operator task and not by the application program.

Initialization also involves loading “initial” application keys and local vendor dependent keys. These can be supplied, for example, by an operator through a keyboard, a local configuration file, remote RSA key management or possibly by means of some secure hardware that can be attached to the device. The application “initial” keys would normally get updated by the application during a WFS_CMD_PIN_IMPORT_KEY command as soon as possible. Local vendor dependent static keys (e.g. storage, firmware and offset keys) would normally be transparent to the application and by definition can not be dynamically changed.

Where initial keys are not available immediately when this command is issued (i.e. when operator intervention is required), the Service Provider returns WFS_ERR_PIN_ACCESS_DENIED and the application must await the WFS_SRVE_PIN_INITIALIZED event.

During initialization an optional encrypted ID key can be stored in the HW module. The ID key and the corresponding encryption key can be passed as parameters; if not, they are generated automatically by the encryption module. The encrypted ID is returned to the application and serves as authorization for the key import function. The WFS_INF_PIN_CAPABILITIES command indicates whether or not the device will support this feature.

This function also resets the HSM terminal data, except session key index and trace number.

This function resets all certificate data and authentication public/private keys back to their initial states at the time of production. Key-pairs created with WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR are deleted. Any keys installed during production, which have been permanently replaced, will not be reset. Any Verification certificates that may have been loaded must be reloaded. The Certificate state will remain the same, but the WFS_CMD_PIN_LOAD_CERTIFICATE or WFS_CMD_REPLACE_CERTIFICATE commands must be called again.

Input Param

```
LPWFSPININIT  lpInit;

typedef struct _wfs_pin_init
   {
   LPWFSXDATA       lpxIdent;
```

```
    LPWFSXDATA       lpxKey;
    } WFSPININIT, * LPWFSPININIT;
```

lpxIdent
Pointer to the value of the ID key. Null if not required.

lpxKey
Pointer to the value of the encryption key. Null if not required.

Output Param `LPWFSXDATA    lpxIdentification;`

lpxIdentification
Pointer to the value of the ID key encrypted by the encryption key. Can be used as authorization for the WFS_CMD_PIN_IMPORT_KEY command, can be NULL if no authorization required.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized (or not ready for some vendor specific reason).
WFS_ERR_PIN_INVALIDID	The ID passed was not valid.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_INITIALIZED	The encryption module is now initialized.
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.13 WFS_CMD_PIN_LOCAL_BANKSYS

Description The PIN Block previously built by the WFS_CMD_PIN_GET_PINBLOCK according to the BANKSYS specifications is combined with the ATMVAC code for local validation.

Input Param `LPWFSPINLOCALBANKSYS              lpLocalBanksys;`

```
typedef struct _wfs_pin_local_banksys
    {
    LPWFSXDATA       lpxATMVAC;
    } WFSPINLOCALBANKSYS, * LPWFSPINLOCALBANKSYS;
```

lpxATMVAC
The ATMVAC code calculated by the BANKSYS Security Control Module.

Output Param `LPBOOL        lpbResult;`

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared without building the Banksys PIN Block.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxATMVAC* is not supported.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.14 WFS_CMD_PIN_BANKSYS_IO

Description This command sends a single command to the Banksys Security Control Module.

Input Param

```
LPWFSPINBANKSYSIO   lpBANKSYSIoIn;

typedef struct _wfs_pin_BANKSYS_io
    {
    ULONG     ulLength;
    LPBYTE    lpbData;
    } WFSPINBANKSYSIO, * LPWFSPINBANKSYSIO;
```

ulLength
Specifies the length of the following field *lpbData*.

lpbData
Points to the data sent to the BANKSYS Security Control Module.

Output Param

```
LPWFSPINBANKSYSIO          lpBANKSYSIoOut;

typedef struct _wfs_pin_BANKSYS_io
    {
    ULONG     ulLength;
    LPBYTE    lpbData;
    } WFSPINBANKSYSIO, * LPWFSPINBANKSYSIO;
```

ulLength
Specifies the length of the following field *lpbData*.

lpbData
Points to the data responded by the BANKSYS Security Control Module.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_INVALIDDATA	An error occurred while communicating with the device.

Events Only the generic events defined in [Ref. 1] can be generated by this command.

Comments The Banksys command and response message data are defined in the Banksys document “SCM DKH Manual Rel 2.x ”

5.1.15 WFS_CMD_PIN_RESET

Description Sends a service reset to the service provider.

Input Param None

Output Param None.

Error Codes Only the generic error codes defined in [Ref. 1] can be generated by this command.

Events Only the generic events defined in [Ref. 1] can be generated by this command.

Comments This command is used by an application control program to cause a device to reset itself to a known good condition. It does not delete any keys.

5.1.16 WFS_CMD_PIN_HSM_SET_TDATA

Description This function allows to set the HSM terminal data except keys, trace number and session key index. The data must be provided as a series of “tag/length/value” items.
Terminal data that are set but are not supported by the hardware will be ignored.

Input Param `LPWFSXDATA    lpxTData;`

lpxTData
Specifies which parameter(s) is(are) to be set. lpxTData is a series of “tag/length/value” items where each item consists of
- one byte tag (see the list of tags below),
- one byte specifying the length of the following data as an unsigned binary number
- n bytes data (see the list below for formatting)

with no separators.

The following tags are supported:

tag (hexadecimal)	Format	Length (in bytes)	Meaning
C2	BCD	4	Terminal ID ISO BMP 41
C3	BCD	4	Bank code ISO BMP 42 (rightmost 4 bytes)
C4	BCD	9	Account data for terminal account ISO BMP 60 (load against other card)
C5	BCD	9	Account data for fee account ISO BMP 60 ("Laden vom Kartenkonto")
C6	EBCDIC	40	Terminal location ISO BMP 43
C7	ASCII	3	Terminal currency
C8	BCD	7	Online date and time (YYYYMMDDHHMMSS) ISO BMP 61
C9	BCD	4	Minimum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42,
CA	BCD	4	Maximum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42,
CB	BIN	3	logical HSM binary coded serial number (starts with 1; 0 means that there are no logical HSMs).
CC	EBCDIC	16	ZKA ID (is filled during the pre-initialisation of the HSM).

CD	BIN	1	HSM status (1 = irreversibly out of order 2 = out of order, K_UR is not loaded 3 = not pre-initialized, K_UR is loaded 4 = pre-initialized, K_INIT is loaded 5 = initialized/personalized, K_PERS is loaded).
CE	EBCDIC	variable, min. 16	HSM-ID (6 byte Manufacturer- ID + min. 10 Byte serial number), as needed for ISO BMP57 of a pre-initialisation

The parameters CB, CC, CD and CE cannot be set. They can only be read using the command WFS_INF_PIN_HSM_TDATA.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this command.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_HSM_TDATA_CHANGED	The terminal data has changed.

Comments None.

5.1.17 WFS_CMD_PIN_SECURE_MSG_SEND

Description This command handles all messages that should be send through a secure messaging to a authorization system, German "Ladezentrale", personalization system or the chip. The encryption module adds the security relevant fields to the message and returns the modified message in the output structure. All messages must be presented to the encryptor via this command even if they do not contain security fields in order to keep track of the transaction status in the internal state machine.

Input Param

```
LPWFSPINSECMSG      lpSecMsgIn;

typedef struct _wfs_pin_secure_message
   {
   WORD      wProtocol;
   ULONG     ulLength;
   LPBYTE    lpbMsg;
   } WFSPINSECMSG, * LPWFSPINSECMSG;
```

wProtocol
Specifies the protocol the message belongs to. Specified as one of the following flags:

Value	Meaning
WFS_PIN_PROTISOAS	ISO 8583 protocol for the authorization system
WFS_PIN_PROTISOLZ	ISO 8583 protocol for the German "Ladezentrale"
WFS_PIN_PROTISOPS	ISO 8583 protocol for the personalization system
WFS_PIN_PROTCHIPZKA	ZKA chip protocol
WFS_PIN_PROTRAWDATA	raw data protocol

WFS_PIN_PROTPBM	PBM protocol (see [Ref. 8] –[Ref. 13])
WFS_PIN_PROTHSMLDI	HSM LDI protocol

ulLength
Specifies the length in bytes of the message in *lpbMsg*. This parameter is ignored for the WFS_PIN_PROTHSMLDI protocol.

lpbMsg
Specifies the message that should be send. This parameter is ignored for the WFS_PIN_PROTHSMLDI protocol.

Output Param
```
LPWFSPINSECMSG      lpSecMsgOut;
```

lpSecMsgOut
pointer to a WFSPINSECMSG structure that contains the modified message that can now be send to a authorization system, German "Ladezentrale", personalization system or the chip.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this message.
WFS_ERR_PIN_PROTINVALID	The specified protocol is invalid.
WFS_ERR_PIN_CONTENTINVALID	The contents of one of the security relevant fields are invalid.

Events Only the generic events defined in [Ref. 1] can be generated by this command.

Comments None.

5.1.18 WFS_CMD_PIN_SECURE_MSG_RECEIVE

Description This command handles all messages that are received through a secure messaging from a authorization system, German "Ladezentrale", personalization system or the chip. The encryption module checks the security relevant fields. All messages must be presented to the encryptor via this command even if they do not contain security relevant fields in order to keep track of the transaction status in the internal state machine.

Input Param
```
LPWFSPINSECMSG      lpSecMsgIn;

typedef struct _wfs_pin_secure_message
   {
   WORD      wProtocol;
   ULONG     ulLength;
   LPBYTE    lpbMsg;
   } WFSPINSECMSG, * LPWFSPINSECMSG;
```

wProtocol
Specifies the protocol the message belongs to. Specified as one of the following flags:

Value	Meaning
WFS_PIN_PROTISOAS	ISO 8583 protocol for the authorization system
WFS_PIN_PROTISOLZ	ISO 8583 protocol for the German "Ladezentrale"
WFS_PIN_PROTISOPS	ISO 8583 protocol for the personalization system
WFS_PIN_PROTCHIPZKA	ZKA chip protocol
WFS_PIN_PROTRAWDATA	raw data protocol
WFS_PIN_PROTPBM	PBM protocol (see [Ref. 8] –[Ref. 13])

ulLength
Specifies the length in bytes of the message in *lpbMsg*.

lpbMsg
Specifies the message that was received. Can be NULL if during a specified time period no

response was received from the communication partner (necessary to set the internal state machine to the correct state).

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this message.
WFS_ERR_PIN_MACINVALID	The MAC of the message is not correct.
WFS_ERR_PIN_PROTINVALID	The specified protocol is invalid.
WFS_ERR_PIN_FORMATINVALID	The format of the message is invalid.
WFS_ERR_PIN_CONTENTINVALID	The contents of one of the security relevant fields are invalid.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_HSM_TDATA_CHANGED	The terminal data has changed.

Comments None.

5.1.19 WFS_CMD_PIN_GET_JOURNAL

Description This command is used to get journal data from the encryption module. It retrieves cryptographically secured information about the result of the last transaction that was done with the indicated protocol. When the service provider supports journaling (see Capabilities) then it is impossible to do any WFS_CMD_PIN_SECURE_MSG_SEND/RECEIVE with this protocol, unless the journal data is retrieved. It is possible - especially after restarting a system - to get the same journal data again.

Input Param
```
LPWORD        lpwProtocol;
```

lpwProtocol
Specifies the protocol the journal data belong to. Specified as one of the following flags:

Value	Meaning
WFS_PIN_PROTISOAS	Get authorization system journal data
WFS_PIN_PROTISOLZ	Get German "Ladezentrale" journal data
WFS_PIN_PROTISOPS	Get personalization system journal data
WFS_PIN_PROTPBM	Get PBM protocol data

Output Param
```
LPWFSXDATA    lpxJournalData;
```

lpxJournalData
Pointer to the journal data

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to return journal data.
WFS_ERR_PIN_PROTINVALID	The specified protocol is invalid.

Events Only the generic events defined in [Ref. 1] can be generated by this command.

Comments None.

5.1.20 WFS_CMD_PIN_IMPORT_KEY_EX

Description The key passed by the application is loaded in the encryption module. The key can be passed in clear text mode or encrypted with an accompanying "key encryption key". The dwUse parameter is needed to separate the keys in several parts of the encryption module to avoid the manipulation of a key. A key can be loaded in multiple unencrypted parts by combining the WFS_PIN_USECONSTRUCT value with the final usage flag within the dwUse field.

Input Param `LPWFSPINIMPORTKEYEX lpImportKeyEx;`

```
typedef struct _wfs_pin_import_key_ex
    {
    LPSTR            lpsKey;
    LPSTR            lpsEncKey;
    LPWFSXDATA       lpxValue;
    LPWFSXDATA       lpxControlVector;
    DWORD            dwUse;
    WORD             wKeyCheckMode;
    LPWFSXDATA       lpxKeyCheckValue;
    } WFSPINIMPORTKEYEX, * LPWFSPINIMPORTKEYEX;
```

lpsKey
Specifies the name of key being loaded.

lpsEncKey
lpsEncKey specifies a key name which was used to encrypt the key string passed in *lpxValue*. If *lpsEncKey* is NULL the key is loaded directly into the encryption module. *lpsEncKey* must be NULL if *dwUse* contains WFS_PIN_USECONSTRUCT.

lpxValue
Specifies the value of key to be loaded. If it is an RSA key the first 4 bytes contain the exponent and the following 128 the modulus.

lpxControlVector
Specifies the control vector of the key to be loaded. It contains the attributes of the key. If this parameter is NULL the keys is only specified by its use.

dwUse
Specifies the type of access for which the key can be used. If this parameter equals zero, the key is deleted. Otherwise the parameter can be one of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key is used for encryption and decryption
WFS_PIN_USEFUNCTION	key is used for PIN block creation
WFS_PIN_USEMACING	key is used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USEPINLOCAL	key is used for local PIN check
WFS_PIN_USERSAPUBLIC	key is used as a public key for RSA encryption including EMV PIN block creation
WFS_PIN_USERSAPRIVATE	key is used as a private key for RSA decryption (it is not recommend that private keys are imported with this function).
WFS_PIN_USECONSTRUCT	key is under construction through the import of multiple parts. This value is used in combination with one of the other key usage flags

If *dwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* are ignored.

wKeyCheckMode
Specifies the mode that is used to create the key check value. It can be one of the following flags:

Value	Meaning
WFS_PIN_KCVNONE	There is no key check value verification required.
WFS_PIN_KCVSELF	The key check value is created by an encryption of the key with itself.
WFS_PIN_KCVZERO	The key check value is created by an encryption of the key with a zero value.

lpxKeyCheckValue
Specifies a check value to verify that the value of the imported key is correct. It can be NULL, if no key check value verification is required and *wKeyCheckMode* equals WFS_PIN_KCVNONE.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key encryption key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOVALUE	The specified key encryption key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use conflicts with a previously for the same key specified one.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_KEYINVALID	The key value is invalid. The key check value verification failed.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments When keys are loaded in multiple parts, all parts of the key loaded must set the WFS_PIN_USECONSTRUCT value in the *dwUse* field along with any usage's needed for the final key use. The usage flag must be consistent for all parts of the key. Activation of a key entered in multiple parts is indicated through an additional final call to this command, where WFS_USECONSTRUCT is removed from *dwUse* but those other usage's defined during the key part loading must still be used. No key data is passed during the final activation of the key.

When the WFS_PIN_USECONSTRUCT flag is set, the optional KCV applies to the key part being imported. If the KVC provided for a key part fails verification, the key part will not be accepted. When the key is being activated, the optional KCV applies to the complete key already stored. If the KVC provided during activation fails verification, the key will not be activated.

When the first part of the key is received, it is stored directly in the device. All subsequent parts are combined with the existing value in the device through XOR. No sub-parts of the key are maintained separately. While a key still has a *dwUse* value that is combined with WFS_PIN_USECONSTRUCT (i.e. it is still being loaded), it cannot be used for cryptographic functions.

5.1.21 WFS_CMD_PIN_ENC_IO

Description This command is used to communicate with the encryption module. Transparent data is sent from the application to the encryption module and the response is returned transparently to the application.

Input Param

```
LPWFSPINENCIO       lpEncIoIn;

typedef struct _wfs_pin_enc_io
   {
   WORD         wProtocol;
   ULONG        ulDataLength;
   LPVOID       lpvData;
   } WFSPINENCIO, *LPWFSPINENCIO;
```

wProtocol
Identifies the protocol that is used to communicate with the encryption module.
The following protocol numbers are defined:

Value	Meaning
WFS_PIN_ENC_PROT_CH	For Swiss specific protocols. The document specification for Swiss specific protocols is "CMD_ENC_IO - CH Protocol.doc". This document is available at the following address: *EUROPAY (Switzerland) SA* *Terminal Management* *Hertistrasse 27* *CH-8304 Wallisellen*
WFS_PIN_ENC_PROT_GIECB	Protocol for “Groupement des Cartes Bancaires” (France)

ulDataLength
Specifies the length in bytes of the structure pointed to by the following field *lpvData*.

lpvData
Points to a structure containing the data to be sent to the encryption module.

Output Param

```
LPWFSPINENCIO       lpEncIoOut;

typedef struct _wfs_pin_enc_io
   {
   WORD         wProtocol;
   ULONG        ulDataLength;
   LPVOID       lpvData;
   } WFSPINENCIO, *LPWFSPINENCIO;
```

wProtocol
Identifies the protocol that is used to communicate with the encryption module. This field contains the same value as the corresponding field in the input structure.

ulDataLength
Specifies the length in bytes of the structure pointed to by the following field *lpvData*.

lpvData
Points to a structure containing the data responded by the encryption module.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_PROTOCOLNOTSUPP	The specified protocol is not supported by the service provider. For *wProtocol*=WFS_PIN_ENC_PROT_GIECB
WFS_ERR_INVALID_DATA	The input data is not valid for the specified protocol, e.g. inconsistent TLV encoding
WFS_ERR_PIN_RANDOMINVALID	The encrypted random number in the input data does not decrypt to the one previously provided by the EPP
WFS_ERR_PIN_SIGNATUREINVALID	The signature in the input data is invalid
WFS_ERR_PIN_SNSCDINVALID	The SCD serial number in the input data is invalid
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this command.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.
WFS_ERR_PIN_KEYINVALID	The key value is invalid.
WFS_ERR_PIN_KEY_GENERATION_ERROR	The EPP is unable to generate a key pair

Events None.

Comments None.

5.1.22 WFS_CMD_PIN_HSM_INIT

Description This command is used to set an HSM out of order. At the same time the online time can be set to control when the OPT online dialog (see Protocol WFS_PIN_PROTISOPS) shall be started to initialize the HSM again. When this time is reached a WFS_SRVE_PIN_OPT_REQUIRED event will be sent.

Input Param

```
LPWFSPINHSMINIT     lpHsmInit;

typedef struct _wfs_pin_hsm_init
    {
    WORD          wInitMode;
    LPWFSXDATA    lpxOnlineTime;
    } WFSPINHSMINIT, * LPWFSPINHSMINIT;
```

wInitMode
Specifies the init mode as one of the following flags:

Value	Meaning
WFS_PIN_INITTEMP	Initialize the HSM temporarily (K_UR remains loaded)
WFS_PIN_INITDEFINITE	Initialize the HSM definitely (K_UR is deleted)
WFS_PIN_INITIRREVERSIBLE	Initialize the HSM irreversibly (can only be restored by the vendor)

lpxOnlineTime
Specifies the Online date and time in the format YYYYMMDDHHMMSS like in ISO BMP 61 as BCD packed characters. This parameter is ignored when the init mode equals WFS_PIN_INITDEFINITE or WFS_PIN_INITIRREVERSIBLE. If this parameter is NULL, *ulLength* is zero or the value is 0x00 0x00 0x00 0x00 0x00 0x00 0x00 the online time will be set to a value in the past.

Output Param None.

Error Codes The following additional error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_MODENOTSUPPORTED	The specified init mode is not supported.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this command.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_HSM_TDATA_CHANGED	The terminal data has changed.

Comments None.

5.2 Common commands for Remote Key Loading Schemes

This section describes those commands that are common between the two Remote Key Loading Schemes. The commands defined within this section can be used for both the Remote Key Loading Scheme using Signatures and the Remote Key Loading Scheme using Certificates. Section 8 provides additional explanation on how these commands are used.

5.2.1 WFS_CMD_PIN_START_KEY_EXCHANGE

Description This command is used to start the transfer of the host's Key Transport Key.

This output value is returned to the host and is used in the WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY and WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY commands to verify that the encryptor is talking to the proper host.

The WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY and WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY commands end the key exchange process.

Input Param None

Output Param
```
LPWFSPINSTARTKEYEXCHANGE         lpStartKeyExchange;

typedef struct _wfs_pin_start_key_exchange
{
LPWFSXDATA        lpxRandomItem;
} WFSPINSTARTKEYEXCHANGE, * LPWFSPINSTARTKEYEXCHANGE;
```

lpxRandomItem
Pointer to a randomly generated number created by the encryptor, which will be used to verify the Key Transport message sent from the host. If the PIN device does not support random number generation and verification, a zero length random number is returned and a NULL *lpbData* pointer is returned.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.

Events None.

Comments None.

5.3 Remote Key Loading Using Signatures

This section contains commands that are used for Remote Key Loading with Signatures. Applications wishing to use such functionality must use these commands. Section 8.1 provides additional explanation on how these commands are used. Section 8.1.7 defines the fixed names for the Security Item and RSA keys that must be loaded during manufacture.

5.3.1 WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY

Description The Public RSA key passed by the application is loaded in the encryption module. The *dwUse* parameter restricts the cryptographic functions that the imported key can be used for.

This command provides similar public key import functionality to that provided with WFS_CMD_PIN_IMPORT_KEY_EX. The primary advantage gained through using this function is that the imported key can be verified as having come from a trusted source. If a Signature algorithm is specified that is not supported by the PIN SP, then the request will not be accepted and the command fails.

Input Param

```
LPWFSPINIMPORTRSAPUBLICKEY       lpImportRSAPublicKey;
```

```
typedef struct _wfs_pin_import_rsa_public_key
{
LPSTR               lpsKey;
LPWFSXDATA          lpxValue;
DWORD               dwUse;
LPSTR               lpsSigKey;
DWORD               dwRSASignatureAlgorithm;
LPWFSXDATA          lpxSignature;
} WFSPINIMPORTRSAPUBLICKEY, * LPWFSPINIMPORTRSAPUBLICKEY;
```

lpsKey
Specifies the name of key being loaded
lpxValue
Contains the PKCS #1 formatted RSA Public Key to be loaded, represented in DER encoded ASN.1.

dwUse
Specifies the type of access for which the key can be used. If this parameter equals zero, the key is deleted. Otherwise the parameter can be one of the following flags:

Value	Meaning
WFS_PIN_USERSAPUBLIC	key is used as a public key for RSA Encryption including EMV PIN block creation
WFS_PIN_USERSAPUBLICVERIFY	key is used as a public key for RSA signature verification and/or data decryption.

If *dwUse* equals zero the specified key is deleted. In that case, all parameters but *lpsKey* are ignored. WFS_CMD_PIN_IMPORT_KEY, WFS_CMD_PIN_IMPORT_KEY_EX, WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY and WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY can be used to delete a key that has been imported with this command. The equivalent commands in the certificate scheme must not be used to delete a key imported through the signature scheme.

lpsSigKey
lpsSigKey specifies the name of a previously loaded asymmetric key (i.e. an RSA Public Key) which will be used to verify the signature passed in *lpxSignature*. The default Signature Issuer public key (installed in a secure environment during manufacture) will be used, if *lpsSigKey* is either NULL or contains the name of the default Signature issuer as defined in section 8.1.7.

dwRSASignatureAlgorithm
Defines the algorithm used to generate the Signature specified in *lpxSignature*. Contains one of the following values:

Value	Meaning
WFS_PIN_SIGN_NA	No signature algorithm specified. No signature verification will take place and the contents of *lpsSigKey* and *lpxSignature* are ignored.
WFS_PIN_SIGN_RSASSA_PKCS1_V1_5	Use the RSASSA-PKCS1-v1.5 algorithm.
WFS_PIN_SIGN_RSASSA_PSS	Use the RSASSA-PSS algorithm.

lpxSignature
Contains the Signature associated with the key being imported. The Signature is used to validate the key has been received from a trusted sender. Contains NULL when no key validation is required.

Output Param LPWFSPINIMPORTRSAPUBLICKEYOUTPUT lpImportRSAPublicKeyOutput;

```
typedef struct _wfs_pin_import_rsa_public_key_output
{
DWORD        dwRSAKeyCheckMode;
LPWFSXDATA   lpxKeyCheckValue;
} WFSPINIMPORTRSAPUBLICKEYOUTPUT, * LPWFSPINIMPORTRSAPUBLICKEYOUTPUT;
```

dwRSAKeyCheckMode
Defines algorithm/method used to generate the public key check value/thumb print. The check value can be used to verify that the public key has been imported correctly. It can be can be one of the following flags:

Value	Meaning
WFS_PIN_RSA_KCV_NONE	No check value is returned in *lpxKeyCheckValue.*
WFS_PIN_RSA_KCV_SHA1	*lpxKeyCheckValue* contains a SHA-1 digest of the public key

lpxKeyCheckValue
Contains the public key check value as defined by the *dwRSAKeyCheckMode* flag.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOTFOUND	The key name supplied in *lpsSigKey* was not found.
WFS_ERR_PIN_USEVIOLATION	An invalid use was specified for the key being imported.

WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.
WFS_ERR_PIN_SIG_NOT_SUPP	The SP does not support the Signature Algorithm requested. The key was discarded
WFS_ERR_PIN_SIGNATUREINVALID	The imported key failed its signature verification. It is not stored in the PIN.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments **None.**

5.3.2 WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM

Description This command is used to export data elements from the PIN device, which have been signed by an offline Signature Issuer. This command is used when the default keys and Signature Issuer signatures, installed during manufacture, are to be used for remote key loading.

This command allows the following data items are to be exported:

- the Security Item which uniquely identifies the PIN device. This value may be used to uniquely identify a PIN device and therefore confer trust upon any key or data obtained from this device.
- the RSA Public key component of a public/private key pair that exists within the PIN device. These public/private key pairs are installed during manufacture Typically, an exported public key is used by the host to encipher the symmetric key.

See section 8.1.7 for the default names and the description of the keys installed during manufacture. These names are defined to ensure multi-vendor applications can be developed.

The WFS_INF_PIN_KEY_DETAIL_EX command can be used to determine the valid uses for the exported public key.

Input Param `LPWFSPINEXPORTRSAISSUERSIGNEDITEM lpExportRSAIssuerSignedItem;`

```
typedef struct _wfs_pin_export_rsa_issuer_signed_item
{
WORD                wExportItemType;
LPSTR               lpsName;

} WFSPINEXPORTRSAISSUERSIGNEDITEM, *
LPWFSPINEXPORTRSAISSUERSIGNEDITEM;
```

wExportItemType
Defines the type of data item to be exported from the PIN. Contains one of the following values:

Value	Meaning
WFS_PIN_EXPORT_EPP_ID	The Unique ID for the PIN will be exported, *lpsName* is ignored.
WFS_PIN_EXPORT_PUBLIC_KEY	The public key identified by *lpsName* will be exported.

lpsName
Specifies the name of the public key to be exported. The private/public key pair was installed during manufacture, see section 8.1.7 for a definition of these default keys. If *lpsName* is NULL , then the default EPP public key that is used for symmetric key encryption is exported.

Output Param

```
LPWFSPINEXPORTRSAISSUERSIGNEDITEMOUTPUT
       lpExportRSAIssuerSignedItemOutput;
```

```
typedef struct _wfs_pin_export_rsa_issuer_signed_item_output
    {
    LPWFSXDATA      lpxValue;
    DWORD           dwRSASignatureAlgorithm;
    LPWFSXDATA      lpxSignature;
    } WFSPINEXPORTRSAISSUERSIGNEDITEMOUTPUT, *
    LPWFSPINEXPORTRSAISSUERSIGNEDITEMOUTPUT;
```

lpxValue
If a public key was requested then *lpxValue* contains the PKCS #1 formatted RSA Public Key represented in DER encoded ASN.1 format. If the security item was requested then *lpxValue* contains the PIN's Security Item which may be vendor specific.

dwRSASignatureAlgorithm.
Specifies the algorithm used to generate the Signature returned in *lpxSignature*. Contains one of the following values:

Value	Meaning
WFS_PIN_SIGN_NA	No signature algorithm used, no signature will be provided in *lpxSignature*, the data item may still be exported.
WFS_PIN_SIGN_RSASSA_PKCS1_V1_5	RSASSA-PKCS1-v1.5 algorithm used.
WFS_PIN_SIGN_RSASSA_PSS	RSASSA-PSS algorithm used.

lpxSignature
Specifies the RSA signature of the data item exported. NULL can be returned when key Signatures are not supported

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_NORSAKEYPAIR	The PIN device does not have a private key.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOTFOUND	The data item identified by *lpsName* was not found.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3.3 WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY

Description This command is used to load a Symmetric Key that is either a single or double DES length key into the encryptor. The key passed by the application is loaded in the encryption module, the (optional) signature is used during validation, the key is decrypted using the device's RSA Private Key, and is then stored. The loaded key will be discarded at any stage if any of the above fails.

The random number previously obtained from the WFS_CMD_PIN_START_KEY_EXCHANGE command and sent to the host is included in the signed data. This random number (when present) is verified during the load process. This command ends the Key Exchange process.

The *dwUse* parameter restricts the cryptographic functions that the imported key can be used for.

If a Signature algorithm is specified that is not supported by the PIN SP, then the message will not be decrypted and the command fails.

Input Param

```
LPWFSPINIMPORTRSASIGNEDDESKEY     lpImportRSASignedDESKey;

typedef struct _wfs_pin_import_rsa_signed_des_key
   {
   LPSTR                  lpsKey;
   LPSTR                  lpsDecryptKey;
   DWORD                  dwRSAEncipherAlgorithm;
   LPWFSXDATA             lpxValue;
   DWORD                  dwUse;
   LPSTR                  lpsSigKey;
   DWORD                  dwRSASignatureAlgorithm;
   LPWFSXDATA             lpxSignature;
   } WFSPINIMPORTRSASIGNEDDESKEY, * LPWFSPINIMPORTRSASIGNEDDESKEY;
```

lpsKey
Specifies the name of key being loaded.

lpsDecryptKey
Specifies the name of the RSA private key used to decrypt the symmetric key. See section 8.1.7 for a description of the fixed name defined for the default decryption private key. If *lpsDecryptKey* is NULL then the default decryption private key is used.

dwRSAEncipherAlgorithm
Specifies the RSA algorithm that is used, along with the private key, to decipher the imported key. Contains one of the following values:

Value	Meaning
WFS_PIN_CRYPT_RSAES_PKCS1_V1_5	Use the RSAAES_PKCS1-v1.5 algorithm.
WFS_PIN_CRYPT_RSAES_OAEP	Use the RSAAES_OAEP algorithm.

lpxValue
Specifies the enciphered value of the key to be loaded. *lpxValue* contains the concatenation of the random number(when present) and enciphered key.

dwUse
Specifies the type of access for which the key can be used. If this parameter equals zero, the key is deleted. Otherwise, the parameter can be a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key is used for encryption and decryption
WFS_PIN_USEFUNCTION	key is used for PIN block creation
WFS_PIN_USEMACING	key is used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USEPINLOCAL	key is used for local PIN check
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key

If *dwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* are ignored. WFS_CMD_PIN_IMPORT_KEY, WFS_CMD_PIN_IMPORT_KEY_EX, WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY and WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY can be used to delete a key that has been imported with this command. The equivalent commands in the certificate scheme must not be used to delete a key imported through the signature scheme.

lpsSigKey
If *lpsSigKey* is NULL then the key signature will not be used for validation and *lpxSignature* is ignored. Otherwise *lpsSigKey* specifies the name of an Asymmetric Key (i.e. an RSA Public Key) previously loaded which will be used to verify the signature passed in *lpxSignature*.

dwRSASignatureAlgorithm
Specifies the algorithm used to generate the Signature specified in *lpxSignature*. Contains one of the following values:

Value	Meaning
WFS_PIN_SIGN_NA	No signature algorithm specified. No signature verification will take place and the contents of *lpxSignature* is ignored.
WFS_PIN_SIGN_RSASSA_PKCS1_V1_5	Use the RSASSA-PKCS1-v1.5 algorithm.
WFS_PIN_SIGN_RSASSA_PSS	Use the RSASSA-PSS algorithm.

lpxSignature
Contains the Signature associated with the key being imported. The Signature is used to validate the key has been received from a trusted sender. The signature is generated over the contents of the *lpxValue*. The *lpxSignature* signature contains NULL when no key validation is required.

Output Param `LPWFSPINIMPORTRSASIGNEDDESKEYOUTPUT     lpImportRSASignedDESKeyOutput;`

```
typedef struct _wfs_pin_import_rsa_signed_des_key_output
{
 WORD              wKeyLength;
 WORD              wKeyCheckMode;
 LPWFSXDATA        lpxKeyCheckValue;
} WFSPINIMPORTRSASIGNEDDESKEYOUTPUT, *
LPWFSPINIMPORTRSASIGNEDDESKEYOUTPUT;
```

wKeyLength
Specifies the length of the key loaded. It can be one of the following flags:

Value	Meaning
WFS_PIN_KEYSINGLE	The imported key is single length.
WFS_PIN_KEYDOUBLE	The imported key is double length.

wKeyCheckMode
Specifies the mode that is used to create the key check value. It can be one of the following flags:

Value	Meaning
WFS_PIN_KCVNONE	There is no key check value provided.
WFS_PIN_KCVSELF	The key check value is calculated by an encryption of the key with itself.
WFS_PIN_KCVZERO	The key check value is calculated by an encryption of a zero value with the key.

lpxKeyCheckValue
pointer to the key verification data that can be used for verification of the loaded key, NULL if device does not have that capability.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key encryption key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOTFOUND	One of the keys specified were not found.
WFS_ERR_PIN_KEYNOVALUE	The specified key encryption key is not loaded.

WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.
WFS_ERR_PIN_SIG_NOT_SUPP	The SP does not support the Signature Algorithm requested. The key was discarded.
WFS_ERR_PIN_SIGNATUREINVALID	The signature in the input data is invalid. The key is not stored in the PIN.
WFS_ERR_PIN_RANDOMINVALID	The encrypted random number in the input data does not match the one previously provided by the EPP. The key is not stored in the PIN.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3.4 WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR

Description This command will generate a new RSA key pair. The public key generated as a result of this command can subsequently be obtained by calling WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED_ITEM

The newly generated key pair can only be used for the use defined in the *dwUse* flag. This flag defines the use of the private key, it’s public key can only be used for the inverse function.

Input Param

```
LPWFSPINGENERATERSAKEYPAIR       lpGenerateRSAKeyPair;

typedef struct _wfs_pin_generate_rsa_key
    {
    LPSTR           lpsKey;
    DWORD           dwUse;
    WORD            wModulusLength;
    WORD            wExponentValue;
    } WFSPINGENERATERSAKEYPAIR, * LPWFSPINGENERATERSAKEYPAIR;
```

lpsKey
Specifies the name of the new key-pair to be generated. Details of the generated key-pair can be obtained through the WFS_INF_PIN_KEY_DETAIL_EX command.

dwUse
Specifies what the private key component of the key pair can be used for. The public key part can only be used for the inverse function. For example, if the WFS_PIN_USERSAPRIVATESIGN use is specified, then the private key can only be used for signature generation and the partner public key can only be used for verification. *dwUse* can take one of the following values:

Value	Meaning
WFS_PIN_USERSAPRIVATE	key is used as a private key for RSA decryption
WFS_PIN_USERSAPRIVATESIGN	key is used as a private key for RSA Signature generation. Only data generated within the device can be signed.

wModulusLength
Specifies the number of bits for the modulus of the RSA key pair to be generated. When zero is specified then the PIN device will be responsible for defining the length:

wExponentValue
Specifies the value of the exponent of the RSA key pair to be generated. The following defines valid values the exponent:

Value	Meaning
WFS_PIN_DEFAULT	The device will decide the exponent.
WFS_PIN_EXPONENT_1	Exponent of 2^1+1 (3)
WFS_PIN_EXPONENT_4	Exponent of 2^4+1 (17)
WFS_PIN_EXPONENT_16	Exponent of 2^{16}+1 (65537)

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_INVALID_MOD_LEN	The modulus length specified is invalid
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEY_GENERATION_ERROR	The EPP is unable to generate a key pair

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3.5 WFS_CMD_PIN_EXPORT_ RSA_EPP_SIGNED_ITEM

Description This command is used to export data elements from the PIN device that have been signed by a private key within the EPP. This command is used in place of the WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM command, when a private key generated within the PIN device is to be used to generate the signature for the data item. This command allows an application to define which of the following data items are to be exported:

- The Security Item which uniquely identifies the PIN device. This value may be used to uniquely identify a PIN device and therefore confer trust upon any key or data obtained from this device.
- The RSA Public key component of a public/private key pair that exists within the PIN device.

See section 8.1.7 for the default names and the description of the keys installed during manufacture. These names are defined to ensure multi-vendor applications can be developed.

The public/private key pairs exported by this command are either installed during manufacture or generated through the WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR command.

The WFS_INF_PIN_KEY_DETAIL_EX command can be used to determine the valid uses for the exported public key.

Input Param

```
LPWFSPINEXPORTRSAEPPSIGNEDITEM    lpExportRSAEPPSignedItem;

typedef struct _wfs_pin_export_rsa_epp_signed_item
{
WORD               wExportItemType;
LPSTR              lpsName;
LPSTR              lpsSigKey;
DWORD              dwSignatureAlgorithm;
} WFSPINEXPORTRSAEPPSIGNEDITEM, * LPWFSPINEXPORTRSAEPPSIGNEDITEM;
```

wExportItemType
Defines the type of data item to be exported from the PIN. Contains one of the following values:

Value	Meaning
WFS_PIN_EXPORT_EPP_ID	The Unique ID for the PIN will be exported, *lpsName* is ignored.
WFS_PIN_EXPORT_PUBLIC_KEY	The public key identified by *lpsName* will be exported.

lpsName
Specifies the name of the public key to be exported. This can either be the name of a key-pair generated through WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR or the name of one of the default key-pairs installed during manufacture.

lpsSigKey
Specifies the name of the private key to use to sign the exported item.

dwSignatureAlgorithm
Specifies the algorithm to use to generate the Signature returned in both *lpxSelfSignature* and *lpxSignature*. Contains one of the following values:

Value	Meaning
WFS_PIN_SIGN_NA	No signature algorithm used, no signature will be provided in *lpxSelfSignature* or *lpxSignature*, the requested item may still be exported.
WFS_PIN_SIGN_RSASSA_PKCS1_V1_5	RSASSA-PKCS1-v1.5 algorithm used.
WFS_PIN_SIGN_RSASSA_PSS	RSASSA-PSS algorithm used.

Output Param

```
LPWFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT    lpExportRSAEPPSignedItemOutput;

typedef struct _wfs_pin_export_rsa_epp_signed_item_output
{
LPWFSXDATA    lpxValue;
LPWFSXDATA    lpxSelfSignature;
LPWFSXDATA    lpxSignature;
} WFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT, *
LPWFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT;
```

lpxValue
If a public key was requested then *lpxValue* contains the PKCS #1 formatted RSA Public Key represented in DER encoded ASN.1 format. If the security item was requested then *lpxValue* contains the PIN's Security Item, which may be vendor specific.

lpxSelfSignature
If a public key was requested then *lpxSelfSignature* contains the RSA signature of the public key exported, generated with the key-pair's private component. NULL can be returned when key Self-Signatures are not supported/required.

lpxSignature
Specifies the RSA signature of the data item exported. NULL can be returned when signatures are not supported/required.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_NORSAKEYPAIR	The PIN device does not have a private key.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOTFOUND	The data item identified by *lpsName* was not found.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.4 Remote Key Loading with Certificates

This section contains commands that are used for Remote Key Loading with Certificates. Applications wishing to use such functionality must use these commands.

5.4.1 WFS_CMD_PIN_LOAD_CERTIFICATE

Description This command is used to load a host certificate or to load a new encryptor certificate from a Certificate Authority to make remote key loading possible. This command can be called only once if there are no plans for a new CA to take over the duties. If a new CA does take over the duties, then this command should be called after the WFS_CMD_REPLACE_CERTIFICATE command. The type of certificate (Primary or Secondary) to be loaded will be embedded within the actual certificate structure.

Input Param `LPWFSPINLOADCERTIFICATE    lpLoadCertificate;`

```
typedef struct _wfs_pin_load_certificate
{
LPWFSXDATA        lpxLoadCertificate;
} WFSPINLOADCERTIFICATE, *LPWFSPINLOADCERTIFICATE;
```

lpxLoadCertificate
Pointer to the structure that contains the certificate that is to be loaded represented in DER encoded ASN.1 notation.. This data should be in a binary encoded PKCS #7 using the degenerate certificate only case of the signed-data content type in which the inner content's data file is omitted and there are no signers.

Output Param `LPWFSPINLOADCERTIFICATEOUTPUT    lpLoadCertificateOutput;`

```
 typedef struct _wfs_pin_load_certificate_output
{
 LPWFSXDATA       lpxCertificateData;
} WFSPINLOADCERTIFICATEOUTPUT, *LPWFSPINLOADCERTIFICATEOUTPUT;
```

lpxCertificateData
Pointer to a PKCS #7 structure using a Digested-data content type. The digest parameter should contain the thumb print value.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_FORMATINVALID	The format of the message is invalid.
WFS_ERR_PIN_INVALIDCERTSTATE	The certificate module is in a state in which the request is invalid.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_CERTIFICATE_CHANGE	The certificate module state has changed.

Comments None.

5.4.2 WFS_CMD_PIN_GET_CERTIFICATE

Description This command is used to read out the encryptor's certificate, which has been signed by the trusted Certificate Authority and is sent to the host. This command only needs to be called once if no new Certificate Authority has taken over. The output of this command will specify in the PKCS #7 message the resulting Primary or Secondary certificate.

Input Param
```
LPWFSPINGETCERTIFICATE          lpGetCertificate;
```

```
typedef struct _wfs_pin_get_certificate
 {
 WORD            wGetCertificate;
 } WFSPINGETCERTIFICATE, *LPWFSPINGETCERTIFICATE;
```

wGetCertificate
Specifies which public key certificate is requested. If the WFS_INF_PIN_STATUS command indicates Primary Certificates are accepted, then the Primary Public Encryption Key or the Primary Public Verification Key will be read out. If the WFS_INF_PIN_STATUS command indicates Secondary Certificates are accepted, then the Secondary Public Encryption Key or the Secondary Public Verification Key will be read out.

Value	Meaning
WFS_PIN_PUBLICENCKEY	The corresponding encryption key is to be returned.
WFS_PIN_PUBLICVERIFICATIONKEY	The corresponding verification key is to be returned.

Output Param
```
LPWFSPINGETCERTIFICATEOUPUT             lpGetCertificateOutput;
```

```
 typedef struct _wfs_pin_get_certificate_output
{
 LPWFSXDATA        lpxCertificate;
} WFSPINGETCERTIFICATEOUTPUT, *LPWFSPINGETCERTIFICATEOUTPUT;
```

lpxCertificate
Pointer to the structure that contains the certificate that is to be loaded represented in DER encoded ASN.1 notation. This data should be in a binary encoded PKCS #7 using the degenerate certificate only case of the signed-data content type in which the inner content's data file is omitted and there are no signers.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_INVALIDCERTSTATE	The certificate module is in a state in which the request is invalid.

Events None.

Comments None.

5.4.3 WFS_CMD_PIN_REPLACE_CERTIFICATE

Description This command is used to replace the existing primary or secondary Certificate Authority certificate already loaded into the encryptor. This operation must be done by an Initial Certificate Authority or by a Sub-Certificate Authority. These operations will replace either the primary or secondary Certificate Authority public verification key inside of the encryptor. After this command is complete, the application should send the WFS_CMD_PIN_LOAD_CERTIFICATE and WFS_CMD_GET_CERTIFICATE commands to ensure that the new HOST and the encryptor have all the information required to perform the remote key loading process.

Input Param

```
LPWFSPINREPLACECERTIFICATE       lpReplaceCertificate;

typedef struct _wfs_pin_replace_certificate
{
 LPWFSXDATA       lpxReplaceCertificate;
} WFSPINREPLACECERTIFICATE, *LPWFSPINREPLACECERTIFICATE;
```

lpxReplaceCertificate
Pointer to the a PKCS # 7 message that will replace the current Certificate Authority. The outer content uses the Signed-data content type, the inner content is a degenerate certificate only content containing the new CA certificate and Inner Signed Data type The certificate should be in a format represented in DER encoded ASN.1 notation..

Output Param

```
LPWFSPINREPLACECERTIFICATEOUTPUT        lpReplaceCertificateOuput

 typedef struct _wfs_pin_replace_certificate_output
 {
 LPWFSXDATA       lpxNewCertificateData;
 } WFSPINREPLACECERTIFICATEOUTPUT,
 *LPWFSPINREPLACECERTIFICATEOUTPUT;
```

lpxNewCertificateData
Pointer to a PKCS #7 structure using a Digested-data content type. The digest parameter should contain the thumb print value.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_FORMATINVALID	The format of the message is invalid.
WFS_ERR_PIN_INVALIDCERTSTATE	The certificate module is in a state in which the request is invalid.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_CERTIFICATE_CHANGE	The certificate module state has changed.

Comments None.

5.4.4 WFS_CMD_PIN_ IMPORT_RSA_ENCIPHERED_PKCS7_KEY

Description This command is used to load a Key Transport Key that is either a single or double DES length key into the encryptor. The Key Transport Key should be destroyed if the entire process is not completed. In addition, a new Key Transport Key should be generated each time this protocol is executed. This method ends the Key Exchange process.

Input Param

```
LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEY     lpImportRSAEncipheredPKCS7Key;

typedef struct _wfs_pin_import_rsa_enciphered_pkcs7_key
{
 LPWFSXDATA   lpxImportRSAKeyIn;
 LPSTR        lpsKey;
 DWORD                dwUse;
}WFSPINIMPORTRSAENCIPHEREDPKCS7KEY, *LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEY;
```

lpxImportRSKeyIn
Pointer to a binary encoded PKCS #7 represented in a DER encoded ASN.1 notation. This allows the Host to verify that key was imported correctly and to the correct encryptor The message has an outer Signed-data content type with the SignerInfo encryptedDigest field containing the HOST's signature. The random numbers are included as authenticatedAttributes within the SignerInfo. The inner content is an Enveloped-data content type. The ATM identifier is included as the issuerAndSerialNumber within the RecipientInfo. The enciphered KTK is included within RecipientInfo. The encryptedContent is omitted.

lpsKey
Specifies the name of the key to be stored.

dwUse
Specifies the type of access for which the key can be used as a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key can be used for encryption/decryption
WFS_PIN_USEFUNCTION	key can be used for PIN functions
WFS_PIN_USEMACING	key can be used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key

If *dwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* are ignored. . WFS_CMD_PIN_IMPORT_KEY, WFS_CMD_PIN_IMPORT_KEY_EX, WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY
can be used to delete a key that has been imported with this command. The equivalent commands in the signature scheme must not be used to delete a key imported through the certificate scheme.

Output Param `LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT lpImportRSAEncipheredKeyOut;`

```
typedef struct _wfs_pin_import_rsa_enciphered_pkcs7_key_output
{
WORD                wKeyLength;
LPWFSXDATA          lpxRSAData;
}WFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT,
*LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT;
```

wKeyLength
Specifies the length of the key loaded. It can be one of the following flags:

Value	Meaning
WFS_PIN_KEYSINGLE	The imported key is single length.
WFS_PIN_KEYDOUBLE	The imported key is double length.

lpxRSAData
Pointer to a binary encoded PKCS #7, represented in DER encoded ASN.1 notation. The message has an outer Signed-data content type with the SignerInfo encryptedDigest field containing the ATM's signature. The random numbers are included as authenticatedAttributes within the SignerInfo. The inner content is a data content type which contains the HOST identifier as an issuerAndSerialNumber sequence.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason
WFS_ERR_PIN_FORMATINVALID	The format of the message is invalid
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_INVALIDID	The ID passed was not valid.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.
WFS_ERR_PIN_FORMATINVALID	The format of the message is invalid.
WFS_ERR_PIN_USEVIOLATION	The specified use conflicts with a previously for the same key specified one.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None

5.5 EMV

This section defines the commands needed to import the EMV RSA keys provided either by a Certification Authority (for example VISA or MASTERCARD EUROPE) or by the chip card itself (ISSUER KEY, ICC KEY and ICC PIN KEY).

5.5.1 WFS_CMD_PIN_EMV_IMPORT_PUBLIC_KEY

Description The Certification Authority and the Chip Card RSA public keys needed for EMV are loaded or deleted in/from the encryption module. This command is similar to the WFS_CMD_PIN_IMPORT_KEY_EX command, but it is specifically designed to address the key formats and security features defined by EMV. Mainly the extensive use of “signed certificate” or “EMV certificate” (which is a compromise between signature and a pure certificate) to provide the public key is taken in account. The service provider is responsible for all EMV public key import validation. Once loaded, the service provider is not responsible for key/certificate expiry, this is an application responsibility.

Input Param

```
LPWFSPINEMVIMPORTPUBLICKEY       lpEMVImportPublicKey;

typedef struct _wfs_pin_emv_import_public_key
   {
   LPSTR                  lpsKey;
   DWORD                  dwUse;
   WORD                   wImportScheme;
   LPWFSXDATA             lpxImportData;
   LPSTR                  lpsSigKey;
   } WFSPINEMVIMPORTPUBLICKEY, * LPWFSPINEMVIMPORTPUBLICKEY;
```

lpsKey
Specifies the name of key being loaded.

dwUse
Specifies the type of access for which the key can be used. If this parameter equals zero, the key is deleted. Otherwise the parameter can be one of the following flags:

Value	Meaning
WFS_PIN_USERSAPUBLIC	key is used as a public key for RSA encryption including EMV PIN block creation
WFS_PIN_USERSAPUBLICVERIFY	key is used as a public key for RSA signature verification and/or data decryption.

If *dwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* areignored.

wImportScheme
Defines the import scheme used. Contains one of the following values:

Value	Meaning
WFS_PIN_EMV_IMPORT_PLAIN_CA	This scheme is used by VISA. A plain text CA public key is imported with no verification. The two parts of the key (modulus and exponent) are passed in clear mode as a DER encoded PKCS#1 public key. The key is loaded directly in the security module
WFS_PIN_EMV_IMPORT_CHKSUM_CA	This scheme is used by VISA. A plain text CA public key is imported using the EMV 2000 Book II verification algorithm and it is verified before being loaded in the security module. (See [Ref. 4] under references section for this document)

WFS_PIN_EMV_IMPORT_EPI_CA	This scheme is used by Mastercard Europe. A CA public key is imported using the self-signed scheme defined in [Ref. 5].
WFS_PIN_EMV_IMPORT_ISSUER	An Issuer public key is imported as defined in EMV 2000 Book II, reference 4. (See [Ref. 4] under references section for this document)
WFS_PIN_EMV_IMPORT_ICC	An ICC public key is imported as defined in EMV 2000 Book II, (See [Ref. 4] under references section for this document)
WFS_PIN_EMV_IMPORT_ICC_PIN	An ICC PIN public key is imported as defined in EMV 2000 Book II, reference 4. (See [Ref. 4] under references section for this document)
WFS_PIN_EMV_IMPORT_PKCSV1_5_CA	A CA public key is imported and verified using a signature generated with a private key for which the public key is already loaded.

lpxImportData
The *lpxImportData* parameter contains all the necessary data to complete the import using the scheme specified within *wImportScheme*.

If *wImportScheme* is WFS_PIN_EMV_IMPORT_PLAIN_CA then *lpxImportData* contains a DER encoded PKCS#1 public key. No verification is possible. *lpsSigKey* is ignored.

If *wImportScheme* is WFS_PIN_EMV_IMPORT_CHKSUM_CA then *lpxImportData* contains table 23 data, as specified in EMV 2000 Book 2 (See [Ref. 4] under the reference section for this document). The plain text key is verified as defined within EMV 2000 Book 2, page 73. *lpsSigKey* is ignored (See [Ref. 4] under the reference section for this document).

If *wImportScheme* is WFS_PIN_EMV_IMPORT_EPI_CA then *lpxImportData* contains the concatenation of tables 4 and 13, as specified in [Ref. 5], Europay International, EPI CA Module Technical – Interface specification Version 1.4. These tables are also described in the EMV Support Appendix. The self-signed public key is verified as defined by the reference document. *lpsSigKey* is ignored.

If *wImportScheme* is WFS_PIN_EMV_IMPORT_ISSUER then *lpxImportData* contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the Service Provider. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value: '9F32'), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value: '90') , the remainder length (1 byte). The remainder value (variable length – EMV Tag value: '92'), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value : '5A'). The service provider will compare the leftmost three-eight digits of the PAN to the Issuer Identification Number retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book2 – Security & Key Management Version 4.0, Table 4 (See [Ref. 4] under the reference section for this document). *lpsSigKey* defines the previously loaded key used to verify the signature.

If *wImportScheme* is WFS_PIN_EMV_IMPORT_ICC then *lpxImportData* contains the EMV public key certificate. . Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the Service Provider. The data consists of the concatenation of : the key exponent length (1 byte), the key exponent value (variable length– EMV Tag value : '9F47'), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value :'9F46'), the remainder length (1 byte), the remainder value (variable length – EMV Tag value : '9F48'), the SDA length (1 byte), the SDA value (variable length), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value : '5A'),. The service provider will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book2 – Security & Key Management Version 4.0, Table 9 (See [Ref. 4] under the reference section for this document). *lpsSigKey* defines the previously loaded key used to verify the signature.

If *wImportScheme* is WFS_PIN_EMV_IMPORT_ICC_PIN then *lpxImportData* contains the EMV public key certificate. . Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the Service Provider. The data consists of the concatenation of : the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value : '9F2E'), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value :'9F2D'), the remainder length (1 byte), the remainder value (variable length – EMV Tag value : '9F2F'), the SDA length (1 byte), the SDA value (variable length), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value : '5A'),. The service provider will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book2 – Security & Key Management Version 4.0, Table 9 (See [Ref. 4] under the reference section for this document). *lpsSigKey* defines the previously loaded key used to verify the signature.

If *wImportScheme* is WFS_PIN_EMV_IMPORT_PKCSV1_5_CA then *lpxImportData* contains the CA public key signed with the previously loaded public key specified in *lpsSigKey*. *lpxImportData* consists of the concatenation of EMV 2000 Book II Table 23 + 8 byte random number + Signature (See [Ref. 4] under the reference section for this document). The 8 byte random number is not used for validation, it is used to ensure the signature is unique. The Signature consists of all the bytes in the *lpxImportData* buffer after table 23 and the 8 byte random number.

lpsSigKey
This field specifies the name of the previously loaded key used to verify the signature, as detailed in the descriptions above.

Output Param `LPWFSPINEMVIMPORTPUBLICKEYOUTPUT lpEMVImportPublicKeyOutput;`

```
typedef struct _wfs_pin_emv_import_public_key_output
  {
  LPSTR     lpsExpiryDate;
  } WFSPINEMVIMPORTPUBLICKEYOUTPUT, * LPWFSPINEMVIMPORTPUBLICKEYOUTPUT;
```

lpxExpiryDate
Contains the expiry date of the certificate in the following format MMYY. If no expiry date applies then *lpsExpiryDate* is NULL.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.
WFS_ERR_PIN_EMV_VERIFY_FAILED	The verification of the imported key failed and the key was discarded.
WFS_ERR_PIN_KEYNOTFOUND	The specified key name is not found.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments This command only imports one key per use. If the same key value has to be imported for two different uses, this command must be called twice and different key names must be specified.

5.5.2 WFS_CMD_PIN_DIGEST

Description: This command is used to compute a hash code on a stream of data using the specified hash algorithm. This command can be used to verify EMV static and dynamic data.

Input:

```
LPWFSPINDIGEST     lpDigest;
typedef struct _wfs_pin_digest
{
WORD              wHashAlgorithm;
LPWFSXDATA        lpxDigestInput;
} WFSPINDIGEST, * LPWFSPINDIGEST;
```

wHashAlgorithm
Specifies which hash algorithm should be used to calculate the hash. See the Capabilities section for valid algorithms.

lpxDigestInput
Pointer to the structure that contains the length and the data to be hashed

Output Param:

```
LPWFSPINDIGESTOUPUT  lpDigestOutput;
typedef struct _wfs_pin_digest_output
{
LPWFSXDATA        lpxDigestOutput
} WFSPINDIGESTOUTPUT, * LPWFSPINDIGESTOUTPUT;
```

lpxDigestOuput
Pointer to the structure that contains the length and the data containing the calculated hash.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.

Events: None

6. Events

6.1 WFS_EXEE_PIN_KEY

Description This event specifies that any active key has been pressed at the PIN pad. It is used if the device has no internal display unit and the application has to manage the display of the entered digits.

It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

Event Param

```
LPWFSPINKEY   lpKey;

typedef struct _wfs_pin_key
   {
   WORD      wCompletion;
   ULONG     ulDigit;
   } WFSPINKEY, * LPWFSPINKEY;
```

wCompletion
Specifies the reason for completion or continuation of the entry. Possible values are:
(see command WFS_CMD_PIN_GET_PIN)

ulDigit
Specifies the digit entered by the user. When working in encryption mode (WFS_CMD_PIN_GET_PIN), the value of this field is zero. For each key pressed, the corresponding FK or FDK mask value is stored in this field.

Comments None.

6.2 WFS_SRVE_PIN_INITIALIZED

Description This event specifies that, as a result of a WFS_CMD_PIN_INITIALIZATION, the encryption module is now initialized and the master key (where required) and any other initial keys are loaded; ready to import other keys.

Event Param

```
LPWFSPININIT  lpInit;
```

lpInit
For a definition of WFSPININIT see command WFS_CMD_PIN_INITIALIZATION.

Comments None.

6.3 WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS

Description This event specifies that an error occurred accessing an encryption key. Possible situations for generating this event are the encryption key was not found, had no value, or a use violation.

Event Param

```
LPWFSPINACCESS      lpAccess;

typedef struct _wfs_pin_access
   {
   LPSTR     lpsKeyName;
   LONG      lErrorCode;
   } WFSPINACCESS, * LPWFSPINACCESS;
```

lpsKeyName
Specifies the name of the key that caused the error.

lErrorCode
Specifies the type of illegal key access that occurred. Possible values are:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not loaded.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.

Comments None.

6.4 WFS_SRVE_PIN_OPT_REQUIRED

Description This event indicates that the online date/time stored in a HSM has been reached.

Event Param None.

Comments This event may be triggered by the clock reaching a previously stored online time or by the online time being set to a time that lies in the past.

The online time may be set by the command WFS_CMD_PIN_HSM_SET_TDATA or by a command WFS_CMD_PIN_SECURE_MSG_RECEIVE that contains a message from a host system containing a new online date/time.

The event does not mean that any keys or other data in the HSM is out of date now. It just indicates that the terminal should communicate with a "Personalisierungsstelle" as soon as possible using the commands WFS_CMD_PIN_SECURE_MSG_SEND / _RECEIVE and *wProtocol*=WFS_PIN_PROTISOPS.

6.5 WFS_SRVE_PIN_CERTIFICATE_CHANGE

Description This event indicates that the certificate module state has changed from Primary to Secondary.

Event Param `LPWORD        lpwCertificateChange`

lp*wCertificateChange*
Specifies change of the certificate state inside of the encryptor as one of the following:

Value	Meaning
WFS_PIN_CERT_SECONDARY	The certificate state of the encryptor is now Secondary and Primary Certificates will no longer be accepted.

Comments None

6.6 WFS_SRVE_PIN_HSM_TDATA_CHANGED

Description This event indicates that one of the values of the terminal data has changed (these are the data that can be set using WFS_CMD_PIN_SET_HSM_TDATA). I.e. this event will be sent especially when the online time or the HSM status is changed because of a WFS_CMD_PIN_HSM_INIT command or an OPT online dialog (WFS_CMD_PIN_SECURE_MSG_SEND/_RECEIVE with WFS_PIN_PROTPS).

Event Param `LPWFSXDATA    lpxTData;`

lpxTData
Contains the parameter settings as a series of “tag/length/value” items. See command WFS_CMD_PIN_HSM_SET_TDATA for the tags supported.

Comments None.

7. C - Header File

```
/******************************************************************************
*                                                                             *
*xfspin.h XFS - Personal Identification Number Keypad (PIN) definitions      *
*                                                                             *
*            Version 3.02  (17/04/03)                                         *
*                                                                             *
******************************************************************************/

#ifndef __INC_XFSPIN__H
#define __INC_XFSPIN__H

#ifdef __cplusplus
extern "C" {
#endif

#include <xfsapi.h>

/*   be aware of alignment   */
#pragma pack(push,1)


/* values of WFSPINCAPS.wClass */

#define WFS_SERVICE_CLASS_PIN               (4)
#define WFS_SERVICE_CLASS_VERSION_PIN       (0x0203) /* Version 3.02 */
#define WFS_SERVICE_CLASS_NAME_PIN          "PIN"

#define PIN_SERVICE_OFFSET                  (WFS_SERVICE_CLASS_PIN * 100)

/* PIN Info Commands */

#define WFS_INF_PIN_STATUS                  (PIN_SERVICE_OFFSET + 1)
#define WFS_INF_PIN_CAPABILITIES            (PIN_SERVICE_OFFSET + 2)
#define WFS_INF_PIN_KEY_DETAIL              (PIN_SERVICE_OFFSET + 4)
#define WFS_INF_PIN_FUNCKEY_DETAIL          (PIN_SERVICE_OFFSET + 5)
#define WFS_INF_PIN_HSM_TDATA               (PIN_SERVICE_OFFSET + 6)
#define WFS_INF_PIN_KEY_DETAIL_EX           (PIN_SERVICE_OFFSET + 7)

/* PIN Command Verbs */

#define WFS_CMD_PIN_CRYPT                   (PIN_SERVICE_OFFSET + 1)
#define WFS_CMD_PIN_IMPORT_KEY              (PIN_SERVICE_OFFSET + 3)
#define WFS_CMD_PIN_GET_PIN                 (PIN_SERVICE_OFFSET + 5)
#define WFS_CMD_PIN_GET_PINBLOCK            (PIN_SERVICE_OFFSET + 7)
#define WFS_CMD_PIN_GET_DATA                (PIN_SERVICE_OFFSET + 8)
#define WFS_CMD_PIN_INITIALIZATION          (PIN_SERVICE_OFFSET + 9)
#define WFS_CMD_PIN_LOCAL_DES               (PIN_SERVICE_OFFSET + 10)
#define WFS_CMD_PIN_LOCAL_EUROCHEQUE        (PIN_SERVICE_OFFSET + 11)
#define WFS_CMD_PIN_LOCAL_VISA              (PIN_SERVICE_OFFSET + 12)
#define WFS_CMD_PIN_CREATE_OFFSET           (PIN_SERVICE_OFFSET + 13)
#define WFS_CMD_PIN_DERIVE_KEY              (PIN_SERVICE_OFFSET + 14)
#define WFS_CMD_PIN_PRESENT_IDC             (PIN_SERVICE_OFFSET + 15)
#define WFS_CMD_PIN_LOCAL_BANKSYS           (PIN_SERVICE_OFFSET + 16)
#define WFS_CMD_PIN_BANKSYS_IO              (PIN_SERVICE_OFFSET + 17)
#define WFS_CMD_PIN_RESET                   (PIN_SERVICE_OFFSET + 18)
#define WFS_CMD_PIN_HSM_SET_TDATA           (PIN_SERVICE_OFFSET + 19)
#define WFS_CMD_PIN_SECURE_MSG_SEND         (PIN_SERVICE_OFFSET + 20)
#define WFS_CMD_PIN_SECURE_MSG_RECEIVE      (PIN_SERVICE_OFFSET + 21)
#define WFS_CMD_PIN_GET_JOURNAL             (PIN_SERVICE_OFFSET + 22)
#define WFS_CMD_PIN_IMPORT_KEY_EX           (PIN_SERVICE_OFFSET + 23)
#define WFS_CMD_PIN_ENC_IO                  (PIN_SERVICE_OFFSET + 24)
#define WFS_CMD_PIN_HSM_INIT                (PIN_SERVICE_OFFSET + 25)
#define WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY  (PIN_SERVICE_OFFSET + 26)
#define WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM (PIN_SERVICE_OFFSET + 27)
#define WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY (PIN_SERVICE_OFFSET + 28)
#define WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR (PIN_SERVICE_OFFSET + 29)
#define WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED_ITEM (PIN_SERVICE_OFFSET + 30)
#define WFS_CMD_PIN_LOAD_CERTIFICATE        (PIN_SERVICE_OFFSET + 31)
#define WFS_CMD_PIN_GET_CERTIFICATE         (PIN_SERVICE_OFFSET + 32)
#define WFS_CMD_PIN_REPLACE_CERTIFICATE     (PIN_SERVICE_OFFSET + 33)
#define WFS_CMD_PIN_START_KEY_EXCHANGE      (PIN_SERVICE_OFFSET + 34)
```

```
#define WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY (PIN_SERVICE_OFFSET + 35)
#define WFS_CMD_PIN_EMV_IMPORT_PUBLIC_KEY (PIN_SERVICE_OFFSET + 36)
#define WFS_CMD_PIN_DIGEST                (PIN_SERVICE_OFFSET + 37)


/* PIN Messages */

#define WFS_EXEE_PIN_KEY                  (PIN_SERVICE_OFFSET + 1)
#define WFS_SRVE_PIN_INITIALIZED          (PIN_SERVICE_OFFSET + 2)
#define WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS   (PIN_SERVICE_OFFSET + 3)
#define WFS_SRVE_PIN_OPT_REQUIRED         (PIN_SERVICE_OFFSET + 4)
#define WFS_SRVE_PIN_HSM_TDATA_CHANGED    (PIN_SERVICE_OFFSET + 5)
#define WFS_SRVE_PIN_CERTIFICATE_CHANGE   (PIN_SERVICE_OFFSET + 6)

/* values of WFSPINSTATUS.fwDevice */

#define WFS_PIN_DEVONLINE                 WFS_STAT_DEVONLINE
#define WFS_PIN_DEVOFFLINE                WFS_STAT_DEVOFFLINE
#define WFS_PIN_DEVPOWEROFF               WFS_STAT_DEVPOWEROFF
#define WFS_PIN_DEVNODEVICE               WFS_STAT_DEVNODEVICE
#define WFS_PIN_DEVHWERROR                WFS_STAT_DEVHWERROR
#define WFS_PIN_DEVUSERERROR              WFS_STAT_DEVUSERERROR
#define WFS_PIN_DEVBUSY                   WFS_STAT_DEVBUSY

/* values of WFSPINSTATUS.fwEncStat */

#define WFS_PIN_ENCREADY                  (0)
#define WFS_PIN_ENCNOTREADY               (1)
#define WFS_PIN_ENCNOTINITIALIZED         (2)
#define WFS_PIN_ENCBUSY                   (3)
#define WFS_PIN_ENCUNDEFINED              (4)
#define WFS_PIN_ENCINITIALIZED            (5)

/* values of WFSPINCAPS.wType */

#define WFS_PIN_TYPEEPP                   (0x0001)
#define WFS_PIN_TYPEEDM                   (0x0002)
#define WFS_PIN_TYPEHSM                   (0x0004)

/* values of WFSPINCAPS.fwAlgorithms, WFSPINCRYPT.wAlgorithm */

#define WFS_PIN_CRYPTDESECB               (0x0001)
#define WFS_PIN_CRYPTDESCBC               (0x0002)
#define WFS_PIN_CRYPTDESCFB               (0x0004)
#define WFS_PIN_CRYPTRSA                  (0x0008)
#define WFS_PIN_CRYPTECMA                 (0x0010)
#define WFS_PIN_CRYPTDESMAC               (0x0020)
#define WFS_PIN_CRYPTTRIDESECB            (0x0040)
#define WFS_PIN_CRYPTTRIDESCBC            (0x0080)
#define WFS_PIN_CRYPTTRIDESCFB            (0x0100)
#define WFS_PIN_CRYPTTRIDESMAC            (0x0200)
#define WFS_PIN_CRYPTMAAMAC               (0x0400)

/* values of WFSPINCAPS.fwPinFormats */

#define WFS_PIN_FORM3624                  (0x0001)
#define WFS_PIN_FORMANSI                  (0x0002)
#define WFS_PIN_FORMISO0                  (0x0004)
#define WFS_PIN_FORMISO1                  (0x0008)
#define WFS_PIN_FORMECI2                  (0x0010)
#define WFS_PIN_FORMECI3                  (0x0020)
#define WFS_PIN_FORMVISA                  (0x0040)
#define WFS_PIN_FORMDIEBOLD               (0x0080)
#define WFS_PIN_FORMDIEBOLDCO             (0x0100)
#define WFS_PIN_FORMVISA3                 (0x0200)
#define WFS_PIN_FORMBANKSYS               (0x0400)
#define WFS_PIN_FORMEMV                   (0x0800)
#define WFS_PIN_FORMISO3                  (0x2000)

/* values of WFSPINCAPS.fwDerivationAlgorithms */

#define WFS_PIN_CHIP_ZKA                  (0x0001)

/* values of WFSPINCAPS.fwPresentationAlgorithms */
```

```
#define WFS_PIN_PRESENT_CLEAR             (0x0001)

/* values of WFSPINCAPS.fwDisplay */

#define WFS_PIN_DISPNONE                  (1)
#define WFS_PIN_DISPLEDTHROUGH            (2)
#define WFS_PIN_DISPDISPLAY               (3)

/* values of WFSPINCAPS.fwIDKey */

#define WFS_PIN_IDKEYINITIALIZATION       (0x0001)
#define WFS_PIN_IDKEYIMPORT               (0x0002)

/* values of WFSPINCAPS.fwValidationAlgorithms */

#define WFS_PIN_DES                       (0x0001)
#define WFS_PIN_EUROCHEQUE                (0x0002)
#define WFS_PIN_VISA                      (0x0004)
#define WFS_PIN_DES_OFFSET                (0x0008)
#define WFS_PIN_BANKSYS                   (0x0010)

/* values of WFSPINCAPS.fwKeyCheckModes and
          WFSPINIMPORTKEYEX.wKeyCheckMode */

#define WFS_PIN_KCVNONE                   (0x0000)
#define WFS_PIN_KCVSELF                   (0x0001)
#define WFS_PIN_KCVZERO                   (0x0002)

/* values of WFSPINKEYDETAIL.fwUse and values of WFSPINKEYDETAILEX.dwUse */

#define WFS_PIN_USECRYPT                  (0x0001)
#define WFS_PIN_USEFUNCTION               (0x0002)
#define WFS_PIN_USEMACING                 (0x0004)
#define WFS_PIN_USEKEYENCKEY              (0x0020)
#define WFS_PIN_USENODUPLICATE            (0x0040)
#define WFS_PIN_USESVENCKEY               (0x0080)
#define WFS_PIN_USECONSTRUCT              (0x0100)
#define WFS_PIN_USEPINLOCAL              (0x10000)
#define WFS_PIN_USERSAPUBLIC             (0x20000)
#define WFS_PIN_USERSAPRIVATE            (0x40000)
#define WFS_PIN_USECHIPINFO             (0x100000)
#define WFS_PIN_USECHIPPIN              (0x200000)
#define WFS_PIN_USECHIPPS               (0x400000)
#define WFS_PIN_USECHIPMAC              (0x800000)
#define WFS_PIN_USECHIPLT              (0x1000000)
#define WFS_PIN_USECHIPMACLZ           (0x2000000)
#define WFS_PIN_USECHIPMACAZ           (0x4000000)
#define WFS_PIN_USERSAPUBLICVERIFY     (0x8000000)
#define WFS_PIN_USERSAPRIVATESIGN     (0x10000000)


/* values of WFSPINFUNCKEYDETAIL.ulFuncMask */

#define WFS_PIN_FK_0                      (0x00000001)
#define WFS_PIN_FK_1                      (0x00000002)
#define WFS_PIN_FK_2                      (0x00000004)
#define WFS_PIN_FK_3                      (0x00000008)
#define WFS_PIN_FK_4                      (0x00000010)
#define WFS_PIN_FK_5                      (0x00000020)
#define WFS_PIN_FK_6                      (0x00000040)
#define WFS_PIN_FK_7                      (0x00000080)
#define WFS_PIN_FK_8                      (0x00000100)
#define WFS_PIN_FK_9                      (0x00000200)
#define WFS_PIN_FK_ENTER                  (0x00000400)
#define WFS_PIN_FK_CANCEL                 (0x00000800)
#define WFS_PIN_FK_CLEAR                  (0x00001000)
#define WFS_PIN_FK_BACKSPACE              (0x00002000)
#define WFS_PIN_FK_HELP                   (0x00004000)
#define WFS_PIN_FK_DECPOINT               (0x00008000)
#define WFS_PIN_FK_00                     (0x00010000)
#define WFS_PIN_FK_000                    (0x00020000)
#define WFS_PIN_FK_RES1                   (0x00040000)
#define WFS_PIN_FK_RES2                   (0x00080000)
```

```
#define WFS_PIN_FK_RES3                     (0x00100000)
#define WFS_PIN_FK_RES4                     (0x00200000)
#define WFS_PIN_FK_RES5                     (0x00400000)
#define WFS_PIN_FK_RES6                     (0x00800000)
#define WFS_PIN_FK_RES7                     (0x01000000)
#define WFS_PIN_FK_RES8                     (0x02000000)
#define WFS_PIN_FK_OEM1                     (0x04000000)
#define WFS_PIN_FK_OEM2                     (0x08000000)
#define WFS_PIN_FK_OEM3                     (0x10000000)
#define WFS_PIN_FK_OEM4                     (0x20000000)
#define WFS_PIN_FK_OEM5                     (0x40000000)
#define WFS_PIN_FK_OEM6                     (0x80000000)

/* values of WFSPINFUNCKEY.ulFDK */

#define WFS_PIN_FK_FDK01                    (0x00000001)
#define WFS_PIN_FK_FDK02                    (0x00000002)
#define WFS_PIN_FK_FDK03                    (0x00000004)
#define WFS_PIN_FK_FDK04                    (0x00000008)
#define WFS_PIN_FK_FDK05                    (0x00000010)
#define WFS_PIN_FK_FDK06                    (0x00000020)
#define WFS_PIN_FK_FDK07                    (0x00000040)
#define WFS_PIN_FK_FDK08                    (0x00000080)
#define WFS_PIN_FK_FDK09                    (0x00000100)
#define WFS_PIN_FK_FDK10                    (0x00000200)
#define WFS_PIN_FK_FDK11                    (0x00000400)
#define WFS_PIN_FK_FDK12                    (0x00000800)
#define WFS_PIN_FK_FDK13                    (0x00001000)
#define WFS_PIN_FK_FDK14                    (0x00002000)
#define WFS_PIN_FK_FDK15                    (0x00004000)
#define WFS_PIN_FK_FDK16                    (0x00008000)
#define WFS_PIN_FK_FDK17                    (0x00010000)
#define WFS_PIN_FK_FDK18                    (0x00020000)
#define WFS_PIN_FK_FDK19                    (0x00040000)
#define WFS_PIN_FK_FDK20                    (0x00080000)
#define WFS_PIN_FK_FDK21                    (0x00100000)
#define WFS_PIN_FK_FDK22                    (0x00200000)
#define WFS_PIN_FK_FDK23                    (0x00400000)
#define WFS_PIN_FK_FDK24                    (0x00800000)
#define WFS_PIN_FK_FDK25                    (0x01000000)
#define WFS_PIN_FK_FDK26                    (0x02000000)
#define WFS_PIN_FK_FDK27                    (0x04000000)
#define WFS_PIN_FK_FDK28                    (0x08000000)
#define WFS_PIN_FK_FDK29                    (0x10000000)
#define WFS_PIN_FK_FDK30                    (0x20000000)
#define WFS_PIN_FK_FDK31                    (0x40000000)
#define WFS_PIN_FK_FDK32                    (0x80000000)

/* values of WFSPINCRYPT.wMode */

#define WFS_PIN_MODEENCRYPT                 (1)
#define WFS_PIN_MODEDECRYPT                 (2)
#define WFS_PIN_MODERANDOM                  (3)

/* values of WFSPINENTRY.wCompletion */

#define WFS_PIN_COMPAUTO                    (0)
#define WFS_PIN_COMPENTER                   (1)
#define WFS_PIN_COMPCANCEL                  (2)
#define WFS_PIN_COMPCONTINUE                (6)
#define WFS_PIN_COMPCLEAR                   (7)
#define WFS_PIN_COMPBACKSPACE               (8)
#define WFS_PIN_COMPFDK                     (9)
#define WFS_PIN_COMPHELP                    (10)
#define WFS_PIN_COMPFK                      (11)
#define WFS_PIN_COMPCONTFDK                 (12)


/* values of WFSPINSECMSG.wProtocol */
```

```
#define WFS_PIN_PROTISOAS                 (1)
#define WFS_PIN_PROTISOLZ                 (2)
#define WFS_PIN_PROTISOPS                 (3)
#define WFS_PIN_PROTCHIPZKA               (4)
#define WFS_PIN_PROTRAWDATA               (5)
#define WFS_PIN_PROTPBM                   (6)
#define WFS_PIN_PROTHSMLDI                (7)

/* values of WFSPINHSMINIT.wInitMode. */
#define WFS_PIN_INITTEMP                  (1)
#define WFS_PIN_INITDEFINITE              (2)
#define WFS_PIN_INITIRREVERSIBLE          (3)

/* values of WFSPINENCIO.wProtocol */
#define WFS_PIN_ENC_PROT_CH               (0x0001)
#define WFS_PIN_ENC_PROT_GIECB            (0x0002)

/* values for WFS_SRVE_PIN_CERTIFICATE_CHANGE */
#define WFS_PIN_CERT_PRIMARY              (0x00000001)
#define WFS_PIN_CERT_SECONDARY            (0x00000002)
#define WFS_PIN_CERT_NOTREADY             (0x00000004)

/* Values for WFSPINCAPS.dwRSAAuthenticationScheme and the fast-track Capabilities
lpszExtra parameter, REMOTE_KEY_SCHEME. */
#define WFS_PIN_RSA_AUTH_2PARTY_SIG       (0x00000001)
#define WFS_PIN_RSA_AUTH_3PARTY_CERT      (0x00000002)

/* Values for WFSPINCAPS.dwSignatureScheme and the fast-track Capabilities lpzExtra
parameter, SIGNATURE_CAPABILITIES. */
#define WFS_PIN_SIG_GEN_RSA_KEY_PAIR          (0x00000001)
#define WFS_PIN_SIG_RANDOM_NUMBER             (0x00000002)
#define WFS_PIN_SIG_EXPORT_EPP_ID             (0x00000004)

/* values of WFSPINIMPORTRSAPUBLICKEY.dwRSASignatureAlgorithm */
#define WFS_PIN_SIGN_NA                   (0)
#define WFS_PIN_SIGN_RSASSA_PKCS1_V1_5    (0x00000001)
#define WFS_PIN_SIGN_RSASSA_PSS           (0x00000002)

/* values of WFSPINIMPORTRSAPUBLICKEYOUTPUT.dwRSAKeyCheckMode */
#define WFS_PIN_RSA_KCV_NONE              (0x00000000)
#define WFS_PIN_RSA_KCV_SHA1              (0x00000001)

/* values of WFSPINEXPORTRSAISSUERSIGNEDITEM.wExportItemType and */
/*           WFSPINEXPORTRSAEPPSIGNEDITEM.wExportItemType        */
#define WFS_PIN_EXPORT_EPP_ID             (0x0001)
#define WFS_PIN_EXPORT_PUBLIC_KEY         (0x0002)

/* values of WFSPINIMPORTRSASIGNEDDESKEY.dwRSAEncipherAlgorithm */
#define WFS_PIN_CRYPT_RSAES_PKCS1_V1_5    (0x00000001)
#define WFS_PIN_CRYPT_RSAES_OAEP          (0x00000002)

/* values of WFSPINGENERATERSAKEYPAIR.wExponentValue */
#define WFS_PIN_DEFAULT                   (0)
#define WFS_PIN_EXPONENT_1                (1)
#define WFS_PIN_EXPONENT_4                (2)
#define WFS_PIN_EXPONENT_16               (3)

/* values of WFSPINIMPORTRSASIGNEDDESKEYOUTPUT.wKeyLength and */
/*           WFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT.wKeyLength */
#define WFS_PIN_KEYSINGLE                 (0x0001)
#define WFS_PIN_KEYDOUBLE                 (0x0002)

/* values of WFSPINGETCERTIFICATE.wGetCertificate  */
#define WFS_PIN_PUBLICENCKEY              (1)
#define WFS_PIN_PUBLICVERIFICATIONKEY     (2)
```

```
/* values for WFSPINEMVIMPORTPUBLICKEY.wImportScheme */
#define WFS_PIN_EMV_IMPORT_PLAIN_CA       (0x0001)
#define WFS_PIN_EMV_IMPORT_CHKSUM_CA      (0x0002)
#define WFS_PIN_EMV_IMPORT_EPI_CA         (0x0003)
#define WFS_PIN_EMV_IMPORT_ISSUER         (0x0004)
#define WFS_PIN_EMV_IMPORT_ICC            (0x0005)
#define WFS_PIN_EMV_IMPORT_ICC_PIN        (0x0006)
#define WFS_PIN_EMV_IMPORT_PKCSV1_5_CA    (0x0007)

/* values for WFSPINDIGEST.wHashAlgorithm */
#define WFS_PIN_HASH_SHA1_DIGEST          (0x0001)


/* XFS PIN Errors */

#define WFS_ERR_PIN_KEYNOTFOUND           (-(PIN_SERVICE_OFFSET + 0))
#define WFS_ERR_PIN_MODENOTSUPPORTED      (-(PIN_SERVICE_OFFSET + 1))
#define WFS_ERR_PIN_ACCESSDENIED          (-(PIN_SERVICE_OFFSET + 2))
#define WFS_ERR_PIN_INVALIDID             (-(PIN_SERVICE_OFFSET + 3))
#define WFS_ERR_PIN_DUPLICATEKEY          (-(PIN_SERVICE_OFFSET + 4))
#define WFS_ERR_PIN_KEYNOVALUE            (-(PIN_SERVICE_OFFSET + 6))
#define WFS_ERR_PIN_USEVIOLATION          (-(PIN_SERVICE_OFFSET + 7))
#define WFS_ERR_PIN_NOPIN                 (-(PIN_SERVICE_OFFSET + 8))
#define WFS_ERR_PIN_INVALIDKEYLENGTH      (-(PIN_SERVICE_OFFSET + 9))
#define WFS_ERR_PIN_KEYINVALID            (-(PIN_SERVICE_OFFSET + 10))
#define WFS_ERR_PIN_KEYNOTSUPPORTED       (-(PIN_SERVICE_OFFSET + 11))
#define WFS_ERR_PIN_NOACTIVEKEYS          (-(PIN_SERVICE_OFFSET + 12))
#define WFS_ERR_PIN_NOTERMINATEKEYS       (-(PIN_SERVICE_OFFSET + 14))
#define WFS_ERR_PIN_MINIMUMLENGTH         (-(PIN_SERVICE_OFFSET + 15))
#define WFS_ERR_PIN_PROTOCOLNOTSUPP       (-(PIN_SERVICE_OFFSET + 16))
#define WFS_ERR_PIN_INVALIDDATA           (-(PIN_SERVICE_OFFSET + 17))
#define WFS_ERR_PIN_NOTALLOWED            (-(PIN_SERVICE_OFFSET + 18))
#define WFS_ERR_PIN_NOKEYRAM              (-(PIN_SERVICE_OFFSET + 19))
#define WFS_ERR_PIN_NOCHIPTRANSACTIVE     (-(PIN_SERVICE_OFFSET + 20))
#define WFS_ERR_PIN_ALGORITHMNOTSUPP      (-(PIN_SERVICE_OFFSET + 21))
#define WFS_ERR_PIN_FORMATNOTSUPP         (-(PIN_SERVICE_OFFSET + 22))
#define WFS_ERR_PIN_HSMSTATEINVALID       (-(PIN_SERVICE_OFFSET + 23))
#define WFS_ERR_PIN_MACINVALID            (-(PIN_SERVICE_OFFSET + 24))
#define WFS_ERR_PIN_PROTINVALID           (-(PIN_SERVICE_OFFSET + 25))
#define WFS_ERR_PIN_FORMATINVALID         (-(PIN_SERVICE_OFFSET + 26))
#define WFS_ERR_PIN_CONTENTINVALID        (-(PIN_SERVICE_OFFSET + 27))
#define WFS_ERR_PIN_SIG_NOT_SUPP          (-(PIN_SERVICE_OFFSET + 29))
#define WFS_ERR_PIN_INVALID_MOD_LEN       (-(PIN_SERVICE_OFFSET + 31))
#define WFS_ERR_PIN_INVALIDCERTSTATE      (-(PIN_SERVICE_OFFSET + 32))
#define WFS_ERR_PIN_KEY_GENERATION_ERROR  (-(PIN_SERVICE_OFFSET + 33))
#define WFS_ERR_PIN_EMV_VERIFY_FAILED     (-(PIN_SERVICE_OFFSET + 34))
#define WFS_ERR_PIN_RANDOMINVALID         (-(PIN_SERVICE_OFFSET + 35))
#define WFS_ERR_PIN_SIGNATUREINVALID      (-(PIN_SERVICE_OFFSET + 36))
#define WFS_ERR_PIN_SNSCDINVALID          (-(PIN_SERVICE_OFFSET + 37))
#define WFS_ERR_PIN_NORSAKEYPAIR          (-(PIN_SERVICE_OFFSET + 38))


/*=================================================================*/
/* PIN Info Command Structures and variables */
/*=================================================================*/

typedef struct _wfs_pin_status
{
    WORD                fwDevice;
    WORD                fwEncStat;
    LPSTR               lpszExtra;
} WFSPINSTATUS, * LPWFSPINSTATUS;

typedef struct _wfs_pin_caps
{
    WORD                wClass;
    WORD                fwType;
    BOOL                bCompound;
    USHORT              usKeyNum;
    WORD                fwAlgorithms;
    WORD                fwPinFormats;
    WORD                fwDerivationAlgorithms;
    WORD                fwPresentationAlgorithms;
    WORD                fwDisplay;
    BOOL                bIDConnect;
    WORD                fwIDKey;
    WORD                fwValidationAlgorithms;
    WORD                fwKeyCheckModes;
    LPSTR               lpszExtra;
} WFSPINCAPS, * LPWFSPINCAPS;

typedef struct _wfs_pin_key_detail
{
    LPSTR               lpsKeyName;
    WORD                fwUse;
    BOOL                bLoaded;
} WFSPINKEYDETAIL, * LPWFSPINKEYDETAIL;

typedef struct _wfs_pin_fdk
{
    ULONG               ulFDK;
    USHORT              usXPosition;
    USHORT              usYPosition;
} WFSPINFDK, * LPWFSPINFDK;

typedef struct _wfs_pin_func_key_detail
{
    ULONG               ulFuncMask;
    USHORT              usNumberFDKs;
    LPWFSPINFDK       * lppFDKs;
} WFSPINFUNCKEYDETAIL, * LPWFSPINFUNCKEYDETAIL;

typedef struct _wfs_pin_key_detail_ex
{
    LPSTR         lpsKeyName;
    DWORD         dwUse;
    BYTE          bGeneration;
    BYTE          bVersion;
    BYTE          bActivatingDate[4];
    BYTE          bExpiryDate[4];
    BOOL          bLoaded;
} WFSPINKEYDETAILEX, * LPWFSPINKEYDETAILEX;

/*=================================================================*/
/* PIN Execute Command Structures */
/*=================================================================*/

typedef struct _wfs_hex_data
{
    USHORT              usLength;
    LPBYTE              lpbData;
} WFSXDATA, * LPWFSXDATA;

typedef struct _wfs_pin_crypt
{
    WORD                wMode;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    WORD                wAlgorithm;
    LPSTR               lpsStartValueKey;
    LPWFSXDATA          lpxStartValue;
    BYTE                bPadding;
    BYTE                bCompression;
    LPWFSXDATA          lpxCryptData;
} WFSPINCRYPT, * LPWFSPINCRYPT;

typedef struct _wfs_pin_import
{
    LPSTR               lpsKey;
    LPSTR               lpsEncKey;
    LPWFSXDATA          lpxIdent;
    LPWFSXDATA          lpxValue;
    WORD                fwUse;
} WFSPINIMPORT, * LPWFSPINIMPORT;

typedef struct _wfs_pin_derive
```

```
{
    WORD                wDerivationAlgorithm;
    LPSTR               lpsKey;
    LPSTR               lpsKeyGenKey;
    LPSTR               lpsStartValueKey;
    LPWFSXDATA          lpxStartValue;
    BYTE                bPadding;
    LPWFSXDATA          lpxInputData;
    LPWFSXDATA          lpxIdent;
 } WFSPINDERIVE, * LPWFSPINDERIVE;

typedef struct _wfs_pin_getpin
{
    USHORT              usMinLen;
    USHORT              usMaxLen;
    BOOL                bAutoEnd;
    CHAR                cEcho;
    ULONG               ulActiveFDKs;
    ULONG               ulActiveKeys;
    ULONG               ulTerminateFDKs;
    ULONG               ulTerminateKeys;
} WFSPINGETPIN, * LPWFSPINGETPIN;

typedef struct _wfs_pin_entry
{
    USHORT              usDigits;
    WORD                wCompletion;
} WFSPINENTRY, * LPWFSPINENTRY;

typedef struct _wfs_pin_local_des
{
    LPSTR               lpsValidationData;
    LPSTR               lpsOffset;
    BYTE                bPadding;
    USHORT              usMaxPIN;
    USHORT              usValDigits;
    BOOL                bNoLeadingZero;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    LPSTR               lpsDecTable;
} WFSPINLOCALDES, * LPWFSPINLOCALDES;

typedef struct _wfs_pin_create_offset
{
    LPSTR               lpsValidationData;
    BYTE                bPadding;
    USHORT              usMaxPIN;
    USHORT              usValDigits;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    LPSTR               lpsDecTable;
} WFSPINCREATEOFFSET, * LPWFSPINCREATEOFFSET;

typedef struct _wfs_pin_local_eurocheque
{
    LPSTR               lpsEurochequeData;
    LPSTR               lpsPVV;
    WORD                wFirstEncDigits;
    WORD                wFirstEncOffset;
    WORD                wPVVDigits;
    WORD                wPVVOffset;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    LPSTR               lpsDecTable;
} WFSPINLOCALEUROCHEQUE, * LPWFSPINLOCALEUROCHEQUE;

typedef struct _wfs_pin_local_visa
{
    LPSTR               lpsPAN;
    LPSTR               lpsPVV;
    WORD                wPVVDigits;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
} WFSPINLOCALVISA, * LPWFSPINLOCALVISA;
```

```
typedef struct _wfs_pin_presentidc
{
    WORD                wPresentAlgorithm;
    WORD                wChipProtocol;
    ULONG               ulChipDataLength;
    LPBYTE              lpbChipData;
    LPVOID              lpAlgorithmData;
} WFSPINPRESENTIDC, * LPWFSPINPRESENTIDC;

typedef struct _wfs_pin_present_result
{
    WORD                wChipProtocol;
    ULONG               ulChipDataLength;
    LPBYTE              lpbChipData;
} WFSPINPRESENTRESULT, * LPWFSPINPRESENTRESULT;

typedef struct _wfs_pin_presentclear
{
    ULONG               ulPINPointer;
    USHORT              usPINOffset;
} WFSPINPRESENTCLEAR, * LPWFSPINPRESENTCLEAR;

typedef struct _wfs_pin_block
{
    LPSTR               lpsCustomerData;
    LPSTR               lpsXORData;
    BYTE                bPadding;
    WORD                wFormat;
    LPSTR               lpsKey;
    LPSTR               lpsKeyEncKey;
} WFSPINBLOCK, * LPWFSPINBLOCK;

typedef struct _wfs_pin_getdata
{
    USHORT              usMaxLen;
    BOOL                bAutoEnd;
    ULONG               ulActiveFDKs;
    ULONG               ulActiveKeys;
    ULONG               ulTerminateFDKs;
    ULONG               ulTerminateKeys;
} WFSPINGETDATA, * LPWFSPINGETDATA;

typedef struct _wfs_pin_key
{
    WORD          wCompletion;
    ULONG         ulDigit;
} WFSPINKEY, * LPWFSPINKEY;

typedef struct _wfs_pin_data
{
    USHORT              usKeys;
    LPWFSPINKEY        *lpPinKeys;
    WORD                wCompletion;
} WFSPINDATA, * LPWFSPINDATA;

typedef struct _wfs_pin_init
{
    LPWFSXDATA          lpxIdent;
    LPWFSXDATA          lpxKey;
} WFSPININIT, * LPWFSPININIT;

typedef struct _wfs_pin_local_banksys
{
    LPWFSXDATA          lpxATMVAC;
} WFSPINLOCALBANKSYS, * LPWFSPINLOCALBANKSYS;

typedef struct _wfs_pin_banksys_io
{
    ULONG               ulLength;
    LPBYTE              lpbData;
} WFSPINBANKSYSIO, * LPWFSPINBANKSYSIO;
```

```
typedef struct _wfs_pin_secure_message
    {
    WORD          wProtocol;
    ULONG         ulLength;
    LPBYTE        lpbMsg;
} WFSPINSECMSG, * LPWFSPINSECMSG;

typedef struct _wfs_pin_import_key_ex
{
    LPSTR       lpsKey;
    LPSTR       lpsEncKey;
    LPWFSXDATA  lpxValue;
    LPWFSXDATA  lpxControlVector;
    DWORD       dwUse;
    WORD        wKeyCheckMode;
    LPWFSXDATA  lpxKeyCheckValue;
} WFSPINIMPORTKEYEX, * LPWFSPINIMPORTKEYEX;

typedef struct _wfs_pin_enc_io
{
    WORD          wProtocol;
    ULONG         ulDataLength;
    LPVOID        lpvData;
} WFSPINENCIO, *LPWFSPINENCIO;

typedef struct _wfs_pin_import_rsa_public_key
{
    LPSTR        lpsKey;
    LPWFSXDATA   lpxValue;
    DWORD        dwUse;
    LPSTR        lpsSigKey;
    DWORD        dwRSASignatureAlgorithm;
    LPWFSXDATA   lpxSignature;
} WFSPINIMPORTRSAPUBLICKEY, * LPWFSPINIMPORTRSAPUBLICKEY;

typedef struct _wfs_pin_import_rsa_public_key_output
{
    DWORD        dwRSAKeyCheckMode;
    LPWFSXDATA   lpxKeyCheckValue;
} WFSPINIMPORTRSAPUBLICKEYOUTPUT, * LPWFSPINIMPORTRSAPUBLICKEYOUTPUT;

typedef struct _wfs_pin_export_rsa_issuer_signed_item
{
    WORD         wExportItemType;
    LPSTR        lpsName;
} WFSPINEXPORTRSAISSUERSIGNEDITEM, * LPWFSPINEXPORTRSAISSUERSIGNEDITEM;

typedef struct _wfs_pin_export_rsa_issuer_signed_item_output
{
    LPWFSXDATA   lpxValue;
    DWORD        dwRSASignatureAlgorithm;
    LPWFSXDATA   lpxSignature;
} WFSPINEXPORTRSAISSUERSIGNEDITEMOUTPUT, * LPWFSPINEXPORTRSAISSUERSIGNEDITEMOUTPUT;

typedef struct _wfs_pin_import_rsa_signed_des_key
{
    LPSTR        lpsKey;
    LPSTR        lpsDecryptKey;
    DWORD        dwRSAEncipherAlgorithm;
    LPWFSXDATA   lpxValue;
    DWORD        dwUse;
    LPSTR        lpsSigKey;
    DWORD        dwRSASignatureAlgorithm;
    LPWFSXDATA   lpxSignature;
} WFSPINIMPORTRSASIGNEDDESKEY, * LPWFSPINIMPORTRSASIGNEDDESKEY;

typedef struct _wfs_pin_import_rsa_signed_des_key_output
{
    WORD         wKeyLength;
    WORD         wKeyCheckMode;
    LPWFSXDATA   lpxKeyCheckValue;
} WFSPINIMPORTRSASIGNEDDESKEYOUTPUT, * LPWFSPINIMPORTRSASIGNEDDESKEYOUTPUT;

typedef struct _wfs_pin_generate_rsa_key
```

```
{
    LPSTR         lpsKey;
    DWORD         dwUse;
    WORD          wModulusLength;
    WORD          wExponentValue;
} WFSPINGENERATERSAKEYPAIR, * LPWFSPINGENERATERSAKEYPAIR;

typedef struct _wfs_pin_export_rsa_epp_signed_item
{
    WORD          wExportItemType;
    LPSTR         lpsName;
    LPSTR         lpsSigKey;
    DWORD         dwSignatureAlgorithm;
} WFSPINEXPORTRSAEPPSIGNEDITEM, * LPWFSPINEXPORTRSAEPPSIGNEDITEM;

typedef struct _wfs_pin_export_rsa_epp_signed_item_output
{
    LPWFSXDATA    lpxValue;
    LPWFSXDATA    lpxSelfSignature;
    LPWFSXDATA    lpxSignature;
} WFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT, * LPWFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT;



typedef struct _wfs_pin_load_certificate
{
    LPWFSXDATA    lpxLoadCertificate;
} WFSPINLOADCERTIFICATE, *LPWFSPINLOADCERTIFICATE;

typedef struct _wfs_pin_load_certificate_output
{
    LPWFSXDATA    lpxCertificateData;
} WFSPINLOADCERTIFICATEOUTPUT, *LPWFSPINLOADCERTIFICATEOUTPUT;

typedef struct _wfs_pin_get_certificate
{
    WORD          wGetCertificate;
} WFSPINGETCERTIFICATE, *LPWFSPINGETCERTIFICATE;

typedef struct _wfs_pin_get_certificate_output
{
    LPWFSXDATA    lpxCertificate;
} WFSPINGETCERTIFICATEOUTPUT, *LPWFSPINGETCERTIFICATEOUTPUT;

typedef struct wfs_pin_replace_certificate
{
    LPWFSXDATA    lpxReplaceCertificate;
} WFSPINREPLACECERTIFICATE, *LPWFSPINREPLACECERTIFICATE;

typedef struct _wfs_pin_replace_certificate_output
{
   LPWFSXDATA     lpxNewCertificateData;
} WFSPINREPLACECERTIFICATEOUTPUT, *LPWFSPINREPLACECERTIFICATEOUTPUT;

typedef struct _wfs_pin_start_key_exchange
{

    LPWFSXDATA    lpxRandomItem;
} WFSPINSTARTKEYEXCHANGE, *LPWFSPINSTARTKEYEXCHANGE;


typedef struct _wfs_pin_import_rsa_enciphered_pkcs7_key
{
    LPWFSXDATA    lpxImportRSAKeyIn;
    LPSTR         lpsKey;
    DWORD         dwUse;
} WFSPINIMPORTRSAENCIPHEREDPKCS7KEY, * LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEY;

typedef struct _wfs_pin_import_rsa_enciphered_pkcs7_key_output
{
    WORD          wKeyLength;
    LPWFSXDATA    lpxRSAData;
}WFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT, *LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT;
```

```
typedef struct _wfs_pin_emv_import_public_key
{
    LPSTR          lpsKey;
    DWORD          dwUse;
    WORD           wImportScheme;
    LPWFSXDATA     lpxImportData;
    LPSTR          lpsSigKey;
} WFSPINEMVIMPORTPUBLICKEY, * LPWFSPINEMVIMPORTPUBLICKEY;

typedef struct _wfs_pin_emv_import_public_key_output
{
    LPSTR          lpsExpiryDate;
} WFSPINEMVIMPORTPUBLICKEYOUTPUT, * LPWFSPINEMVIMPORTPUBLICKEYOUTPUT;

typedef struct _wfs_pin_digest
{
    WORD           wHashAlgorithm;
    LPWFSXDATA     lpxDigestInput;
} WFSPINDIGEST, * LPWFSPINDIGEST;

typedef struct _wfs_pin_digest_output
{
    LPWFSXDATA     lpxDigestOutput;
} WFSPINDIGESTOUTPUT, * LPWFSPINDIGESTOUTPUT;

typedef struct _wfs_pin_hsm_init
{
    WORD           wInitMode;
    LPWFSXDATA     lpxOnlineTime;
} WFSPINHSMINIT, * LPWFSPINHSMINIT;


/*=================================================================*/
/* PIN Message Structures */
/*=================================================================*/

typedef struct _wfs_pin_access
{
    LPSTR          lpsKeyName;
    LONG           lErrorCode;
} WFSPINACCESS, * LPWFSPINACCESS;


/* restore alignment */
#pragma pack(pop)

#ifdef __cplusplus
}       /*extern "C"*/
#endif

#endif    /* __INC_XFSPIN__H */
```

8. Appendix

This section is used to further explain concepts and functionality that needs further clarification.

The terminology as described below is used within the following sections.

Definitions and Abbreviations

ATM	Automated Teller Machine, used here for any type of self-service terminal, regardless whether it actually dispenses cash
CA	Certificate Authority
Certificate	A data structure that contains a public key and a name that allows certification of a public key belonging to a specific individual. This is certified using digital signatures.
Host	The remote system that an ATM communicates with.
KTK	Key Transport Key
PKI	Public Key Infrastructure
Private Key	That key of an entity's key pair that should only be used by that entity.
Public Key	That key of an entity's key pair that can be made public.
Symmetric Key	A key used with symmetric cryptography
Verification Key	A key that is used to verify the validity of a certificate
SignatureIssuer	An entity that signs the ATM´s public key at production time, may be the ATM manufacturer

Notation of Cryptographic Items and Functions

$\mathbf{SK_E}$	The private key belonging to entity E
$\mathbf{PK_E}$	The public belonging to entity E
$\mathbf{SK_{ATM}}$	The private key belonging to the ATM/PIN
$\mathbf{PK_{ATM}}$	The public key belonging to the ATM/PIN
$\mathbf{SK_{HOST}}$	The private key belonging to the Host
$\mathbf{PK_{HOST}}$	The public key belonging to the Host
$\mathbf{SK_{SI}}$	The private key belonging to Signature Issuer
$\mathbf{PK_{SI}}$	The public key belonging to Signature Issuer
$\mathbf{K_{NAME}}$	A symmetric key
$\mathbf{Cert_{HOST}}$	A Certificate that contains the public verification of the host and is signed by a trusted Certificate Authority.
$\mathbf{Cert_{ATM}}$	A Certificate that contains the ATM/PINpublic verification or encipherment key, which is signed by a trusted Certificate Authority.
$\mathbf{Cert_{CA}}$	The Certificate of a new Certificate Authority
$\mathbf{R_{ATM}}$	Random Number of the ATM/PIN
$\mathbf{I_{HOST}}$	Identifier of the Host
$\mathbf{K_{KTK}}$	Key Transport Key
$\mathbf{R_{HOST}}$	Random number of the Host
$\mathbf{I_{ATM}}$	Identifier of the ATM/PIN
$\mathbf{TP_{ATM}}$	Thumb Print of the ATM/PIN
Sign(SK_E)[D]	The signing of data block D, using the private key SK_E
Recover(PK_E)[S]	The recovery of the data block D from the signature S, using the private key PK_E
RSACrypt(PK_E)[D]	RSA Encryption of the data block D using the public key PK_E
Hash [M]	Hashing of a message M of arbitrary length to a 20 Byte hash value
Des(K) [D]	DES encipherment of an 8 byte data block D using the secret key K
Des^{-1}(K)[D]	DES decipherment of an 8 byte data block D using the 8 byte secret key K
Des3(K)[D]	Triple DES encipherment of an 8 byte data block D using the 16 byte secret key K = (K_L || K_R), equivalent to **Des(K_L) [Des^{-1}(K_R) [Des(K_L) [D]]]]**
Des3^{-1} (K) [D]	Triple DES decipherment of an 8 byte data block D using the 16 byte secret key K = (K_L || K_R), equivalent to **Des^{-1} (K_L) [Des (K_R) [Des^{-1} (K_L) [D]]]**
$\mathbf{Rnd_E}$	A random number created by entity E
$\mathbf{UI_E}$	Unique Identifier for entity E
(A || B)	Concatenation of A and B

8.1 Remote Key Loading Using Signatures

8.1.1 RSA Data Authentification & Digital Signatures

Digital signatures rely on a public key infrastructure (PKI). The PKI model involves an entity, such as a Host, having a pair of encryption keys – one private, one public. These keys work in consort to encrypt, decrypt and authenticate data. One way authentication occurs is through the application of a digital signature. For example:

- The Host creates some data that it would like to digitally sign;
- Host runs the data through a hashing algorithm to produce a hash or digest of the data. The digest is unique to every block of data – a digital fingerprint of the data, much smaller and therefore more economical to encrypt than the data itself;
- Digest is encrypted with the Host's private key.

This is the digital signature – a data block digest encrypted with the private key. The Host then sends the following to the ATM:

- data block
- digital signature
- Host's public key

To validate the signature, the ATM performs the following:

- ATM runs data through the standard hashing algorithm – the same one used by the Host – to produce a digest of the data received. Consider this digest$_2$;
- ATM uses the Host's public key to decrypt the digital signature. The digital signature was produced using the Host's private key to encrypt the data digest; therefore, when decrypted with the Host's public key it produces the same digest. Consider this digest$_1$. Incidentally, no other public key in the world would work to decrypt digest$_1$ – only the public key corresponding to the signing private key.
- ATM compares digest$_1$ with digest$_2$.

If digest$_1$ matches digest$_2$ exactly, the ATM has confirmed the following:

- Data was not tampered with in transit. Changing a single bit in the data sent from the Host to the ATM would cause digest$_2$ to be different than digest$_1$. Every data block has a unique digest; therefore, an altered data block is detected by the ATM.
- Public key used to decrypt the digital signature corresponds to the private key used to create it. No other public key could possibly work to decrypt the digital signature, so the ATM was not handed someone else's public key.

This gives an overview of how Digital Signatures can be used in Data Authentication. In particular, Signatures can be used to validate & securely install Encryption Keys. The following section describes Key Exchange and the use of Digital signatures.

8.1.2 RSA Secure Key Exchange using Digital Signatures

In summary, both end points, the ATM & the Host, inform each other of their Public Keys. This information is then used to securely send the PIN device Master Key to the ATM. A trusted third party, the Signature Issuer, is used to generate the signatures for the Public keys of each end point, ensuring their validity.

The detail of this is as follows :-

Purpose: The Host wishes to install a new master key (K_M) on the ATM securely.

Assumptions:

1. The Host has obtained the Public Key (PK_{SI}) from the Signature Issuer
2. The Host has provided the Signature Issuer with its Public Key (PK_{HOST}), and receives the corresponding signature Sign(SK_{SI})[PK_{HOST}]. The signature Issuer uses its own Private Key (SK_{SI}) to create this signature.
4. (Optional) The host obtains a list of the valid PIN device's Unique Identifiers. The Signature Issuer installs a Signature Sign(SK_{SI})[UI_{ATM}] for the Unique Id (UI_{ATM}) on the ATM PIN. The Signature Issuer uses SK_{SI} to do this
5. The Signature Issuer installs its Public Key (PK_{SI}) on the ATM PIN. It also derives and installs the Signature Sign(SK_{SI})[PK_{ATM}] of the ATM PIN's Public Key (PK_{ATM}) on the ATM PIN. The Signature Issuer uses SK_{SI} to do this.
6. The ATM PIN device additionally contains its own Public (PK_{ATM}) & Private Key (SK_{ATM}).

<u>Step 1</u>
The ATM PIN sends its Public Key to the Host in a secure structure:
The ATM PIN sends its ATM Public Key with its associated Signature. When the Host receives this information it will use the Signature Issuer's Public Key to validate the signature and obtain the ATM Public Key.

The XFS command used to export the PIN public key securely as described above is
WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM

<u>Step 2 (Optional)</u>
The Host verifies that the key it has just received is from a valid sender.
It does this by obtaining the PIN device unique identifier. The ATM PIN sends its Unique Identifier with its associated Signature. When the Host receives this information it will use the Signature Issuer's Public Key to validate the signature and retrieve the PIN Unique Identifier. It can then check this against the list it received from the Signature Issuer.

The XFS command used to export the PIN Unique Identifier is
WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM.

<u>Step 3</u>
The Host sends its public key to the ATM PIN:
The Host sends its Public Key and associated Signature. The ATM PIN verifies the signature using PK_{SI} and stores the key.

The XFS command used to export the PIN public key securely as described above is
WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY

<u>Step 4</u>
The ATM PIN receives its Master Key from the Host:
The Host encrypts the Master Key (K_M) with PK_{ATM}. A signature for this is then created using SK_{HOST}. The ATM PIN will then validate the signature using PK_{HOST} and then obtain the master key by decrypting using SK_{ATM}.

The XFS command used to exchange master symmetric keys as described above is
WFS_CMD_PIN_START_KEY_EXCHANGE
WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY.

<u>Step 4 – Alternative including random number</u>

The host requests the ATM PIN to begin the DES key transfer process and generate a random number.

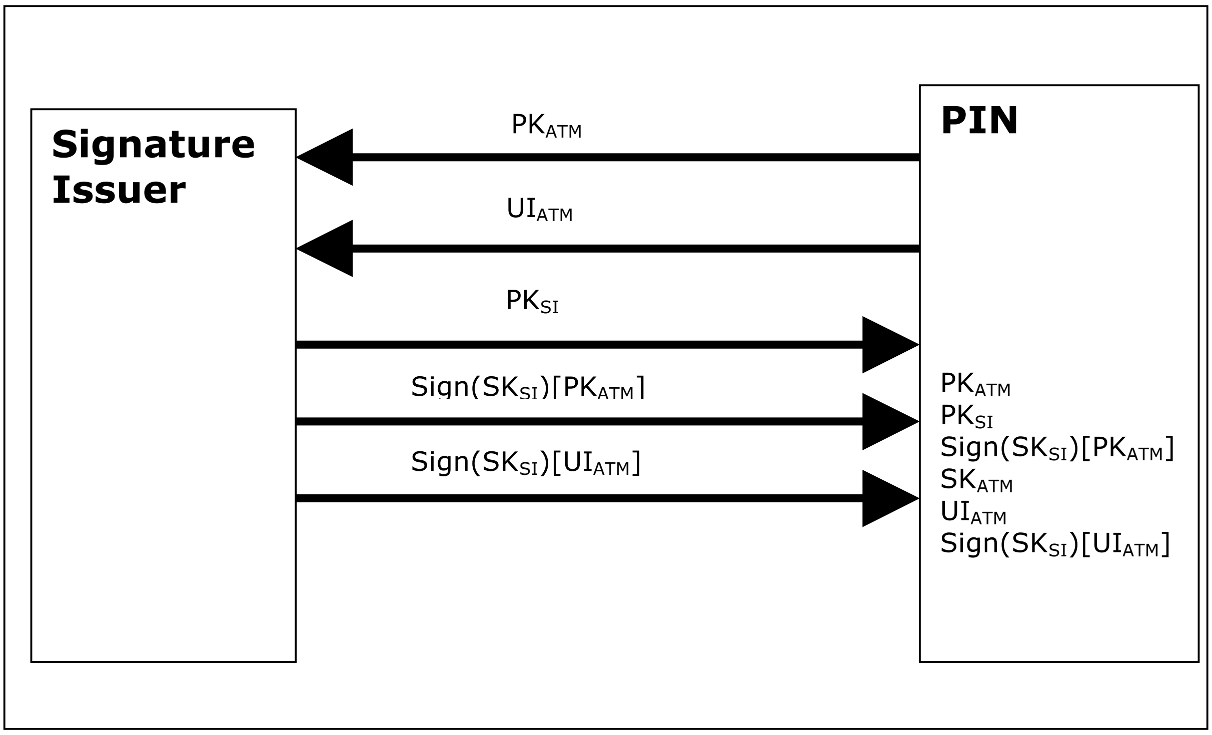
The Host encrypts the Master Key (K_M) with PK_{ATM}. A signature for the random number and encrypted key is then created using SK_{HOST}.
The ATM PIN will then validate the signature using $PK_{HOST,}$ verify the random number and then obtain the master key by decrypting using SK_{ATM}.

The XFS commands used to exchange master symmetric keys as described above is
WFS_CMD_PIN_START_KEY_EXCHANGE
WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY.

The following diagrams summaries the key exchange process described above:

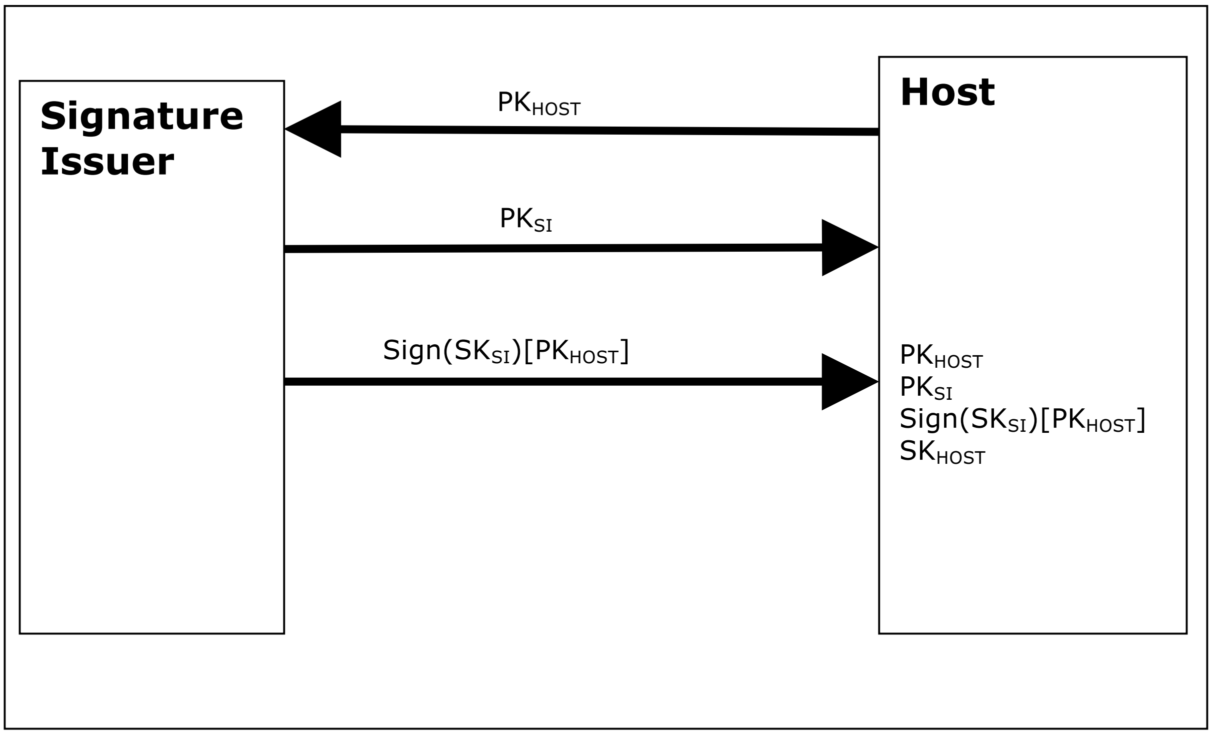
8.1.3 Initialization Phase – Signature Issuer & ATM PIN

This would typically occur in a secure manufacturing environment


Signature Issuer
PIN
PK_{ATM}
UI_{ATM}
PK_{SI}
$Sign(SK_{SI})[PK_{ATM}]$
$Sign(SK_{SI})[UI_{ATM}]$
PK_{ATM}
PK_{SI}
$Sign(SK_{SI})[PK_{ATM}]$
SK_{ATM}
UI_{ATM}
$Sign(SK_{SI})[UI_{ATM}]$

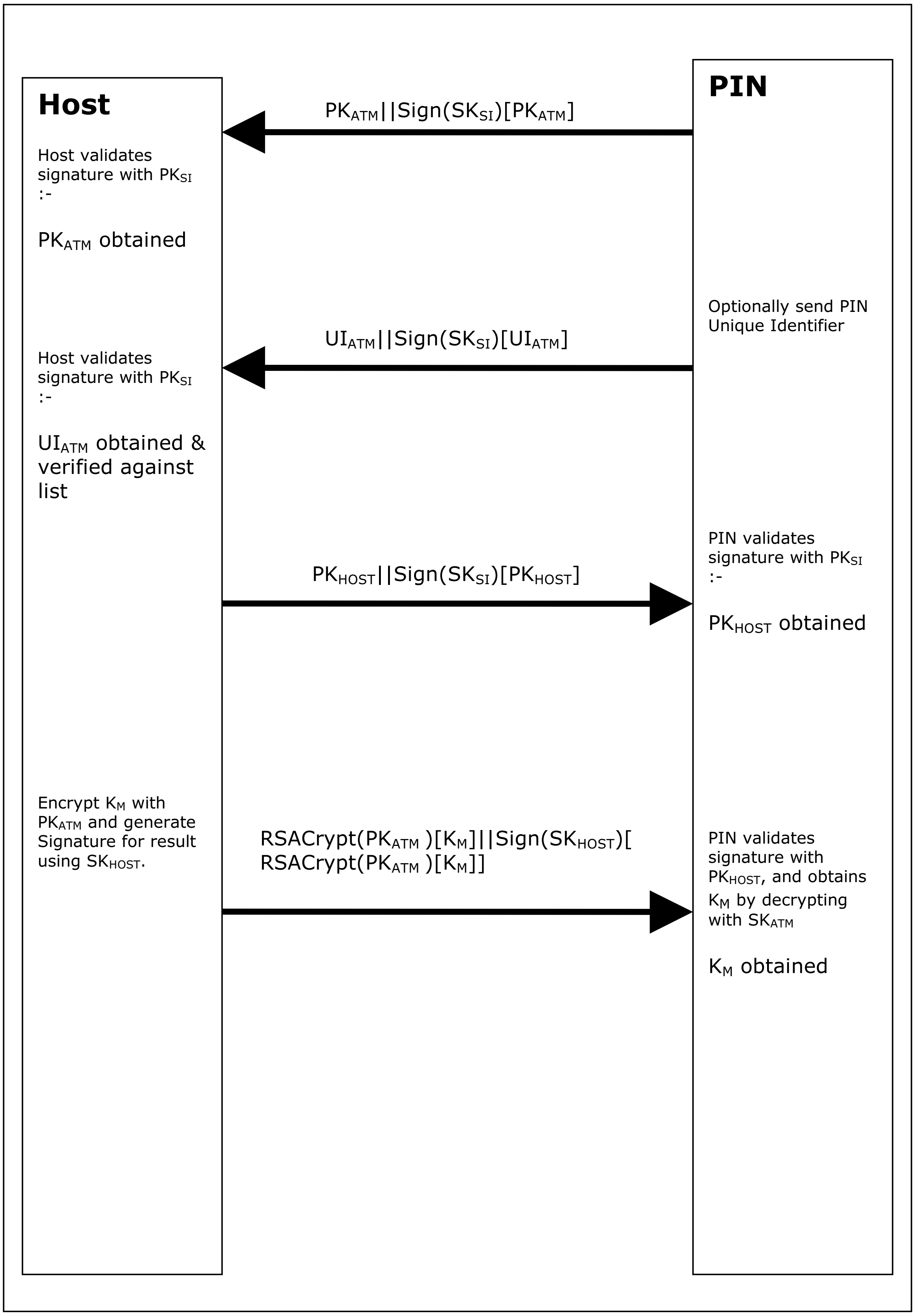
8.1.4 Initialization Phase – Signature Issuer & Host

This would typically occur in a secure offline environment



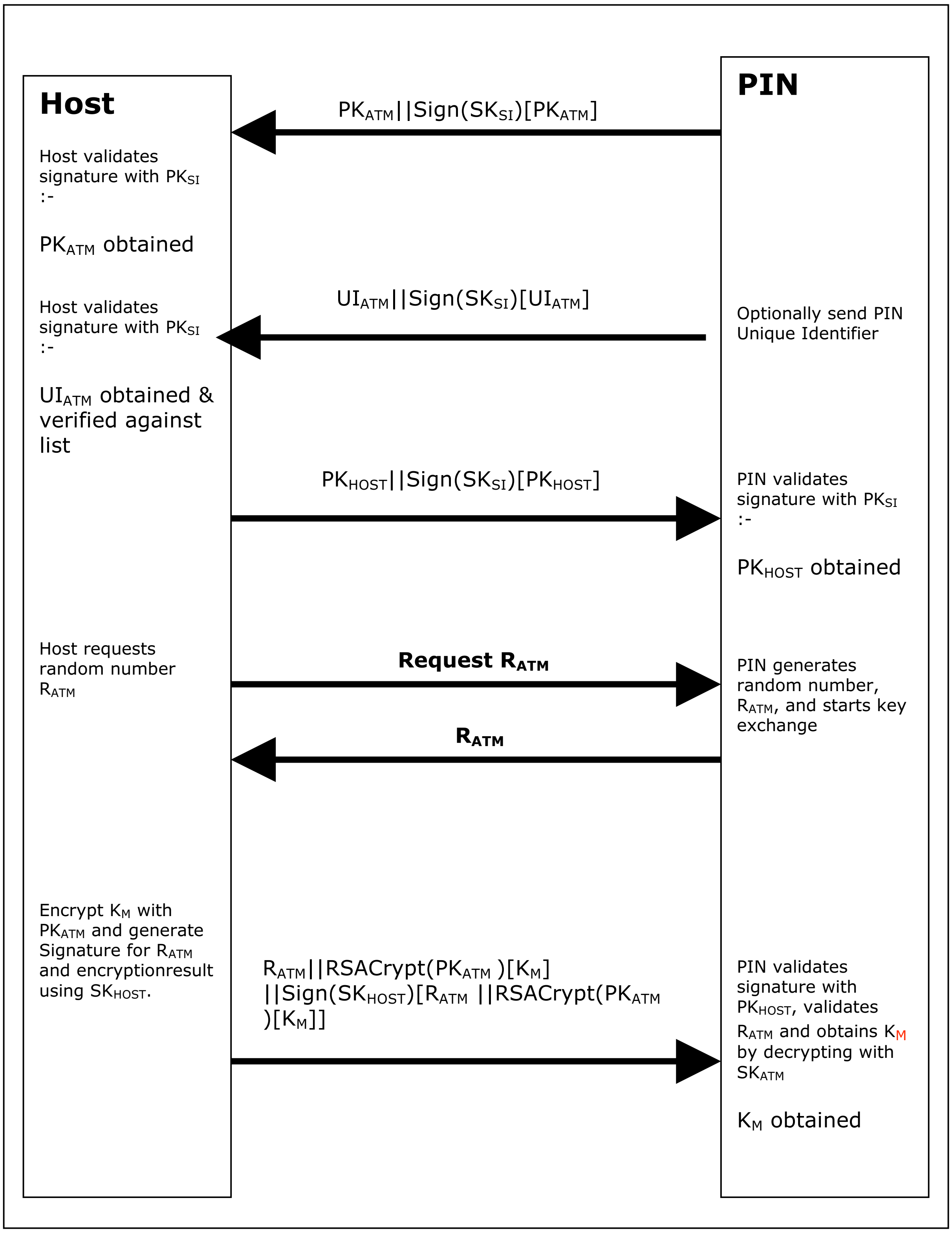
8.1.5 Key Exchange – Host & ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key in a typical ATM Network. The following is the recommended sequence of interchanges


Host
PIN
$PK_{ATM}||Sign(SK_{SI})[PK_{ATM}]$
Host validates signature with PK_{SI} :-
PK_{ATM} obtained
Optionally send PIN Unique Identifier
$UI_{ATM}||Sign(SK_{SI})[UI_{ATM}]$
Host validates signature with PK_{SI} :-
UI_{ATM} obtained & verified against list
PIN validates signature with PK_{SI} :-
$PK_{HOST}||Sign(SK_{SI})[PK_{HOST}]$
PK_{HOST} obtained
Encrypt K_M with PK_{ATM} and generate Signature for result using SK_{HOST}.
$RSACrypt(PK_{ATM})[K_M]||Sign(SK_{HOST})[RSACrypt(PK_{ATM})[K_M]]$
PIN validates signature with PK_{HOST}, and obtains K_M by decrypting with SK_{ATM}
K_M obtained

8.1.6 Key Exchange (with random number) – Host & ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key when the PIN device and service provider supports the WFS_CMD_PIN START_KEY_EXCHANGE command.


Host
PIN
$PK_{ATM}||Sign(SK_{SI})[PK_{ATM}]$
Host validates signature with PK_{SI} :-
PK_{ATM} obtained
$UI_{ATM}||Sign(SK_{SI})[UI_{ATM}]$
Optionally send PIN Unique Identifier
Host validates signature with PK_{SI} :-
UI_{ATM} obtained & verified against list
$PK_{HOST}||Sign(SK_{SI})[PK_{HOST}]$
PIN validates signature with PK_{SI} :-
PK_{HOST} obtained
Host requests random number R_{ATM}
Request R_{ATM}
PIN generates random number, R_{ATM}, and starts key exchange
R_{ATM}
Encrypt K_M with PK_{ATM} and generate Signature for R_{ATM} and encryptionresult using SK_{HOST}.
$R_{ATM}||RSACrypt(PK_{ATM})[K_M]||Sign(SK_{HOST})[R_{ATM}||RSACrypt(PK_{ATM})[K_M]]$
PIN validates signature with PK_{HOST}, validates R_{ATM} and obtains K_M by decrypting with SK_{ATM}
K_M obtained

8.1.7 Default Keys and Security Item loaded during manufacture

Several keys and a security item which are mandatory for the 2 party/Signature authentication scheme are installed during manufacture. These items are given fixed names so multi-vendor applications can be developed without the need for vendor specific configuration tools.

Item Name	Item Type	Signed by	Description
"_SigIssuerVendor"	Public Key	N/A	The public key of the signature issuer, ie PK_{SI}
"_EPPCryptKey"	Public/Private key-pair	The private key associated with _SigIssuerVendor	The key-pair used to encrypt and decrypt the symmetric key, ie SK_{ATM} and PK_{ATM}.The public key is used for encryption by the host and the private for decryption by the EPP.

In addition the following optional keys can be loaded during manufacture.

Item Name	Item Type	Signed by	Description
"_EPPSignKey"	Public/Private key-pair	The private key associated with _SigIssuerVendor	A key-pair where the private key is used to sign data, e.g. other generated key pairs.

8.2 Remote Key Loading Using Certificates

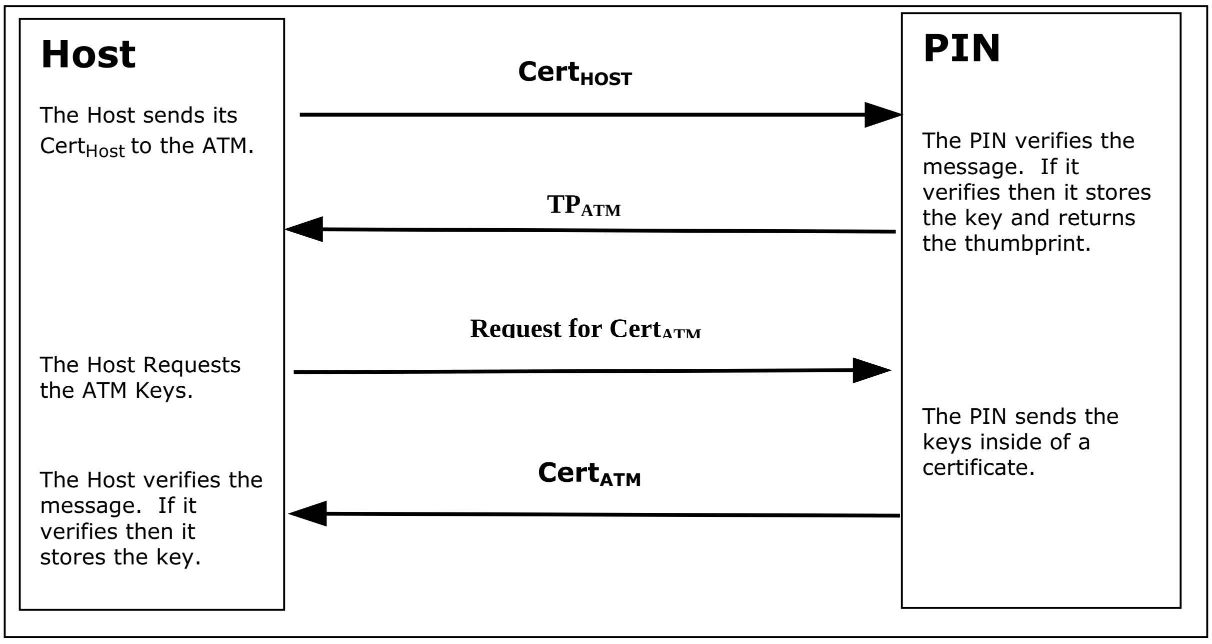
8.2.1 Certificate Exchange and Authentication

In summary, both end points, the ATM & the Host, inform each other of their Public Keys. This information is then used to securely send the PIN device Master Key to the ATM. A trusted third party, Certificate Authority (or a HOST if it becomes the new CA), is used to generate the certificates for the Public Keys of each end point, ensuring their validity. NOTE: The WFS_CMD_PIN_LOAD_CERTIFICATE and WFS_CMD_PIN_GET_CERTIFICATE do not necessarily need to be called in the order below. This way though is the recommend way.
The following flow is how the exchange authentication takes place:

1) WFS_CMD_PIN_LOAD_CERTIFICATE is called. In this message contains the host certificate, which has been signed by the trusted CA. The encryptor uses the Public Key of the CA (loaded at the time of production) to verify the validity of the certificate. If the certificate is valid, the encryptor stores the HOST's Public Verification Key.

2) Next, WFS_CMD_PIN_GET_CERTIFICATE is called. The encryptor then sends a message that contains a certificate, which is signed by the CA and is sent to the HOST. The HOST uses the Public Key from the CA to verify the certificate. If valid then the HOST stores the encryptors verification or encryption key (primary or secondary this depends on the state of the encryptor).

The following diagram below shows how the Host and ATM Load and Get each others information to make Remote Key Loading possible:


Host
The Host sends its Cert$_{Host}$ to the ATM.
The Host Requests the ATM Keys.
The Host verifies the message. If it verifies then it stores the key.
Cert$_{HOST}$
TP$_{ATM}$
Request for Cert$_{ATM}$
Cert$_{ATM}$
PIN
The PIN verifies the message. If it verifies then it stores the key and returns the thumbprint.
The PIN sends the keys inside of a certificate.

8.2.2 Remote Key Exchange

After the above has been completed, the HOST is ready to load the key into the encryptor. The following is done to complete this and the application must complete the Remote Key Exchange in this order:

1) First, the WFS_CMD_PIN_START_KEY_EXCHANGE is called. This returns R_{ATM} from the encryptor to be used in the authenticating the WFS_CMD_PIN_ IMPORT_RSA_ENCHIPERED_PKCS7_KEY message.

2) Next, WFS_CMD_PIN_ IMPORT_RSA_ENCIPHERED_PKCS7_KEY is called. This commands sends down the KTK to the encryptor. The following items below shows how this is accomplished.

a) HOST has obtained a Key Transport Key and wants to transfer it to the encryptor. HOST constructs a key block containing an identifier of the HOST, I_{HOST}, and the key, K_{KTK}, and enciphers the block, using the encryptor's Public Encryption Key from the WFS_CMD_PIN_GET CERTIFICATE command.

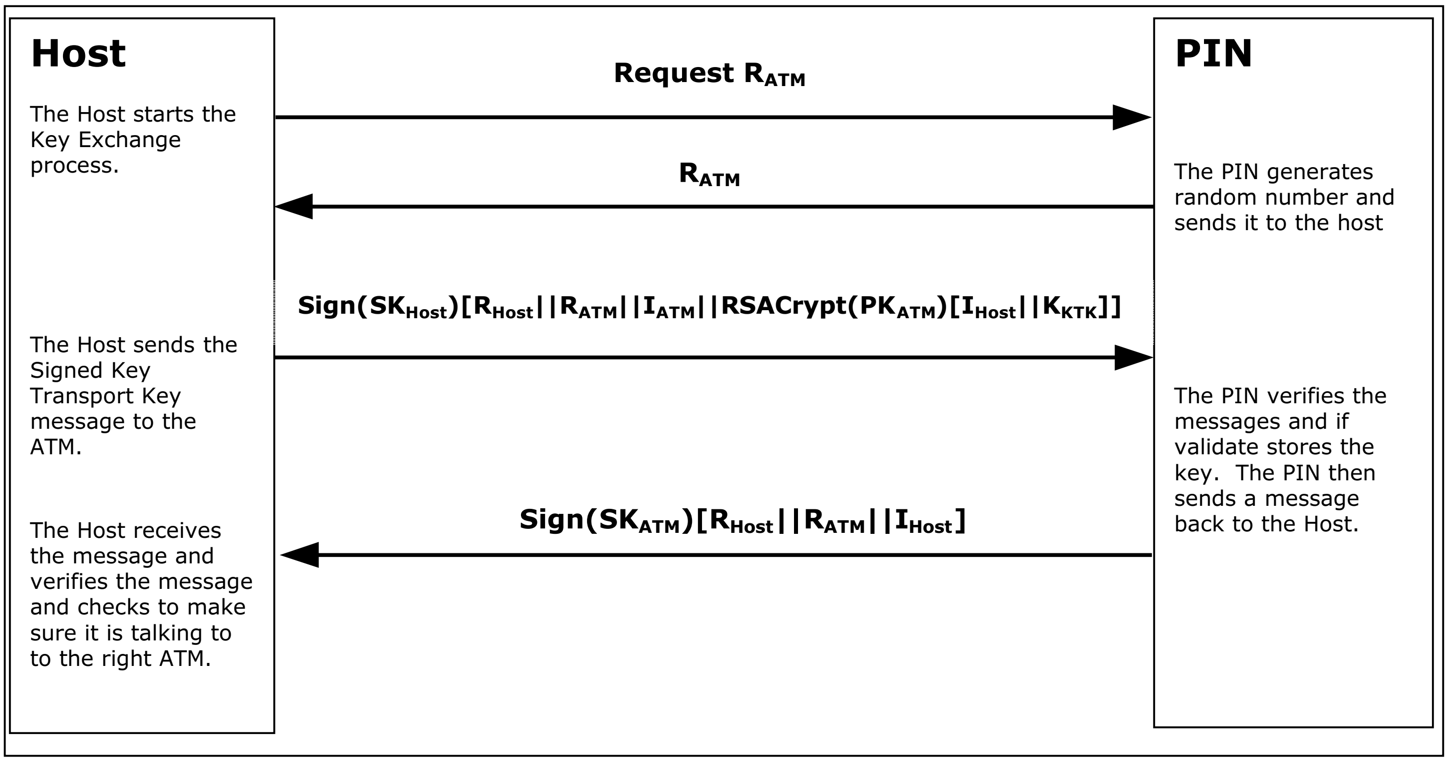
b) After completing the above, the HOST generates random data and builds the outer message containing the random number of the host, R_{HOST}, the random number of the encryptor returned in the WFS_CMD_PIN_START_KEY_EXCHANGE command, R_{ATM} , the identifier of the encryptor, I_{ENC}, and the enciphered key block. The HOST signs the whole block using its private signature key and sends the message down to the encryptor.

The encryptor then verifies the HOST's signature on the message by using the HOST's Public Verification Key. Then checks the identifier and the random number of the encryptor passed in the message to make sure that the encryptor is talking to the right HOST. The encryptor then deciphers the enciphered block using it's private verification key. After the message has been deciphered, the encryptor checks the Identifier of the HOST. Finally, if everything checks out to this point the encryptor will load the Key Transport Key. NOTE: If one step of this verification occurs the encryptor will return the proper error to the HOST.

c) After the Key Transport Key has been accepted, the encryptor constructs a message that contains the random number of the host, the random number of the encryptor and the HOST identifier all signed by the private signature key of the encyrptor. This message is sent to the host.

d) The HOST verifies the message sent from the encryptor by using the ATM's public verification key. The HOST then checks the identifier of the host and then compares the identifier in the message with the one stored in the HOST. Then checks the random number sent in the message and to the one stored in the HOST. The HOST finally checks the encryptor's random number with the one received in received in the WFS_CMD_PIN_START_KEY_EXCHANGE command.

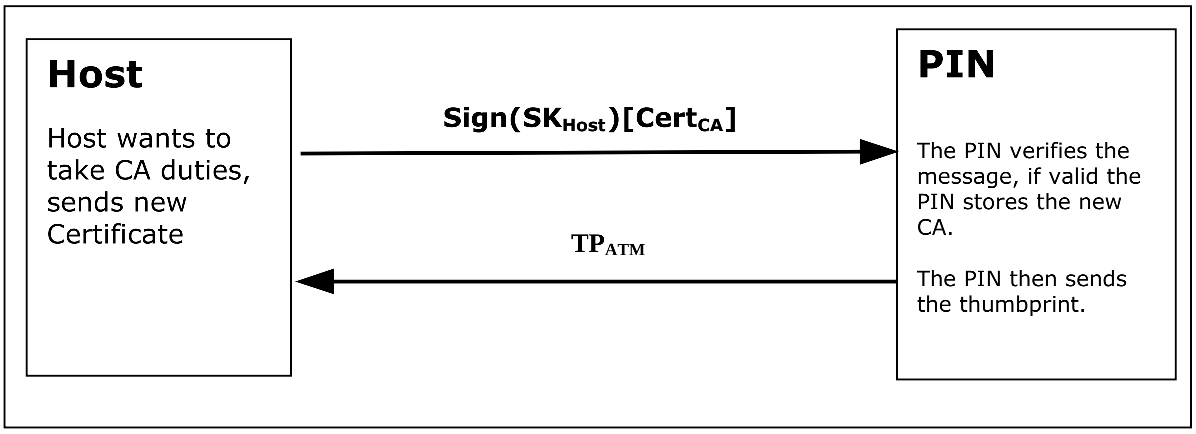
The following diagram below shows how the Host and ATM transmits the Key Transport Key.


Host
PIN
Request R_{ATM}
The Host starts the Key Exchange process.
R_{ATM}
The PIN generates random number and sends it to the host
$Sign(SK_{Host})[R_{Host}||R_{ATM}||I_{ATM}||RSACrypt(PK_{ATM})[I_{Host}||K_{KTK}]]$
The Host sends the Signed Key Transport Key message to the ATM.
The PIN verifies the messages and if validate stores the key. The PIN then sends a message back to the Host.
$Sign(SK_{ATM})[R_{Host}||R_{ATM}||I_{Host}]$
The Host receives the message and verifies the message and checks to make sure it is talking to to the right ATM.

8.2.3 Replace Certificate

After the key is been loaded into the encryptor, the following could be completed:

1) (Optional) WFS_CMD_PIN_REPLACE_CERTIFICATE. This is called by entity that would like to take over the job of being the CA. The new CA requests a Certificate from the previous Certificate Authority. The HOST must over-sign the message to take over the role of the CA to ensure that the encryptor accepts the new Certificate Authority. The HOST sends the message to the encryptor. The encryptor uses the HOST's Public Verification Key to verify the HOST's signature. The encryptor uses the previous CA's Public Verification Key to verify the signature on the new Certificate sent down in the message. If valid, the EPP stores the new CA's certificate and uses the new CA's Public Verification Key, as it's new CA verification key. The diagram below shows how the Host and the ATM communicates to load the new CA.


Host
Host wants to take CA duties, sends new Certificate
Sign(SK_Host)[Cert_CA]
TP_ATM
PIN
The PIN verifies the message, if valid the PIN stores the new CA.
The PIN then sends the thumbprint.

8.2.4 Primary and Secondary Certificates

- Primary and Secondary Certificates for both the Public Verification Key and Public Encipherment Key are pre-loaded into the encryptor. Primary Certificates will be used until told otherwise by the HOST via the WFS_CMD_PIN_LOAD_CERTIFICATE or WFS_CMD_PIN_REPLACE_CERTIFICATE commands. This change in state will be specified in the PKCS #7 message of the WFS_CMD_PIN_LOAD_CERTIFICATE or WFS_CMD_PIN_REPLACE_CERTIFICATE commands. The reason why the HOST would want to change states is because the HOST thinks that the Primary Certificates have been compromised.

- After the HOST tells the encryptor to shift to the secondary certificate state, and then only Secondary Certificates can be used. The encryptor will no longer be able to go back to the Primary State and any attempts from the HOST to get or load a Primary Certificate will return an error. When either Primary or Secondary certificates are compromised it is up to the vendor on how the encryptor should be handled with the manufacturer.

8.3 German ZKA GeldKarte

The PIN service is able to handle the German "Geldkarte", which is an electronic purse specified by the ZKA (Zentraler Kreditausschuß).

For anyone attempting to write an application that handles these chip cards, it is essential to read and understand the specifications published by

Bank-Verlag, Köln
Postfach 30 01 91
D-50771 Köln

Phone: +49 221 5490-0

Fax: +49 221 5490-120

8.3.1 How to use the SECURE_MSG commands

This is to describe how an application should use the WFS_CMD_PIN_SECURE_MSG_SEND and WFS_CMD_PIN_SECURE_MSG_RECEIVE commands for transactions involving chipcards with a German ZKA GeldKarte chip.

- Applications must call SECURE_MSG_SEND for every command they send to the chip or to a host system, including those commands that do not actually require secure messaging. This enables the service provider to remember security-relevant data that may be needed or checked later in the transaction.
- Applications must pass a complete message as input to SECURE_MSG_SEND, with all fields - including those that will be filled by the service provider - being present in the correct length. All fields that are not filled by the service provider must be filled with the ultimate values in order to enable MACing by the service provider.
- Every command SECURE_MSG_SEND that an application issues must be followed by exactly one command SECURE_MSG_RECEIVE that informs the service provider about the response from the chip or host. If no response is received (timeout or communication failure) the application must issue a SECURE_MSG_RECEIVE command with **`lpSecMsgIn->lpbMsg = NULL`** to inform the service provider about this fact.
- If a system is restarted after a SECURE_MSG_SEND was issued to the service provider but before the SECURE_MSG_RECEIVE was issued, the restart has the same effect as a SECURE_MSG_RECEIVE command with **`lpSecMsgIn->lpbMsg = NULL`**.
- Between a SECURE_MSG_SEND and the corresponding SECURE_MSG_RECEIVE no SECURE_MSG_SEND with the same **`lpSecMsgIn->wProtocol`** must be issued. Other WFS_CMD_PIN... commands – including SECURE_MSG_SEND / RECEIVE with different **`wProtocol`** – may be used.

8.3.2 Protocol WFS_PIN_PROTISOAS

This protocol handles ISO8583 messages between an ATM and an authorization system (AS).

Only messages in the new ISO format, with new PAC/MAC-format using session keys and Triple-DES are supported.

Authorization messages may be used to dispense the amount authorized in cash or to load the amount into an electronic purse (GeldKarte).

For loading a GeldKarte the only type of authorization supported is a transaction originating from track 3 of a German ec-card (message types 0200/0210 for authorization and 0400/0410 for reversal)

For dispensing cash, transactions originating from international cards (message types 0100/0110 and 0400/0410) are supported as well.

The following bitmap positions are filled by the service provider:

- BMP11 Trace-Nummer
- BMP52 PAC
- BMP57 Verschlüsselungsparameter (only the challenge values RND_{MES} and RND_{PAC})
- BMP64 MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the service provider and have to be filled by the application:

- Nachrichtentyp
- BMP3 Abwicklungskennzeichen (only for GeldKarte, not for cash)
- BMP4 Transaktionsbetrag (only for GeldKarte, not for cash)
- BMP41 Terminal-ID
- BMP42 Betreiber-BLZ

For a documentation of authorization messages see:

Regelwerk für das deutsche ec-Geldautomaten-System
Stand: 22. Nov. 1999

Bank-Verlag, Köln
Autorisierungszentrale GA/POS der privaten Banken
Spezifikation für GA-Betreiber
Version 3.12
31. Mai 2000

dvg Hannover
Schnittstellenbeschreibung für Autorisierungsanfragen bei nationalen GA-Verfügungen unter Verwendung der Spur 3
Version 2.5
Stand: 15.03.2000

dvg Hannover
Schnittstellenbeschreibung für Autorisierungsanfragen bei internationalen Verfügungen unter Verwendung der Spur 2
Version 2.6
Stand: 30.03.2000

8.3.3 Protocol WFS_PIN_PROTISOLZ

This protocol handles ISO8583 messages between a „Ladeterminal" and a „Ladezentrale" (LZ).

Only messages in the new ISO format, with new MAC-format using session keys and Triple-DES are supported.

Both types of GeldKarte chip (type 0 = DEM, type 1 = EUR) are supported.

The following bitmap positions are filled by the service provider:

- BMP11: Trace-Nummer
- BMP57: Verschlüsselungsparameter (only the challenge value $\mathrm{RND}_{\mathrm{MES}}$)
- BMP64: MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the service provider and have to be filled by the application:

- Nachrichtentyp
- BMP3: Abwicklungskennzeichen
- BMP4: Transaktionsbetrag
- BMP12: Uhrzeit
- BMP13: Datum
- BMP25: Konditionscode
- BMP41: Terminal-ID
- BMP42: Betreiber-BLZ (caution: "Ladeentgelt" also in BMP42 is not set by the EPP)
- BMP61: Online-Zeitpunkt
- BMP62: Chipdaten

The following bitmap positions are only checked if they are available:

- BMP43: Standort
- BMP60: Kontodaten Ladeterminal

For a documentation of the Ladezentrale interface see [Ref. 6].

8.3.4 Protocol WFS_PIN_PROTISOPS

This protocol handles ISO8583 messages between a terminal and a "Personalisierungsstelle" (PS). These messages are about OPT.

The service provider creates the whole message with WFS_CMD_PIN_SECURE_MSG_SEND, including message type and bitmap.

For a documentation of the Personalisierungsstelle interface see [Ref. 6] and [Ref. 7].

8.3.5 Protocol WFS_PIN_PROTCHIPZKA

This protocol is intended to handle messages between the application and a GeldKarte.

Both types of GeldKarte are supported.

Both types of load transactions ("Laden vom Kartenkonto" and "Laden gegen andere Zahlungsmittel") are supported.

See the chapter "Command Sequence" below for the actions that service providers take for the various chip card commands.

Only the command APDUs to and the response APDUs from the chip must be passed to the service provider, the ATR (answer to reset) data from the chip is not passed to the service provider.

For a documentation of the chip commands used to load a GeldKarte see [Ref. 6].

8.3.6 Protocol WFS_PIN_PROTRAWDATA

This protocol is intended for vendor-specific purposes. Generally the use of this protocol is not recommended and should be restricted to issues that are impossible to handle otherwise.

For example a HSM that requires vendor-specific, cryptographically secured data formats for importing keys or terminal data may use this protocol.

Applicaton programmers should be aware that the use of this command may prevent their applications from running on different hardware.

8.3.7 Protocol WFS_PIN_PROTPBM

This protocol handles host messages between a terminal and a host system, as specified by PBM protocol.

For a documentation of this protocol see [Ref. 8] – [Ref. 13].

Some additions are defined to the PBM protocol in order to satisfy the German ZKA 3.0 PAC/MAC standard. See [Ref. 14].

The commands WFS_CMD_PIN_SECURE_MSG_SEND and WFS_CMD_PIN_SECURE_MSG_RECEIVE handle the PAC and MAC in the VARDATA ‘K’ or ‘Q’ subfield of transactions records and responses. The MAC in the traditional MACODE field is not affected.

In order to enable the service provider to understand the messages, the application must provide the messages according to the following rules:

- All alphanumeric fields must be coded in EBCDIC
- Pre-Edit (padding and blank compression) must not be done by the application. The service provider will check the MACMODE field and will perform the pre-edit according to what the MACMODE field intends.

- In order to enable the service provider to find the vardata subfield ‘K’ or ‘Q’, it must be included in the message by the application, with the indicator ‘K’ or ‘Q’ and its length set.
- Because CARDDATA (track 2) and T3DATA (track 3) fields always take part in the MAC computation for a transaction record, these fields must be included in the message, even if they already have been sent to the host in a previous transaction record and the CI-Option SHORTREC prevents them from being sent again.

8.3.8 Protocol WFS_PIN_PROTHSMLDI

With this protocol an application can request information about the personalized OPT groups.

The information returned consists of personalization record like in BMP62 of an OPT response but without MAC.

Data format:

```
XX XX VV        group ID and version number (BCD format)
XX              number of LDIs within the group (BCD format)
...
first LDI of the group
...
last LDI of the group
XX XX VV        group ID and version number (BCD format)
...
etc. for several groups
```

Each LDI consists of

NN	Number of the LDI
00	Alg. code
LL	Length of the following data
XX...XX	data of the LDI

For each group ID the Service Provider must always return the standard LDI. LDI 01 must also be returned for groups AF XX VV. Further LDIs can be returned optionally.

8.3.9 Command Sequence

The following list shows the sequence of actions an application has to take for the various GeldKarte Transactions. Please note that this is a summary and is just intended to clarify the purpose of the chipcard-related WFS_CMD_PIN_... commands. In no way it can replace the ZKA specifications mentioned above.

Command WFS_CMD_PIN_...	**wProtocol WFS_PIN_PROT...**	**lpbMsg**	**Service Provider´s actions**
Preparation for Load/Unload			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	recognize type of chip
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_ID	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_ID	store EF_ID
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_LLOG	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_LLOG	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ_RECORD EF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_BÖRSE	

Command WFS_CMD_PIN_...	**wProtocol WFS_PIN_PROT...**	**lpbMsg**	**Service Provider´s actions**
SECURE_MSG_SEND	CHIPZKA	Command APDU READ_RECORD EF_BETRAG	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_BETRAG	
Load against other ec-Card			
SECURE_MSG_SEND	CHIPZKA	**for type 0 chips only** Command APDU READ RECORD EF_KEYD	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_KEYD	
SECURE_MSG_SEND	CHIPZKA	**for type 1 chips only** Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND1 from Chip	store RND1
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN with Secure Msg.	fill -Terminal ID -Traceno. -RND2 -MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	store response APDU for later check of ISOLZ message, BMP 62
SECURE_MSG_SEND	ISOAZ	ISO8583 message 0200 Authorization Request	fill - Traceno. (BMP 11) - PAC (BMP 52) - RND_{MES} + RND_{PAC} (BMP 57) - MAC (BMP 64) check other security relevant fields
SECURE_MSG_RECEIVE	ISOAZ	ISO8583 message 0210 Authorization Response	check MAC and other security relevant fields
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0200 Ladeanfrage	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND3 from chip	store RND3
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN with Secure Msg.	provide complete command from BMP62 of ISOLZ response , compute command MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check response MAC
GET_JOURNAL	ISOLZ	Vendor specific	
GET_JOURNAL	ISOAZ	Vendor specific	
Reversal of a Load against other ec-Card			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND5 from chip	store RND5

Command WFS_CMD_PIN_...	**wProtocol WFS_PIN_PROT...**	**lpbMsg**	**Service Provider´s actions**
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN with Secure Msg.	fill -Terminal ID -Traceno. -RND6 -Keyno. KGK_{LT} -MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	store response APDU for later check of ISOLZ message, BMP 62
SECURE_MSG_SEND	ISOAZ	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - PAC (BMP 52) - RND_{MES} + RND_{PAC} (BMP 57) - MAC (BMP 64) check other security relevant fields
SECURE_MSG_RECEIVE	ISOAZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields.
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND7 from chip	store RND7
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN with Secure Msg.	provide complete command from BMP62 of ISOLZ response , compute command MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check response MAC
GET_JOURNAL	ISOLZ	Vendor specific	
GET_JOURNAL	ISOAZ	Vendor specific	

PIN Verification Type 0			
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND0 from chip	store RND0
SECURE_MSG_SEND	CHIPZKA	Command APDU EXTERNAL AUTHENTICATE	fill -Keyno. K_{INFO} -ENCRND
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU PUT DATA	fill RND1
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_INFO with Secure Messaging	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_INFO	check MAC
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND2 from chip	store RND2
SECURE_MSG_SEND	CHIPZKA	Command APDU VERIFY	provide complete command APDU
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
PIN Verification Type 1			
SECURE_MSG_SEND	CHIPZKA	Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND0 from chip	store RND0
SECURE_MSG_SEND	CHIPZKA	Command APDU MUTUAL AUTHENTICATE	fill ENC0
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check ENC1
SECURE_MSG_SEND	CHIPZKA	Command APDU VERIFY	provide complete command APDU
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check MAC
„Laden vom Kartenkonto“ (both types)			
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0200 Ladeanfrage	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields.
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	

Reversal of a „Laden vom Kartenkonto“			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN	fill -Terminal ID -Traceno.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	
Unload			
SECURE_MSG_SEND	CHIPZKA	ENTLADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message Entladeanfrage 0200	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Entladeantwort 0210	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	ENTLADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	ENTLADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message Entladequittung 0202	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Entladebestätigung 0212	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	Command APDU ENTLADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	

Repeated Messages (Stornowiederholung / Entladequittungswiederholung)			
SECURE_MSG_SEND	ISOLZ	ISO8583 message Stornowiederholung 0401 or Entladequittungswiederholung 0203	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Stornoantwort 410 or Entladebestätigung 0212	check MAC and other security relevant fields
GET_JOURNAL	ISOLZ	Vendor specific	

8.4 EMV Support

EMV support by this specification consists in the ability of importing Certification Authority and Chip Card Public Keys, creating the PIN Blocks for offline PIN verification and verifying static and dynamic data.
This section is used to further explain concepts and functionality that needs further clarification.

The PIN service is able to manage the EMV chip card regarding the card authentication and the RSA local PIN verification. Two steps are mandatory in order to reach these two functions: The loading of the keys which come from the Certification Authorities or from the card itself, and the EMV PIN block management.

The service provider is responsible for all key validation during the import process. The application is responsible for management of the key lifetime and expiry after the key is successfully imported.

8.4.1 Keys loading

The final goal of an application is to retrieve the keys located on card to perform the operations of authentication or local PIN check (RSA encrypted). These keys are provided by the card using EMV certificates and can be retrieved using a Public Key provided by a Certification Authority. The application should first load the keys issued by the Certification Authority. At transaction time the application will use these keys to load the keys that the application has retrieved from the chip card.

Certification Authority keys

These keys are provided in the following formats:

- Plain text
- Plain Text with EMV 2000 Verification Data (See [Ref. 4] under the reference section for this document)
- EPI CA (or self signed) format as specified in the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5] under the reference section for this document).
- PKCSV1_5 encrypted (as used by GIECB in France) (See [Ref. 15] under the reference section for this document)..

.

EPI CA format
The following table corresponds to table 4 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5]) and identifies the Europay Public Key (self-certified) and the associated data:

Field name	Length	Description	Format
ID of Certificate Subject	5	RID for Europay	Binary
Europay public key Index	1	Europay public key Index	Binary
Subject public key Algorithm Indicator	1	Algorithm to be used with the Europay public key Index, set to 0x01	Binary
Subject public key Length	1	Length of the Europay public key Modulus (equal to *Nca)*	Binary
Subject public key Exponent Length	1	Length of the Europay public key Exponent	Binary
Leftmost Digits of Subject public key	*Nca*-37	*Nca*-37 most significant bytes of the Europay public key Modulus	Binary
Subject public key Remainder	37	37 least significant bytes of the Europay public key Modulus	Binary
Subject public key Exponent	1	Exponent for Europay public key	Binary
Subject public key Certificate	*Nca*	Output of signature algorithm	Binary

Table 1

The following table corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 and identifies the Europay Public Key Hash code and associated data.:

Field name	Length	Description	Format
ID of Certificate Subject	5	RID for Europay	Binary
Europay public key Index	1	Europay public key Index	Binary
Subject public key Algorithm Indicator	1	Algorithm to be used with the Europay public key Index, set to 0x01	Binary
Certification Authority public key Check Sum	20	Hash-code for Europay public key	Binary

Table 2

Table 2 corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5]).

8.41.1 Chip card keys

These keys are provided as EMV certificates which come from the chip card in a multiple layer structure (issuer key first, then the ICC keys). Two kinds of algorithm are used with these certificates in order to retrieve the keys: One for the issuer key and the other for the ICC keys (ICC Public Key and ICC PIN encipherment key). The associated data with these algorithms – The PAN (Primary Account Number) and the SDA (Static Data to be Authenticated) - come also from the chip card.

8.4.2 PIN block management

The PIN block management is done through the command WFS_CMD_PIN_GET_PINBLOCK. A new format WFS_PIN_FORMEMV has been added to indicate to the PIN service that the PIN block must follow the requirements of the EMVco, Book2 – Security & Key management Version 4.0 document The parameter *lpsCustomerData* is used in this case to transfer to the PIN service the challenge number coming from the chip card. The final encryption must be done using a RSA Public Key. Please note that the application is responsible to send the PIN block to the chip card inside the right APDU.

8.4.3 SHA-1 Digest

The SHA-1 Digest is a hash algorithm used by EMV in validating ICC static and dynamic data item. The SHA-1 Digest is supported through the WFS_CMD_PIN_ DIGEST command. The application will pass the data to be hashed to the service provider. Once the encryptor completes the SHA-1 hash code, the Service Provider will return the 20-byte hash value back to the application.

8.5 French Cartes Bancaires

"*Groupement des Cartes Bancaires*" from France has specified a cryptographic architecture for ATM networks. See the document [Ref. 15] for details.

The XFS command WFS_CMD_PIN_ENC_IO with the protocol WFS_PIN_ENC_PROT_GIECB is used for

- ATM initialization
- Renewal of ATM master key
- Renewal of HOST master key
- Generation and loading of key transport key

Keys loaded or generated with WFS_CMD_PIN_ENC_IO get names like any other keys in a XFS PIN service. WFS_INF_PIN_KEY_DETAIL_[EX] shows the key with this name and the name may be used with WFS_CMD_PIN_IMPORT_KEY[_EX] to delete a key.

8.5.1 Data Structure for WFS_CMD_PIN_ENC_IO

Data will be transferred as tag-length-value (TLV) structure, encoded according to the distinguished encoding rules (DER) defined in [Ref. 16]

The following is a list of top level tags defined for the use with WFS_PIN_ENC_PROT_GIECB. All these tags have the APPLICATION class, therefore the Identifier Octets are (binary)

0 1 0 n n n n n	for the primitive types
0 1 1 n n n n n	for the constructed types

Tag Number	Primitive / Constructed	Identifier Octet	Contents
0	**P**	**0x40**	**Protocol Version** **The INTEGER value zero for this version of the protocol**
1	**P**	**0x41**	**Interchange Code** **An ASCII string holding one of the interchange codes defined in [Ref. 15], e.g. "HRN-H1"**
2	**C**	**0x62**	**Interchange Data** **The data items as defined by [Ref.15], see table below for details**
3	**P**	**0x43**	**Key Name** **An ASCII string holding the name for the key being loaded or generated.**

The Interchange Data (Tag 2) is constructed from data items where tag numbers of the sub-tags from 1 to 23 correspond to the data item numbers ("N^{o} donnée") as defined in section 3.1 of [Ref. 15]. Some of the data items consist of data elements, for these the constructed encoding will be used. For data items with no data elements the primitive encoding will be used.

All Tags have the CONTEXT class, therefore the Identifier Octets are (binary)

```
            1 0 0 n   n n n n          for the primitive types
            1 0 1 n   n n n n          for the constructed types
```

Tag (=Data Item No)	Primitive / Constructed	Identifier Octet	Data Item Label
1	**C**	**0xA1**	**IdKG**
2	**C**	**0xA2**	**KTK-encrypted**
3	**C**	**0xA3**	**KGp**
4	**C**	**0xA4**	**KDp**
5	**C**	**0xA5**	**SnSCD**
6	**P**	**0x86**	**Rand**
7	**P**	**0x87**	**HOST authentication**
8	**P**	**0x88**	**KDp signature**
9	**P**	**0x89**	**KGp signature**
10	**P**	**0x8A**	**KTK signature**
11	**P**	**0x8B**	**KT-encrypted**
12	**P**	**0x8C**	**Ksc-encrypted**
13	**P**	**0x8D**	**PIN cryptogram**
14	**P**	**0x8E**	**Seal**
15	**P**	**0x8F**	**Thumbprint of KDp**
16	**P**	**0x90**	**Thumbprint of KGp**
17	**C**	**0xB1**	**IdKD**
18	**C**	**0xB2**	**IdKTK**
19	**C**	**0xB3**	**IdKT**
20	**C**	**0xB4**	**IdKSC**
21	**P**	**0x95**	**Manufacturer**
22	**C**	**0xB6**	**SCD type**
23	**C**	**0xB7**	**Firmware version**

Inside the constructed data items, primitive encoding is used for the data elements, all tags having CONTEXT class with tag numbers corresponding to the data element numbers (“N^{o} d’élément de donnée”) as defined in section 3.1 of [Ref. 15].

Example :

The example shows the DER encoding of the input for a WFS_CMD_PIN_ENCIO command, for the interchange “GIN-H5”. All data except the 128 byte content of data item 7 is shown in hexadecimal (0x omitted for the sake of readability).

```
40 01 00                              (tag / length / value for Protocol Version 0)
41 06 47 49 4E 2D 47 34               (tag / length / value for Interchange Code “GIN-H5”)
62 81 B5                                     (tag / length for Interchange Data)
   A1 14                                     (tag / length for data item 1)
      81 01 00                                      (data element 1)
      82 0C 00 00 00 00 00 00 00 00 00 00 00 00     (data element 2)
      83 01 00                                      (data element 3)
   A5 10                                     (tag / length for data item 5)
      81 03 00 00 00                                (data element 1)
      82 09 00 00 00 00 00 00 00 00 00              (data element 2)
   86 08 00 00 00 00 00 00 00 00             (tag / length / value for data item 6)
```

```
          87 81 80 <128 bytes>                        (tag / length / value for data item 7)
      43 05 4D 59 4B 45 59                    (tag / length / value for Key Name “MYKEY”)
```

8.5.2 Command Sequence

The following list shows the sequence of actions an application has to take for the various *Cartes Bancaires* interchanges.

- **GIN (ATM initialization)**

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
Thumbprint supplied by host via external channel (GIN-H1)				
WFS_CMD_PIN_ENCIO	GIN-G2			21,22,23
Host Communication (GIN-G2 / GIN-H3)				
WFS_CMD_PIN_ENCIO	GIN-H3	Key Name for KG	3	16
WFS_CMD_PIN_ENCIO	GIN-G4			5,6,1
Host Communication (GIN-G4 / GIN-H5)				
WFS_CMD_PIN_ENCIO	GIN-H5	Key Name for KD	5,6,1,7	
WFS_CMD_PIN_ENCIO	GIN-G6			5,4,8
Host Communication (GIN-G6)				
WFS_CMD_PIN_ENCIO	GIN-G7			15
Send thumbprint to host via external channel (GIN-G7)				

- **GRN (Renewal of ATM Master Key)**

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
WFS_CMD_PIN_ENCIO	GRN-G1			5,6,1
Host Communication (GRN-G1 / GRN-H2)				
WFS_CMD_PIN_ENCIO	GRN-H2	Key Name for KD	5,6,1,7	
WFS_CMD_PIN_ENCIO	GRN-G3			5,4,8,17
Host Communication (GRN-G3)				
WFS_CMD_PIN_ENCIO	GRN-C or GRN-R		17	

The Interchange codes “GRN-C” to commit the transaction resp. “GRN-R” to roll back the transactions are an addition to those defined in [Ref. 15]

- **HRN (Renewal of HOST Master Key)**

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
Host Communication (HRN-H1)				
WFS_CMD_PIN_ENCIO	HRN-H1	Key Name for KG	3,9,1	

- **DKT (Generation and Loading of KTK)**

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
WFS_CMD_PIN_ENCIO	DKT-G1			5,6
Host Communication (DKT-G1 / DKT-H2)				
WFS_CMD_PIN_ENCIO	DKT-H2	Key Name for KTK	5,6,2,10,1,17	