CEN

WORKSHOP

AGREEMENT

CWA 14050-42

January 2005

ICS 35.200; 35.240.15; 35.240.40

English version

Extensions for Financial Services (XFS) interface specification – Release 3.03 – Part 42: PIN Keypad Device Class Interface - Migration from Version 3.02 to Version 3.03 - 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, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Slovakia, Slovenia, 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

Ref. No. CWA 14050-42:2005 D/E/F

Table of Contents

Foreword

This CWA is revision 3.03 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 CWA was formally approved by the XFS Workshop meeting on 2004-09-24. 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 3.03.

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 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 17: Printer Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 18: Identification Card Device Class Interface - Migration from Version 2.0 (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 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 20: PIN Keypad Device Class Interface - Migration from Version 2.0 (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 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 22: Text Terminal Unit Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 23: Sensors and Indicators Unit Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.01 (this CWA) - Programmer's Reference

Part 24: Camera Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (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 3.0 (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 3.0 (see CWA 14050-6:2000; superseded) to Version 3.02 (see CWA 14050-6:2003; superseded) - Programmer's Reference

Part 28: Cash In Module Device Class Interface - Migration from Version 3.0 (see CWA 14050-15:2000; superseded) to Version 3.02 (this CWA) - Programmer's Reference

Part 42: PIN Keypad Device Class Interface - Migration from Version 3.02 (see CWA 14050-6:2003; superseded) to Version 3.03 (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.

Parts 29 through 41 constitute an optional addendum to this CWA. They define the integration between the SNMP standard and the set of status and statistical information exported by the service providers.

Part 29: XFS MIB Architecture and SNMP Extensions – Programmer's Reference

Part 30: XFS MIB Device Specific Definitions - Printer Device Class

Part 31: XFS MIB Device Specific Definitions - Identification Card Device Class

Part 32: XFS MIB Device Specific Definitions - Cash Dispenser Device Class

Part 33: XFS MIB Device Specific Definitions - PIN Keypad Device Class

Part 34: XFS MIB Device Specific Definitions - Check Reader/Scanner Device Class

Part 35: XFS MIB Device Specific Definitions - Depository Device Class

Part 36: XFS MIB Device Specific Definitions - Text Terminal Unit Device Class

Part 37: XFS MIB Device Specific Definitions - Sensors and Indicators Unit Device Class

Part 38: XFS MIB Device Specific Definitions - Camera Device Class

Part 39: XFS MIB Device Specific Definitions - Alarm Device Class

Part 40: XFS MIB Device Specific Definitions - Card Embossing Unit Class

Part 41: XFS MIB Device Specific Definitions - Cash In Module Device Class

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.

1. General

The PIN has been enhanced with the following functionality:

- The capability to load a symmetric DES key using a secure manual multi-part encryption key entry process.
- The capability to generate a Key Check Value (KCV) for a symmetric key.
- The capability to authenticate the request to delete a public key loaded through Signature based Remote Key Loading scheme.
- Support for the ZKA PROTGENAS protocol.

2. Backwards Compatability

2.1 Secure Manual Key Entry

Secure Manual Key Entry was added through the addition of two new commands and with the definition of additional flag values for four existing commands.

The new commands are WFS_INF_PIN_SECUREKEY_DETAIL and WFS_CMD_PIN_SECUREKEY_ENTRY.

The modified commands are WFS_INF_PIN_KEY_DETAIL, WFS_INF_PIN_KEY_DETAIL, WFS_CMD_PIN_IMPORT_KEY and WFS_CMD_PIN_IMPORT_KEY_EX.

2.2 Public Key Deletion Authentication on RKL Signature Scheme

The capability to authenticate the deletion of a public key loaded through the Signature based Remote Key Loading scheme was added without changing the interface in any way. The descriptions of the WFS_CMD_PIN_INITIALIZATION and WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY were modified.

2.3 Symmetric Key Key Check Value Generation

The capability to generate a Key Check Value for a symmetric key was added through the new command WFS_CMD_PIN_GENERATE_KCV.

2.4 ZKA PROTGENAS Protocol

The ZKA protocol PROTGENAS was added to the existing WFS_CMD_PIN_SECURE_MSG_SEND and WFS_CMD_PIN_SECURE_MSG_RECEIVE commands. The functionality was added through the definition of a new protocol literal WFS_PIN_PROTGENAS and the addition of existing PIN error codes to these commands.

3. New Chapters

3.1 German ZKA GeldKarte

3.1.1 Protocol WFS_PIN_ PROTGENAS

This protocol provides the capability to create a PAC (encrypted Pin-Block) and to create and verify a MAC for a proprietary message. As the service provider doesn't know the message format, it cannot complete the message by adding security relevant fields like random values, PAC and MAC, like it does for the protocol WFS_PIN_PROTISOAS. Only the application is able to place these fields into the proper locations. Using this protocol, an application can generate the PAC and the random values in separate steps, adds them to the proprietary send-message, and finally lets the service provider generate the MAC. The generated MAC can then be added to the send-message as well.
For a received message, the application extracts the MAC and the associated random value and passes them along with the entire message data to the service provider for MAC verification.
PAC generation supports Pin-Block ISO-Format 0 and 1.

Command description:
The first byte of field `lpbMsg` of `WFSPINSECMSG` contains a subcommand, which is used to qualify the type of operation. The remaining bytes of the command data are depending on the value of the subcommand.

The following sub-commands are defined:

- GeneratePAC (Code 0x01)
  Returns the encrypted Pin-Block together with generation and version values of the Master Key and the PAC random value

- GetMACRandom (Code 0x02)
  Returns the generation and version values of the Master Key and the MAC random value

- GenerateMAC (Code 0x03)
  Returns the generated MAC for the message data passed in. Note, that the MAC is generated for exactly the data that is presented (contents and sequence). Data, that should not go into MAC calculation must not be passed in.

- VerifyMAC (Code 0x04)
  Generates a MAC for the data passed in and compares it with the provided MAC value. MAC random value, key generation and key version must be passed in separately.

Command/Message sequence:

Command WFS_CMD_PIN_...	**lpbMsg in lpbSecMsgIn**	**lpbMsg in lpbSecMsgOut**	**Service Provider´s actions**
SECURE_MSG_SEND	Byte 0: 0x01 (Generate PAC) Byte 1: format (0 or 1) Byte 2-9: ANF (Primary Account Number, if length is less than 12 digits, value must be left padded with binary 0, only applicable for format 0)	Byte 0: key generation Byte 1: key version Byte 2-17: PAC random Byte 18-25: PAC value (all values are binary values)	Generates a session key for PAC generation and finally the PAC itself. Determine generation and version values of Master-Key and return them along with the random value.
SECURE_MSG_SEND	Byte 0: 0x02 (Get MAC Random)	Byte 0: key generation Byte 1: key version Byte 2-17: MAC random (all values are binary values)	Generates a session key for MAC generation (see next step below) Determine generation and version values of Master-

			Key and return them along with the random value
SECURE_MSG_SEND	Byte 0: 0x03 (Generate MAC) Byte 1-n: Message to be mac'ed (all values are binary values)	Byte 0-7: generated MAC (binary value)	Generates MAC over bytes 1-n of the inbound message using the session key created in the previous step.
SECURE_MSG_RECEIVE	Byte 0: 0x04 (Verify MAC) Byte 1: key generation Byte 2: key version Byte 3-18: MAC random Byte 19-26: MAC Byte 27-n: Message to be verified (all values are binary values) Note: If no message has been received, this function must be called by omitting Bytes 1-n	N/a	Generates a session key using the Master key identified by key generation and version by using the random value passed in. Generates a MAC for the message data passed in and compare the resulting MAC with the MAC passed in.

Returns:

The error code WFS_ERR_PIN_FORMATINVALID is returned when

- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_SEND with protocol WFS_PIN_PROTGENAS is not 01, 02 or 03.
- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_RECEIVE with protocol WFS_PIN_PROTGENAS is not 04.
- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_SEND with protocol WFS_PIN_PROTGENAS is 01 and Byte 1 is not 00 and not 01 (Pin-Block format is not ISO-0 and ISO-1)
- the individual command data length for a subcommand is less than specified

The error code WFS_ERR_PIN_HSMSTATEINVALID is returned when

- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_SEND with protocol WFS_PIN_PROTGENAS is 03 (Generate MAC) without a preceding GetMACRandom (WFS_CMD_PIN_SECURE_MSG_SEND with subcommand 02).

The error code WFS_ERR_PIN_MACINVALID is returned when

- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_RECEIVE with protocol WFS_PIN_PROTGENAS is 04 (Verify MAC) and the MACs didn't match.

The error code WFS_ERR_PIN_KEYNOTFOUND is returned when

- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_SEND with protocol WFS_PIN_PROTGENAS is 01 (Generate PAC) and the service provider doesn't find a master key.
- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_SEND with protocol WFS_PIN_PROTGENAS is 02 (Get MAC Random) and the service provider doesn't find a master key.
- the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_RECEIVE with protocol WFS_PIN_PROTGENAS is 04 (Verify MAC) and the service provider doesn't find a key for the provided key generation and key version values.

The error code WFS_ERR_PIN_NOPIN is returned when
the subcommand in Byte 0 of `lpbMsg` for Execute Command WFS_CMD_PIN_SECURE_MSG_SEND with protocol WFS_PIN_PROTGENAS is 01 (Generate PAC) and no PIN or insufficient PIN-digits have been entered.

3.2 Secure Key Entry

This section provides additional information to describe how encryption keys are entered securely through the pinpad keyboard and also provides examples of possible keyboard layouts.

3.2.1 Keyboard Layout

The following sections describe what is returned within the WFS_INF_PIN_SECUREKEY_DETAIL output parameters to describe the physical keyboard layout. These descriptions are purely examples to help understand the usage of the parameters they do not indicate a specific layout per Key Entry Mode.

In the following section all references to parameters relate to the output fields of the WFS_INF_PIN_SECUREKEY_DETAIL command.

When f*wKeyEntryMode* represents a regular shaped pin pad (WFS_PIN_SECUREKEY_REG_UNIQUE or WFS_PIN_SECUREKEY_REG_SHIFT) then *lppHexKeys* must contain one entry for each physical key on the pinpad (i.e. the product of *wRows* by *wColumns*). On a regular shaped pinpad the application can choose to ignore the position and size data and just use the *wRows* and *wColumns* parameters to define the layout. However, a service provider must return the position and size data for each key.

3.2.1.1 *fwKeyEntryMode* == WFS_PIN_SECUREKEY_REG_UNIQUE

When *fwKeyEntryMode* is WFS_PIN_SECUREKEY_REG_UNIQUE then the values in the array report which physical keys are associated with the function keys 0-9, A-F and any other function keys that can be enabled as defined in the lp*FuncKeyDetail* parameter. Any positions on the pinpad that are not used must be defined as a WFS_PIN_FK_UNUSED in the *ulFK* and *ulShiftFK* field of the *lppHexKeys* structure.

1	2	3	Clear (A)
4	5	6	Cancel (B)
7	8	9	Enter (C)
(D)	0	(E)	(F)

In the above example, where all keys are the same size and the hex digits are located as shown the *lppHexKeys* will contain the entries in the array as defined in the following table.

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
0	0	0	250	250	FK_1	FK_UNUSED
1	250	0	250	250	FK_2	FK_UNUSED
2	500	0	250	250	FK_3	FK_UNUSED
3	750	0	250	250	FK_A	FK_UNUSED
4	0	250	250	250	FK_4	FK_UNUSED
5	250	250	250	250	FK_5	FK_UNUSED
6	500	250	250	250	FK_6	FK_UNUSED
7	750	250	250	250	FK_B	FK_UNUSED
8	0	500	250	250	FK_7	FK_UNUSED
9	250	500	250	250	FK_8	FK_UNUSED
10	500	500	250	250	FK_9	FK_UNUSED
11	750	500	250	250	FK_C	FK_UNUSED
12	0	750	250	250	FK_D	FK_UNUSED
13	250	750	250	250	FK_0	FK_UNUSED
14	500	750	250	250	FK_E	FK_UNUSED
15	750	750	250	250	FK_F	FK_UNUSED

3.2.1.2 *fwKeyEntryMode* == WFS_PIN_SECUREKEY_REG_SHIFT

When *fwKeyEntryMode* is WFS_PIN_SECUREKEY_REG_SHIFT then the values in the array report which physical keys are associated with the function keys 0-9, A-F and the shift key as defined in the lp*FuncKeyDetail* parameter. Other function keys as defined by the l*pFuncKeyDetail* parameter that can be enabled must also be reported. Any positions on the pinpad that are not used must be defined as a WFS_PIN_FK_UNUSED in the *ulFK* and *ulShiftFK* field of the *lppHexKeys* structure. Digits 0 to 9 are accessed through the numeric keys as usual. Digits A - F are accessed by using the shift key in combination with another function key, e.g. shift-0(zero) is hex digit A.

1 (B)	2 (C)	3 (D)	Clear
4 (E)	5 (F)	6	Cancel
7	8	9	Enter
SHIFT	0 (A)		

In the above example, where all keys are the same size and the hex digits 'A' to 'F' are accessed through shift '0' to '5', then the *lppHexKeys* will contain the entries in the array as defined in the following table.

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
0	0	0	250	250	FK_1	FK_B
1	250	0	250	250	FK_2	FK_C
2	500	0	250	250	FK_3	FK_D
3	750	0	250	250	FK_CLEAR	FK_UNUSED
4	0	250	250	250	FK_4	FK_E
5	250	250	250	250	FK_5	FK_F
6	500	250	250	250	FK_6	FK_UNUSED
7	750	250	250	250	FK_CANCEL	FK_UNUSED
8	0	500	250	250	FK_7	FK_UNUSED
9	250	500	250	250	FK_8	FK_UNUSED
10	500	500	250	250	FK_9	FK_UNUSED
11	750	500	250	250	FK_ENTER	FK_UNUSED
12	0	750	250	250	FK_SHIFT	FK_UNUSED
13	250	750	250	250	FK_0	FK_A
14	500	750	250	250	FK_UNUSED	FK_UNUSED
15	750	750	250	250	FK_UNUSED	FK_UNUSED

3.2.1.3 *fwKeyEntryMode* == WFS_PIN_SECUREKEY_IRREG_SHIFT

When *fwKeyEntryMode* represents an irregular shaped pin pad the *wRows* and *wColumns* parameters define the ratio of the width to height, i.e. square if the parameters are the same or rectangular if *wColumns* is larger than *wRows*, etc. A service provider must return the position and size data for each key reported.

When *fwKeyEntryMode* is WFS_PIN_SECUREKEY_IRREG_SHIFT then the values in the array must be the function keys codes for 0-9 and the shift key as defined in the l*pFuncKeyDetail* parameter. Other function keys as defined by the l*pFuncKeyDetail* parameter that can be enabled must also be reported. Any positions on the pinpad that are not used must be defined as a WFS_PIN_FK_UNUSED in the *ulFK* and *ulShiftFK* field of the *lppHexKeys* structure. Digits 0 to 9 are accessed through the numeric keys as usual. Digits A - F are accessed by using the shift key in combination with another function key,e.g. shift-0(zero) is hex digit A.

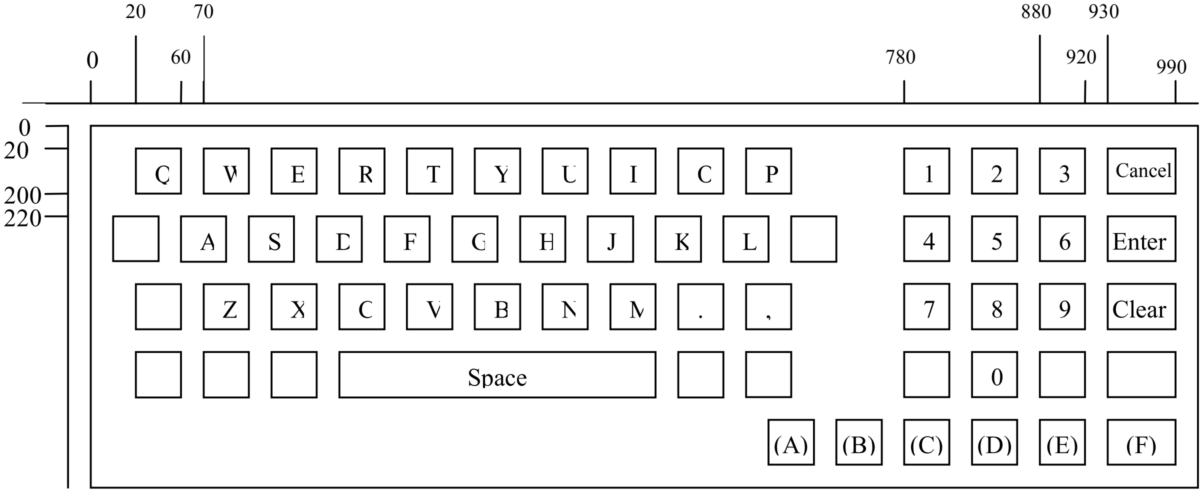
1 (B)	2 (C)	3 (D)	Clear
4 (E)	5 (F)	6	Cancel
7	8	9	Enter
	0 (A)		
SHIFT			

In the above example, where the hex digits 'A' to 'F' are accessed through shift '0' to '5' , *wColumns* will be 4, *wRows* will be 5 and the *lppHexKeys* will contain the entries in the array as defined in the following table.

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
0	0	0	250	200	FK_1	FK_B
1	250	0	250	200	FK_2	FK_C
2	500	0	250	200	FK_3	FK_D
3	750	0	250	200	FK_CLEAR	FK_UNUSED
4	0	200	250	200	FK_4	FK_E
5	250	200	250	200	FK_5	FK_F
6	500	200	250	200	FK_6	FK_UNUSED
7	750	200	250	200	FK_CANCEL	FK_UNUSED
8	0	400	250	200	FK_7	FK_UNUSED
9	250	400	250	200	FK_8	FK_UNUSED
10	500	400	250	200	FK_9	FK_UNUSED
11	750	400	250	200	FK_ENTER	FK_UNUSED
12	0	600	250	200	FK_UNUSED	FK_UNUSED
13	250	600	250	200	FK_0	FK_A
14	500	600	250	200	FK_UNUSED	FK_UNUSED
15	750	600	250	200	FK_UNUSED	FK_UNUSED
16	0	800	1000	200	FK_SHIFT	FK_UNUSED

3.2.1.4 *fwKeyEntryMode* == WFS_PIN_SECUREKEY_IRREG_UNIQUE

When *fwKeyEntryMode* is WFS_PIN_SECUREKEY_REG_UNIQUE then the values in the array report which physical keys are associated with the function keys 0-9, A-F and any other function keys that can be enabled as defined in the lp*FuncKeyDetail* parameter. The *wRows* and *wColumns* parameters define the ratio of the width to height, ie square if the parameters are the same or rectangular if if *wColumns* is larger than *wRows*, etc. A service provider must return the position and size data for each key.

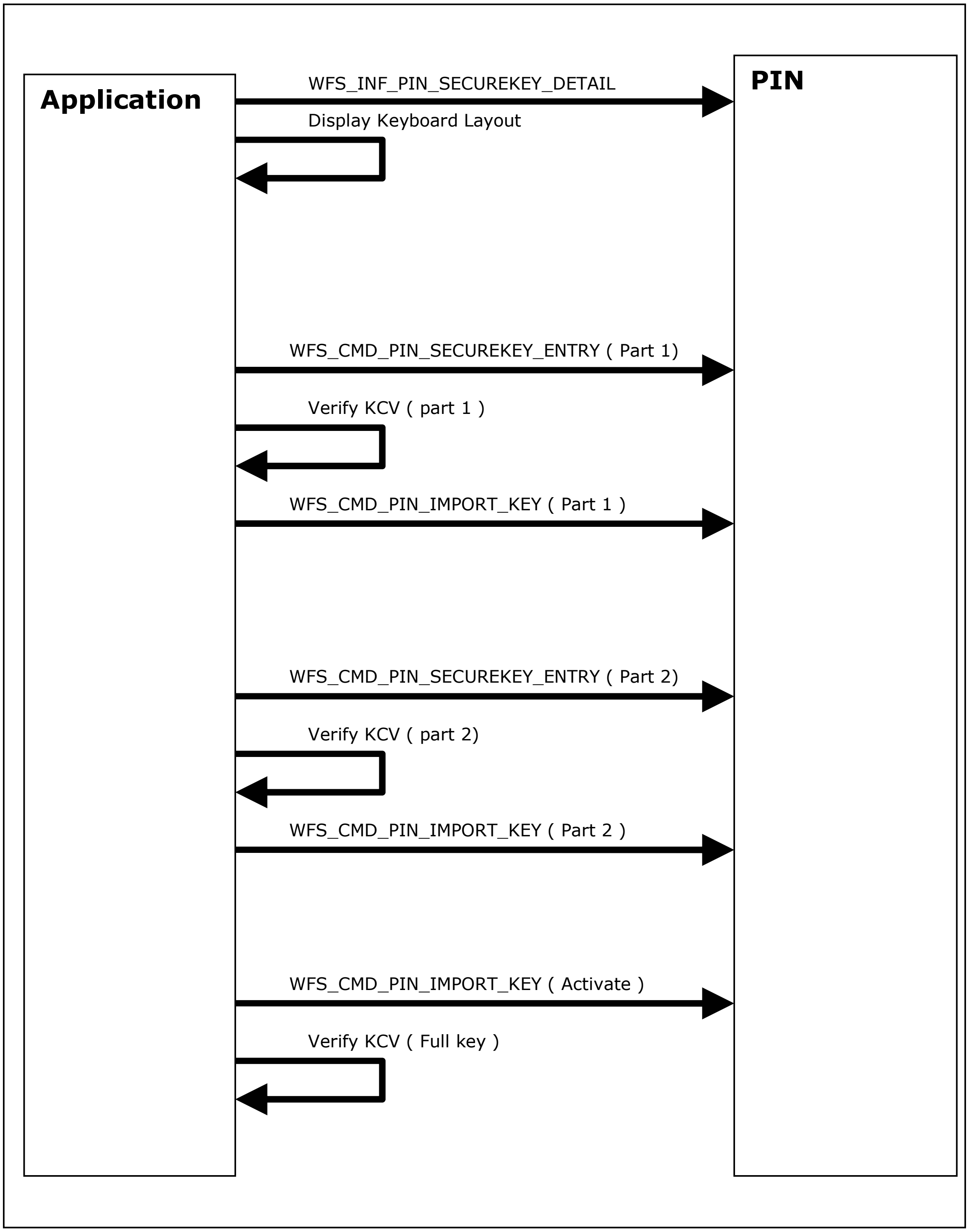


In the above example, where an alphanumeric keyboard supports secure key entry and the hex digits are located as shown, the *lppHexKeys* will contain the entries in the array as defined in the following table. All the hex digits and function keys that can be enabled must be included in the array; in addition any keys that would help an application display an image of the keyboard can be included. In this example only the pinpad digits(the keys on the right) and the unique hex digits are reported. Note that the position data in this example may not be 100% accurate as the diagram is not to scale.

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
0	780	18	40	180	FK_1	FK_UNUSED
1	830	18	40	180	FK_2	FK_UNUSED
2	880	18	40	180	FK_3	FK_UNUSED
3	930	18	60	180	FK_CANCEL	FK_UNUSED
4	780	216	40	180	FK_4	FK_UNUSED
5	830	216	40	180	FK_5	FK_UNUSED
6	880	216	40	180	FK_6	FK_UNUSED
7	930	216	60	180	FK_ENTER	FK_UNUSED
8	780	414	40	180	FK_7	FK_UNUSED
9	830	414	40	180	FK_8	FK_UNUSED
10	880	414	40	180	FK_9	FK_UNUSED
11	930	414	60	180	FK_CLEAR	FK_UNUSED
12	780	612	40	180	FK_UNUSED	FK_UNUSED
13	830	612	40	180	FK_0	FK_UNUSED
14	880	612	40	180	FK_UNUSED	FK_UNUSED
15	930	612	60	180	FK_UNUSED	FK_UNUSED
16	680	810	40	180	FK_A	FK_UNUSED
17	730	810	40	180	FK_B	FK_UNUSED
18	780	810	40	180	FK_C	FK_UNUSED
19	830	810	40	180	FK_D	FK_UNUSED
20	880	810	40	180	FK_E	FK_UNUSED
21	930	810	60	180	FK_F	FK_UNUSED

3.2.2 Command Usage

This section provides an example of the sequence of commands required to enter an encryption key securely. In the following sequence, the application retrieves the keyboard secure key entry mode and associated keyboard layout and displays an image of the keyboard for the user. It then gets the first key part, verifies the KCV for the key part and stores it. The sequence is repeated for the second key part and then finally the key part is activated.


Application
PIN
WFS_INF_PIN_SECUREKEY_DETAIL
Display Keyboard Layout
WFS_CMD_PIN_SECUREKEY_ENTRY (Part 1)
Verify KCV (part 1)
WFS_CMD_PIN_IMPORT_KEY (Part 1)
WFS_CMD_PIN_SECUREKEY_ENTRY (Part 2)
Verify KCV (part 2)
WFS_CMD_PIN_IMPORT_KEY (Part 2)
WFS_CMD_PIN_IMPORT_KEY (Activate)
Verify KCV (Full key)

4. New Info Commands

4.1 WFS_INF_PIN_SECUREKEY_DETAIL

Description This command reports the secure key entry method used by the device. This allows an application to enable the relevant keys and inform the user how to enter the hex digits 'A' to 'F', e.g by displaying an image indicating which key pad locations correspond to the 16 hex digits and/or shift key. It reports the following information:

- The secure key entry mode (uses a shift key to access the hex digit 'A' to 'F' or each hex digit has a specific key assigned to it).
- The function keys and FDKs available during secure key entry
- The FDKs that are configured as function keys (Enter, Cancel, Clear and Backspace)
- The physical keyboard layout

The keys that are active during the secure key entry command are vendor specific but must be sufficient to enter a secure encryption key. On some systems a unique key is assigned to each encryption key digit. On some systems encryption key digits are entered by pressing a shift key and then a numeric digit, e.g. to enter 'A' the shift key (WFS_PIN_FK_SHIFT) is pressed followed by the zero key (WFS_PIN_FK_0). On these systems WFS_PIN_FK_SHIFT is not returned to the application in a WFS_EXEE_PIN_KEY event. The exact behavior of the shift key is vendor dependent, some devices will require the shift to be used before every key and some may require the shift key to enter and exit shift mode.

There are many different styles of pinpads in operation. Most have a regular shape with all keys having the same size and are laid out in a regular matrix. However, some devices have a layout with keys of different sizes and different numbers of keys on some rows and columns. This command returns information that allows an application to provide user instructions and an image of the keyboard layout to assist with key entry.

Input Param None.

Output Param `LPWFSPINSECUREKEYDETAIL lpSecureKeyDetail;`

```
typedef struct _wfs_pin_secure_key_detail
        {
        WORD                        fwKeyEntryMode;
        LPWFSPINFUNCKEYDETAIL       lpFuncKeyDetail;
         ULONG                      ulClearFDK;
        ULONG                       ulCancelFDK;
        ULONG                       ulBackspaceFDK;
        ULONG                       ulEnterFDK;
        WORD                        wColumns;
        WORD                        wRows;
        LPWFSPINHEXKEYS             * lppHexKeys;
        } WFSPINSECUREKEYDETAIL, * LPWFSPINSECUREKEYDETAIL;
```

fwKeyEntryMode
Specifies the method to be used to enter the encryption key digits (including 'A' to 'F') during secure key entry. The value can be one of the following.

Value	Meaning
WFS_PIN_SECUREKEY_NOTSUPP	Secure key entry is not supported, all other parameters are undefined.
WFS_PIN_SECUREKEY_REG_SHIFT	Secure key hex digits 'A' – 'F' are accessed through the shift key. Digits 'A' – 'F' are accessed through the shift key followed by one of the other function keys. The keys associated with 'A' to 'F' are defined within

the *lppHexKeys* parameter. The keyboard has a regular shaped key layout where all rows have the same number of keys and all columns have the same number of keys, e.g. 5x4. The *lppHexKeys* parameter must contain one entry for each key on the pinpad (i.e. the product of *wRows* by *wColumns*).

WFS_PIN_SECUREKEY_IRREG_SHIFT — Secure key hex digits 'A' – 'F' are accessed through the shift key. Digits 'A' – 'F' are accessed through the shift key followed by one of the other function keys. The keys associated with 'A' to 'F' are defined within the *lppHexKeys* parameter. The keyboard has an irregular shaped key layout, e.g there are more or less keys on one row or column than on the others. The *lppHexKeys* parameter must contain one entry for each key on the pinpad.

WFS_PIN_SECUREKEY_REG_UNIQUE — Secure key hex digits are accessed through specific keys assigned to each hex digit. The keyboard has a regular shaped key layout where all rows have the same number of keys and all columns have the same number of keys, e.g. 5x4. The *lppHexKeys* parameter must contain one entry for each key on the pinpad (i.e. the product of *wRows* by *wColumns*).

WFS_PIN_SECUREKEY_IRREG_UNIQUE — Secure key hex digits are accessed through specific keys assigned to each hex digit. The keyboard has an irregular shaped key layout, e.g. there are more or less keys on one row or column than on the others. The *lppHexKeys* must contain one entry for each key on the pinpad.

lpFuncKeyDetail

Contains information about the Function Keys and FDKs supported by the device while in secure key entry mode. This structure is the same as the output structure of the WFS_INF_PIN_FUNCKEY_DETAIL command with information always returned for every FDK valid during secure key entry. It describes the function keys that represent the hex digits and shift key, but also reports any other keys that can be enabled while in secure key entry mode.

The double zero, triple zero and decimal point function keys are not valid during secure key entry so are never reported.

On a pinpad where the physical Enter, Clear, Cancel and Backspace keys are used for hex digits (e.g WFS_PIN_SECUREKEY_REG_UNIQUE mode), the logical function keys WFS_PIN_FK_ENTER, WFS_PIN_FK_CLEAR, WFS_PIN_FK_CANCEL and WFS_PIN_FK_BACKSPACE will not be reported by this command (unless there is another physical key offering this functionality).

In addition to the existing definition for WFS_INF_PIN_FUNCKEY_DETAIL, the following definitions replace function keys WFS_PIN_FK_RES1 to WFS_PIN_FK_RES7:

WFS_PIN_FK_A	(hex digit A)
WFS_PIN_FK_B	(hex digit B)
WFS_PIN_FK_C	(hex digit C)
WFS_PIN_FK_D	(hex digit D)
WFS_PIN_FK_E	(hex digit E)
WFS_PIN_FK_F	(hex digit F)

WFS_PIN_FK_SHIFT (Shift key used during hex entry)

ulClearFDK
The FDK code mask reporting any FDKs associated with Clear. If this field is 0 then Clear through an FDK is not supported, otherwise the bit mask reports which FDKs are associated with Clear.

ulCancelFDK
The FDK code mask reporting any FDKs associated with Cancel. If this field is 0 then Cancel through an FDK is not supported, otherwise the bit mask reports which FDKs are associated with Cancel.

ulBackspaceFDK
The FDK code mask reporting any FDKs associated with Backspace. If this field is 0 then Backspace through an FDK is not supported, otherwise the bit mask reports which FDKs are associated with Backspace.

ulEnterFDK
The FDK code mask reporting any FDKs associated with Enter. If this field is 0 then Enter through an FDK is not supported, otherwise the bit mask reports which FDKs are associated with Enter.

wColumns
Specifies the maximum number of columns on the pinpad (the columns are defined by the x co-ordinate values within the *lppHexKeys* structure below). When the *fwKeyEntryMode* parameter represents an irregular shaped keyboard the *wRows* and *wColumns* parameters define the ratio of the width to height, i.e.square if the parameters are the same or rectangular if *wColumns* is larger than *wRows*, etc.

wRows
Specifies the maximum number of rows on the pinpad(the rows are defined by the y co-ordinate values within the *lppHexKeys* structure below). When the *fwKeyEntryMode* parameter represents an irregular shaped keyboard the *wRows* and *wColumns* parameters define the ratio of the width to height, ie square if the parameters are the same or rectangular if *wColumns* is larger than *wRows*, etc.

lppHexKeys
A NULL terminated array of pointers to key layout structures describing the physical keys on the pinpad, it does not include FDKs.

```
typedef struct _wfs_pin_hex_keys
     {
     USHORT            usXPos;
     USHORT            usYPos;
     USHORT            usXSize;
     USHORT            usYSize;
     ULONG             ulFK;
     ULONG             ulShiftFK;
     } WFSPINHEXKEYS, * LPWFSPINHEXKEYS;
```

This array defines the keys associated with the hex digits. Each structure entry describes the position, size and function key associated with a key. This data must be returned by the service provider. This array represents the pinpad keys ordered left to right and top to bottom.

usXPos
Specifies the position of the top left corner of the FK relative to the left hand side of the keyboard expressed as a value between 0 and 999, where 0 is the left edge and 999 is the right edge.

usYPos
Specifies the position of the top left corner of the FK relative to the top of the keyboard expressed as a value between 0 and 999, where 0 is the top edge and 999 is the bottom edge.

usXSize
Specifies the FK width expressed as a value between 1 and 1000, where 1 is the smallest possible size and 1000 is the full width of the keyboard.

usYSize
Specifies the FK height as expressed as a value between 1and 1000, where 1 is the smallest possible size and 1000 is the full height of the keyboard,

ulFK
Specifies the FK code associated with the physical key in non shifted mode, WFS_PIN_FK_UNUSED if the key is not used.

ulShiftFK
Specifies the FK code associated with the physical key in shifted mode, WFS_PIN_FK_UNUSED if the key is not used in shifted mode. This field will always be WFS_PIN_FK_UNUSED when the *fwKeyEntryMode* parameter indicates that keyboard does not use a shift mode.

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

Comments Examples keyboard layouts are provided in section 3.2 to explain the use of the *lppHexKeys* parameter. In addition section 3.2 also provides an example of a command flow required to enter encryption keys securely.

5. Changes to Existing Info Commands

5.1 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. This value can be returned in combination with any of the other key usage flags (other than WFS_PIN_USESECURECONSTRUCT).
WFS_PIN_USESECURECONSTRUCT	key is under construction through the import of multiple parts from a secure encryption key entry buffer. This value can be returned in combination with any of the other key usage flags (other than WFS_PIN_USECONSTRUCT).

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.

5.2 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 (other than WFS_PIN_USESECURECONSTRUCT).
WFS_PIN_USESECURECONSTRUCT	key is under construction through the import of multiple parts from a secure encryption key entry buffer. This value can be returned in combination with any of the other key usage flags (other than WFS_PIN_USECONSTRUCT).

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.

6. New Execute Commands

6.1 Normal PIN Commands

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

6.1.1 WFS_CMD_PIN_SECUREKEY_ENTRY

Description This command allows a full length symmetric encryption key part to be entered directly into the pinpad without being exposed outside of the pinpad. From the point this function is invoked, encryption key digits (WFS_PIN_FK_0 to WFS_PIN_FK_9 and WFS_PIN_FK_A to WFS_PIN_FK_F) are ***not*** passed to the application. For each encryption key digit, or any other active key entered(except for shift), an execute notification event WFS_EXEE_PIN_KEY is sent in order to allow an application to perform the appropriate display action (i.e. when the pinpad has no integrated display). When an encryption key digit is entered the application is not informed of the value entered, instead zero is returned.

The keys that can be enabled by this command are defined by the l*pFuncKeyDetail* parameter of the WFS_INF_PIN_SECUREKEY_DETAIL command. Function keys which are not associated with an encryption key digit may be enabled but will not contribute to the secure entry buffer (unless they are Cancel, Clear or Backspace) and will not count towards the length of the key entry. The Cancel and Clear keys will cause the encryption key buffer to be cleared. The Backspace key will cause the last encryption key digit in the encryption key buffer to be removed**.**

If *bAutoEnd* is TRUE the command will automatically complete when the required number of encryption key digits have been added to the buffer.
If *bAutoEnd* is FALSE then the command will not automatically complete and Enter, Cancel or any terminating key must be pressed. When *usKeyLen* hex encryption key digits have been entered then all encryption key digits keys are disabled. If the Clear or Backspace key is pressed to reduce the number of entered encryption key digits below *usKeyLen* , the same keys will be re-enabled.

Terminating keys have to be active keys to operate.

If an FDK is associated with Enter, Cancel, Clear or Backspace then the FDK must be activated to operate. The Enter and Cancel FDKs must also be marked as a terminator if they are to terminate entry. These FDKs are reported as normal FDKs within the WFS_EXEE_PIN_KEY event, applications must be aware of those FDKs associated with Cancel, Clear, Backspace and Enter and handle any user interaction as required. For example, if the WFS_PIN_FK_FDK01 is associated with Clear, then the application must include the WFS_PIN_FK_FDK01 FDK code in the *ulActiveFDK* parameter (if the clear functionality is required). In addition when this FDK is pressed the WFS_EXEE_PIN_KEY event will contain the WFS_PIN_FK_FDK01 mask value in the *ulDigit* field. The application must update the user interface to reflect the effect of the clear on the encryption key digits entered so far.

On some devices that are configured as either WFS_PIN_SECUREKEY_REG_UNIQUE or WFS_PIN_SECUREKEY_IRREG_UNIQUE all the function keys on the pinpad will be associated with hex digits and there may be no FDKs available either. On these devices there may be no way to correct mistakes or cancel the key encryption entry before all the encryption key digits are entered, so the application must set the *bAutoEnd* flag to TRUE and wait for the command to auto-complete. Applications should check the KCV to avoid storing an incorrect key component.

Encryption key parts entered with this command are stored through either the WFS_CMD_PIN_IMPORT_KEY or WFS_CMD_PIN_IMPORT_KEY_EX. Each key part can only be stored once after which the secure key buffer will be cleared automatically**.**

Input Param

```
LPWFSPINSECUREKEYENTRY     lpSecureKeyEntry;

typedef struct _wfs_pin_secure_key_entry
    {
    USHORT           usKeyLen;
    BOOL             bAutoEnd;
    ULONG            ulActiveFDKs;
    ULONG            ulActiveKeys;
    ULONG            ulTerminateFDKs;
    ULONG            ulTerminateKeys;
    WORD             wVerificationType;
    } WFSPINSECUREKEYENTRY, * LPWFSPINSECUREKEYENTRY;
```

usKeyLen
Specifies the number of digits which must be entered for the encryption key, 16 for a single length key and 32 for a double length key. The only valid values are 16 and 32.

bAutoEnd
If *bAutoEnd* is set to true, the service provider terminates the command when the maximum number of encryption key digits are entered. Otherwise, the input is terminated by the user using Enter, Cancel or any terminating key. When *usKeyLen* is reached, the service provider will disable all keys associated with an encryption key digit.

ulActiveFDKs
Specifies those FDKs which are active during the execution of the command. This parameter should include those FDKs mapped to edit functions.

ulActiveKeys

Specifies all Function Keys(not FDKs) which are active during the execution of the command. This should be the complete set or a subset of the keys returned in the l*pFuncKeyDetail* parameter of the WFS_INF_PIN_SECUREKEY_DETAIL command. This should include WFS_PIN_FK_0 to WFS_PIN_FK_9 and WFS_PIN_FK_A to WFS_PIN_FK_F for all modes of secure key entry, but should also include WFS_PIN_FK_SHIFT on shift based systems. The WFS_PIN_FK_00, WFS_PIN_FK_000 and WFS_PIN_FK_DECPOINT function keys must not be included in the list of active or terminate keys.

ulTerminateFDKs

Specifies those FDKs which must terminate the execution of the command. This should include the FDKs associated with Cancel and Enter.

ulTerminateKeys

Specifies those all Function Keys(not FDKs) which must terminate the execution of the command. This does not include the FDKs associated with Enter or Cancel.

wVerificationType

Specifies the type of verification to be done on the entered key. Possible values are as follows

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 a zero value with the key.

Output Param

```
LPWFSPINSECUREKEYENTRYOUT       lpSecureKeyEntryOut;

typedef struct _wfs_pin_secure_key_entry_out
   {
   USHORT           usDigits;
   WORD             wCompletion;
   LPWFSXDATA       lpxKCV;
   } WFSPINSECUREKEYENTRYOUT, * LPWFSPINSECUREKEYENTRYOUT;
```

usDigits

Specifies the number of key digits entered. Applications must ensure all required digits have been entered before trying to store the key.

wCompletion

Specifies the reason for completion of the entry. Possible values are described in WFS_CMD_PIN_GET_PIN.

lpxKCV

Contains the key check value data that can be used for verification of the entered key. This parameter is NULL if device does not have this capability, or the key entry was not fully entered, e.g. the entry was terminated by Enter before the required number of digits was entered.

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_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 *bAutoEnd* is FALSE.
WFS_ERR_PIN_INVALIDKEYLENGTH	The *usKeyLen* key length is not supported.

WFS_ERR_PIN_MODENOTSUPPORTED	The KCV mode 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_EXEE_PIN_KEY	A key has been pressed at the pinpad. Applications must be aware of the association between FDKs and the edit functions reported within the WFS_INF_PIN_SECUREKEY_DETAIL command.

Comments None

6.1.2 WFS_CMD_PIN_GENERATE_KCV

Description This command returns the Key Check Value (KCV) for the specified key.

Input Param

```
LPWFSPINGENERATEKCV       lpGenerateKCV;
```

```
typedef struct _wfs_pin_generate_KCV
   {
   LPSTR            lpsKey;
   WORD             wKeyCheckMode;
   } WFSPINGENERATEKCV, * LPWFSPINGENERATEKCV;
```

lpsKey
Specifies the name of key that should be used to generate the KCV.

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_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 a zero value with this key.

Output Param

```
LPWFSPINKCV  lpKCV;
```

```
typedef struct _wfs_pin_kcv
   {
      LPWFSXDATA   lpxKCV;
   } WFSPINKCV, * LPWFSPINKCV;
```

lpxKCV
Contains the key check value data that can be used for verification of the 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_KEYNOTFOUND	The specified key encryption key was not found.
WFS_ERR_PIN_KEYNOVALUE	The specified key exists but has no value loaded.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_MODENOTSUPPORTED	The KCV mode 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.

7. Changes to Existing Execute Commands

7.1 Normal PIN Commands

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

7.1.1 WFS_CMD_PIN_IMPORT_KEY

Description

The encryption key in the secure key buffer or 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 or WFS_PIN_USESECURECONSTRUCT value with the final usage flags within the *fwUse* field. If the WFS_PIN_USECONSTRUCT flag is used then the application must provide the key data through the *lpxValue* parameter, If WFS_PIN_USESECURECONSTRUCT is used then the encryption key part in the secure key buffer previously populated with the WFS_CMD_PIN_SECUREKEY_ENTRY command is used and *lpxValue* is ignored. Key parts loaded with the WFS_PIN_USESECURECONSTRUCT flag can only be stored once as the encryption key in the secure key buffer is no longer available after this command has been executed. The WFS_PIN_USECONSTRUCT and WFS_PIN_USESECURECONSTRUCT construction flags cannot be used in combination.

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 or WFS_PIN_USESECURECONSTRUCT.

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.
WFS_PIN_USESECURECONSTRUCT	key is under construction through the import of multiple parts. This value is used in combination with the actual usage flags for the key. *lpxValue* is ignored as the encryption key part is taken from the secure key buffer.

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

Output Param `LPWFSXDATA    lpxKVC;`

lpxKVC
Contains 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 or the encryption key in the secure key buffer is invalid (or has not been entered).
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 relevant construction 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 the construction flag 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. A WFS_ERR_PIN_ACCESSDENIED error will be returned if the key cannot be activated, e.g. if only one key part has been entered.

The optional KCV is only returned during the final activation step. Applications wishing to verify the KCV for each key part (and passing keys as a parameter to this command) 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. It is not possible to verify a key part being loaded from a secure key buffer with this command. This is achieved through the WFS_CMD_PIN_SECUREKEY_ENTRY command.

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 indicates it is under construction, it cannot be used for cryptographic functions.

7.1.2 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 or public keys imported under the RSA Signature based remote key loading scheme when public key deletion authentication is required. 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 (except for those public keys imported under the RSA Signature based remote key loading scheme when public key deletion authentication is required). 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.

7.1.3 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
WFS_PIN_PROTGENAS	Generic PAC/MAC for non-ISO8583 message formats

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_FORMATINVALID	The format of the message is invalid.
WFS_ERR_PIN_CONTENTINVALID	The contents of one of the security relevant fields are invalid.
WFS_ERR_PIN_KEYNOTFOUND	No key was found for PAC/MAC generation.
WFS_ERR_PIN_NOPIN	No PIN or insufficient PIN-digits have been entered.

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

Comments None.

7.1.4 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])
WFS_PIN_PROTGENAS	Generic PAC/MAC for non-ISO8583 message formats

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.
WFS_ERR_PIN_KEYNOTFOUND	No key was found for MAC verification.

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.

7.1.5 WFS_CMD_PIN_IMPORT_KEY_EX

Description The encryption key in the secure key buffer or 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 or WFS_PIN_USESECURECONSTRUCT value with the final usage flag within the *dwUse* field. If the WFS_PIN_USECONSTRUCT flag is used then the application must provide the key data through the *lpxValue* parameter, If WFS_PIN_USESECURECONSTRUCT is used then the encryption key part in the secure key buffer previously populated with the WFS_CMD_PIN_SECUREKEY_ENTRY command is used and *lpxValue* is ignored. Key parts loaded with the WFS_PIN_USESECURECONSTRUCT flag can only be stored once as the encryption key in the secure key buffer is no longer available after this command has been executed. The WFS_PIN_USECONSTRUCT and WFS_PIN_USESECURECONSTRUCT construction flags cannot be used in combination.

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 or WFS_PIN_USESECURECONSTRUCT.

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 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_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.
WFS_PIN_USESECURECONSTRUCT	key is under construction through the import of multiple parts. This value is used in combination with one of the other key usage flags. *lpxValue* is ignored as the encryption key part is taken from the secure key buffer.

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 a zero value with the key.

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 or the encryption key in the secure key buffer is invalid (or has not been entered).
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 relevant construction 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 the construction flag 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. A WFS_ERR_PIN_ACCESSDENIED error will be returned if the key cannot be activated, e.g. if only one key part has been entered.

When a construction 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 indicates it is under construction, it cannot be used for cryptographic functions.

7.2 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 **Error! Reference source not found.** provides additional explanation on how these commands are used. Section **Error! Reference source not found.** defines the fixed names for the Security Item and RSA keys that must be loaded during manufacture.

7.2.1 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.

When no signature is required to authenticate the deletion of a public key all parameters but *lpsKey* are ignored. In addition, 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.

When a signature is required to authenticate the deletion of the public key, all parameters in the command are used. *lpxValue* must contain the concatenation of the public key to be deleted and the Security Item which uniquely identifies the PIN device (see the WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM command). *lpxSignature* contains the signature generated from lpxValue using the private key component of the public key being deleted.

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 **Error! Reference source not found.**.

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 or deleted. The Signature is used to validate the key request 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 signature verification failed. The key has not been stored or deleted.

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.**

8. New Events

None.

9. Changes to Existing Events

9.1 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 or secure key entry mode (WFS_CMD_PIN_GET_PIN and WFS_CMD_PIN_SECUREKEY_ENTRY), the value of this field is zero for the function keys 0-9 and A-F. Otherwise, for each key pressed, the corresponding FK or FDK mask value is stored in this field.

Comments None.

10. C - Header File

```
/****************************************************************************
*                                                                           *
*xfspin.h XFS - Personal Identification Number Keypad (PIN) definitions    *
*                                                                           *
*            Version 3.03  (24/09/04)                                       *
*                                                                           *
****************************************************************************/

#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         (0x0303) /* Version 3.03 */
#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)
#define WFS_INF_PIN_SECUREKEY_DETAIL          (PIN_SERVICE_OFFSET + 8)


/* 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)
#define WFS_CMD_PIN_SECUREKEY_ENTRY       (PIN_SERVICE_OFFSET + 38)
#define WFS_CMD_PIN_GENERATE_KCV          (PIN_SERVICE_OFFSET + 39)


/* 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 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_USESECURECONSTRUCT        (0x0200)

/* Additional values of WFSPINKEYDETAILEX.dwUse */

#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)

/* additional values of WFSPINFUNCKEYDETAIL.ulFuncMask */
#define WFS_PIN_FK_UNUSED               (0x00000000)

#define WFS_PIN_FK_A                    WFS_PIN_FK_RES1
#define WFS_PIN_FK_B                    WFS_PIN_FK_RES2
#define WFS_PIN_FK_C                    WFS_PIN_FK_RES3
#define WFS_PIN_FK_D                    WFS_PIN_FK_RES4
#define WFS_PIN_FK_E                    WFS_PIN_FK_RES5
#define WFS_PIN_FK_F                    WFS_PIN_FK_RES6
#define WFS_PIN_FK_SHIFT                WFS_PIN_FK_RES7

/* 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)
#define WFS_PIN_PROTGENAS                 (8)


/* 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)

/* values of WFSPINSECUREKEYDETAIL.fwKeyEntryMode */
#define WFS_PIN_SECUREKEY_NOTSUPP      (0x0000)
#define WFS_PIN_SECUREKEY_REG_SHIFT    (0x0001)
#define WFS_PIN_SECUREKEY_REG_UNIQUE   (0x0002)
#define WFS_PIN_SECUREKEY_IRREG_SHIFT  (0x0004)
#define WFS_PIN_SECUREKEY_IRREG_UNIQUE (0x0008)


/* 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;

/* WFS_INF_PIN_SECUREKEY_DETAIL command key layout output structure */
typedef struct _wfs_pin_hex_keys
{
    USHORT         usXPos;
    USHORT         usYPos;
    USHORT         usXSize;
    USHORT         usYSize;
    ULONG          ulFK;
    ULONG          ulShiftFK;
} WFSPINHEXKEYS, * LPWFSPINHEXKEYS;

/* WFS_INF_PIN_SECUREKEY_DETAIL command output structure */
typedef struct _wfs_pin_secure_key_detail
{
    WORD                       fwKeyEntryMode;
    LPWFSPINFUNCKEYDETAIL  lpFuncKeyDetail;
    ULONG                      ulClearFDK;
    ULONG                   ulCancelFDK;
    ULONG                   ulBackspaceFDK;
    ULONG                   ulEnterFDK;
    WORD                    wColumns;
    WORD                    wRows;
    LPWFSPINHEXKEYS       * lppHexKeys;
} WFSPINSECUREKEYDETAIL, * LPWFSPINSECUREKEYDETAIL;


/*=================================================================*/
/* 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;


/* WFS_CMD_PIN_SECUREKEY_ENTRY command input structure */
typedef struct _wfs_pin_secure_key_entry
{
    USHORT        usKeyLen;
    BOOL          bAutoEnd;
    ULONG          ulActiveFDKs;
    ULONG          ulActiveKeys;
    ULONG          ulTerminateFDKs;
    ULONG          ulTerminateKeys;
    WORD           wVerificationType;
} WFSPINSECUREKEYENTRY, * LPWFSPINSECUREKEYENTRY;

/* WFS_CMD_PIN_SECUREKEY_ENTRY command output structure */
typedef struct _wfs_pin_secure_key_entry_out
{
    USHORT        usDigits;
    WORD          wCompletion;
    LPWFSXDATA    lpxKCV;
} WFSPINSECUREKEYENTRYOUT, * LPWFSPINSECUREKEYENTRYOUT;

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;


typedef struct _wfs_pin_generate_KCV
{
    LPSTR         lpsKey;
    WORD          wKeyCheckMode;
} WFSPINGENERATEKCV, * LPWFSPINGENERATEKCV;

typedef struct _wfs_pin_kcv
{
    LPWFSXDATA    lpxKCV;
} WFSPINKCV, * LPWFSPINKCV;

/*=================================================================*/
/* 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 */
```