Extensions for Financial Services (XFS) interface specification
Release 3.20 - Part 6: PIN Keypad Device Class Interface
Programmer's Reference

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

The formal process followed by the Workshop in the development of this Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN-CENELEC 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, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.
# Table of Contents

## Foreword ....................................................................................................................... 5

## 1. Introduction................................................................................................................. 8
### 1.1 Background to Release 3.20 .................................................................................. 8
### 1.2 XFS Service-Specific Programming ........................................................................ 8

## 2. PIN Keypad .................................................................................................................. 9

## 3. References .................................................................................................................. 11

## 4. Info Commands .......................................................................................................... 13
### 4.1 WFS_INF_PIN_STATUS ......................................................................................... 13
### 4.2 WFS_INF_PIN_CAPABILITIES ............................................................................. 17
### 4.3 WFS_INF_PIN_KEYDETAIL .................................................................................. 25
### 4.4 WFS_INF_PIN_FUNCKEYDETAIL .......................................................................... 27
### 4.5 WFS_INF_PIN_HSM_TDATA .................................................................................. 30
### 4.6 WFS_INF_PIN_KEYDETAIL_EX ........................................................................... 31
### 4.7 WFS_INF_PIN_SECUREKEYDETAIL ....................................................................... 33
### 4.8 WFS_INF_PIN_QUERY_LOGICAL_HSMDETAIL ....................................................... 37
### 4.9 WFS_INF_PIN_QUERY_PCMPTSDEVICEID ............................................................. 38

## 5. Execute Commands ................................................................................................... 39
### 5.1 Normal PIN Commands .......................................................................................... 39
#### 5.1.1 WFS_CMD_PIN_CRYPT .................................................................................... 39
#### 5.1.2 WFS_CMD_PIN_IMPORT_KEY .......................................................................... 42
#### 5.1.3 WFS_CMD_PIN_DERIVE_KEY .......................................................................... 45
#### 5.1.4 WFS_CMD_PIN_GET_PIN ................................................................................ 47
#### 5.1.5 WFS_CMD_PIN_LOCAL_DES ............................................................................ 50
#### 5.1.6 WFS_CMD_PIN_CREATE_OFFSET ..................................................................... 52
#### 5.1.7 WFS_CMD_PIN_LOCAL_EUROCHEQUE ............................................................. 54
#### 5.1.8 WFS_CMD_PIN_LOCAL_VISA ........................................................................... 56
#### 5.1.9 WFS_CMD_PIN_PRESENT_IDC ......................................................................... 58
#### 5.1.10 WFS_CMD_PIN_GET_PINBLOCK ..................................................................... 60
#### 5.1.11 WFS_CMD_PIN_GET_DATA ............................................................................ 62
#### 5.1.12 WFS_CMD_PIN_INITIALIZATION ..................................................................... 65
#### 5.1.13 WFS_CMD_PIN_LOCAL_BANKSYS ................................................................. 67
#### 5.1.14 WFS_CMD_PIN_BANKSYSIO .......................................................................... 68
#### 5.1.15 WFS_CMD_PIN_RESET .................................................................................. 69
#### 5.1.16 WFS_CMD_PIN_HSMSET_TDATA ................................................................. 70
#### 5.1.17 WFS_CMD_PIN_SECUREMSG_SEND ............................................................... 72
#### 5.1.18 WFS_CMD_PIN_SECUREMSG_RECEIVE ......................................................... 74
#### 5.1.19 WFS_CMD_PIN_GETJOURNAL ........................................................................ 76
#### 5.1.20 WFS_CMD_PIN_IMPORTKEY_EX ................................................................. 77
#### 5.1.21 WFS_CMD_PIN_ENCIO ................................................................................. 80
#### 5.1.22 WFS_CMD_PIN_HSMINIT ............................................................................. 82
#### 5.1.23 WFS_CMD_PIN_SECUREKEYENTRY .............................................................. 83
#### 5.1.24 WFS_CMD_PIN_GENERATE_KCV ................................................................. 86
#### 5.1.25 WFS_CMD_PIN_SET_GUIDANCE_LIGHT ....................................................... 87
#### 5.1.26 WFS_CMD_PIN_MAINTAIN_PIN .................................................................... 89
#### 5.1.27 WFS_CMD_PIN_KEYPRESSBEEP ................................................................. 90
#### 5.1.28 WFS_CMD_PIN_SET_PINBLOCKDATA .......................................................... 91
5.2 Common commands for Remote Key Loading Schemes ......................................................... 96
5.2.1 WFS_CMD_PIN_START_KEY_EXCHANGE ........................................................................ 96

5.3 Remote Key Loading Using Signatures .............................................................................. 97
5.3.1 WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY ...................................................................... 97
5.3.2 WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM ...................................................... 100
5.3.3 WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY ............................................................... 102
5.3.4 WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR ................................................................. 105
5.3.5 WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED_ITEM ......................................................... 107

5.4 Remote Key Loading with Certificates .............................................................................. 109
5.4.1 WFS_CMD_PIN_LOAD_CERTIFICATE ............................................................................. 109
5.4.2 WFS_CMD_PIN_GET_CERTIFICATE ............................................................................... 110
5.4.3 WFS_CMD_PIN_REPLACE_CERTIFICATE ...................................................................... 111
5.4.4 WFS_CMD_PIN_IMPORT_RSA_ENCRYPTED_PKCS7_KEY .................................................. 112

5.5 EMV .............................................................................................................................. 114
5.5.1 WFS_CMD_PIN_EMV_IMPORT_PUBLIC_KEY ................................................................. 114
5.5.2 WFS_CMD_PIN_DIGEST .............................................................................................. 117

6. Events .............................................................................................................................. 118
6.1 WFS_EXEE_PIN_KEY ........................................................................................................ 118
6.2 WFS_SRVE_PIN_INITIALIZED .......................................................................................... 119
6.3 WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS ........................................................................... 120
6.4 WFS_SRVE_PIN_OPT_REQUIRED ..................................................................................... 121
6.5 WFS_SRVE_PIN_CERTIFICATE_CHANGE ......................................................................... 122
6.6 WFS_SRVE_PIN_HSM_TDATA_CHANGED ......................................................................... 123
6.7 WFS_SRVE_PIN_HSM_CHANGED ..................................................................................... 124
6.8 WFS_EXEE_PIN_ENTERDATA ......................................................................................... 125
6.9 WFS_SRVE_PIN_DEVICEPOSITION ................................................................................ 126
6.10 WFS_SRVE_PIN_POWER_SAVE_CHANGE ....................................................................... 127

7. C - Header File ................................................................................................................ 128

8. Appendix-A ..................................................................................................................... 145
8.1 Remote Key Loading Using Signatures ........................................................................... 146
8.1.1 RSA Data Authentication and Digital Signatures ......................................................... 146
8.1.2 RSA Secure Key Exchange using Digital Signatures .................................................. 147
8.1.3 Initialization Phase – Signature Issuer and ATM PIN ................................................... 149
8.1.4 Initialization Phase – Signature Issuer and Host ......................................................... 150
8.1.5 Key Exchange – Host and ATM PIN ........................................................................ 151
8.1.6 Key Exchange (with random number) – Host and ATM PIN ....................................... 152
8.1.7 Enhanced RKL, Key Exchange (with random number) – Host and ATM PIN ............... 153
8.1.8 Default Keys and Security Item loaded during manufacture ....................................... 154

8.2 Remote Key Loading Using Certificates ...................................................................... 155
8.2.1 Certificate Exchange and Authentication ..................................................................... 155
8.2.2 Remote Key Exchange ............................................................................................... 156
8.2.3 Replace Certificate ..................................................................................................... 157
8.2.4 Primary and Secondary Certificates ........................................................................... 158

8.3 German ZKA GeldKarte ............................................................................................... 159
8.3.1 How to use the SECURE_MSG commands ................................................................. 159
8.3.2 Protocol WFS_PIN_PROTISOAS ................................................................................. 160

3
8.3.3 Protocol WFS_PIN_PROTISOLZ .................................................................161
8.3.4 Protocol WFS_PIN_PROTISOPS .................................................................162
8.3.5 Protocol WFS_PIN_PROTCHIPZKA .........................................................163
8.3.6 Protocol WFS_PIN_PROTRAWDATA .......................................................164
8.3.7 Protocol WFS_PIN_PROTPBM .................................................................165
8.3.8 Protocol WFS_PIN_PROTHSMLDI ...........................................................166
8.3.9 Protocol WFS_PIN_PROTGENAS .............................................................167
8.3.10 Protocol WFS_PIN_PROTCHIPINCHG ....................................................170
8.3.11 Protocol WFS_PIN_PROTPINCMP ..........................................................171
8.3.12 Protocol WFS_PIN_PROTISOPINCHG ....................................................173
8.3.13 Command Sequence ..............................................................................174

8.4 EMV Support ...............................................................................................181
8.4.1 Keys loading .............................................................................................181
8.4.2 PIN Block Management ............................................................................183
8.4.3 SHA-1 Digest .............................................................................................184

8.5 French Cartes Bancaires ..............................................................................185
8.5.1 Data Structure for WFS_CMD_PIN_ENC_IO .............................................185
8.5.2 Command Sequence ..................................................................................187

8.6 Secure Key Entry ..............................................................................................189
8.6.1 Keyboard Layout .......................................................................................189
8.6.2 Command Usage .........................................................................................193

9. Appendix-B (Country Specific WFS_CMD_PIN_ENC_IO protocols) .......... 194
9.1 Luxembourg Protocol ..................................................................................194
9.1.1 WFS_CMD_ENC_IO_LUX_LOAD_APPKEY ...........................................196
9.1.2 WFS_CMD_ENC_IO_LUX_GENERATE_MAC .......................................198
9.1.3 WFS_CMD_ENC_IO_LUX_CHECK_MAC .............................................199
9.1.4 WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK ......................................200
9.1.5 WFS_CMD_ENC_IO_LUX_DECRYPT_TDES .......................................201
9.1.6 WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES .......................................202
9.1.7 Luxembourg-specific Header File .............................................................203

10. Appendix-C (Standardized lpszExtra fields) ............................................206
10.1 WFS_INF_PIN_STATUS .............................................................................206
10.2 WFS_INF_PIN_CAPABILITIES ...............................................................207
Foreword

This CWA is revision 3.20 of the XFS interface specification.

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties on 2011-06-29, the constitution of which was supported by CEN following the public call for participation made on 1998-06-24. 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 3.20.

A list of the individuals and organizations which supported the technical consensus represented by the CEN Workshop Agreement is available to purchasers from the CEN-CENELEC Management Centre. These organizations were drawn from the banking sector. The CEN/ISSS XFS Workshop gathered suppliers as well as banks and other financial service companies.

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 and Scanning 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: Card Dispenser Device Class Interface - Programmer's Reference
Part 17: Barcode Reader Device Class Interface - Programmer's Reference
Part 18: Item Processing Module Device Class Interface- Programmer's Reference
Parts 19 - 28: Reserved for future use.

Parts 29 through 47 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
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 Device Class
Part 41: XFS MIB Device Specific Definitions - Cash-In Module Device Class
Part 42: Reserved for future use.
Part 43: XFS MIB Device Specific Definitions - Vendor Dependent Mode Class
Part 44: XFS MIB Application Management
Part 45: XFS MIB Device Specific Definitions - Card Dispenser Device Class
Part 46: XFS MIB Device Specific Definitions - Barcode Reader Device Class
Part 47: XFS MIB Device Specific Definitions - Item Processing Module Device Class
Parts 48 - 60 are reserved for future use.
Part 61: Application Programming Interface (API) - Service Provider Interface (SPI) - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 62: Printer and Scanning Device Class Interface - Migration from Version 3.10 (CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 63: Identification Card Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 64: Cash Dispenser Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 65: PIN Keypad Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 66: Check Reader/Scanner Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 67: Depository Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 68: Text Terminal Unit Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 69: Sensors and Indicators Unit Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 70: Vendor Dependent Mode Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 71: Camera Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 72: Alarm Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 73: Card Embossing Unit Device Class Interface - Migration from Version 3.10 (CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 74: Cash-In Module Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 75: Card Dispenser Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 76: Barcode Reader Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (this CWA) - Programmer's Reference
Part 77: Item Processing Module Device Class Interface - Migration from Version 3.10 (see CWA 15748) to Version 3.20 (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.cen.eu/cen/pages/default.aspx.
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.

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

The final review/endorsement round for this CWA was started on 2011-06-23 and was successfully closed on 2011-07-23. The final text of this CWA was submitted to CEN for publication on 2011-08-26.

This CEN Workshop Agreement is publicly available as a reference document from the National Members of CEN: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

Comments or suggestions from the users of the CEN Workshop Agreement are welcome and should be addressed to the CEN-CENELEC Management Centre.

Revision History:

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.00</td>
<td>October 18, 2000</td>
<td>Initial release.</td>
</tr>
<tr>
<td>3.02</td>
<td>May 21, 2003</td>
<td>For a description of changes from version 3.00 to version 3.02 see the PIN 3.02 Migration document.</td>
</tr>
<tr>
<td>3.03</td>
<td>September 24, 2004</td>
<td>For a description of changes from version 3.02 to version 3.03 see the PIN 3.03 Migration document.</td>
</tr>
<tr>
<td>3.10</td>
<td>November 29, 2007</td>
<td>For a description of changes from version 3.03 to version 3.10 see the PIN 3.10 Migration document.</td>
</tr>
<tr>
<td>3.20</td>
<td>March 2nd, 2011</td>
<td>For a description of changes from version 3.10 to version 3.20 see the PIN 3.20 Migration document.</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Background to Release 3.20

The CEN/ISSS XFS Workshop aims to promote a clear and unambiguous specification defining a multi-vendor software interface to financial peripheral devices. The XFS (eXtensions for Financial Services) specifications are developed within the CEN/ISSS (European Committee for Standardization/Information Society Standardization System) Workshop environment. CEN/ISSS Workshops aim to arrive at a European consensus on an issue that can be published as a CEN Workshop Agreement (CWA).

The CEN/ISSS XFS Workshop encourages the participation of both banks and vendors in the deliberations required to create an industry standard. The CEN/ISSS XFS Workshop achieves its goals by focused sub-groups working electronically and meeting quarterly.

Release 3.20 of the XFS specification is based on a C API and is delivered with the continued promise for the protection of technical investment for existing applications. This release of the specification extends the functionality and capabilities of the existing devices covered by the specification, but does not include any new device classes. Notable major enhancements include Mixed Media processing to allow mixed cash and check accepting, as well as the addition of new commands to the CIM, PTR and IDC to allow better support of the Japanese marketplace.

1.2 XFS Service-Specific Programming

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

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

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

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

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability is not considered to be fundamental to the service. In this case, the Service Provider returns a successful completion, but does no operation. An example would be a request from an application to turn on a control indicator on a passbook printer; the Service Provider recognizes the command, but since the passbook printer it is managing does not include that indicator, the Service Provider does no operation and returns a successful completion to the application.

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability is considered to be fundamental to the service. In this case, a WFS_ERR_UNSUPP_COMMAND error is returned to the calling application. An example would be a request from an application to a cash dispenser to dispense coins; the Service Provider recognizes the command but, since the cash dispenser it is managing dispenses only notes, returns this error.

The requested capability is not defined for the class of Service Providers by the XFS specification. In this case, a WFS_ERR_INVALID_COMMAND error is returned to the calling application.

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

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

This section describes the general interface for the following functions:

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

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

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

Important Notes:

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

This specification also supports the Hardware Security Module (HSM), which is necessary for the German ZKA Electronic Purse transactions. Furthermore the HSM stores terminal specific data.

This data will be compared against the message data fields (Sent and Received ISO8583 messages) prior to HSM-MAC generation/verification. HSM-MACs are generated/verified only if the message fields match the data stored.

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

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

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

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

When hex values are passed to the API within strings, the hex digits 0xA to 0xF can be represented by characters in the ranges ‘a’ to ‘f’ or ‘A’ to ‘F’.

The following commands and events were initially added to support the German ZKA standard, but may also be used for other national standards:

- `WFS_INF_PIN_HSM_TDATA`
- `WFS_CMD_PIN_HSM_SET_TDATA`
- `WFS_CMD_PIN_SECURE_MSG_SEND`
• WFS_CMD_PIN_SECURE_MSG_RECEIVE
• WFS_CMD_PIN_GET_JOURNAL
• WFS_SRVE_PIN_OPT_REQUIRED
• WFS_CMD_PIN_HSM_INIT
• WFS_SRVE_PIN_HSM_TDATA_CHANGED
3. References

1. XFS Application Programming Interface (API)/Service Provider Interface (SPI), Programmer’s Reference Revision 3.20
2. RSA Laboratories, PKCS #7: Cryptographic Message Syntax Standard, Version 1.5, November 1993
6. ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ec-Karte mit Chip, Online-Personalisierung von Terminal-HSMs, Version 3.0, 2. 4. 1998
7. ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ZKA-Chipkarte, Online-Vor-Initialisierung und Online-Anzeige einer Außerbetriebnahme von Terminal-HSMs, Version 1.0, 04.08.2000
8. 473x Programmers Reference Volume 1 - TP-820399-001A
9. 473x Programmers Reference Volume 2 - TP-820403-001A
10. 473x Programmers Reference Volume 3 - TP-820400-001A
11. 473x Programmers Reference Volume 4 - TP-820404-001A
12. 473x P-Model Programmers Reference - TP-820397-001A
13. 473x Log Reference Guide - TP-820398-001A
14. Diebold’s Specification for support of Online Preinitialization and Personalization of Terminal HSMs (OPT) and support for the PAC/MAC standards for the 473x Protocol, Diebold USA, Revision 1.10, revised on May 2002
17. German ZKA specification, published by: Bank-Verlag Koeln, Post Box 300191, 50771 Cologne, Germany; Tel: +49 221 5490-0; Fax: +49 221 5490-120
18. Banksys document “SCM DKH Manual Rel 2.x”
19. Diebold’s and IBM’s Specification for support of Online Preinitialization and Personalization of Terminal HSMs (OPT) and support for the PAC/MAC standards for th 473x Protocol, Diebold USA, Revision 1.8, revised on Jan-03-2001
20. ANSI X3.92, American National Standard for Data Encryption Algorithm (DEA), American National Standards Institute, 1983
29. dvg Hannover, Schnittstellenbeschreibung für Autorisierungsanfragen bei nationalen GA-Verfügungen unter Verwendung der Spur 3, Version 2.5, Stand: 15.03.2000
30. dvg Hannover, Schnittstellenbeschreibung für Autorisierungsanfragen bei internationalen Verfügungen unter Verwendung der Spur 2, Version 2.6, Stand: 30.03.2000
31. ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ec-Karte mit Chip, Geldkarte Ladeterminals, Version 3.0, 2. 4. 1998
32. ISO/IEC 9797-1: 1999
33. ISO 8731-2
34. ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ec-Karte mit Chip
PIN-Änderungsfunktion, Version 3.0, 12.05.1999
35. ANSI X9 TR-31 2005, Interoperable Secure Key Exchange Key Block Specification for Symmetric Algorithms
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>36.</td>
<td>Oliself2 Specifiche Tecniche, PIN Block Detail for WFS_PIN_FORMAT</td>
</tr>
<tr>
<td>37.</td>
<td>PCI Security Standards Council PCI PTS approval list</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.pcisecuritystandards.org/security_standards/ped/pedapprovallist.html">https://www.pcisecuritystandards.org/security_standards/ped/pedapprovallist.html</a></td>
</tr>
</tbody>
</table>
4. Info Commands

4.1 WFS_INF_PIN_STATUS

Description  This command returns several kinds of status information.

Input Param  None.

Output Param  LPWFSPINSTATUS lpStatus;

```
typedef struct _wfs_pin_status {
    WORD     fwDevice;
    WORD     fwEncStat;
    LPSTR     lpszExtra;
    DWORD     dwGuidLights[WFS_PIN_GUIDLIGHTS_SIZE];
    WORD     fwAutoBeepMode;
    DWORD     dwCertificateState;
    WORD     wDevicePosition;
    USHORT    usPowerSaveRecoveryTime;
    WORD     wAntiFraudModule;
} WFSPINSTATUS, *LPWFSPINSTATUS;
```

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_DEVONLINE</td>
<td>The device is online (i.e. powered on and operable).</td>
</tr>
<tr>
<td>WFS_PIN_DEVOFFLINE</td>
<td>The device is offline (e.g. the operator has taken the device offline by turning a switch or pulling out the device).</td>
</tr>
<tr>
<td>WFS_PIN_DEVPOWEROFF</td>
<td>The device is powered off or physically not connected.</td>
</tr>
<tr>
<td>WFS_PIN_DEVNODEVICE</td>
<td>There is no device intended to be there; e.g. this type of self service machine does not contain such a device or it is internally not configured.</td>
</tr>
<tr>
<td>WFS_PIN_DEVHWERROR</td>
<td>The device is inoperable due to a hardware error.</td>
</tr>
<tr>
<td>WFS_PIN_DEVUSERERROR</td>
<td>The device is present but a person is preventing proper device operation.</td>
</tr>
<tr>
<td>WFS_PIN_DEVBUSY</td>
<td>The device is busy and unable to process an execute command at this time.</td>
</tr>
<tr>
<td>WFS_PIN_DEVFRAUDATTEMPT</td>
<td>The device is present but is inoperable because it has detected a fraud attempt.</td>
</tr>
<tr>
<td>WFS_PIN_DEVPOTENTIALFRAUD</td>
<td>The device has detected a potential fraud attempt and is capable of remaining in service. In this case the application should make the decision as to whether to take the device offline.</td>
</tr>
</tbody>
</table>

**fwEncStat**
Specifies the state of the encryption module as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_ENCREADY</td>
<td>The encryption module is initialized and ready (at least one key is imported into the encryption module).</td>
</tr>
<tr>
<td>WFS_PIN_ENCNOTREADY</td>
<td>The encryption module is not available or not ready due to hardware error or communication error.</td>
</tr>
<tr>
<td>WFS_PIN_ENCNOTINITIALIZED</td>
<td>The encryption module is not initialized (no master key loaded).</td>
</tr>
</tbody>
</table>
The encryption module is busy (implies that the device is busy).

The encryption module state is undefined.

The encryption module is initialized and master key (where required) and any other initial keys are loaded; ready to import other keys.

Specifies a list of vendor-specific, or any other extended, information. The information is returned as a series of “key=value” strings so that it is easily extendable by Service Providers. Each string will be null-terminated, the whole list terminated with an additional null character. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters.

A number of lpszExtra key value pairs have been standardized during previous releases of the PIN specification. These values have now been added to the main status structure but the standardized key value pairs in lpszExtra must still be supported by the Service Provider when the functionality is supported. Section 10 defines the standardized lpszExtra key value pairs.

dwGuidLights [...] specifies the state of the guidance light indicators. A number of guidance light types are defined below. Vendor specific guidance lights are defined starting from the end of the array. The maximum guidance light index is WFS_PIN_GUIDLIGHTS_MAX.

Specifies the state of the guidance light indicator as WFS_PIN_GUIDANCE_NOT_AVAILABLE, WFS_PIN_GUIDANCE_OFF or a combination of the following flags consisting of one type B, and optionally one type C.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_GUIDANCE_NOT_AVAILABLE</td>
<td>The status is not available.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_OFF</td>
<td>The light is turned off.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_SLOW_FLASH</td>
<td>The light is blinking slowly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_MEDIUM_FLASH</td>
<td>The light is blinking medium frequency.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_QUICK_FLASH</td>
<td>The light is blinking quickly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_CONTINUOUS</td>
<td>The light is turned on continuous (steady).</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_RED</td>
<td>The light is red.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_GREEN</td>
<td>The light is green.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_YELLOW</td>
<td>The light is yellow.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_BLUE</td>
<td>The light is blue.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_CYAN</td>
<td>The light is cyan.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_MAGENTA</td>
<td>The light is magenta.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_WHITE</td>
<td>The light is white.</td>
<td>C</td>
</tr>
</tbody>
</table>

dwGuidLights [WFS_PIN_GUIDANCE_PINPAD]

Specifies the state of the guidance light indicator on the PIN pad unit.

fwAutoBeepMode

Specifies whether automatic beep tone on key press is active or not. Active and in-active key beeping is reported independently. fwAutoBeepMode can take a combination of the following values, if the flag is not set auto beeping is not activated (or not supported) for that key type (i.e. active or in-active keys):

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_BEEP_ON_ACTIVE</td>
<td>An automatic tone will be generated for all active keys.</td>
</tr>
<tr>
<td>WFS_PIN_BEEP_ON_INACTIVE</td>
<td>An automatic tone will be generated for all in-active keys.</td>
</tr>
</tbody>
</table>

dwCertificateState

Specifies the state of the public verification or encryption key in the PIN certificate modules as one of the following flags:
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_CERT_UNKNOWN</td>
<td>The state of the certificate module is unknown or the device does not have this capability.</td>
</tr>
<tr>
<td>WFS_PIN_CERT_PRIMARY</td>
<td>All pre-loaded certificates have been loaded and that primary verification certificates will be accepted for the commands WFS_CMD_PIN_LOAD_CERTIFICATE or WFS_CMD_PIN_REPLACE_CERTIFICATE.</td>
</tr>
<tr>
<td>WFS_PIN_CERT_SECONDARY</td>
<td>Primary verification certificates will not be accepted and only secondary verification certificates will be accepted. If primary certificates have been compromised (which the certificate authority or the host detects), then secondary certificates should be used in any transaction. This is done by calling the WFS_CMD_PIN_LOAD_CERTIFICATE command or the WFS_CMD_PIN_REPLACE_CERTIFICATE.</td>
</tr>
<tr>
<td>WFS_PIN_CERT_NOTREADY</td>
<td>The certificate module is not ready. (The device is powered off or physically not present).</td>
</tr>
</tbody>
</table>

**wDevicePosition**

Specifies the device position. The device position value is independent of the `fwDevice` value, e.g. when the device position is reported as WFS_PIN_DEVICESPOSITION, `fwDevice` can have any of the values defined above (including WFS_PIN_DEVONLINE or WFS_PIN_DEVOFFLINE). This value is one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_DEVICEINPOSITION</td>
<td>The device is in its normal operating position, or is fixed in place and cannot be moved.</td>
</tr>
<tr>
<td>WFS_PIN_DEVICENOTINPOSITION</td>
<td>The device has been removed from its normal operating position.</td>
</tr>
<tr>
<td>WFS_PIN_DEVICEPOSUNKNOWN</td>
<td>Due to a hardware error or other condition, the position of the device cannot be determined.</td>
</tr>
<tr>
<td>WFS_PIN_DEVICEPOSNOTSUPP</td>
<td>The physical device does not have the capability of detecting the position.</td>
</tr>
</tbody>
</table>

**usPowerSaveRecoveryTime**

Specifies the actual number of seconds required by the device to resume its normal operational state from the current power saving mode. This value is zero if either the power saving mode has not been activated or no power save control is supported.

**wAntiFraudModule**

Specifies the state of the anti-fraud module as one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_AFMNOTSUPP</td>
<td>No anti-fraud module is available.</td>
</tr>
<tr>
<td>WFS_PIN_AFMOOK</td>
<td>Anti-fraud module is in a good state and no foreign device is detected.</td>
</tr>
<tr>
<td>WFS_PIN_AFMINOP</td>
<td>Anti-fraud module is inoperable.</td>
</tr>
<tr>
<td>WFS_PIN_AFMDEVICEDETECTED</td>
<td>Anti-fraud module detected the presence of a foreign device.</td>
</tr>
<tr>
<td>WFS_PIN_AFMUNKNOWN</td>
<td>The state of the anti-fraud module cannot be determined.</td>
</tr>
</tbody>
</table>

**Error Codes**

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

**Comments**

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

In the case where communications with the device have been lost, the `fwDevice` field will report WFS_PIN_DEVPOWEROFF when the device has been removed or WFS_PIN_DEVHWERROR.
if the communications are unexpectedly lost. All other fields should contain a value based on the following rules and priority:

1. Report the value as unknown.
2. Report the value as a general h/w error.
3. Report the value as the last known value.
4.2 WFS_INF_PIN_CAPABILITIES

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

Input Param
None.

Output Param
LPWFSPINCAPS lpCaps;

typedef struct _wfs_pin_caps
{
    WORD    wClass;
    WORD    fwType;
    BOOL    bCompound;
    USHORT  usKeyNum;
    WORD    fwAlgorithms;
    WORD    fwPinFormats;
    WORD    fwDerivationAlgorithms;
    WORD    fwPresentationAlgorithms;
    WORD    fwDisplay;
    BOOL    bIDConnect;
    WORD    fwIDKey;
    WORD    fwValidationAlgorithms;
    WORD    fwKeyCheckModes;
    LPSTR   lpszExtra;
    DWORD   dwGuidLights[WFS_PIN_GUIDLIGHTS_SIZE];
    BOOL    bPINCanPersistAfterUse;
    WORD    fwAutoBeep;
    LPSTR   lpszHSMVendor;
    BOOL    bHSMJournaling;
    DWORD   dwRSAAuthenticationScheme;
    DWORD   dwRSASignatureAlgorithm;
    DWORD   dwRSACryptAlgorithm;
    DWORD   dwRSAKeyCheckMode;
    DWORD   dwSignatureScheme;
    LPWORD  lpwEMVImportSchemes;
    WORD    fwEMVHashAlgorithm;
    BOOL    bKeyImportThroughParts;
    WORD    fwENCIOProtocols;
    BOOL    bTypeCombined;
    BOOL    bSetPinblockDataRequired;
    WORD    fwKeyBlockImportFormats;
    BOOL    bPowerSaveControl;
    BOOL    bAntiFraudModule;
} WFSPINCAPS, *LPWFSPINCAPS;

wClass
Specifies the logical service class as WFS_SERVICE_CLASS_PIN.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_TYPEEPP</td>
<td>Electronic PIN pad (keyboard data entry device).</td>
</tr>
<tr>
<td>WFS_PIN_TYPEEDM</td>
<td>Encryption/decryption module.</td>
</tr>
<tr>
<td>WFS_PIN_TYPEHSM</td>
<td>Hardware security module (electronic PIN pad and encryption module within the same physical unit).</td>
</tr>
</tbody>
</table>

bCompound
Specifies whether the logical device is part of a compound physical device.

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_CRYPTDESECB</td>
<td>Electronic Code Book.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTDESCBC</td>
<td>Cipher Block Chaining.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTDESCFB</td>
<td>Cipher Feed Back.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTRSA</td>
<td>RSA Encryption.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTECMA</td>
<td>ECMA Encryption.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTDESMAC</td>
<td>MAC calculation using CBC.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTTRIDESECB</td>
<td>Triple DES with Electronic Code Book.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTTRIDESCBC</td>
<td>Triple DES with Cipher Block Chaining.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTTRIDESCFB</td>
<td>Triple DES with Cipher Feed Back.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTTRIDESMAC</td>
<td>Last Block Triple DES MAC as defined in ISO/IEC 9797-1:1999 [Ref. 32], using: block length n=64, Padding Method 1 (when bPadding=0), MAC Algorithm 3, MAC length m where 32&lt;=m&lt;=64.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPTMAAMAC</td>
<td>MAC calculation using the Message authenticator algorithm as defined in ISO 8731-2 [Ref. 33].</td>
</tr>
</tbody>
</table>

## FWPinFormats

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_FORM3624</td>
<td>PIN left justified, filled with padding characters, PIN length 4-16 digits. The padding character is a hexadecimal digit in the range 0x00 to 0x0F.</td>
</tr>
<tr>
<td>WFS_PIN_FORMANSI</td>
<td>PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number).</td>
</tr>
<tr>
<td>WFS_PIN_FORMISO0</td>
<td>PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number without check number, no minimum length specified, missing digits are filled with 0x00).</td>
</tr>
<tr>
<td>WFS_PIN_FORMISO1</td>
<td>PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits). (similar to WFS_PIN_FORM3624), PIN only 4 digits.</td>
</tr>
<tr>
<td>WFS_PIN_FORMECI2</td>
<td>PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from 0x0 to 0xF.</td>
</tr>
<tr>
<td>WFS_PIN_FORMECI3</td>
<td>PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with 0x0 to the length of six, the padding character can range from 0x0 through 0x9 (This format is also referred to as VISA2).</td>
</tr>
<tr>
<td>WFS_PIN_FORMVISA</td>
<td>PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted.</td>
</tr>
</tbody>
</table>
WFS_PIN_FORMDIEBOLDCO
PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is preceded by the one-digit coordination number with a value from 0x0 to 0xF, padded with the padding character with a value from 0x0 to 0xF and may be not encrypted, single encrypted or double encrypted.

WFS_PIN_FORMVISA3
PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is followed by a delimiter with the value of 0xF and then padded by the padding character with a value between 0x0 to 0xF.

WFS_PIN_FORMBANKSYS
PIN is encrypted and formatted according to the Banksys PIN block specifications.

WFS_PIN_FORMEMV
The PIN block is constructed as follows: PIN is preceded by 0x02 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, formatted up to 248 bytes of other data as defined within the EMV 4.0 specifications and finally encrypted with an RSA key.

WFS_PIN_FORMISO3
PIN is preceded by 0x03 and the length of the PIN (0x04 to 0x0C), padding characters sequentially or randomly chosen, XORed with digits from PAN.

WFS_PIN_FORMAP
PIN is formatted according to the Italian Bancomat specifications. It is known as the Authentication Parameter PIN block and is created with a 5 digit PIN, an 18 digit PAN, and the 8 digit CCS from the track data.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_CHIP_ZKA</td>
<td>Algorithm for the derivation of a chip card individual key as described by the German ZKA.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_PRESENT_CLEAR</td>
<td>Algorithm for the presentation of a clear text PIN to a chipcard. Each digit of the clear text PIN is inserted as one nibble (=halfbyte) into lpbChipData. See WFS_CMD_PIN_PRESENT_IDC for a detailed description.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_DISPNONE</td>
<td>No display unit.</td>
</tr>
<tr>
<td>WFS_PIN_DISPLEDTHROUGH</td>
<td>Lights next to text guide user.</td>
</tr>
<tr>
<td>WFS_PIN_DISPDISPLAY</td>
<td>A real display is available (this doesn’t apply for self-service).</td>
</tr>
</tbody>
</table>

**bIDConnect**
Specifies whether the PIN pad is directly physically connected to the ID card unit. If the value is TRUE, the PIN will be transported securely during the command WFS_CMD_PIN_PRESENT_IDC.
fwIDKey
Specifies if key owner identification (in commands referenced as lpxIdent), which authorizes access to the encryption module, is required. A zero value is returned if the encryption module does not support this capability. Otherwise it will be a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_IDKEYINITIALIZATION</td>
<td>ID key is returned by the WFS_CMD_PIN_INITIALIZATION command.</td>
</tr>
<tr>
<td>WFS_PIN_IDKEYIMPORT</td>
<td>ID key is required as input for the WFS_CMD_PIN_IMPORT_KEY and WFS_CMD_PIN_DERIVE_KEY command.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_DES</td>
<td>DES algorithm.</td>
</tr>
<tr>
<td>WFS_PIN_EUROCHEQUE</td>
<td>EUROCHEQUE algorithm.</td>
</tr>
<tr>
<td>WFS_PIN_VISA</td>
<td>VISA algorithm.</td>
</tr>
<tr>
<td>WFS_PIN_DES_OFFSET</td>
<td>DES offset generation algorithm.</td>
</tr>
<tr>
<td>WFS_PIN_BANKSYS</td>
<td>Banksys algorithm.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_KCVSELF</td>
<td>The key check value is created by an encryption of the key with itself. For a double length key the KCV is generated using 3DES encryption using the first half of the key as the source data for the encryption.</td>
</tr>
<tr>
<td>WFS_PIN_KCVZERO</td>
<td>The key check value is created by encrypting a zero value with the key.</td>
</tr>
</tbody>
</table>

lpszExtra
Points to a list of vendor-specific, or any other extended, information. The information is returned as a series of “key=value” strings so that it is easily extendable by Service Providers. Each string is null-terminated, the whole list terminated with an additional null character. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters.

A number of lpszExtra key value pairs have been standardized during previous releases of the PIN specification. These values have now been added to the main capabilities structure but the standardized key value pairs in lpszExtra must still be supported by the Service Provider when the functionality is supported. Section 10 defines the standardized lpszExtra key value pairs.

dwGuidLights [...] Specifies which guidance lights are available. A number of guidance light types are defined below. Vendor specific guidance lights are defined starting from the end of the array. The maximum guidance light index is WFS_PIN_GUIDLIGHTS_MAX.

The elements of this array are specified as a combination of the following flags and indicate all of the possible flash rates (type B) and colors (type C) that the guidance light indicator is capable of handling. A value of WFS_PIN_GUIDANCE_NOT_AVAILABLE indicates that the device has no guidance light indicator or the device controls the light directly with no application control possible.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_GUIDANCE_NOT_AVAILABLE</td>
<td>There is no guidance light control available at this position.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_OFF</td>
<td>The light can be off.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_SLOW_FLASH</td>
<td>The light can blink slowly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_MEDIUM_FLASH</td>
<td>The light can blink medium frequency.</td>
<td>B</td>
</tr>
</tbody>
</table>
WFS_PIN_GUIDANCE_QUICK_FLASH  The light can blink quickly.  B
WFS_PIN_GUIDANCE_CONTINUOUS  The light can be continuous (steady).  B
WFS_PIN_GUIDANCE_RED  The light can be red.  C
WFS_PIN_GUIDANCE_GREEN  The light can be green.  C
WFS_PIN_GUIDANCE_YELLOW  The light can be yellow.  C
WFS_PIN_GUIDANCE_BLUE  The light can be blue.  C
WFS_PIN_GUIDANCE_CYAN  The light can be cyan.  C
WFS_PIN_GUIDANCE_MAGENTA  The light can be magenta.  C
WFS_PIN_GUIDANCE_WHITE  The light can be white.  C

dwGuidLights [WFS_PIN_GUIDANCE_PINPAD]
Specifies whether the guidance light indicator on the PIN pad unit is available.

bPINCanPersistAfterUse
Specifies whether the device can retain the PIN after a PIN processing command, e.g. WFS_CMD_PIN_GET_PINBLOCK, WFS_CMD_PIN_LOCAL_DES, WFS_CMD_PIN_PRESENT_IDC, etc:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Applications may request, through the WFS_CMD_PIN_MAINTAIN_PIN command, that the PIN continues to be held within the device after use by a PIN processing command.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The PIN will always be cleared by the device after processing. The WFS_CMD_PIN_MAINTAIN_PIN is not supported.</td>
</tr>
</tbody>
</table>

fwAutoBeep
Specifies whether the PIN device will emit a key beep tone on key presses (of active keys or inactive keys), and if so, which mode it supports. Specified as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_BEEP_ACTIVE_AVAILABLE</td>
<td>Automatic beep tone on active key key-press is supported. If this flag is not set then automatic beeping for active keys is not supported.</td>
</tr>
<tr>
<td>WFS_PIN_BEEP_ACTIVE_SELECTABLE</td>
<td>Automatic beeping for active keys can be controlled (i.e. turned on and off) by the application. If this flag is not set then automatic beeping for active keys cannot be controlled by an application.</td>
</tr>
<tr>
<td>WFS_PIN_BEEP_INACTIVE_AVAILABLE</td>
<td>Automatic beep tone on in-active key key-press is supported. If this flag is not set then automatic beeping for in-active keys is not supported.</td>
</tr>
<tr>
<td>WFS_PIN_BEEP_INACTIVE_SELECTABLE</td>
<td>Automatic beeping for in-active keys can be controlled (i.e. turned on and off) by the application. If this flag is not set then automatic beeping for in-active keys cannot be controlled by an application.</td>
</tr>
</tbody>
</table>

lpsHSMVendor
Identifies the HSM Vendor. lpsHSMVendor is NULL when the HSM Vendor is unknown or the HSM is not supported.

The following is a list of known vendors’ strings that lpsHSMVendor can contain for the support of German HSMs:

“KRONE”
“ASCOM”
“IBM”
“NCR”

bHSMJournaling
Specifies whether the HSM supports journaling by the WFS_CMD_PIN_GET_JOURNAL command. The value of this parameter is either TRUE or FALSE. TRUE means the HSM supports journaling by WFS_CMD_GET_JOURNAL.

dwRSAAuthenticationScheme
Specifies which type(s) of Remote Key Loading/Authentication is supported as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_RSA_AUTH_2PARTY_SIG</td>
<td>Two-party Signature based authentication.</td>
</tr>
<tr>
<td>WFS_PIN_RSA_AUTH_3PARTY_CERT</td>
<td>Three-party Certificate based authentication.</td>
</tr>
</tbody>
</table>


dwRSASignatureAlgorithm
Specify which type(s) of RSA Signature Algorithm(s) is supported as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PKCS1_V1_5</td>
<td>SSA_PKCS_V1_5 Signatures supported.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PSS</td>
<td>SSA_PSS Signatures supported.</td>
</tr>
</tbody>
</table>


dwRSAEncryptAlgorithm
Specify which type(s) of RSA Encipherment Algorithm(s) is supported as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_CRYPT_RSAES_PKCS1_V1_5</td>
<td>AES_PKCS_V1_5 algorithm supported.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPT_RSAES_OAEP</td>
<td>AES_OAEP algorithm supported.</td>
</tr>
</tbody>
</table>


dwRSAKeyCheckMode
Specifies which algorithm/method used to generate the public key check value/thumb print as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_RSA_KCV_SHA1</td>
<td>SHA-1 is supported as defined in Ref. 3.</td>
</tr>
</tbody>
</table>


dwSignatureScheme
Specifies which capabilities are supported by the Signature scheme as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIG_GEN_RSA_KEY_PAIR</td>
<td>Specifies if the Service Provider supports the RSA Signature Scheme WFS_CMD_PIN_GENERATE_RSA_KEYPAIR and WFS_CMD_PIN_EXPORT_RSA_EPP_SIG NED commands.</td>
</tr>
<tr>
<td>WFS_PIN_SIG_RANDOM_NUMBER</td>
<td>Specifies if the Service Provider returns a random number from the WFS_CMD_PIN_START_KEY_EXCHANGE command within the RSA Signature Scheme.</td>
</tr>
<tr>
<td>WFS_PIN_SIG_EXPORT_EPP_ID</td>
<td>Specifies if the Service Provider supports exporting the EPP Security Item within the RSA Signature Scheme.</td>
</tr>
<tr>
<td>WFS_PIN_SIG_ENHANCED_RKL</td>
<td>Specifies that the Service Provider supports the Enhanced Signature Remote Key Scheme. This scheme allows the customer to manage their own public keys independently of the Signature Issuer. When this mode is supported then the key loaded signed with the Signature Issuer key is the host root public key $PK_{ROOT}$, rather than $PK_{HOST}$. See Section 8.1 for a full description.</td>
</tr>
</tbody>
</table>
lpwEMVImportSchemes
Identifies the supported EMV Import Scheme(s) as a zero terminated array of modes.
lpwEMVImportSchemes is set to NULL if the Import Scheme(s) are unknown or not supported.
Otherwise lpwEMVImportSchemes lists all Import Scheme(s) supported by the PIN Service Provider from the following possible values:

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

fwEMVHashAlgorithm
Specifies which hash algorithm is supported for the calculation of the HASH as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_HASH_SHA1_DIGEST</td>
<td>The SHA 1 digest algorithm is supported by the WFS_CMD_PIN_DIGEST command.</td>
</tr>
</tbody>
</table>

bKeyImportThroughParts
Specifies whether the device is capable of importing keys in multiple parts. TRUE means the device supports the key import in multiple parts.

fwENCIOProtocols
Specifies the ENC_IO protocols supported to communicate with the encryption module as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_ENC_PROT_CH</td>
<td>For Swiss specific protocols. The document specification for Swiss specific protocols is &quot;CMD_ENC_IO - CH Protocol.doc&quot;. This document is available at the following address: EUROPAY (Switzerland) SA Terminal Management Hertistrasse 27 CH-8304 Wallisellen</td>
</tr>
<tr>
<td>WFS_PIN_ENC_PROT_GIECB</td>
<td>Protocol for “Groupement des Cartes Bancaires” (France).</td>
</tr>
<tr>
<td>WFS_PIN_ENC_PROT_LUX</td>
<td>Protocol for Luxembourg commands. The reference for this specific protocol is the Authorization Center in Luxembourg (CETREL.) Cryptography Management Postal address: CETREL Société Coopérative Centre de Transferts Electroniques L-2956 Luxembourg</td>
</tr>
</tbody>
</table>
**bTypeCombined**
Specifies whether the keypad used in the secure PIN pad module is integrated within a generic Win32 keyboard.

TRUE means the secure PIN keypad is integrated within a generic Win32 keyboard and standard Win32 key events will be generated for any key when there is no ‘active’ GET_DATA or GET_PIN command. Note that XFS continues to support defined PIN keys only, and is not extended to support new alphanumeric keys.

This feature assists in developing generic browser based applications which need to access both PIN and generic keyboards.

- When an application wishes to receive XFS-based key information then it can use the WFS_CMD_PIN_GET_DATA and WFS_CMD_PIN_GET_PIN commands.
- No Win32 keystrokes are generated for any key (active or not) in a combined device when WFS_CMD_PIN_GET_DATA or WFS_CMD_PIN_GET_PIN are ‘active’.
- When no WFS_CMD_PIN_GET_DATA or WFS_CMD_PIN_GET_PIN command is ‘active’ then any key press will result in a Win32 key event. These events can be ignored by the application, if required.

Note that this does not compromise secure PIN entry – there will be no Win32 keyboard events during PIN collection.

On terminals and kiosks with separate PIN and Win32 keyboards, the Win32 keyboard behaves purely as a PC keyboard and the PIN device behaves only as an XFS device.

**bSetPinblockDataRequired**
Specifies whether the command WFS_CMD_PIN_SET_PINBLOCK_DATA must be called before the PIN is entered via WFS_CMD_PIN_GET_PIN and retrieved via WFS_CMD_PIN_GET_PINBLOCK.

**fwKeyBlockImportFormats**
Supported key block formats; a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_ANSTR31KEYBLOCK</td>
<td>Supports ANS TR-31 Keyblock format key import.</td>
</tr>
</tbody>
</table>

**bPowerSaveControl**
Specifies whether power saving control is available. This can either be TRUE if available or FALSE if not available.

**bAntiFraudModule**
Specifies whether the anti-fraud module is available. This can either be TRUE if available or FALSE if not available.

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

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

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

Details relating to the keys loaded using OPT (via the ZKA WFS_PIN_PROTISOPS protocol) are retrieved using the ZKA WFS_PIN_PROTHSMLDI protocol. These keys are not reported by this command.

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
LPWFS_PINKEYDETAIL *lppKeyDetail;

Pointer to a NULL-terminated array of pointers to WFSPINKEYDETAIL structures.

typedef struct _wfs_pin_key_detail
{       
    LPSTR lpsKeyName;
    WORD   fwUse;
    BOOL   bLoaded;
    LPWFSXDATA lpxKeyBlockHeader;
} 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USECRYPT</td>
<td>Key can be used for encryption/decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USEFUNCTION</td>
<td>Key can be used for PIN functions.</td>
</tr>
<tr>
<td>WFS_PIN_USEMACING</td>
<td>Key can be used for MACing.</td>
</tr>
<tr>
<td>WFS_PIN_USEKEYENCKEY</td>
<td>Key is used as key encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USENEODYM</td>
<td>Key can be imported only once.</td>
</tr>
<tr>
<td>WFS_PIN_USESVCMB</td>
<td>Key is used as CBC Start Value encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USECONSTRUCT</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_PIN_USESECURECONSTRUCT</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_PIN_USEANSTR31MASTER</td>
<td>Key is an ANS X9 TR-31 key block master key (see reference 35).</td>
</tr>
</tbody>
</table>

bLoaded
Specifies whether the key has been loaded (imported from Application or locally from Operator).

lpxKeyBlockHeader
Contains the key block header of keys imported within an ANS TR-31 keyblock. This data is encoded in the same format that it was imported in, and contains all mandatory and optional header fields. lpxKeyBlockHeader is NULL if the key was not imported within a key block or has not been loaded yet. The fwUse field provides a summary of the key use.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key name is not found.</td>
</tr>
</tbody>
</table>

Comments

None.
4.4 WFS_INF_PIN_FUNCKEY_DETAIL

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

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

**Input Param**
LPULONG lpulFDKMask;

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

**Output Param**
LPWFSFUNCKEYDETAIL lpFuncKeyDetail;

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

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

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

The remaining 6 bit masks may be used as vendor dependent keys.
- WFS_PIN_FK_OEM1
- WFS_PIN_FK_OEM2
- WFS_PIN_FK_OEM3
- WFS_PIN_FK_OEM4
- WFS_PIN_FK_OEM5
- WFS_PIN_FK_OEM6
usNumberFDKs
This value indicates the number of FDK structures returned. Only supported FDKs are returned.

lppFDKs
Pointer to an array of pointers to WFSPINFSDK structures. It is the responsibility of the
application to identify the mapping between the FDK code and the physical location of the FDK.
lppFDKs is NULL if no FDKs are requested or supported.

typedef struct _wfs_pin_fdk
{
    ULONG     ulFDK;
    USHORT    usXPosition;
    USHORT    usYPosition;
} WFSPINFSDK, *LPWFSPINFSDK;

ulFDK
Specifies the code returned by this FDK, defined as one of the following values:
    WFS_PIN_FK_FDK01
    WFS_PIN_FK_FDK02
    WFS_PIN_FK_FDK03
    WFS_PIN_FK_FDK04
    WFS_PIN_FK_FDK05
    WFS_PIN_FK_FDK06
    WFS_PIN_FK_FDK07
    WFS_PIN_FK_FDK08
    WFS_PIN_FK_FDK09
    WFS_PIN_FK_FDK10
    WFS_PIN_FK_FDK11
    WFS_PIN_FK_FDK12
    WFS_PIN_FK_FDK13
    WFS_PIN_FK_FDK14
    WFS_PIN_FK_FDK15
    WFS_PIN_FK_FDK16
    WFS_PIN_FK_FDK17
    WFS_PIN_FK_FDK18
    WFS_PIN_FK_FDK19
    WFS_PIN_FK_FDK20
    WFS_PIN_FK_FDK21
    WFS_PIN_FK_FDK22
    WFS_PIN_FK_FDK23
    WFS_PIN_FK_FDK24
    WFS_PIN_FK_FDK25
    WFS_PIN_FK_FDK26
    WFS_PIN_FK_FDK27
    WFS_PIN_FK_FDK28
    WFS_PIN_FK_FDK29
    WFS_PIN_FK_FDK30
    WFS_PIN_FK_FDK31
    WFS_PIN_FK_FDK32

usXPosition
For FDKs, specifies the screen position the FDK relates to. This position is relative to the Left
Hand side of the screen expressed as a percentage of the width of the screen.
For FDKs along the side of the screen this will be 0 (left side) or 100 (right side, user’s view).

usYPosition
For FDKs, specifies the screen position the FDK relates to. This position is relative to the top
of the screen expressed as a percentage of the height of the screen.
For FDKs above or below the screen this will be 0 (above) or 100 (below).
Diagram: Shows how \textit{usXPosition} and \textit{usYPosition} are set.

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

**Comments**
None.
4.5   WFS-INF_PIN_HSM_TDATA

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

Input Param  None.

Output Param  LPWFSXDATA lpxTData;

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

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

Comments  None.
4.6 WFS_INF_PIN_KEY_DETAIL_EX

Description
This command returns extended detailed information about the keys in the encryption module, including DES, private and public keys. This command will also return information on all keys loaded during manufacture that can be used by applications.

Details relating to the keys loaded using OPT (via the ZKA WFS_PIN_PROTISOPS protocol) are retrieved using the ZKA WFS_PIN_PROTHSMLDI protocol. These keys are not reported by this command.

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
LPWFS_PIN_KEYDETAIL_EX *lppKeyDetailEx;

Pointer to a null-terminated array of pointers to WFS_PIN_KEYDETAIL_EX structures.

typedef struct _wfs_pin_key_detail_ex
{
    LPSTR     lpsKeyName;
    DWORD     dwUse;
    BYTE      bGeneration;
    BYTE      bVersion;
    BYTE      bActivatingDate[4];
    BYTE      bExpiryDate[4];
    BOOL      bLoaded;
    LPWFS_XDATA lpxKeyBlockHeader;
} WFS_PIN_KEYDETAIL_EX, *LPWFS_PIN_KEYDETAIL_EX;

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USECRYPT</td>
<td>Key can be used for encryption/decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USEFUNCTION</td>
<td>Key can be used for PIN functions.</td>
</tr>
<tr>
<td>WFS_PIN_USEMACING</td>
<td>Key can be used for MACing.</td>
</tr>
<tr>
<td>WFS_PIN_USEKEYENCEY</td>
<td>Key is used as key encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USEENODUPLICATE</td>
<td>Key can be imported only once.</td>
</tr>
<tr>
<td>WFS_PIN_USEMACKEY</td>
<td>Key is used as a CBC Start Value encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USEPINLOCAL</td>
<td>Key is used for local PIN check.</td>
</tr>
<tr>
<td>WFS_PIN_USERSAPUBLIC</td>
<td>Key is used as a public key for RSA encryption including EMV PIN block creation.</td>
</tr>
<tr>
<td>WFS_PIN_USERSAPRIVATE</td>
<td>Key is used as a private key for RSA decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USERSAPRIVATE_SIGN</td>
<td>Key is used as a private key for RSA Signature generation. Only data generated within the device can be signed.</td>
</tr>
<tr>
<td>WFS_PIN_USECHIPINFO</td>
<td>Key is used as KGKINFO key (only ZKA standard).</td>
</tr>
<tr>
<td>WFS_PIN_USECHIPPIN</td>
<td>Key is used as KGKPIN key (only ZKA standard).</td>
</tr>
<tr>
<td>WFS_PIN_USECHIPPSS</td>
<td>Key is used as K_pp key (only ZKA standard).</td>
</tr>
<tr>
<td>WFS_PIN_USECHIPMAC</td>
<td>Key is used as K_MAC key (only ZKA standard).</td>
</tr>
<tr>
<td>WFS_PIN_USECHIPLT</td>
<td>Key is used as KGK_LT key (only ZKA standard).</td>
</tr>
<tr>
<td>WFS_PIN_USECHIPMACLZ</td>
<td>Key is used as K_PACMAC key (only ZKA standard).</td>
</tr>
<tr>
<td>Key Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>WFS_PIN_USECHIPMACAZ</td>
<td>Key is used as $K_{\text{MASTER}}$ key (only ZKA standard).</td>
</tr>
<tr>
<td>WFS_PIN_USERSAPUBLICVERIFY</td>
<td>Key is used as a public key for RSA signature verification and/or data decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USECONSTRUCT</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_PIN_USESECURECONSTRUCT</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_PIN_USEANSTR31MASTER</td>
<td>Key is an ANS X9 TR-31 key block master key (see reference 35).</td>
</tr>
</tbody>
</table>

**bGeneration**
Specifies the generation of the key as BCD value. Different generations might correspond to different environments (e.g. test or production environment). The content is vendor specific. This value will be 0xFF if no such information is available for the key.

**bVersion**
Specifies the version of the key (the year in which the key is valid, e.g. 01 for 2001) as BCD value. This 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. This value 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. This value 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).

**lpxKeyBlockHeader**
Contains the key block header of keys imported within an ANS TR-31 keyblock. This data is encoded in the same format that it was imported in, and contains all mandatory and optional header fields. lpxKeyBlockHeader is NULL if the key was not imported within a key block or has not been loaded yet. The dwUse field provides a summary of the key use.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key name is not found.</td>
</tr>
</tbody>
</table>

### Comments
When the encryption module contains a public/private key-pair, only the private part of the key will be reported. Every private key in the encryption module 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.
4.7 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 PIN pads 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SECUREKEY_NOTSUPP</td>
<td>Secure key entry is not supported, all other parameters are undefined.</td>
</tr>
</tbody>
</table>
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 PIN pad (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 PIN pad.

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 PIN pad (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 PIN pad.

lFuncKeyDetail

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 PIN pad 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:

<table>
<thead>
<tr>
<th>Function Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_FK_A</td>
<td>(hex digit A)</td>
</tr>
<tr>
<td>WFS_PIN_FK_B</td>
<td>(hex digit B)</td>
</tr>
<tr>
<td>WFS_PIN_FK_C</td>
<td>(hex digit C)</td>
</tr>
<tr>
<td>WFS_PIN_FK_D</td>
<td>(hex digit D)</td>
</tr>
<tr>
<td>WFS_PIN_FK_E</td>
<td>(hex digit E)</td>
</tr>
</tbody>
</table>
ulClearFDK
The FDK code mask reporting any FDKs associated with Clear. If this field is zero 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 zero 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 zero 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 zero 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 PIN pad (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 PIN pad (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, i.e. square if the parameters are the same or rectangular if wColumns is larger than wRows, etc.

lppHexKeys
A NULL-terminated array of pointers to WFSPINHEXKEYS structures describing the physical keys on the PIN pad, 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 PIN pad 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 expressed as a value between 1 and 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 8.6 to explain the use of the lppHexKeys parameter. In addition section 8.6 also provides an example of a command flow required to enter encryption keys securely.
4.8 WFS_INF_PIN_QUERY_LOGICAL_HSM_DETAIL

**Description**
This command reports the ZKA logical HSMs available within the EPP. It also reports which logical HSM is currently active.

**Input Param**
None.

**Output Param**
LPWFSPINHSMDETAIL lpHSMDetail;

```c
typedef struct _wfs_pin_hsm_detail
{
    WORD     wActiveLogicalHSM;
    LPWFSPINHSMINFO   *lppHSMInfo;
} WFSPINHSMDETAIL, *LPWFSPINHSMDETAIL;
```

- `wActiveLogicalHSM`
  Specifies the serial number of the logical HSM that is currently active. This value is the HSM serial number (tag CB in the HSM TDATA) encoded as a normal binary value (i.e. it is not a BCD). If no logical HSMs are present or logical HSMs are not supported then this value is zero.

- `lppHSMInfo`
  Pointer to a NULL terminated array of pointers to WFSPINHSMINFO structures (one for each logical HSM). A NULL pointer is returned if no logical HSMs are supported/present.

```c
typedef struct _wfs_pin_hsm_info
{
    WORD    wHSMSerialNumber;
    LPSTR    lpsZKAID;
} WFSPINHSMINFO, *LPWFSPINHSMINFO;
```

- `wHSMSerialNumber`
  Specifies the Serial Number of the Logical HSM (tag CB in the HSM TDATA). This value is encoded as a normal binary value (i.e. it is not a BCD).

- `lpsZKAID`
  A null-terminated string containing the ZKA ID of the logical HSM (defined by tag CC in the HSM TDATA). The characters in the string are EBCDIC characters.

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

**Comments**
None.
## 4.9 WFS_INF_PIN_QUERY_PCIPTS_DEVICE_ID

**Description**
This command is used to report information in order to verify the PCI Security Standards Council PIN transaction security (PTS) certification held by the PIN device. The command provides detailed information in order to verify the certification level of the device. Support of this command by the Service Provider does not imply in anyway the certification level achieved by the device.

**Input Param**
None.

**Output Param**

```c
typedef struct _wfs_pin_pcipts_deviceid {
    LPSTR    lpszManufacturerIdentifier;
    LPSTR    lpszModelIdentifier;
    LPSTR    lpszHardwareIdentifier;
    LPSTR    lpszFirmwareIdentifier;
    LPSTR    lpszApplicationIdentifier;
} WFSPINPCIPTSDEVICEID, *LPWFSPINPCIPTSDEVICEID;
```

- **lpszManufacturerIdentifier**
  Returns an ASCII string containing the manufacturer identifier of the PIN device. This value is NULL if the manufacturer identifier is not available. This field is distinct from the HSM key pair that may be reported in the `lpszExtra` field by the WFS_INF_PIN_CAPABILITIES command.

- **lpszModelIdentifier**
  Returns an ASCII string containing the model identifier of the PIN device. This value is NULL if the model identifier is not available.

- **lpszHardwareIdentifier**
  Returns an ASCII string containing the hardware identifier of the PIN device. This value is NULL if the hardware identifier is not available.

- **lpszFirmwareIdentifier**
  Returns an ASCII string containing the firmware identifier of the PIN device. This value is NULL if the firmware identifier is not available.

- **lpszApplicationIdentifier**
  Returns an ASCII string containing the application identifier of the PIN device. This value is NULL if the application identifier is not available.

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

**Comments**
The string contained in `lpszManufacturerIdentifier`, `lpszModelIdentifier`, `lpszHardwareIdentifier`, `lpszFirmwareIdentifier`, and `lpszApplicationIdentifier` are expected to match those submitted to the PCI Security Standards Council in order for the certification level to be determined. The PCI PTS certification levels for PIN devices are available on the PCI Security Standards Council website (see Reference 37).
5. Execute Commands

5.1 Normal PIN Commands

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

5.1.1 WFS_CMD_PIN_CRYPT

Description

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

This command can also be used for random number generation.

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

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

Input Param

LPWFSPINCYPRT lpCrypt;

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

wMode

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_MODEENCRIPT</td>
<td>Encrypt with key.</td>
</tr>
<tr>
<td>WFS_PIN_MODEDECRYPT</td>
<td>Decrypt with key.</td>
</tr>
<tr>
<td>WFS_PIN_MODERANDOM</td>
<td>An 8 byte random value shall be returned (in this case all the other input parameters are ignored).</td>
</tr>
</tbody>
</table>

This parameter does not apply to MACing.

lpsKey

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

lpxKeyEncKey

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

wAlgorithm

Specifies the encryption algorithm. Possible values are those described in WFS_INF_PIN_CAPABILITIES. This value is ignored, if wMode equals WFS_PIN_MODERANDOM.
lpsStartValueKey
Specifies the name of the stored key used to decrypt the lpxStartValue to obtain the Initialization Vector. If this parameter is NULL, lpxStartValue is used as the Initialization Vector. This value is ignored, if wMode equals WFS_PIN_MODERANDOM.

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

bPadding
Specifies the padding character. The padding character is a full byte, e.g. 0xFF. This value is ignored, if wMode equals WFS_PIN_MODERANDOM. The valid range is 0x00 to 0xFF.

bCompression
Specifies whether data is to be compressed (blanks removed) before building the MAC. If bCompression is 0x00 no compression is selected, otherwise bCompression holds the representation of the blank character (e.g. 0x20 in ASCII or 0x40 in EBCDIC). This value is ignored, if wMode equals WFS_PIN_MODERANDOM.

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

Output Param
LPWFSXDATA lpxCryptData;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_MODENOTSUPPORTED</td>
<td>The specified mode is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key name was found but the corresponding key value has not been loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpxKeyEncKey or lpxStartValue is not supported or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOCHIPTRANSACTIVE</td>
<td>A chipcard key is used as encryption key and there is no chip transaction active.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ALGORITHMNOTSUPP</td>
<td>The specified algorithm is not supported by this key.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

Comments
The key used for encryption/decryption must be a double length key when used for Triple DES encryption/decryption. If a double-length key is used when a DES encryption algorithm is specified, or a single-length key is used when Triple DES is specified, the WFS_ERR_PIN_INVALIDKEYLENGTH error is returned. Users of this specification must adhere to local regulations when using Triple DES.

The data type LPWFSXDATA is used to pass hexadecimal data and is defined as follows:
typedef struct _wfs_hex_data
{
    USHORT     usLength;
    LPBYTE     lpbData;
} WFSXDATA, *LPWFSXDATA;

usLength
Length of the byte stream pointed to by lpbData.

lpbData
Pointer to the binary data stream.
5.1.2 WFS_CMD_PIN_IMPORT_KEY

Description

The 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 lpValue 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 lpValue 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 was used to encrypt (in ECB mode) 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. It is a handle to the encryption module and is returned to the application in the WFS_CMD_PIN_INITIALIZATION command. See fwIDKey in WFS_INF_PIN_CAPABILITIES for whether this value is required. If not required lpxIdent should be NULL. 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USECRYPT</td>
<td>Key can be used for encryption/decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USEFUNCTION</td>
<td>Key can be used for PIN functions (PIN block creation and local PIN check).</td>
</tr>
<tr>
<td>WFS_PIN_USEMACING</td>
<td>Key can be used for MACing.</td>
</tr>
<tr>
<td>WFS_PIN_USEKEYENCKEY</td>
<td>Key is used as key encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USENODUPLICATE</td>
<td>Key can be imported only once.</td>
</tr>
<tr>
<td>WFS_PIN_USESVENCKEY</td>
<td>Key is used as CBC Start Value encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USECONSTRUCT</td>
<td>Key is under construction through the import of multiple parts. This value is used in combination with the actual usage flags for the key.</td>
</tr>
</tbody>
</table>
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.

Key can be used for importing keys packaged within an ANS TR-31 key block. This key usage can only be combined with WFS_PIN_USECONSTRUCT and WFS_PIN_USESECURECONSTRUCT.

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

**Output Param**

LPWSFXDATA 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key encryption key was not found or attempting to delete a non-existent key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDID</td>
<td>The ID passed was not valid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_DUPLICATEKEY</td>
<td>A key exists with that name and cannot be overwritten.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNVALUE</td>
<td>The specified key encryption key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpxValue is not supported or the encryption key in the secure key buffer is invalid (or has not been entered) or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOKEYRAM</td>
<td>There is no space left in the key RAM for a key of the specified type.</td>
</tr>
</tbody>
</table>

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**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 usages 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.
5.1.3 WFS_CMD_PIN_DERIVE_KEY

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

Input Param
LPWFSPINDERIVE lpDerive;
ytypedef struct _wfs_pin_derive
{
WORD wDerivationAlgorithm;
LPSTR lpsKey;
LPSTR lpsKeyGenKey;
LPSTR lpsStartValueKey;
LPWFSXDATA lpxStartValue;
BYTE bPadding;
LPWFSXDATA lpxInputData;
LPWFSXDATA lpxIdent;
} WFSPINDERIVE, *LPWFSPINDERIVE;

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

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

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

lpsStartValueKey
Specifies the name of the stored key used to decrypt the lpxStartValue to obtain the Initialization
Vector. If this parameter is NULL, lpxStartValue is used as the Initialization Vector.

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

bPadding
Specifies the padding character for the encryption step within the derivation. The valid range is
0x00 to 0xFF.

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

lpxIdent
Specifies the key owner identification. It is a handle to the encryption module and is returned to
the application in the WFS_CMD_PIN_INITIALIZATION command. See fwIDKey in
WFS_INF_PIN_CAPABILITIES for whether this value is required. If not required lpxIdent
should be NULL. The use of this parameter is vendor dependent.

Output Param
None.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
</tbody>
</table>
| WFS_ERR_PIN_ACCESSDENIED| The encryption module is either not initialized (or not ready for some vendor
  specific reason).                                                       |
| WFS_ERR_PIN_INVALIDID   | The ID passed was not valid.                                            |
| WFS_ERR_PIN_DUPLICATEKEY| A key exists with that name and cannot be overwritten.                  |
| WFS_ERR_PIN_KEYNOVALUE  | The specified key is not loaded.                                        |
| WFS_ERR_PIN_USEVIOLATION| The specified use is not supported by this key.                         |
WFS_ERR_PIN_INVALIDKEYLENGTH  The length of lpxStartValue is not supported or the length of an encryption key is not compatible with the encryption operation required.

WFS_ERR_PIN_ALGORITHMNOTSUPP  The specified algorithm is not supported.

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**

None.
5.1.4 WFS_CMD_PIN_GET_PIN

Description

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

The WFS_EXEE_PIN_ENTERDATA event will be generated when the PIN pad is ready for the user to start entering data.

Some PIN pad devices do not inform the application as each PIN digit is entered, but locally process the PIN entry based upon minimum PIN length and maximum PIN length input parameters.

When the maximum number of PIN digits is entered and the flag bAutoEnd is true, or a terminating key is pressed after the minimum number of PIN digits is entered, the command completes. If the <Cancel> key is a terminator key and is pressed, then the command will complete successfully even if the minimum number of PIN digits has not been entered.

Terminating FDKs can have the functionality of <Enter> (terminates only if minimum length has been reached) or <Cancel> (can terminate before minimum length is reached). The configuration of this functionality is vendor specific.

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

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

Terminating keys have to be active keys to operate.

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

If usMaxLen has been met and bAutoEnd is set to False, then all numeric keys will automatically be disabled. If the CLEAR or BACKSPACE key is pressed to reduce the number of entered keys, the numeric keys will be re-enabled.

If the ENTER key (or FDK representing the ENTER key – note that the association of an FDK to ENTER functionality is vendor specific) is pressed prior to usMinLen being met, then the ENTER key or FDK is ignored. In some cases the PIN pad device cannot ignore the ENTER key then the command will complete normally. To handle these types of devices the application should use the output parameter usDigits field to check that sufficient digits have been entered. The application should then get the user to re-enter their PIN with the correct number of digits.

If the application makes a call to WFS_CMD_PIN_GET_PINBLOCK or a local verification command without the minimum PIN digits having been entered, either the command will fail or the PIN verification will fail.

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

Input Param

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

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

usMaxLen
Specifies the maximum number of digits which can be entered for the PIN. A value of zero
indicates no maximum PIN length verification.

bAutoEnd
If bAutoEnd is set to true, the Service Provider terminates the command when the maximum
number of digits are entered. Otherwise, the input is terminated by the user using one of the
termination keys. bAutoEnd is ignored when usMaxLen is set to zero.

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

ulActiveFDKs
Specifies a mask of those FDKs which are active during the execution of the command (see
WFS_INF_PIN_FUNCKEY_DETAIL).

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

ulTerminateFDKs
Specifies a mask of those FDKs which must terminate the execution of the command (see
WFS_INF_PIN_FUNCKEY_DETAIL).

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

Output Param
LPWFSPINENTRY lpEntry;

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

usDigits
Specifies the number of PIN digits entered.

wCompletion
Specifies the reason for completion of the entry. Unless otherwise specified the following values
must not be used in the execute event WFS_EXEE_PIN_KEY or in the array of keys in the
completion of WFS_PIN_CMD_GET_DATA. Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_COMPAUTO</td>
<td>The command terminated automatically, because maximum length was reached.</td>
</tr>
<tr>
<td>WFS_PIN_COMPENTER</td>
<td>The ENTER Function Key was pressed as terminating key.</td>
</tr>
<tr>
<td>WFS_PIN_COMPCANCEL</td>
<td>The CANCEL Function Key was pressed as terminating key.</td>
</tr>
</tbody>
</table>
**WFS_PIN_COMPCONTINUE**  
A function key was pressed and input may continue unless the command completes (this value is only used in the execute event WFS_EXEE_PIN_KEY and in the array of keys in the completion of WFS_PIN_CMD_GET_DATA).

**WFS_PIN_COMPCLEAR**  
The CLEAR Function Key was pressed as terminating key and the previous input is cleared.

**WFS_PIN_COMPBACKSPACE**  
The last input digit was cleared and the key was pressed as terminating key.

**WFS_PIN_COMPFDK**  
Indicates input is terminated only if the FDK pressed was set to be a terminating FDK.

**WFS_PIN_CMPHELP**  
The HELP Function Key was pressed as terminating key.

**WFS_PIN_COMPFK**  
A Function Key (FK) other than ENTER, CLEAR, CANCEL, BACKSPACE, HELP was pressed as terminating key.

**WFS_PIN_COMPCONTFDK**  
An FDK was pressed and input may continue unless the command completes (this value is only used in the execute event WFS_EXEE_PIN_KEY and in the array of keys in the completion of WFS_PIN_CMD_GET_DATA).

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYINVALID</td>
<td>At least one of the specified function keys or FDKs is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOTSUPPORTED</td>
<td>At least one of the specified function keys or FDKs is not supported by the Service Provider.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOACTIVEKEYS</td>
<td>There are no active function keys specified.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOTERMINATEKEYS</td>
<td>There are no terminate keys specified and usMaxLen is not set to zero and bAutoEnd is FALSE.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_MINIMUMLENGTH</td>
<td>The minimum PIN length field is invalid or greater than the maximum PIN length field when the maximum PIN length is not zero.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_EXEE_PIN_KEY</td>
<td>A key has been pressed at the PIN pad.</td>
</tr>
<tr>
<td>WFS_EXEE_PIN_ENTERDATA</td>
<td>The PIN pad is ready for the user to start entering data.</td>
</tr>
</tbody>
</table>

### Comments
None.
5.1.5 WFS_CMD_PIN_LOCAL_DES

Description
The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the DES validation algorithm and locally verified for correctness. The result of the verification is returned to the application. This command will clear the PIN unless the application has requested that the PIN be maintained through the WFS_CMD_PIN_MAINTAIN_PIN command.

Input Param
LPWFSPINLOCALDES lpLocalDES;

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

lpsValidationData
Customer specific data (normally obtained from card track data) used to validate the correctness of the PIN. The validation data should be an ASCII string.

lpsOffset
ASCII string defining the offset data for the PIN block as an ASCII string; if NULL then no offset is used. The character must be in the ranges ‘0’ to ‘9’, ‘a’ to ‘f’ and ‘A’ to ‘F’.

bPadding
Specifies the padding character for the validation data. If the validation data is less than 16 characters long then it will be padded with this character. If bPadding is in the range 0x00 to 0x0F, padding is applied after the validation data has been compressed. If the bPadding character is in the range ‘0’ to ‘9’, ‘a’ to ‘f’, or ‘A’ to ‘F’, padding is applied before the validation data is compressed.

usMaxPIN
Maximum number of PIN digits to be used for validation. This parameter corresponds to PINMINL in the IBM 3624 specification.

usValDigits
Number of Validation digits from the validation data to be used for validation. This is the length of the lpsValidationData string.

bNoLeadingZero
If set to TRUE and the first digit of result of the modulo 10 addition is a 0x0, it is replaced with 0x1 before performing the verification against the entered PIN. If set to FALSE, a leading zero is allowed in entered PINs.

lpsKey
Name of the key to be used for validation.

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

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

Output Param
LPBOOL lpbResult;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOPIN</td>
<td>PIN has not been entered or has been cleared.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpxKeyEncKey is not supported or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
</tbody>
</table>

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**

The PINMAXL value as defined in the IBM 3624 specification is the length of the PIN entered during the WFS_CMD_PIN_GET_PIN command.
5.1.6 WFS_CMD_PIN_CREATE_OFFSET

Description
This function is used to generate a PIN Offset that is typically written to a card and later used to verify the PIN with the WFS_CMD_PIN_LOCAL_DES command. The PIN offset is computed by combining validation data with the keypad entered PIN. This command will clear the PIN unless the application has requested that the PIN be maintained through the WFS_CMD_PIN_MAINTAIN_PIN command.

Input Param
LPWFSPINCREATEOFFSET lpPINOffset;

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

lpsValidationData
Validation data. The validation data should be an ASCII string.

bPadding
Specifies the padding character for validation data. If bPadding is in the range 0x00 to 0x0F, padding is applied after the validation data has been compressed. If the bPadding character is in the range ‘0’ to ‘9’, ‘a’ to ‘f’, or ‘A’ to ‘F’, padding is applied before the validation data is compressed.

usMaxPIN
Maximum number of PIN digits to be used for PIN Offset creation. This parameter corresponds to PINMINL in the IBM 3624 specification.

usValDigits
Number of Validation Data digits to be used for PIN Offset creation. This is the length of the lpsValidationData string.

lpsKey
Name of the validation key.

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

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

Output Param
LPSTR lpsOffset;

lpsOffset
Computed PIN Offset.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOPIN</td>
<td>PIN has not been entered or has been cleared.</td>
</tr>
</tbody>
</table>
Events

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

Comments

The list of ‘forbidden’ PINs (values that cannot be chosen as a PIN, e.g. 1111) is configured in the device in a vendor dependent way during the configuration of the system. The PINMAXL value as defined in the IBM 3624 specification is the length of the PIN entered during the WFS_CMD_PIN_GET_PIN command.
5.1.7  WFS_CMD_PIN_LOCAL_EUROCHEQUE

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

Input Param
LPWFSPINLOCALEUROCHEQUE lpLocalEurocheque;

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

lpsEurochequeData
Track-3 Eurocheque data.

lpsPVV
PIN Validation Value from track data.

wFirstEncDigits
Number of digits to extract after first encryption.

wFirstEncOffset
Offset of digits to extract after first encryption.

wPVVDigits
Number of digits to extract for PVV.

wPVVOffset
Offset of digits to extract for PVV.

lpsKey
Name of the validation key.

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

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

Output Param
LPBOOL lpbResult;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
</tbody>
</table>
WFS_ERR_PIN_NOPIN

PIN has not been entered or has been cleared.

WFS_ERR_PIN_INVALIDKEYLENGTH

The length of lpxKeyEncKey is not supported or the length of an encryption key is not compatible with the encryption operation required.

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**

None.
5.1.8 WFS_CMD_PIN_LOCAL_VISA

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

Input Param
LPWFSPINLOCALVISA lpLocalVISA;

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

lpsPAN
Primary Account Number from track data, as an ASCII string. lpsPAN should contain the eleven rightmost digits of the PAN (excluding the check digit), followed by the PVKI indicator in the 12th byte.

lpsPVV
PIN Validation Value from track data, as an ASCII string with characters in the range ‘0’ to ‘9’. This string should contain 4 digits.

wPVVDigits
Number of digits of PVV.

lpsKey
Name of the validation key.

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

Output Param
LPBOOL lpbResult;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOPIN</td>
<td>PIN has not been entered or has been cleared.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpxKeyEncKey is not supported or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>
Comments

None.
5.1.9 WFS_CMD_PIN_PRESENT_IDC

Description
The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the IDC presentation algorithm and presented to the smartcard contained in the ID card unit. The result of the presentation is returned to the application. This command will clear the PIN unless the application has requested that the PIN be maintained through the WFS_CMD_PIN_MAINTAIN_PIN command.

Input Param
LPWFSPINPRESENTIDC lpPresentIDC;

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

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

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

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

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

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

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

ulPINPointer
The byte offset where to start inserting the PIN into lpbChipData. The leftmost byte is numbered zero. See below for an example.

usPINOffset
The bit offset within the byte specified by ulPINPointer where to start inserting the PIN. The leftmost bit numbered zero. See below for an example.

Output Param
LPWFSPINPRESENTRESULT lpPresentResult;

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

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

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

lpbChipData
Points to the data responded from the chip.

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

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

**Events**

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

**Comments**

Example for the use of the algorithm WFS_PIN_PRESENT_CLEAR:

The structure of a VERIFY command for a French B0 chip is:

<table>
<thead>
<tr>
<th>Bytes 0 to 4</th>
<th>Bytes 5 to 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>PIN block</td>
</tr>
<tr>
<td>0xBC</td>
<td>0x20</td>
</tr>
<tr>
<td>A1</td>
<td>0x00</td>
</tr>
<tr>
<td>A2</td>
<td>0x00</td>
</tr>
<tr>
<td>Lc</td>
<td>0x04</td>
</tr>
<tr>
<td>0xXX</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>0xXX</td>
<td>0xXX</td>
</tr>
<tr>
<td>0xXX</td>
<td></td>
</tr>
</tbody>
</table>

Where the 4 byte PIN block consists of 2 bits that are always zero, 16 bits for the 4 PIN digits (each digit being coded in 4 bits) and 14 bits that are always one:

<table>
<thead>
<tr>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

In order to insert the PIN into such a command, the application calls WFS_CDM_PIN_PRESENT_IDC with:

- ulChipDataLength: 9
- lpChipData: 0xBC2000000400003FFF
- ulPINPointer: 5
- usPINOffset: 2

For a sample PIN “1234” the PIN block is:

<table>
<thead>
<tr>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Resulting in a chip card command of:

<table>
<thead>
<tr>
<th>Bytes 0 to 4</th>
<th>Bytes 5 to 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>PIN block</td>
</tr>
<tr>
<td>0xBC</td>
<td>0x20</td>
</tr>
<tr>
<td>A1</td>
<td>0x00</td>
</tr>
<tr>
<td>A2</td>
<td>0x00</td>
</tr>
<tr>
<td>Lc</td>
<td>0x04</td>
</tr>
<tr>
<td>0x04</td>
<td>0x8D</td>
</tr>
<tr>
<td>0x3F</td>
<td>0xFF</td>
</tr>
</tbody>
</table>
5.1.10 WFS_CMD_PIN_GET_PINBLOCK

Description
This function takes the account information and a PIN entered by the user to build a formatted PIN. Encrypting this formatted PIN once or twice returns a PIN block which can be written on a magnetic card or sent to a host. The PIN block can be calculated using one of the formats specified in the WFS_INF_PIN_CAPABILITIES command. This command will clear the PIN unless the application has requested that the PIN be maintained through the WFS_CMD_PIN_MAINTAIN_PIN command.

Input Param

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

lpsCustomerData
The customer data should be an ASCII string. Used for ANSI, ISO-0 and ISO-1 algorithm to build the formatted PIN. For ANSI and ISO-0 the PAN (Primary Account Number, without the check number) is supplied, for ISO-1 a ten digit transaction field is required. If not used a NULL is required.

Used for DIEBOLD with coordination number, as a two digit coordination number.

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

For AP PIN blocks, the data must be a concatenation of the PAN (18 digits including the check digit), and the CCS (8 digits).

lpsXORData
If the formatted PIN is encrypted twice to build the resulting PIN block, this data can be used to modify the result of the first encryption by an XOR-operation. This parameter is a string of hexadecimal data that must be converted by the application, e.g. 0123456789ABCDEF must be converted to 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45 0x46 and terminated with 0x00. In other words the application would set lpsXORData to “0123456789ABCDEF0”. The hex digits 0A to 0F can be represented by characters in the ranges ‘a’ to ‘f’ or ‘A’ to ‘F’. If this value is NULL no XOR-operation will be performed. If the formatted PIN is not encrypted twice (i.e. if lpsKeyEncKey is NULL) this parameter is ignored.

bPadding
Specifies the padding character. The valid range is 0x00 to 0x0F. Only the least significant nibble is used.

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

lpsKey
Specifies the key used to encrypt the formatted PIN for the first time, NULL if no encryption is required. If this specifies a double length key, triple DES encryption will be performed. The key referenced by lpsKey must have the WFS_PIN_USEFUNCTION attribute. If this specifies an RSA key, RSA encryption will be performed.

lpsKeyEncKey
Specifies the key used to format the once encrypted formatted PIN, NULL if no second encryption required. The key referenced by lpsKeyEncKey must have the WFS_PIN_USEFUNCTION attribute. If this specifies a double-length key, triple DES encryption will be performed.

Output Param

LPWFSXDATA lpxPinBlock;
**Error Codes**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOPIN</td>
<td>The PIN has not been entered was not long enough or has been cleared.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_FORMATNOTSUPP</td>
<td>The specified format is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpsKeyEncKey or lpsKey is not supported by this key or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
</tbody>
</table>

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**

None.
5.1.11 WFS_CMD_PIN_GET_DATA

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

The WFS_EXEE_PIN_ENTERDATA event will be generated when the PIN pad is ready for the user to start entering data.

When the maximum number of digits is entered and the flag bAutoEnd is true, or a terminate key is pressed after the minimum number of digits is entered, the command completes. If the <Cancel> key is a terminator key and is pressed, the command will complete successfully even if the minimum number of digits has not been entered.

Terminating FDKs can have the functionality of <Enter> (terminates only if minimum length has been reached) or <Cancel> (can terminate before minimum length is reached). The configuration of this functionality is vendor specific.

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

If usMaxLen has been met and bAutoEnd is set to False, then all keys or FDKs that add data to the contents of the WFSPINDATA output parameter will automatically be disabled. If the CLEAR or BACKSPACE key is pressed to reduce the number of entered keys below usMaxLen, the same keys will be re-enabled.

Where applications want direct control of the data entry and the key interpretation, usMaxLen can be set to zero allowing the application to provide tracking and counting of key presses until a terminate key or terminate FDK is pressed or WFSCancel has been issued.

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

WFS_PIN_FK_ENTER
WFS_PIN_FKCANCEL
WFS_PIN_FK_CLEAR
WFS_PIN_FK_BACKSPACE

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

Terminating keys have to be active keys to operate.

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

Input Param
LPWFSPINGETDATA lpPinGetData;

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

usMaxLen
Specifies the maximum number of digits which can be returned to the application in the output parameter.
If `bAutoEnd` is set to true, the Service Provider terminates the command when the maximum number of digits are entered. Otherwise, the input is terminated by the user using one of the termination keys. `bAutoEnd` is ignored when `usMaxLen` is set to zero.

### ulActiveFDKs
Specifies a mask of those FDKs which are active during the execution of the command (see `WFS_INF_PIN_FUNCKEY_DETAIL`).

### ulActiveKeys
Specifies a mask of those (other) Function Keys which are active during the execution of the command (see `WFS_INF_PIN_FUNCKEY_DETAIL`).

### ulTerminateFDKs
Specifies a mask of those FDKs which must terminate the execution of the command (see `WFS_INF_PIN_FUNCKEY_DETAIL`).

### ulTerminateKeys
Specifies a mask of those (other) Function Keys which must terminate the execution of the command (see `WFS_INF_PIN_FUNCKEY_DETAIL`).

### Output Param
```c
LPWFSPINDATA lpPinData;

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

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

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYINVALID</td>
<td>At least one of the specified function keys or FDKs is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOTSUPPORTED</td>
<td>At least one of the specified function keys or FDKs is not supported by the Service Provider.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOACTIVEKEYS</td>
<td>There are no active keys specified.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_EXEE_PIN_KEY</td>
<td>A key has been pressed at the PIN pad.</td>
</tr>
<tr>
<td>WFS_EXEE_PIN_ENTERDATA</td>
<td>The PIN pad is ready for the user to start entering data.</td>
</tr>
</tbody>
</table>

Comments
If the triple zero key is pressed one `WFS_EXEE_PIN_KEY` event is sent that contains the `WFS_PIN_FK_000` code and three `WFS_PIN_FK_0` elements are added to the output buffer.

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

If the backspace key is pressed after the triple zero key only one zero is deleted out of the output buffer.
If the double zero key is pressed one WFS_EXEE_PIN_KEY event is sent that contains the WFS_PIN_FK_00 code and two WFS_PIN_FK_0 elements are added to the output buffer.

If the double zero key is pressed when 3 keys are already inserted and \textit{usMaxLen} equals 4 the key is not accepted and no event is sent to the application.

If the backspace key is pressed after the double zero key only one zero is deleted out of the output buffer.
### 5.1.12 WFS_CMD_PIN_INITIALIZATION

**Description**

The encryption module must be initialized before any encryption function can be used. Every call to WFS_CMD_PIN_INITIALIZATION destroys all application keys that have been loaded or imported; it does not affect those keys loaded during manufacturing 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 cannot 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_ACCESSDENIED 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_PIN_REPLACE_CERTIFICATE commands must be called again.

When multiple ZKA HSMs are present, this command deletes all keys loaded within all ZKA logical HSMs.

**Input Param**

LPWFSPININIT lpInit;

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

*lpIdent*  
Pointer to the value of the ID key. NULL if not required.

*lpKey*  
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. This value can be used as authorization for the WFS_CMD_PIN_IMPORT_KEY command, but 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:
### Events

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_INITIALIZED</td>
<td>The encryption module is now initialized.</td>
</tr>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

### Comments

None.
5.1.13 WFS_CMD_PIN_LOCAL_BANKSYS

Description
The PIN block previously built by the WFS_CMD_PIN_GET_PINBLOCK command is sent to
the BANKSYS security control module using the WFS_CMD_PIN_BANKSYS_IO command.
The BANKSYS security control module will return an ATMVAC code, which is then used in this
command to locally validate the PIN. The key referenced by lpsKey within the most recent
successful WFS_CMD_PIN_GET_PINBLOCK command is reused by the
WFS_CMD_PIN_LOCAL_BANKSYS command for the local validation.

Input Param
LPWFSPINLOCALBANKSYS lpLocalBanksys;

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

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

Output Param
LPBOOL lpbResult;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOPIN</td>
<td>PIN has not been entered or has been cleared without building the Banksys PIN block.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpxATMVAC is not supported or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

Comments
None.
5.1.14 WFS_CMD_PIN_BANKSYS_IO

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

Input Param
LPWFSPINBANKSYSIO lpBanksysIoIn;

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

ulLength
Specifies the length of the following field lpbData.

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

Output Param
LPWFSPINBANKSYSIO lpBanksysIoOut;

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

ulLength
Specifies the length of the following field lpbData.

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_INVALIDDATA</td>
<td>An error occurred while communicating with the device.</td>
</tr>
</tbody>
</table>

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

Comments
The Banksys command and response message data are defined in [Ref. 18].
### 5.1.15 WFS_CMD_PIN_RESET

<table>
<thead>
<tr>
<th>Description</th>
<th>Sends a service reset to the Service Provider.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Param</strong></td>
<td>None.</td>
</tr>
<tr>
<td><strong>Output Param</strong></td>
<td>None.</td>
</tr>
<tr>
<td><strong>Error Codes</strong></td>
<td>Only the generic error codes defined in [Ref. 1] can be generated by this command.</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>Only the generic events defined in [Ref. 1] can be generated by this command.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>This command is used by an application control program to cause a device to reset itself to a known good condition. It does not delete any keys.</td>
</tr>
</tbody>
</table>
5.1.16 WFS_CMD_PIN_HSM_SET_TDATA

Description
This function allows the application to set the HSM terminal data (except keys, trace number and session key index). The data must be provided as a series of “tag/length/value” items.

Terminal data that are set but are not supported by the hardware will be ignored.

Input Param
LPWFSXDATA lpxTData;

𝑙𝑝𝑥𝑇𝐷𝑎𝑡𝑎
Specifies which parameter(s) is(are) to be set. 𝑙𝑝𝑥𝑇𝐷𝑎𝑡𝑎 is a series of “tag/length/value” items where each item consists of:

• One byte tag (see the list of tags below).
• One byte specifying the length of the following data as an unsigned binary number.
• N bytes data (see the list below for formatting) with no separators.

The following tags are supported:

<table>
<thead>
<tr>
<th>Tag (hexl)</th>
<th>Format</th>
<th>Length (bytes)</th>
<th>Meaning</th>
<th>Read / Write</th>
<th>EPP / HSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>BCD</td>
<td>4</td>
<td>Terminal ID</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO BMP 41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>BCD</td>
<td>4</td>
<td>Bank code</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO BMP 42 (rightmost 4 bytes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>BCD</td>
<td>9</td>
<td>Account data for terminal account</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO BMP 60 (load against other card)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>BCD</td>
<td>9</td>
<td>Account data for fee account</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO BMP 60 (&quot;Laden vom Kartenkonto&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>EBCDIC</td>
<td>40</td>
<td>Terminal location</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO BMP 43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C7</td>
<td>ASCII</td>
<td>3</td>
<td>Terminal currency</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(YYYYMMDDHHMMSS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>BCD</td>
<td>7</td>
<td>Online date and time</td>
<td>R/W</td>
<td>HSM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ISO BMP 61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C9</td>
<td>BCD</td>
<td>4</td>
<td>Minimum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td>CA</td>
<td>BCD</td>
<td>4</td>
<td>Maximum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42</td>
<td>R/W</td>
<td>EPP</td>
</tr>
<tr>
<td>CB</td>
<td>BIN</td>
<td>3</td>
<td>logical HSM binary coded serial number (starts with 1; 0 means that there are no logical HSMs)</td>
<td>R</td>
<td>HSM</td>
</tr>
<tr>
<td>CC</td>
<td>EBCDIC</td>
<td>16</td>
<td>ZKA ID (is filled during the pre-initialization of the HSM)</td>
<td>R</td>
<td>HSM</td>
</tr>
<tr>
<td>CD</td>
<td>BIN</td>
<td>1</td>
<td>HSM status</td>
<td>R</td>
<td>HSM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = irreversibly out of order</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 = out of order, K_UR is not loaded</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 = not pre-initialized, K_UR is loaded</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 = pre-initialized, K_INIT is loaded</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 = initialized/personalized, K_PERS is loaded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>EBCDIC</td>
<td>variable, min. 16</td>
<td>HSM-ID (6 byte Manufacturer- ID + min. 10 Byte serial number), as needed for ISO BMP57 of a pre-initialization</td>
<td>R</td>
<td>EPP</td>
</tr>
</tbody>
</table>

In the table above, the fifth column indicates if the variable is read only or both read and write.
The sixth column indicates if the variable is unique per logical HSM or common across all logical HSMs within an EPP.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_HSMSTATEINVALID</td>
<td>The HSM is not in a correct state to handle this command.</td>
</tr>
</tbody>
</table>

## Events

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_HSM_TDATA_CHANGED</td>
<td>The terminal data has changed.</td>
</tr>
</tbody>
</table>

## Comments

None.
5.1.17 WFS_CMD_PIN_SECURE_MSG_SEND

Description
This command handles all messages that should be sent through a secure messaging to an 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_PROTISOAS</td>
<td>ISO 8583 protocol for the authorization system.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOLZ</td>
<td>ISO 8583 protocol for the German “Ladezentrale”.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOPS</td>
<td>ISO 8583 protocol for the personalization system.</td>
</tr>
<tr>
<td>WFS_PIN_PROTCHIPZKA</td>
<td>ZKA chip protocol.</td>
</tr>
<tr>
<td>WFS_PIN_PROTRAWDATA</td>
<td>Raw data protocol.</td>
</tr>
<tr>
<td>WFS_PIN_PROTPBM</td>
<td>PBM protocol (see [Ref. 8] –[Ref. 13])</td>
</tr>
<tr>
<td>WFS_PIN_PROThsmdl</td>
<td>HSM LDI protocol.</td>
</tr>
<tr>
<td>WFS_PIN_PROTGENAS</td>
<td>Generic PAC/MAC for non-ISO8583 message formats.</td>
</tr>
<tr>
<td>WFS_PIN_PROTCHIPINCHG</td>
<td>ZKA chip protocol for changing the PIN on a GeldKarte.</td>
</tr>
<tr>
<td>WFS_PIN_PROTPINCHG</td>
<td>Protocol for comparing PIN numbers entered in the PIN pad during a PIN Change transaction.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOPINCHG</td>
<td>ISO8583 authorization system protocol for changing the PIN on a GeldKarte.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_HSMSTATEINVALID</td>
<td>The HSM is not in a correct state to handle this message.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_PROTINVALID</td>
<td>The specified protocol is invalid.</td>
</tr>
</tbody>
</table>
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.
5.1.18 WFS_CMD_PIN_SECURE_MSG_RECEIVE

Description
This command handles all messages that are received through a secure messaging from an 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_PROTISOAS</td>
<td>ISO 8583 protocol for the authorization system.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOLZ</td>
<td>ISO 8583 protocol for the German &quot;Ladezentrale&quot;.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOPS</td>
<td>ISO 8583 protocol for the personalization system.</td>
</tr>
<tr>
<td>WFS_PIN_PROTCHIPZKA</td>
<td>ZKA chip protocol.</td>
</tr>
<tr>
<td>WFS_PIN_PROTRAWDATA</td>
<td>Raw data protocol.</td>
</tr>
<tr>
<td>WFS_PIN_PROTPBM</td>
<td>PBM protocol (see [Ref. 8] – [Ref. 13]).</td>
</tr>
<tr>
<td>WFS_PIN_PROTGENAS</td>
<td>Generic PAC/MAC for non-ISO8583 message formats.</td>
</tr>
<tr>
<td>WFS_PIN_PROTCHIPINCHG</td>
<td>ZKA chip protocol for changing the PIN on a GeldKarte.</td>
</tr>
<tr>
<td>WFS_PIN_PROTPINCMP</td>
<td>Protocol for comparing PIN numbers entered in the PIN pad during a PIN Change transaction.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOPINCHG</td>
<td>ISO8583 authorization system protocol for changing the PIN on a GeldKarte.</td>
</tr>
</tbody>
</table>

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

lpbMsg
Specifies the message that was received. This value 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_HSMSTATEINVALID</td>
<td>The HSM is not in a correct state to handle this message.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_MACINVALID</td>
<td>The MAC of the message is not correct.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_PROTINVALID</td>
<td>The specified protocol is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_FORMATINVALID</td>
<td>The format of the message is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_CONTENTINVALID</td>
<td>The contents of one of the security relevant fields are invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>No key was found for MAC verification.</td>
</tr>
</tbody>
</table>
### Events
In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_HSM_TDATA_CHANGED</td>
<td>The terminal data has changed.</td>
</tr>
</tbody>
</table>

### Comments
None.
5.1.19 WFS_CMD_PIN_GET_JOURNAL

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

Input Param
LPWORD lpwProtocol;

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_PROTISOAS</td>
<td>Get authorization system journal data.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOLZ</td>
<td>Get German &quot;Ladezentrale&quot; journal data.</td>
</tr>
<tr>
<td>WFS_PIN_PROTISOPS</td>
<td>Get personalization system journal data.</td>
</tr>
<tr>
<td>WFS_PIN_PROTPBM</td>
<td>Get PBM protocol data.</td>
</tr>
</tbody>
</table>

Output Param
LPWFSXDATA lpxJournalData;

lpxJournalData
Pointer to the journal data.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_HSMSTATEINVALID</td>
<td>The HSM is not in a correct state to return journal data.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_PROTINVALID</td>
<td>The specified protocol is invalid.</td>
</tr>
</tbody>
</table>

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

Comments
None.
5.1.20 WFS_CMD_PIN_IMPORT_KEY_EX

Description
The 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 \textit{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 \texttt{WFS\_PIN\_USECONSTRUCT} or \texttt{WFS\_PIN\_USESECURECONSTRUCT} value with the final usage flag within the \textit{dwUse} field.

If the \texttt{WFS\_PIN\_USECONSTRUCT} flag is used then the application must provide the key data through the \textit{lpxValue} parameter, If \texttt{WFS\_PIN\_USESECURECONSTRUCT} is used then the encryption key part in the secure key buffer previously populated with the \texttt{WFS\_CMD\_PIN\_SECUREKEY\_ENTRY} command is used and \textit{lpxValue} is ignored. Key parts loaded with the \texttt{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 \texttt{WFS\_PIN\_USECONSTRUCT} and \texttt{WFS\_PIN\_USESECURECONSTRUCT} construction flags cannot be used in combination.

Input Param
\begin{verbatim}
LPWFSPINIMPORTKEYEX lpImportKeyEx;
typedef struct _wfs_pin_import_key_ex
{    LPSTR     lpsKey;
    LPSTR     lpsEncKey;
    LPWFSXDATA lpxValue;
    LPWFSXDATA lpxControlVector;
    DWORD     dwUse;
    WORD      wKeyCheckMode;
    LPWFSXDATA lpxKeyCheckValue;
} WFS_PIN_IMPORTKEYEX, *LPWFS_PIN_IMPORTKEYEX;
\end{verbatim}

\texttt{lpsKey}
Specifies the name of key being loaded.

\texttt{lpsEncKey}
\texttt{lpsEncKey} specifies a key name which was used to encrypt (in ECB mode) the key string passed in \texttt{lpxValue}. If \texttt{lpsEncKey} is NULL the key is loaded directly into the encryption module. \texttt{lpsEncKey} must be NULL if \texttt{dwUse} contains \texttt{WFS\_PIN\_USECONSTRUCT} or \texttt{WFS\_PIN\_USESECURECONSTRUCT}.

\texttt{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.

\texttt{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 \texttt{dwUse}. See also [Ref. 26].

\texttt{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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{WFS_PIN_USECRYPT}</td>
<td>Key is used for encryption and decryption.</td>
</tr>
<tr>
<td>\texttt{WFS_PIN_USEFUNCTION}</td>
<td>Key is used for PIN block creation.</td>
</tr>
<tr>
<td>\texttt{WFS_PIN_USEMACING}</td>
<td>Key is used for MACing.</td>
</tr>
<tr>
<td>\texttt{WFS_PIN_USEKEYENCEY}</td>
<td>Key is used as key encryption key.</td>
</tr>
<tr>
<td>\texttt{WFS_PIN_USERPINLOCAL}</td>
<td>Key is used for local PIN check.</td>
</tr>
<tr>
<td>\texttt{WFS_PIN_USERSAPUBLIC}</td>
<td>Key is used as a public key for RSA encryption including EMV PIN block creation.</td>
</tr>
<tr>
<td>\texttt{WFS_PIN_USERSAPRIVATE}</td>
<td>Key is used as a private key for RSA decryption (it is not recommend that private keys are imported with this function).</td>
</tr>
</tbody>
</table>
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.

WFS_PIN_USEANSTR31MASTER Key can be used for importing keys packaged within an ANS TR-31 key block. This key usage can only be combined with WFS_PIN_USECONSTRUCT and WFS_PIN_USESECURECONSTRUCT.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_KCVNONE</td>
<td>There is no key check value verification required.</td>
</tr>
<tr>
<td>WFS_PIN_KCVSELF</td>
<td>The key check value is created by an encryption of the key with itself. For a double length key the KCV is generated using 3DES encryption using the first half of the key as the source data for the encryption.</td>
</tr>
<tr>
<td>WFS_PIN_KCVZERO</td>
<td>The key check value is created by an encryption of a zero value with the key.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key encryption key was not found or attempting to delete a non-existent key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_DUPLICATEKEY</td>
<td>A key exists with that name and cannot be overwritten.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key encryption key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use conflicts with a previously for the same key specified one.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpxValue is not supported or the encryption key in the secure key buffer is invalid (or has not been entered) or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYINVALID</td>
<td>The key value is invalid. The key check value verification failed.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOKEYRAM</td>
<td>There is no space left in the key RAM for a key of the specified type.</td>
</tr>
</tbody>
</table>

Events In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

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 usages 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 usages 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.
5.1.21 WFS_CMD_PIN_ENC_IO

Description
This command is used to communicate with the encryption module. Transparent data is sent from
the application to the encryption module and the response is returned transparently to the
application.
This command is used to add support for country-specific protocols.

Input Param
LPWFSPINENCIO lpEncIoIn;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_ENC_PROT_CH</td>
<td>For Swiss specific protocols. The document specification for Swiss specific protocols is &quot;CMD_ENC_IO - CH Protocol.doc&quot;. This document is available at the following address:</td>
</tr>
<tr>
<td></td>
<td>EUROPAY (Switzerland) SA</td>
</tr>
<tr>
<td></td>
<td>Terminal Management</td>
</tr>
<tr>
<td></td>
<td>Hertistrasse 27</td>
</tr>
<tr>
<td></td>
<td>CH-8304 Wallisellen</td>
</tr>
<tr>
<td>WFS_PIN_ENC_PROT_GIECB</td>
<td>Protocol for “Groupement des Cartes Bancaires” (France).</td>
</tr>
<tr>
<td>WFS_PIN_ENC_PROT_LUX</td>
<td>Protocol for Luxemburg commands. The reference for this specific protocol is the Authorization Center in Luxemburg (CETREL.)</td>
</tr>
<tr>
<td></td>
<td>Cryptography Management</td>
</tr>
<tr>
<td></td>
<td>Postal address:</td>
</tr>
<tr>
<td></td>
<td>CETREL Société Coopérative</td>
</tr>
<tr>
<td></td>
<td>Centre de Transferts Electroniques</td>
</tr>
<tr>
<td></td>
<td>L-2956 Luxembourg</td>
</tr>
</tbody>
</table>

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

lpvData
Points to a structure containing the data to be sent to the encryption module. This structure
depends on the wProtocol field where each protocol may contain a different structure.

Output Param
LPWFSPINENCIO lpEncIoOut;

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

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

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

lpvData
Points to a structure containing the data responded by the encryption module.
Error Codes
In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_PROTOCOLNOTSUPP</td>
<td>The specified protocol is not supported by the Service Provider. For wProtocol=WFS_PIN_ENC_PROT_GIECB.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_RANDOMINVALID</td>
<td>The encrypted random number in the input data does not decrypt to the one previously provided by the EPP.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_SIGNATUREINVALID</td>
<td>The signature in the input data is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_SNSCDINVALID</td>
<td>The SCD serial number in the input data is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_HSMSTATEINVALID</td>
<td>The HSM is not in a correct state to handle this command.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOKEYRAM</td>
<td>There is no space left in the key RAM for a key of the specified type.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYINVALID</td>
<td>The key value is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEY_GENERATION_ERROR</td>
<td>The EPP is unable to generate a key pair.</td>
</tr>
</tbody>
</table>

Events
None.

Comments
For the WFS_PIN_ENC_PROT_CH and the WFS_PIN_ENC_PROT_LUX protocols, the WFS_CMD_PIN_ENC_IO command only returns generic error codes. Protocol specific error codes will be returned by the hResult within the output data.
5.1.22 WFS_CMD_PIN_HSM_INIT

Description
This command is used to set the HSM out of order. If multiple logical HSMs are configured then
the command sets the currently active logical HSM out of order. At the same time the online time
can be set to control when the OPT online dialog (see WFS_PIN_PROTISOPS protocol) shall be
started to initialize the HSM again. When this time is reached a
WFS_SRVE_PIN_OPT_REQUIRED event will be sent.

Input Param
LPWFSPINHSMINIT lpHsmInit;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_INITTEMP</td>
<td>Initialize the HSM temporarily (K_UR remains loaded).</td>
</tr>
<tr>
<td>WFS_PIN_INITDEFINITE</td>
<td>Initialize the HSM definitely (K_UR is deleted).</td>
</tr>
<tr>
<td>WFS_PIN_INITIRREVERSIBLE</td>
<td>Initialize the HSM irreversibly (can only be restored by the vendor).</td>
</tr>
</tbody>
</table>

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

Output Param
None.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_MODENOTSUPPORTED</td>
<td>The specified init mode is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_HSMSTATEINVALID</td>
<td>The HSM is not in a correct state to handle this command.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_HSM_TDATA_CHANGED</td>
<td>The terminal data has changed.</td>
</tr>
</tbody>
</table>

Comments
None.
5.1.23 WFS_CMD_PIN_SECUREKEY_ENTRY

**Description**

This command allows a full length symmetric encryption key part to be entered directly into the PIN pad without being exposed outside of the PIN pad. 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 PIN pad 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 WFS_EXEE_PIN_ENTERDATA event will be generated when the PIN pad is ready for the user to start entering data.

The keys that can be enabled by this command are defined by the lpFuncKeyDetail 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 encryption 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 ulActiveFDKs 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 PIN pad 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**

`LPWFS_PINSECUREKEYENTRY lpSecureKeyEntry;`

```c
typedef struct __wfs_pin_secure_key_entry {
    USHORT usKeyLen;
    BOOL bAutoEnd;
    ULONG ulActiveFDKs;
    ULONG ulActiveKeys;
    ULONG ulTerminateFDKs;
    ULONG ulTerminateKeys;
    WORD wVerificationType;
} WFS_PINSECUREKEYENTRY, *LPWFS_PINSECUREKEYENTRY;
```
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 lpFuncKeyDetail 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_KCVSELF</td>
<td>The key check value is created by an encryption of the key with itself. For a double length key the KCV is generated using 3DES encryption using the first half of the key as the source data for the encryption.</td>
</tr>
<tr>
<td>WFS_PIN_KCVZERO</td>
<td>The key check value is created by an encryption of a zero value with the key.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYINVALID</td>
<td>At least one of the specified function keys or FDKs is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOTSUPPORTED</td>
<td>At least one of the specified function keys or FDKs is not supported by the Service Provider.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOACTIVEKEYS</td>
<td>There are no active function keys specified.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOTERMINATEKEYS</td>
<td>There are no terminate keys specified and bAutoEnd is FALSE.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The usKeyLen key length is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_MODENOTSUPPORTED</td>
<td>The KCV mode is not supported.</td>
</tr>
</tbody>
</table>

### Events

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| WFS_EXEE_PIN_KEY           | A key has been pressed at the PIN pad.  
Applications must be aware of the association between FDKs and the edit functions reported within the WFS_INF_PIN_SECUREKEY_DETAIL command. |
| WFS_EXEE_PIN_ENTERDATA     | The PIN pad is ready for the user to start entering data.               |

### Comments

None.
5.1.24 WFS_CMD_PIN_GENERATE_KCV

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

**Input Param**

```c
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_KCVSELF</td>
<td>The key check value is created by an encryption of the key with itself. For a double length key the KCV is generated using 3DES encryption using the first half of the key as the source data for the encryption.</td>
</tr>
<tr>
<td>WFS_PIN_KCVZERO</td>
<td>The key check value is created by an encryption of a zero value with this key.</td>
</tr>
</tbody>
</table>

**Output Param**

```c
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key encryption key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key exists but has no value loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_MODENOTSUPPORTED</td>
<td>The KCV mode is not supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**
None.
5.1.25 WFS_CMD_PIN_SET_GUIDANCE_LIGHT

Description
This command is used to set the status of the PIN guidance lights. This includes defining the flash rate and the color. When an application tries to use a color that is not supported then the Service Provider will return the generic error WFS_ERR_UNSUPP_DATA.

Input Param
LPWFSPINSETGUIDLIGHT lpSetGuidLight;

typedef struct _wfs_pin_set_guidlight
{
    WORD wGuidLight;
    DWORD dwCommand;
} WFSPINSETGUIDLIGHT, *LPWFSPINSETGUIDLIGHT;

wGuidLight
Specifies the index of the guidance light to set as one of the values defined within the capabilities section:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_GUIDANCE_OFF</td>
<td>The light indicator is turned off.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_SLOW_FLASH</td>
<td>The light indicator is set to flash slowly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_MEDIUM_FLASH</td>
<td>The light is blinking medium frequency.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_QUICK_FLASH</td>
<td>The light indicator is set to flash quickly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_CONTINUOUS</td>
<td>The light indicator is turned on continuously (steady).</td>
<td>B</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_RED</td>
<td>The light indicator color is set to red.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_GREEN</td>
<td>The light indicator color is set to green.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_YELLOW</td>
<td>The light indicator color is set to yellow.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_BLUE</td>
<td>The light indicator color is set to blue.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_CYAN</td>
<td>The light indicator color is set to cyan.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_MAGENTA</td>
<td>The light indicator color is set to magenta.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_PIN_GUIDANCE_WHITE</td>
<td>The light indicator color is set to white.</td>
<td>C</td>
</tr>
</tbody>
</table>

dwCommand
Specifies the state of the guidance light indicator as WFS_PIN_GUIDANCE_OFF or a combination of the following flags consisting of one type B, and optionally one type C. If no value of type C is specified then the default color is used. The Service Provider determines which color is used as the default color.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_INVALID_PORT</td>
<td>An attempt to set a guidance light to a new value was invalid because the guidance light does not exist.</td>
</tr>
</tbody>
</table>

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

Comments
Guidance light support was added into the PIN primarily to support guidance lights for workstations where more than one instance of a PIN is present. The original SIU guidance light mechanism was not able to manage guidance lights for workstations with multiple PINS. This command can also be used to set the status of the PIN guidance lights when only one instance of a PIN is present.
The slow and medium flash rates must not be greater than 2.0 Hz. It should be noted that in order to comply with American Disabilities Act guidelines only a slow or medium flash rate must be used.
5.1.26 WFS_CMD_PIN_Maintain_PIN

**Description**
This command is used to control if the PIN is maintained after a PIN processing command for subsequent use by other PIN processing commands. This command is also used to clear the PIN buffer when the PIN is no longer required.

**Input Param**
LPWFSPINMAINTAINPIN lpMaintainPinIn;

typedef struct _wfs_pin_maintain_pin
{
    BOOL          bMaintainPIN;
} WFSPINMAINTAINPIN, *LPWFSPINMAINTAINPIN;

*bMaintainPIN*
Specifies if the PIN should be maintained after a PIN processing command. Once set, this setting applies until changed through another call to this command. This value is not persistent across reboots.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>The PIN should be maintained after PIN processing commands for multiple uses.</td>
</tr>
<tr>
<td>FALSE</td>
<td>The PIN will be cleared and subsequent PINs will not be maintained for multiple uses.</td>
</tr>
</tbody>
</table>

**Output Param**
None.

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

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

**Comments**
When using this command to maintain a PIN for multiple transactions/PIN processing commands, applications should ensure that a customer’s PIN is cleared after they have completed all their transactions. The PIN is cleared by calling this command with *bMaintainPIN* set to FALSE.
5.1.27 WFS_CMD_PIN_KEYPRESS_BEEP

Description
This command is used to enable or disable the PIN device from emitting a beep tone on subsequent key presses of active or in-active keys. This command is valid only on devices which have the capability to support application control of automatic beeping. See WFS_INF_PIN_CAPABILITIES structure for information.

Input Param
LPWORD lpwMode;

lpwMode
Specifies whether automatic generation of key press beep tones should be activated for any active or in-active key subsequently pressed on the PIN. lpwMode selectively turns beeping on and off for active, in-active or both types of keys. lpwMode contains a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_BEEP_ON_ACTIVE</td>
<td>Specifies that beeping should be enabled for active keys. If this flag is not present then beeping is disabled for active keys.</td>
</tr>
<tr>
<td>WFS_PIN_BEEP_ON_INACTIVE</td>
<td>Specifies that beeping should be enabled for in-active keys. If this flag is not present then beeping is disabled for in-active keys.</td>
</tr>
</tbody>
</table>

Output Param
None.

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

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

Comments
None.
5.1.28 WFS_CMD_PIN_SET_PINBLOCK_DATA

**Description**

This function should be used for devices which need to know the data for the PIN block before the PIN is entered by the user. WFS_CMD_PIN_GET_PIN and WFS_CMD_PIN_GET_PINBLOCK should be called after this command. For all other devices WFS_ERR_UNSUPP_COMMAND will be returned here.

If this command is required and it is not called, the WFS_CMD_PIN_GET_PIN command will fail with the generic error WFS_ERR_SEQUENCE_ERROR.

If the input parameters passed to this command and WFS_CMD_PIN_GET_PINBLOCK are not identical, the WFS_CMD_PIN_GET_PINBLOCK command will fail with the generic error WFS_ERR_INVALID_DATA.

The data associated with this command will be cleared on a WFS_CMD_PIN_GET_PINBLOCK command.

**Input Param**

LPWFSPINBLOCK lpPinSetBlockData;

See WFS_CMD_PIN_GET_PINBLOCK for details.

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_FORMATNOTSUPP</td>
<td>The specified format is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of lpsKeyEncKey or lpsKey is not supported by this key or the length of an encryption key is not compatible with the encryption operation required.</td>
</tr>
</tbody>
</table>

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**

None.
5.1.29 WFS_CMD_PIN_SET_LOGICAL_HSM

Description
This command allows an application select the logical HSM that should be active. If the device
does not support multiple logical HSMs this command returns
WFS_ERR_UNSUPP_COMMAND. The WFS_INF_PIN_QUERY_LOGICAL_HSM_DETAIL
command can be called to determine the current active logical HSM.

Once the active logical HSM is set with this command, that logical HSM remains active until this
command is used to change the logical HSM or the system is re-started.

The selected HSM is not persistent across re-boots, when applications want to address a specific
logical HSM they must ensure that the correct logical HSM is set as the active logical HSM.

The commands affected by this command are as follows:

- WFS_INF_PIN_HSM_TDATA
- WFS_INF_PIN_KEY_DETAIL_EX
- WFS_CMD_PIN_HSM_SET_TDATA
- WFS_CMD_PIN_SECURE_MSG_SEND (only affected for the protocols
  WFS_PIN_PROTHSM_LDI and WFS_PIN_PROTISOPS)
- WFS_CMD_PIN_SECURE_MSG_RECEIVE (only affected for the protocols
  WFS_PIN_PROTHSM_LDI and WFS_PIN_PROTISOPS)
- WFS_CMD_PIN_HSM_INIT
- WFS_CMD_PIN_GET_JOURNAL (only affected for the protocol
  WFS_PIN_PROTISOPS)

If there are multiple XFS applications that manipulate the current logical HSM then applications
must co-operate or use the XFS locking facilities to synchronize access to the logical HSMs. The
current logical HSM is the same for all clients.

Input Param
LPWFSPINHSMIDENTIFIER lpSetHSM;

typedef struct _wfs_pin_hsm_identifier
{
    WORD wHSMSerialNumber;
} WFSPINHSMIDENTIFIER, *LPWFSPINHSMIDENTIFIER;

wHSMSerialNumber
Specifies the serial number of the HSM that should be set as the active HSM. The value passed in
this field corresponds to the wHSMSerialNumber field reported in the
WFS_INF_PIN_QUERY_LOGICAL_HSM_DETAIL command output structure (and hence
corresponds to the CB tag in the HSM TDATA). The wHSMSerialNumber value is encoded as a
standard binary value (i.e. it is not BCD).

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDHSM</td>
<td>The logical HSM serial number specified is not valid.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_HSM_CHANGED</td>
<td>Indicates that the current logical HSM has changed to the HSM identified within the event.</td>
</tr>
</tbody>
</table>
Comments
None.
### Description
The command imports an encryption key that has been passed by the application within an ANSI X9 TR-31 key block (see reference 35).

### Input Param
- **LPWFS_PIN_IMPORT_KEYBLOCK lpImportKeyBlock**;

```c
typedef struct _wfs_pin_import_key_block {
    LPSTR lpsKey;
    LPSTR lpsEncKey;
    LPWFSXDATA lpxKeyBlock;
} WFSPINIMPORTKEYBLOCK, *LPWFSPINIMPORTKEYBLOCK;
```

- **lpsKey**
  Specifies the name of key being loaded.

- **lpsEncKey**
  Specifies a key name which will be used to verify and decrypt the key block passed in `lpxKeyBlock`. This key must have a key usage defined as WFS_PIN_USEANSTR31MASTER.

- **lpxKeyBlock**
  Specifies the complete key block for the key being imported.

### 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key encryption key was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key encryption key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_FORMATINVALID</td>
<td>The format of the key block is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_CONTENTINVALID</td>
<td>The content of the key block is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_FORMATNOTSUPP</td>
<td>The key block version or content is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_DUPLICATEKEY</td>
<td>A key exists with that name and cannot be overwritten.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The key control flags specified within the key block are inconsistent, are not supported by the hardware, or the <code>lpsEncKey</code> is not defined as a WFS_PIN_USEANSTR31MASTER key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of the actual encryption key within <code>lpxKeyBlockValue</code> is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYININVALID</td>
<td>The key block failed its authentication check.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOKEYRAM</td>
<td>There is no space left in the key RAM for a key of the specified type.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

### Comments
None.
5.1.31 WFS_CMD_PIN_POWER_SAVE_CONTROL

Description
This command activates or deactivates the power-saving mode.

If the Service Provider receives another execute command while in power saving mode, the Service Provider automatically exits the power saving mode, and executes the requested command. If the Service Provider receives an information command while in power saving mode, the Service Provider will not exit the power saving mode.

Input Param
LPWFSPINPOWERSAVECONTROL lpPowerSaveControl;

typedef struct _wfs_pin_power_save_control
{
    USHORT     usMaxPowerSaveRecoveryTime;
} WFSPINPOWERSAVECONTROL, *LPWFSPINPOWERSAVECONTROL;

usMaxPowerSaveRecoveryTime
Specifies the maximum number of seconds in which the device must be able to return to its normal operating state when exiting power save mode. The device will be set to the highest possible power save mode within this constraint. If usMaxPowerSaveRecoveryTime is set to zero then the device will exit the power saving mode.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_POWERSAVETOOSHORT</td>
<td>The power saving mode has not been activated because the device is not able to resume from the power saving mode within the specified usMaxPowerSaveRecoveryTime value.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_POWER_SAVE_CHANGE</td>
<td>The power save recovery time has changed.</td>
</tr>
</tbody>
</table>

Comments
None.
5.2 Common commands for Remote Key Loading Schemes

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

5.2.1 WFS_CMD_PIN_START_KEY_EXCHANGE

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

This output value is returned to the host and is used in the
WFS_CMD_PIN_IMPORT_RSA_ENCRYPTED_PKCS7_KEY and
WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY commands to verify that the encryptor is
talking to the proper host.

The WFS_CMD_PIN_IMPORT_RSA_ENCRYPTED_PKCS7_KEY and
WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY commands end the key exchange
process.

**Input Param**
None.

**Output Param**

```c
typedef struct _wfs_pin_start_key_exchange
{
    LPWFSXDATA lpxRandomItem;
} WFS_PIN_START_KEY_EXCHANGE, *LPWFS_PIN_START_KEY_EXCHANGE;
```

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
</tbody>
</table>

**Events**
None.

**Comments**
None.
5.3 Remote Key Loading Using Signatures

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

5.3.1 WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY

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

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

Input Param

```
LPWFSPINIMPORTRSA.PUBLICKEY lpImportRSAPublicKey;
```

```
typedef struct _wfs_pin_import_rsa_public_key
{
   LPSTR     lpsKey;
   LPWFSXDATA    lpxValue;
   DWORD     dwUse;
   LPSTR     lpsSigKey;
   DWORD     dwRSASignatureAlgorithm;
   LPWFSXDATA    lpxSignature;
} WFSPINIMPORTRSA.PUBLICKEY, *LPWFSPINIMPORTRSA.PUBLICKEY;
```

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USERSAPUBLIC</td>
<td>Key is used as a public key for RSA Encryption including EMV PIN block creation.</td>
</tr>
<tr>
<td>WFS_PIN_USERSAPUBLICVERIFY</td>
<td>Key is used as a public key for RSA signature verification and/or data decryption.</td>
</tr>
</tbody>
</table>

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 Security Item which uniquely identifies the PIN device (see the command WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM) and the PKCS #1 formatted RSA public key to be deleted, i.e. UIATM||PKTODELETE. **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**

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

**dwRSASignatureAlgorithm**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIGN_NA</td>
<td>No signature algorithm specified. No signature verification will take place and the contents of <em>lpsSigKey</em> and <em>lpxSignature</em> are ignored.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PKCS1_V1_5</td>
<td>Use the RSASSA-PKCS1-v1.5 algorithm.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PSS</td>
<td>Use the RSASSA-PSS algorithm.</td>
</tr>
</tbody>
</table>

**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. This value contains NULL when no key validation is required.

**Output Param**

*LPWFSPINIMPORTRSAPUBLICKEYOUTPUT lpImportRSAPublicKeyValueOutput;*

```c
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 one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_RSA_KCV_NONE</td>
<td>No check value is returned in <em>lpxKeyCheckValue</em>.</td>
</tr>
<tr>
<td>WFS_PIN_RSA_KCV_SHA1</td>
<td><em>lpxKeyCheckValue</em> contains a SHA-1 digest of the public key.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The key name supplied in <em>lpsSigKey</em> was not found.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>An invalid use was specified for the key being imported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_DUPLICATEKEY</td>
<td>A key exists with that name and cannot be overwritten.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDKEYLENGTH</td>
<td>The length of <em>lpxValue</em> is not supported.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_NOKEYRAM</td>
<td>There is no space left in the key RAM for a key of the specified type.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_SIG_NOT_SUPP</td>
<td>The Service Provider does not support the Signature Algorithm requested. The key was discarded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_SIGNATUREINVALID</td>
<td>The signature verification failed. The key has not been stored or deleted.</td>
</tr>
</tbody>
</table>
In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

Comments: None.
5.3.2 WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM

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

This command allows the following data items to be exported:

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

See section 8.1.7 (Default Keys and Security Item loaded during manufacture) for the default names and the description of the keys installed during manufacture. These names are defined to ensure multi-vendor applications can be developed.

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

Input Param
LPWFS_PINEXPORTRSAISSUERSIGNEDITEM lpExportRSAIssuerSignedItem;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_EXPORT_EPP_ID</td>
<td>The Unique ID for the PIN will be exported, lpsName is ignored.</td>
</tr>
<tr>
<td>WFS_PIN_EXPORT_PUBLIC_KEY</td>
<td>The public key identified by lpsName will be exported.</td>
</tr>
</tbody>
</table>

lpsName
Specifies the name of the public key to be exported. The private/public key pair was installed during manufacture; see section 8.1.7 (Default Keys and Security Item loaded during manufacture) for a definition of these default keys. If lpsName is NULL, then the default EPP public key that is used for symmetric key encryption is exported.

Output Param
LPWFS_PINEXPORTRSAISSUERSIGNEDITEMOUTPUT lpExportRSAIssuerSignedItemOutput;

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

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

dwRSASignatureAlgorithm
Specifies the algorithm used to generate the Signature returned in lpxSignature. Contains one of the following values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIGN_NA</td>
<td>No signature algorithm used, no signature will be provided in lpxSignature, the data item may still be exported.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PKCS1_V1_5</td>
<td>RSASSA-PKCS1-v1.5 algorithm used.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PSS</td>
<td>RSASSA-PSS algorithm used.</td>
</tr>
</tbody>
</table>

\( lpxSignature \)

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

### Error Codes

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_NORSAKEYPAIR</td>
<td>The PIN device does not have a private key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The data item identified by lpsName was not found.</td>
</tr>
</tbody>
</table>

### Events

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

### Comments

None.
5.3.3  WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY

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

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

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

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

Input Param
LPWFSPINIMPORTRSAIMPORTDESSKEY lpImportRSASignedDESKey;

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

lpsKey
Specifies the name of key being loaded.

lpsDecryptKey
Specifies the name of the RSA private key used to decrypt the symmetric key. See section 8.1.7 (Default Keys and Security Item loaded during manufacture) for a description of the fixed name defined for the default decryption private key. If lpsDecryptKey is NULL then the default decryption private key is used.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_CRYPT_RSAES_PKCS1_V1_5</td>
<td>Use the RSAAES_PKCS1-v1.5 algorithm.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPT_RSAES_OAEP</td>
<td>Use the RSAAES_OAEP algorithm.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USECRYPT</td>
<td>Key is used for encryption and decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USEFUNCTION</td>
<td>Key is used for PIN block creation.</td>
</tr>
<tr>
<td>WFS_PIN_USEMACING</td>
<td>Key is used for MACing.</td>
</tr>
<tr>
<td>WFS_PIN_USEKEYENCIPHERKEY</td>
<td>Key is used as key encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USEPINLOCAL</td>
<td>Key is used for local PIN check.</td>
</tr>
<tr>
<td>WFS_PIN_USEENODUPLICATE</td>
<td>Key can be imported only once.</td>
</tr>
<tr>
<td>WFS_PIN_USESVCNCKEY</td>
<td>Key is used as CBC Start Value encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USEANSTR31MASTER</td>
<td>Key can be used for importing keys packaged within an ANS TR-31 key block.</td>
</tr>
</tbody>
</table>
If `dwUse` equals zero the specified key is deleted. In that case all parameters but `lpsKey` are ignored. `WFS_CMD_PIN_IMPORT_KEY`, `WFS_CMD_PIN_IMPORT_KEY_EX`, `WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY` and `WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY` can be used to delete a key that has been imported with this command. The equivalent commands in the certificate scheme must not be used to delete a key imported through the signature scheme.

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIGN_NA</td>
<td>No signature algorithm specified. No</td>
</tr>
<tr>
<td></td>
<td>signature verification will take place and the</td>
</tr>
<tr>
<td></td>
<td>content of <code>lpxSignature</code> is ignored.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PKCS1_V1_5</td>
<td>Use the RSASSA-PKCS1-v1.5 algorithm.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PSS</td>
<td>Use the RSASSA-PSS algorithm.</td>
</tr>
</tbody>
</table>

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

**Output Param**
`LPWFSPINIMPORTRSA_SIGNEDDESKEYOUTPUT lpImportRSASignedDESKeyOutput;`

typedef struct _wfs_pin_import_rsa_signed_des_key_output {
    WORD wKeyLength;
    WORD wKeyCheckMode;
    LPWFSXDATA lpxKeyCheckValue;
} WFSPINIMPORTRSA_SIGNEDDESKEYOUTPUT, *LPWFSPINIMPORTRSA_SIGNEDDESKEYOUTPUT;

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_KEY_SINGLE</td>
<td>The imported key is single length.</td>
</tr>
<tr>
<td>WFS_PIN_KEY_DOUBLE</td>
<td>The imported key is double length.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_KCV_NONE</td>
<td>There is no key check value provided.</td>
</tr>
<tr>
<td>WFS_PIN_KCV_SELF</td>
<td>The key check value is calculated by an encryption of the key with itself. For a double length key the KCV is generated using 3DES encryption using the first half of the key as the source data for the encryption.</td>
</tr>
<tr>
<td>WFS_PIN_KCV_ZERO</td>
<td>The key check value is calculated by an encryption of a zero value with the key.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESS_DENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
</tbody>
</table>
WFS_ERR_PIN_DUPLICATEKEY: A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOTFOUND: One of the keys specified were not found.
WFS_ERR_PIN_KEYNOVALUE: The specified key encryption key is not loaded.
WFS_ERR_PIN_USEVIOLATION: The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH: The length of lpxValue is not supported.
WFS_ERR_PIN_NOKEYRAM: There is no space left in the key RAM for a key of the specified type.
WFS_ERR_PIN_SIG_NOT_SUPP: The Service Provider does not support the Signature Algorithm requested. The key was discarded.
WFS_ERR_PIN_SIGNATUREINVALID: The signature in the input data is invalid. The key is not stored in the PIN.
WFS_ERR_PIN_RANDOMINVALID: The encrypted random number in the input data does not match the one previously provided by the EPP. The key is not stored in the PIN.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

Comments
None.
5.3.4 WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR

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

The newly generated key pair can only be used for the use defined in the dwUse flag. This flag defines the use of the private key; its public key can only be used for the inverse function.

Input Param
LPWFSPINGENERATERSAKEYPAIR lpGenerateRSAKeyPair;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USERSAPRIVATE</td>
<td>Key is used as a private key for RSA decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USERSAPRIVATESIGN</td>
<td>Key is used as a private key for RSA Signature generation. Only data generated within the device can be signed.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_DEFAULT</td>
<td>The device will decide the exponent.</td>
</tr>
<tr>
<td>WFS_PIN_EXPONENT_1</td>
<td>Exponent of $2^3+1$ (3).</td>
</tr>
<tr>
<td>WFS_PIN_EXPONENT_4</td>
<td>Exponent of $2^4+1$ (17).</td>
</tr>
<tr>
<td>WFS_PIN_EXPONENT_16</td>
<td>Exponent of $2^{16}+1$ (65537).</td>
</tr>
</tbody>
</table>

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:

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**  None.
5.3.5 **WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED_ITEM**

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

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

See section 8.1.7 (Default Keys and Security Item loaded during manufacture) for the default names and the description of the keys installed during manufacture. These names are defined to ensure multi-vendor applications can be developed.

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

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

**Input Param**

```c
typedef struct _wfs_pin_export_rsa_eppSignedItem {
    WORD wExportItemType;
    LPSTR lpsName;
    LPSTR lpsSigKey;
    DWORD dwSignatureAlgorithm;
} WFS_PIN_EXPORT_RSA_EPP_SIGNEDITEM, *LPWFS_PIN_EXPORT_RSA_EPP_SIGNEDITEM
```

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_EXPORT_EPP_ID</td>
<td>The Unique ID for the PIN will be exported, lpsName is ignored.</td>
</tr>
<tr>
<td>WFS_PIN_EXPORT_PUBLIC_KEY</td>
<td>The public key identified by lpsName will be exported.</td>
</tr>
</tbody>
</table>

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

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

- **dwSignatureAlgorithm**.
  Specifies the algorithm to use to generate the Signature returned in both the lpSelfSignature and lpSignature fields. Contains one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIGN_NA</td>
<td>No signature algorithm used, no signature will be provided in lpSelfSignature or lpSignature. The requested item may still be exported.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PKCS1_V1_5</td>
<td>RSASSA-PKCS1-v1.5 algorithm used.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PSS</td>
<td>RSASSA-PSS algorithm used.</td>
</tr>
</tbody>
</table>

**Output Param**

```c
typedef struct _wfs_pin_export_rsa_eppSignedItemOutput {
    ...}
```

LPWFS_PIN_EXPORT_RSA_EPP_SIGNEDITEMOUTPUT lpExportRSAEPPSignedItemOutput;
typedef struct _WfsPinExportRSAEPPSignedItemOutput
{
    LPWFSXDATA      lpxValue;
    LPWFSXDATA      lpxSelfSignature;
    LPWFSXDATA      lpxSignature;
} WFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT,
*LPWFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT;

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

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_NORSAKEYPAIR</td>
<td>The PIN device does not have a private key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The data item identified by lpsName was not found.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

Comments
None.
5.4 Remote Key Loading with Certificates

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

5.4.1 WFS_CMD_PIN_LOAD_CERTIFICATE

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

Input Param
LPWFSPINLOADCERTIFICATE lpLoadCertificate;

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

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

Output Param
LPWFSPINLOADCERTIFICATEOUTPUT lpLoadCertificateOutput;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_FORMATINVALID</td>
<td>The format of the message is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDCERTSTATE</td>
<td>The certificate module is in a state in which the request is invalid.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_CERTIFICATE_CHANGE</td>
<td>The certificate module state has changed.</td>
</tr>
</tbody>
</table>

Comments
None.
5.4.2 WFS_CMD_PIN_GET_CERTIFICATE

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

Input Param
LPWFSPINGETCERTIFICATE lpGetCertificate;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_PUBLICENCKEY</td>
<td>The corresponding encryption key is to be returned.</td>
</tr>
<tr>
<td>WFS_PIN_PUBLICVERIFICATIONKEY</td>
<td>The corresponding verification key is to be returned.</td>
</tr>
</tbody>
</table>

Output Param
LPWFSPINGETCERTIFICATEOUTPUT lpGetCertificateOutput;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDCERTSTATE</td>
<td>The certificate module is in a state in which the request is invalid.</td>
</tr>
</tbody>
</table>

Events
None.

Comments
None.
5.4.3 WFS_CMD_PIN_REPLACE_CERTIFICATE

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

Input Param
LPWFSPINREPLACECERTIFICATE lpReplaceCertificate;

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

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

Output Param
LPWFSPINREPLACECERTIFICATEOUTPUT lpReplaceCertificateOuput

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_FORMATINVALID</td>
<td>The format of the message is invalid.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_INVALIDCERTSTATE</td>
<td>The certificate module is in a state in which the request is invalid.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_CERTIFICATE_CHANGE</td>
<td>The certificate module state has changed.</td>
</tr>
</tbody>
</table>

Comments
None.
5.4.4 WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY

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

Input Param
LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEY lpImportRSAEncipheredPKCS7Key;

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

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

lpsKey
Specifies the name of the key to be stored.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USECRYPT</td>
<td>Key can be used for encryption/decryption.</td>
</tr>
<tr>
<td>WFS_PIN_USEFUNCTION</td>
<td>Key can be used for PIN functions.</td>
</tr>
<tr>
<td>WFS_PIN_USEMACING</td>
<td>Key can be used for MACing.</td>
</tr>
<tr>
<td>WFS_PIN_USEKEYENCKEY</td>
<td>Key is used as key encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USEENODUPLICATE</td>
<td>Key can be imported only once.</td>
</tr>
<tr>
<td>WFS_PIN_USESLENCKEY</td>
<td>Key is used as CBC Start Value encryption key.</td>
</tr>
<tr>
<td>WFS_PIN_USEANSTR31MASTER</td>
<td>Key can be used for importing keys packaged within an ANSI TR-31 key block.</td>
</tr>
</tbody>
</table>

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

Output Param
LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT lpImportRSAEncipheredKeyOut;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_KEYESINGLE</td>
<td>The imported key is single length.</td>
</tr>
<tr>
<td>WFS_PIN_KEYDOUBLE</td>
<td>The imported key is double length.</td>
</tr>
</tbody>
</table>
**lpxRSAData**
Pointer to a binary encoded PKCS #7, represented in DER encoded ASN.1 notation. The message has an outer Signed-data content type with the SignerInfo encryptedDigest field containing the ATM’s signature. The random numbers are included as authenticatedAttributes within the SignerInfo. The inner content is a data content type, which contains the HOST identifier as an issuerAndSerialNumber sequence.

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**
The following is a generic structure of how the lpxImportRSAIn field is structured regarding the outer signed data content type and the inner content as an Envelope-data content type:

```plaintext
ContentInfo ::= SEQUENCE
{
    contentType ContentType = signedData
    content
    SignedData ::= SEQUENCE
    {
        version Version,
        DigestAlgorithms DigestAlgorithmIdentifiers,
        contentInfo ContentInfo ::= SEQUENCE,
        {
            contentType ContentType = EnvelopedData
            content
            ::=
        }
    }
}
```
5.5 **EMV**

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

### 5.5.1 WFS_CMD_PIN_EMV_IMPORT_PUBLIC_KEY

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

**Input Param**

```
LPWFSPINEMVIMPORTPUBLICKEY lpEMVImportPublicKey;
```

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

*lpsKey*  
Specifies the name of key being loaded.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_USERSAPUBLIC</td>
<td>Key is used as a public key for RSA encryption including EMV PIN block creation.</td>
</tr>
<tr>
<td>WFS_PIN_USERSAPUBLICVERIFY</td>
<td>Key is used as a public key for RSA signature verification and/or data decryption. If <em>dwUse</em> equals zero the specified key is deleted. In that case all parameters but <em>lpsKey</em> are ignored.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_EMV_IMPORTPLAIN_CA</td>
<td>This scheme is used by VISA. A plain text CA public key is imported with no verification. The two parts of the key (modulus and exponent) are passed in clear mode as a DER encoded PKCS#1 public key. The key is loaded directly in the security module.</td>
</tr>
<tr>
<td>WFS_PIN_EMV_IMPORT_CHKSUM_CA</td>
<td>This scheme is used by VISA. A plain text CA public key is imported using the EMV 2000 Book II verification algorithm and it is verified before being loaded in the security module. (See [Ref. 4] under references section for this document).</td>
</tr>
<tr>
<td>WFS_PIN_EMV_IMPORT_EPI_CA</td>
<td>This scheme is used by MasterCard Europe. A CA public key is imported using the self-signed scheme defined in [Ref. 5].</td>
</tr>
</tbody>
</table>
An Issuer public key is imported as defined in EMV 2000 Book II, reference 4. (See [Ref. 4] under references section for this document).

An ICC public key is imported as defined in EMV 2000 Book II, reference 4. (See [Ref. 4] under references section for this document).

An ICC PIN public key is imported as defined in EMV 2000 Book II, reference 4. (See [Ref. 4] under references section for this document).

A CA public key is imported and verified using a signature generated with a private key for which the public key is already loaded.

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

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

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

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

If *wImportScheme* is `WFS_PIN_EMV_IMPORT_ISSUER` then *lxImportData* contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the Service Provider. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value: ‘9F32’), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value: ‘90’), the remainder length (1 byte), the remainder value (variable length – EMV Tag value: ‘92’), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value: ‘5A’). The Service Provider will compare the leftmost three to eight hex digits (where each byte consists of two hex digits) of the PAN to the Issuer Identification Number retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book 2 – Security & Key Management Version 4.0, Table 4 (See [Ref. 4] under the reference section for this document). *lpsSigKey* defines the previously loaded key used to verify the signature.

If *wImportScheme* is `WFS_PIN_EMV_IMPORT_ICC` then *lxImportData* contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the Service Provider. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value: ‘9F47’), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value: ‘9F46’), the remainder length (1 byte), the remainder value (variable length – EMV Tag value: ‘9F48’), the SDA length (1 byte), the SDA value (variable length), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value: ‘5A’). The Service Provider will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book 2 – Security & Key Management Version 4.0, Table 9 (See [Ref. 4] under the reference section for this document). *lpsSigKey* defines the previously loaded key used to verify the signature.
If `wImportScheme` is `WFS_PIN_EMV_IMPORT_ICC_PIN` then `lpxImportData` contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the Service Provider. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value: ‘9F2E’), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value: ‘9F2D’), the remainder length (1 byte), the remainder value (variable length – EMV Tag value: ‘9F2F’), the SDA length (1 byte), the SDA value (variable length), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value: ‘5A’). The Service Provider will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book2 – Security & Key Management Version 4.0, Table 9 (See [Ref. 4] under the reference section for this document). `lpsSigKey` defines the previously loaded key used to verify the signature.

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

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

**Output Param**

```
LPWFSPINEMVIMPORTPUBLICKEYOUTPUT lpEMVImportPublicKeyOutput;
```

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

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

**Error Codes**

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

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

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS</td>
<td>An error occurred accessing an encryption key.</td>
</tr>
</tbody>
</table>

**Comments**

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

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

**Input Param**

- **LPWFSPINDIGEST lpDigest;**

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

- **wHashAlgorithm** Specifies which hash algorithm should be used to calculate the hash. See the Capabilities section for valid algorithms.
- **lpxDigestInput** Pointer to the structure that contains the length and the data to be hashed.

**Output Param**

- **LPWFSPINDIGESTOUTPUT lpDigestOutput;**

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

- **lpxDigestOutput** Pointer to the structure that contains the length and the data containing the calculated hash.

**Error Codes**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_ACCESSDENIED</td>
<td>The encryption module is either not initialized or not ready for any vendor specific reason.</td>
</tr>
</tbody>
</table>

**Events**

- None.

**Comments**

- None.
6. Events

6.1 WFS_EXEE_PIN_KEY

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

Event Param
LPWFS_PINKEY 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 0x00 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.
6.2 WFS_SRVE_PIN_INITIALIZED

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

**Event Param**
LPWFSPININIT lpInit;

For a definition of the WFSPININIT structure see command WFS_CMD_PIN_INITIALIZATION.

**Comments**
None.
6.3 WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS

Description
This event specifies that an error occurred accessing an encryption key. Possible situations for generating this event are listed in the description of lErrorCode.

Event Param
LPWFSPINACCESS lpAccess;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_PIN_KEYNOTFOUND</td>
<td>The specified key was not loaded or attempting to delete a non-existent key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_KEYNOVALUE</td>
<td>The specified key is not loaded.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_USEVIOLATION</td>
<td>The specified use is not supported by this key.</td>
</tr>
<tr>
<td>WFS_ERR_PIN_ALGORITHMNOTSUPP</td>
<td>The specified algorithm is not supported by this key.</td>
</tr>
</tbody>
</table>

Comments
None.
6.4 WFS_SRVE_PIN_OPT_REQUIRED

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

**Event Param**

```c
LPWFSPINHSMIDENTIFIER lpOPTRequired;

typedef struct _wfs_pin_hsm_identifier
{
    WORD     wHSMSerialNumber;
} WFSPINHSMIDENTIFIER, *LPWFSPINHSMIDENTIFIER;
```

- `wHSMSerialNumber` Specifies the serial number of the logical HSM where the online time has been reached. If logical HSMs are not supported then `lpOPTRequired` is NULL. The `wHSMSerialNumber` value is encoded as a standard binary value (i.e. it is not BCD).

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

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

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

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

**Event Param**
LPWORD lpwCertificateChange

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_CERT_SECONDARY</td>
<td>The certificate state of the encryptor is now Secondary and Primary Certificates will no longer be accepted.</td>
</tr>
</tbody>
</table>

**Comments**
None.
### 6.6 WFS_SRVE_PIN_HSM_TDATA_CHANGED

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

On configurations with multiple logical HSMs, the serial number tag must be included within the data so that the logical HSM that has changed can be identified.

**Event Param**  
LPWFSXDATA lpxTData;

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

**Comments**  
None.
6.7 **WFS_SRVE_PIN_HSM_CHANGED**

**Description**
This event indicates that the currently active logical HSM has been changed. This event will be triggered when an application changes the current HSM through the `WFS_CMD_PIN_SET_LOGICAL_HSM` command. This event is not generated if the HSM is not changed.

**Event Param**

```c
LPWFSPINHSMIDENTIFIER lpHSMChanged;
```

```c
typedef struct _wfs_pin_hsm_identifier
{
    WORD wHSMSerialNumber;
} WFSPINHSMIDENTIFIER, *LPWFSPINHSMIDENTIFIER;
```

- `wHSMSerialNumber` Specifies the serial number of the logical HSM that has been made active. The `wHSMSerialNumber` value is encoded as a standard binary value (i.e. it is not BCD).

**Comments**
None.
### 6.8 WFS_EXEE_PIN_ENTERDATA

<table>
<thead>
<tr>
<th>Description</th>
<th>This mandatory event notifies the application when the device is ready for the user to start entering data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Param</td>
<td>None.</td>
</tr>
<tr>
<td>Comments</td>
<td>None.</td>
</tr>
</tbody>
</table>
6.9 **WFS_SRVE_PIN_DEVICEPOSITION**

**Description**
This service event reports that the device has changed its position status.

**Event Param**

```c
typedef struct _wfs_pin_device_position
{
    WORD    wPosition;
} WFSPINDEVICEPOSITION, *LPWFSPINDEVICEPOSITION;
```

`wPosition`
Position of the device as one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_DEVICEINPOSITION</td>
<td>The device is in its normal operating position.</td>
</tr>
<tr>
<td>WFS_PIN_DEVICENOTINPOSITION</td>
<td>The device has been removed from its normal operating position.</td>
</tr>
<tr>
<td>WFS_PIN_DEVICEPOSUNKNOWN</td>
<td>The position of the device cannot be determined.</td>
</tr>
</tbody>
</table>

**Comments**
None.
6.10 WFS_SRVE_PIN_POWER_SAVE_CHANGE

**Description**  This service event specifies that the power save recovery time has changed.

**Event Param**  LPWFSPINPOWERSAVECHANGE lpPowerSaveChange;

```c
typedef struct _wfs_pin_power_save_change {
    USHORT    usPowerSaveRecoveryTime;
} WFSPINPOWERSAVECHANGE, *LPWFSPINPOWERSAVECHANGE;
```

*usPowerSaveRecoveryTime*
Specifies the actual number of seconds required by the device to resume its normal operational state. This value is zero if the device exited the power saving mode.

**Comments**  If another device class compounded with this device enters into a power saving mode, this device will automatically enter into the same power saving mode and this event will be generated.
7. C - Header File

/******************************************************************************
*                                                                           *
* xfspin.h XFS - Personal Identification Number Keypad (PIN) definitions    *
*                                                                           *
*            Version 3.20  (March 02 2011)                                  *
*                                                                           *
******************************************************************************/

#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               (0x1403) /* Version 3.20 */
#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)
#define WFS_INF_PIN_QUERY_LOGICAL_HSM_DETAIL        (PIN_SERVICE_OFFSET + 9)
#define WFS_INF_PIN_QUERY_PCIPTS_DEVICE_ID          (PIN_SERVICE_OFFSET + 10)

/* 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_ID                      (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_SECUREKEY_ENTRY                 (PIN_SERVICE_OFFSET + 38)
#define WFS_CMD_PIN_GENERATE_KCV                    (PIN_SERVICE_OFFSET + 39)
#define WFS_CMD_PIN_SET_GUIDANCE_LIGHT              (PIN_SERVICE_OFFSET + 41)
#define WFS_CMD_PIN_MAINTAIN_PIN                    (PIN_SERVICE_OFFSET + 42)
#define WFS_CMD_PIN_KEYPRESS_BEEP                   (PIN_SERVICE_OFFSET + 43)
#define WFS_CMD_PIN_SET_PINBLOCK_DATA               (PIN_SERVICE_OFFSET + 44)
#define WFS_CMD_PIN_SET_LOGICAL_HSM                 (PIN_SERVICE_OFFSET + 45)
#define WFS_CMD_PIN_IMPORT_KEYBLOCK                 (PIN_SERVICE_OFFSET + 46)
#define WFS_CMD_PIN_POWER_SAVE_CONTROL              (PIN_SERVICE_OFFSET + 47)

/* 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)
#define WFS_SRVE_PIN_HSM_CHANGED                     (PIN_SERVICE_OFFSET + 7)
#define WFS_EXEE_PIN_ENTERDATA                       (PIN_SERVICE_OFFSET + 8)
#define WFS_SRVE_PIN_DEVICEPOSITION                  (PIN_SERVICE_OFFSET + 9)
#define WFS_SRVE_PIN_POWER_SAVE_CHANGE               (PIN_SERVICE_OFFSET + 10)

/* values of WFSPIINSTATUS.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_DEVUSERERROR                        WFS_STAT_DEVUSERERROR
#define WFS_PIN_DEVBUSY                             WFS_STAT_DEVBUSY
#define WFS_PIN_DEVFRAUDATTEMPT                     WFS_STAT_DEVFRAUDATTEMPT
#define WFS_PIN_DEVPOTENTIALFRAUD                   WFS_STAT_DEVPOTENTIALFRAUD

/* values of WFSPIINSTATUS.fwEncStat */

#define WFS_PIN_ENCREADY                            (0)
#define WFS_PIN_ENCNOTREADY                          (1)
#define WFS_PIN_ENCNOTINITIALIZED                    (2)
#define WFS_PIN_ENCUNDEFINED                         (3)
#define WFS_PIN_ENCUNDEFINED                         (4)
#define WFS_PIN_ENCINITIALIZED                       (5)
#define WFS_PIN_ENCINTAMPERED                        (6)

/* Size and max index of dwGuidLights array */

#define WFS_PIN_GUIDLIGHTS_SIZE                     (32)
#define WFS_PIN_GUIDLIGHTS_MAX                      (WFS_PIN_GUIDLIGHTS_SIZE - 1)

/* Indices of WFSPIINSTATUS.dwGuidLights [...] */

#define WFS_PIN_GUIDANCE_PINPAD                     (0)

/* Values of WFSPIINSTATUS.dwGuidLights [...] */
#define WFS_PIN_GUIDANCE_NOT_AVAILABLE          (0x00000000)
#define WFS_PIN_GUIDANCE_OFF                    (0x00000001)
#define WFS_PIN_GUIDANCE_ON                     (0x00000002)
#define WFS_PIN_GUIDANCE_SLOW_FLASH             (0x00000004)
#define WFS_PIN_GUIDANCE_MEDIUM_FLASH           (0x00000008)
#define WFS_PIN_GUIDANCE_QUICK_FLASH            (0x00000010)
#define WFS_PIN_GUIDANCE_CONTINUOUS             (0x00000080)
#define WFS_PIN_GUIDANCE_RED                    (0x00000100)
#define WFS_PIN_GUIDANCE_GREEN                  (0x00000200)
#define WFS_PIN_GUIDANCE_YELLOW                 (0x00000400)
#define WFS_PIN_GUIDANCE_BLUE                   (0x00000800)
#define WFS_PIN_GUIDANCE_CYAN                   (0x00001000)
#define WFS_PIN_GUIDANCE_MAGENTA                (0x00002000)
#define WFS_PIN_GUIDANCE_WHITE                  (0x00004000)

/* values for WFSPINSTATUS.fwAutoBeepMode and WFS_CMD_PIN_KEYPRESS_BEEP lpwMode parameter */
#define WFS_PIN_BEEP_ON_ACTIVE                   (0x0001)
#define WFS_PIN_BEEP_ON_INACTIVE                 (0x0002)

/* values of WFSPINSTATUS.wDevicePosition WFSPINDEVICEPOSITION.wPosition */
#define WFS_PIN_DEVICEINPOSITION                 (0)
#define WFS_PIN_DEVICENOTINPOSITION               (1)
#define WFS_PINDEVICEPOSUNKNOWN                  (2)
#define WFS_PINDEVICEPOSNOTSUPP                  (3)

/* values of WFSPINCAPS.fwType */
#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_PINFORMISO0                         (0x0004)
#define WFS_PINFORMISO1                         (0x0008)
#define WFS_PINFORMEC12                         (0x0010)
#define WFS_PINFORMEC13                         (0x0020)
#define WFS_PINFORMVISA                         (0x0040)
#define WFS_PINFORMBANKS                        (0x0080)
#define WFS_PINFORMVISM                         (0x0200)
#define WFS_PINFORMBANKSYS                      (0x0400)
#define WFS_PINFORMMV                           (0x0800)
#define WFS_PINFORMISO3                         (0x2000)
#define WFS_PINFORMMAP                          (0x4000)

/* 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 WFSPINCAPS.fwAutoBeep */
#define WFS_PIN_BEEP_ACTIVE_AVAILABLE (0x0001)
#define WFS_PIN_BEEP_ACTIVE_SELECTABLE (0x0002)
#define WFS_PIN_BEEP_INACTIVE_AVAILABLE (0x0004)
#define WFS_PIN_BEEP_INACTIVE_SELECTABLE (0x0008)

/** values of WFSPINCAPS.fwKeyBlockImportFormats */
#define WFS_PIN_ANSTR31KEYBLOCK (0x0001)

/** additional values for WFSPINKEYDETAIL.fwUse and values of WFSPINKEYDETAILEX.dwUse */
#define WFS_PIN_USECRYPT (0x0001)
#define WFS_PIN_USEFUNCTION (0x0002)
#define WFS_PIN_USEMACING (0x0004)
#define WFS_PIN_USEKEYENCKEY (0x0020)
#define WFS_PIN_USENODUPLICATE (0x0040)
#define WFS_PIN_USESVENCKEY (0x0080)
#define WFS_PIN_USECONSTRUCT (0x0100)
#define WFS_PIN_USESECURECONSTRUCT (0x0200)
#define WFS_PIN_USEANSTR31MASTER (0x0400)

/* additional values for WFSPINKEYDETAILEX.dwUse */
#define WFS_PIN_USEPINLOCAL (0x00010000)
#define WFS_PIN_USERSAPUBLIC (0x00020000)
#define WFS_PIN_USERSAPRIVATE (0x00040000)
#define WFS_PIN_USECHIPINFO (0x00100000)
#define WFS_PIN_USECHIP (0x00200000)
#define WFS_PIN_USECHIPMAC (0x00400000)
#define WFS_PIN_USECHIPMACAZ (0x00800000)
#define WFS_PIN_USERSAPUBLICVERIFY (0x08000000)
#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_OEM1 (0x02000000)
#define WFS_PIN_FK_OEM2 (0x04000000)
#define WFS_PIN_FK_OEM3 (0x08000000)
#define WFS_PIN_FK_OEM4 (0x10000000)
#define WFS_PIN_FK_OEM5 (0x20000000)
#define WFS_PIN_FK_OEM6 (0x40000000)

/* 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 WFSPINFDK.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 (0x00000001)
#define WFS_PIN_FK_FDK23 (0x00000002)
#define WFS_PIN_FK_FDK24 (0x00000004)
#define WFS_PIN_FK_FDK25 (0x00000008)
#define WFS_PIN_FK_FDK26 (0x00000010)
#define WFS_PIN_FK_FDK27 (0x00000020)
#define WFS_PIN_FK_FDK28 (0x00000040)
#define WFS_PIN_FK_FDK29 (0x00000080)
#define WFS_PIN_FK_FDK30 (0x00000000)
#define WFS_PIN_FK_FDK31 (0x00000100)
#define WFS_PIN_FK_FDK32 (0x00000200)

/* 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_COMPFK (9)
#define WFS_PIN_COMPHELP (10)
#define WFS_PIN_COMPFF (11)
#define WFS_PIN_COMPCOMTFF (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_PROTGENAS (7)
#define WFS_PIN_PROTPINCHG (8)
#define WFS_PIN_PROTISOPINCHG (9)
#define WFS_PIN_PROTPNF (10)
#define WFS_PIN_PROTISOPNF (11)

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

/* values of WFSPINENCIO.wProtocol and WFSPINCAPS.fwENCIOProtocols */
#define WFS_PIN_ENC_PROT_CH (0x0001)
#define WFS_PIN_ENC_PROT_GIECB (0x0002)
#define WFS_PIN_ENC_PROT_LUX (0x0004)

/* values for WFS_SRVE_PIN_CERTIFICATE_CHANGE and WFSPINSTATUS.dwCertificateState */
#define WFS_PIN_CERT_SECONDARY (0x00000002)

/* values for WFSPINSTATUS.dwCertificateState */
#define WFS_PIN_CERT_UNKNOWN (0x00000000)
#define WFS_PIN_CERT_PRIMARY (0x00000001)
#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
type 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)
#define WFS_PIN_SIG_ENHANCED_RKL        (0x00000008)

/* values of WFSPINIMPORTRSAPUBLICKEY.dwRSASignatureAlgorithm and
WFSPINCAPS.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_EXPONENT_1 (1)
#define WFS_PIN_EXPONENT_4 (2)
#define WFS_PIN_EXPONENT_16 (3)

/* values of WFSPINIMPORTRTSASIGNEDDESKEYOUTPUT.wKeyLength and */
/* WFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT.wKeyLength */

#define WFS_PIN_KEYDOUBLE (0x0002)

/* values of WFSPINGETCERTIFICATE.wGetCertificate */

#define WFS_PIN_PUBLICENCKEY (1)
#define WFS_PIN_PUBLICVERIFICATIONKEY (2)

/* values for WFSPINEMVIMPORTPUBLICKEY.wImportScheme and
WFSPINCAPS.lpEMVImportSchemes */

#define WFS_PIN_EMV_IMPORT_PLAIN_CA (1)
#define WFS_PIN_EMV_IMPORT_CHKSUM_CA (2)
#define WFS_PIN_EMV_IMPORT_EPI_CA (3)
#define WFS_PIN_EMV_IMPORT_ISSUER (4)
#define WFS_PIN_EMV_IMPORT_ICC (5)
#define WFS_PIN_EMV_IMPORT_ICC_PIN (6)
#define WFS_PIN_EMV_IMPORT_PKCSV1_5_CA (7)

/* values for WFSPINDIGEST.wHashAlgorithm and WFSPINCAPS.fwEMVHashAlgorithm */

#define WFS_PIN_HASH_SHA1_DIGEST (0x0001)

/* values of WFSPINSECUREKEYDETAIL.fwEntryMode */
#: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)

/* values of WFSPINSTATUS.wAntiFraudModule */
#:define WFS_PIN_AFMNOTSUPP                          (0)
#:define WFS_PIN_AFMOK                               (1)
#:define WFS_PIN_AFMINOP                             (2)
#:define WFS_PIN_AFMDEVICEDETECTED                   (3)
#:define WFS_PIN_AFMUNKNOWN                          (4)

/* 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_HSMSSTATEINVALID                (-(PIN_SERVICE_OFFSET + 23))
#:define WFS_ERR_PIN_MACINVALID                      (-(PIN_SERVICE_OFFSET + 24))
#:define WFS_ERR_PIN_PROTOCOLINVALID                 (-(PIN_SERVICE_OFFSET + 25))
#:define WFS_ERR_PIN_CONTENTINVALID                  (-(PIN_SERVICE_OFFSET + 26))
#: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))
#:define WFS_ERR_PIN_POWERSAVETOOSHORT               (-(PIN_SERVICE_OFFSET + 40))
#:define WFS_ERR_PIN_INVALIDHSM                      (-(PIN_SERVICE_OFFSET + 41))

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

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

typedef struct _wfs_pin_status
{
    WORD     fwDevice;
    WORD     fwEncStat;

LPSTR lpszExtra;
DWORD dwGuidLights[WFS_PIN_GUIDLIGHTS_SIZE];
WORD fwAutoBeepMode;
DWORD dwCertificateState;
WORD wDevicePosition;
USHORT usPowerSaveRecoveryTime;
WORD wAntiFraudModule;
} 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;
    DWORD dwGuidLights[WFS_PIN_GUIDLIGHTS_SIZE];
    BOOL bPINCanPersistAfterUse;
    WORD fwAutoBeep;
    LPSTR lpsHSMVendor;
    BOOL bHSMJournaling;
    DWORD dwRSAAuthenticationScheme;
    DWORD dwRSASignatureAlgorithm;
    DWORD dwRSACryptAlgorithm;
    DWORD dwRSAKeyCheckMode;
    DWORD dwSignatureScheme;
    LPWORD lpwEMVImportSchemes;
    WORD fwEMVHashAlgorithm;
    BOOL bKeyImportThroughParts;
    WORD fwENCIOProtocols;
    BOOL bTypeCombined;
    BOOL bSetPinblockDataRequired;
    WORD fwKeyBlockImportFormats;
    BOOL bPowerSaveControl;
    BOOL bAntiFraudModule;
} WFSPINCAPS, *LPWFSPINCAPS;

typedef struct _wfs_pin_key_detail
{
    LPSTR lpsKeyName;
    WORD fwUse;
    BOOL bLoaded;
    LPWFSXDATA lpxKeyBlockHeader;
} 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 usNumberOfFDKs;
    LPWFSXDATA *lppFDKs;
} WFSPINFUNCKEYDETAIL, *LPWFSPINFUNCKEYDETAIL;

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

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;

typedef struct _wfs_pin_pcipts_deviceid
{
    LPSTR lpszManufacturerIdentifier;
    LPSTR lpszModelIdentifier;
    LPSTR lpszHardwareIdentifier;
    LPSTR lpszFirmwareIdentifier;
    LPSTR lpszApplicationIdentifier;
} WFSPINPCIPTSDEVICEID, *LPWFSPINPCIPTSDEVICEID;

typedef struct _wfs_pin_crypt
{
    WORD wMode;
    LPSTR lpsKey;
    LPWFSXDATA lpxKeyEncKey;
    WORD wAlgorithm;
    LPSTR lpszStartValueKey;
    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;
} WFSPINCRTATEOFFSET, *LPWFSPINCRTATEOFFSET;

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
{
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
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;

/* WFS_CDM_PIN_IMPORT_KEYBLOCK command input structure */
typedef struct _wfs_pin_import_key_block
{
    LPSTR lpsKey;
    LPSTR lpsEncKey;
    LPWFSXDATA lpxKeyBlock;
} WFSPINIMPORTKEYBLOCK, *LPWFSPINIMPORTKEYBLOCK;

typedef struct _wfs_pin_import_rsa_public_key
{
    LPSTR lpsKey;
    LPWFSXDATA lpxValue;
    DWORD dwUse;
    LPSTR lpsSigKey;
    DWORD dwRSASignatureAlgorithm;
    LPWFSXDATA lpxSignature;
typedef struct _wfs_pin_import_rsa_public_key_output {
    DWORD dwRSAKeyCheckMode;
    LPWFSXDATA lpxKeyCheckValue;
} WFSPINIMPORTRSAPUBLICKEYOUTPUT, *LPWFSPINIMPORTRSAPUBLICKEYOUTPUT;

typedef struct _wfs_pin_import_rsa_public_key {
    DWORD dwRSAKeyCheckMode;
    LPWFSXDATA lpxKeyCheckValue;
} WFSPINIMPORTRSAPUBLICKEY, *LPWFSPINIMPORTRSAPUBLICKEY;

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 dwRSAEncryptionAlgorithm;
    LPWFSXDATA lpxValue;
    DWORD dwUse;
    LPSTR lpsSigKey;
    DWORD dwRSASignatureAlgorithm;
    LPWFSXDATA lpxSignature;
} WFSPINIMPORTRSA SigNEDESKY, *LPWFSPINIMPORTRSA SigNEDESKY;

typedef struct _wfs_pin_import_rsa_signed_des_key_output {
    WORD wKeyLength;
    WORD wKeyCheckMode;
    LPWFSXDATA lpxKeyCheckValue;
} WFSPINIMPORTRSA SigNEDESKYOUTPUT, *LPWFSPINIMPORTRSA SigNEDESKYOUTPUT;

typedef struct _wfs_pin_generate_rsa_key {
    LPSTR lpsKey;
    DWORD dwUse;
    WORD wModulusLength;
    WORD wExponentValue;
} WFSPINGENERATOR RSAKEYPAIR, *LPWFSPINGENERATOR RSAKEYPAIR;

typedef struct _wfs_pin_export_rsa_epp_signed_item {
    WORD wExportItemType;
    LPSTR lpsName;
    LPSTR lpsSigKey;
    DWORD dwSignatureAlgorithm;
} WFSPINEXPORTRSA EPPSIGNEDITEM, *LPWFSPINEXPORTRSA EPPSIGNEDITEM;

typedef struct _wfs_pin_export_rsa_epp_signed_item_output {
    LPWFSXDATA lpxValue;
    LPWFSXDATA lpxSelfSignature;
    LPWFSXDATA lpxSignature;
} WFSPINEXPORTRSA EPPSIGNEDITEMOUTPUT, *LPWFSPINEXPORTRSA EPPSIGNEDITEMOUTPUT;

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

typedef struct _wfs_pin_load_certificate_output {
}
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;

typedef struct _wfs_pin_set_guidlight
{
    WORD     wGuidLight;
    DWORD    dwCommand;
} WFSINSETGUIDLIGHT, *LPWFSINSETGUIDLIGHT;

typedef struct _wfs_pin_maintain_pin
{
    BOOL     bMaintainPIN;
} WFSPINMAINTAINPIN, *LPWFSPINMAINTAINPIN;

typedef struct _wfs_pin_hsm_info
{
    WORD     wHSMSerialNumber;
    LPSTR    lpsZKAID;
} WFSPINHSMINFO, *LPWFSPINHSMINFO;

typedef struct _wfs_pin_hsm_detail
{
    WORD     wActiveLogicalHSM;
    LPWFSINHSMINFO *lppHSMInfo;
} WFSINHSMDETAIL, *LPWFSINHSMDETAIL;

typedef struct _wfs_pin_hsm_identifier
{
    WORD     wHSMSerialNumber;
} WFSPINHSMIDENTIFIER, *LPWFSPINHSMIDENTIFIER;

typedef struct _wfs_pin_power_save_control
{
    USHORT   usMaxPowerSaveRecoveryTime;
} WFSINPOWERSAVECONTROL, *LPWFSINPOWERSAVECONTROL;

/*=================================================================*
/* PIN Message Structures */
/*=================================================================*/
#pragma pack(pop)

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

#ifndef __INC_XFSPIN__H */
8. Appendix-A

This section provides extended explanation of concepts and functionality needing further clarification. The terminology as described below is used within the following sections.

**Definitions and Abbreviations**

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

**Notation of Cryptographic Items and Functions**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK_E</td>
<td>The private key belonging to entity E</td>
</tr>
<tr>
<td>PK_E</td>
<td>The public key belonging to entity E</td>
</tr>
<tr>
<td>SK_ATM</td>
<td>The private key belonging to the ATM/PIN</td>
</tr>
<tr>
<td>PK_ATM</td>
<td>The public key belonging to the ATM/PIN</td>
</tr>
<tr>
<td>SK_HOST</td>
<td>The private key belonging to the Host</td>
</tr>
<tr>
<td>PK_HOST</td>
<td>The public key belonging to the Host</td>
</tr>
<tr>
<td>SK_SI</td>
<td>The private key belonging to Signature Issuer</td>
</tr>
<tr>
<td>PK_SI</td>
<td>The public key belonging to Signature Issuer</td>
</tr>
<tr>
<td>SK_ROOT</td>
<td>The root private key belonging to the Host</td>
</tr>
<tr>
<td>PK_ROOT</td>
<td>The root public key belonging to the Host</td>
</tr>
<tr>
<td>K_NAME</td>
<td>A symmetric key</td>
</tr>
<tr>
<td>Cert_HOST</td>
<td>A Certificate that contains the public verification of the host and is signed by a trusted Certificate Authority.</td>
</tr>
<tr>
<td>Cert_ATM</td>
<td>A Certificate that contains the ATM/PIN public verification or encryption key, which is signed by a trusted Certificate Authority.</td>
</tr>
<tr>
<td>Cert_CA</td>
<td>The Certificate of a new Certificate Authority</td>
</tr>
<tr>
<td>R_ATM</td>
<td>Random Number of the ATM/PIN</td>
</tr>
<tr>
<td>I_HOST</td>
<td>Identifier of the Host</td>
</tr>
<tr>
<td>K_KTK</td>
<td>Key Transport Key</td>
</tr>
<tr>
<td>R_HOST</td>
<td>Random number of the Host</td>
</tr>
<tr>
<td>I_ATM</td>
<td>Identifier of the ATM/PIN</td>
</tr>
<tr>
<td>TP_ATM</td>
<td>Thumb Print of the ATM/PIN</td>
</tr>
<tr>
<td>Sign(SK_E)[D]</td>
<td>The signing of data block D, using the private key SK_E</td>
</tr>
<tr>
<td>Recover(PK_E)[S]</td>
<td>The recovery of the data block D from the signature S, using the private key PK_E</td>
</tr>
<tr>
<td>RSA_Crypt(PK_E)[D]</td>
<td>RSA Encryption of the data block D using the public key PK_E</td>
</tr>
<tr>
<td>Hash [M]</td>
<td>Hashing of a message M of arbitrary length to a 20 Byte hash value</td>
</tr>
<tr>
<td>Des(K)[D]</td>
<td>DES encipherment of an 8 byte data block D using the secret key K</td>
</tr>
<tr>
<td>Des3(K)[D]</td>
<td>Triple DES encipherment of an 8 byte data block D using the 16 byte secret key K = (K_L</td>
</tr>
<tr>
<td>Des3^{-1}(K)[D]</td>
<td>Triple DES decipherment of an 8 byte data block D using the 8 byte secret key K = (K_L</td>
</tr>
<tr>
<td>Rnd_E</td>
<td>A random number created by entity E</td>
</tr>
<tr>
<td>(A</td>
<td></td>
</tr>
</tbody>
</table>
8.1 Remote Key Loading Using Signatures

8.1.1 RSA Data Authentication and Digital Signatures

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

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

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

1. Data block.
2. Digital signature.
3. Host’s public key.

To validate the signature, the ATM performs the following:

1. ATM runs data through the standard hashing algorithm – the same one used by the Host – to produce a digest of the data received. Consider this digest₂;
2. ATM uses the Host’s public key to decrypt the digital signature. The digital signature was produced using the Host’s private key to encrypt the data digest; therefore, when decrypted with the Host’s public key it produces the same digest. Consider this digest₁. Incidentally, no other public key in the world would work to decrypt digest₁ – only the public key corresponding to the signing private key.
3. ATM compares digest₁ with digest₂.

If digest₁ matches digest₂ exactly, the ATM has confirmed the following:

- Data was not tampered with in transit. Changing a single bit in the data sent from the Host to the ATM would cause digest₂ to be different than digest₁. Every data block has a unique digest; therefore, an altered data block is detected by the ATM.
- Public key used to decrypt the digital signature corresponds to the private key used to create it. No other public key could possibly work to decrypt the digital signature, so the ATM was not handed someone else’s public key.

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

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

The detail of this is as follows:

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

**Assumptions:**

1. The Host has obtained the Public Key ($PK_{SI}$) from the Signature Issuer.
2. The Host has provided the Signature Issuer with its Public Key ($PK_{HOST}$), and receives the corresponding signature $\text{Sign}(SK_{SI})[PK_{HOST}]$. The Signature Issuer uses its own Private Key ($SK_{SI}$) to create this signature.
3. In the case where Enhanced Remote Key Loading is used, the host has provided the Signature Issuer with its Public Key ($PK_{ROOT}$), and receives the corresponding signature $\text{Sign}(SK_{SI})[PK_{ROOT}]$. The host has generated another key pair $PK_{HOST}$ and $SK_{HOST}$ and signs the $PK_{HOST}$ with the $SK_{ROOT}$.
4. (Optional) The host obtains a list of the valid PIN device’s Unique Identifiers. The Signature Issuer installs a Signature $\text{Sign}(SK_{SI})[UI_{ATM}]$ for the Unique Id ($UI_{ATM}$) on the ATM PIN. The Signature Issuer uses $SK_{SI}$ to do this.
5. The Signature Issuer installs its Public Key ($PK_{SI}$) on the ATM PIN. It also derives and installs the Signature $\text{Sign}(SK_{SI})[PK_{ATM}]$ of the ATM PIN’s Public Key ($PK_{ATM}$) on the ATM PIN. The Signature Issuer uses $SK_{SI}$ to do this.
6. The ATM PIN device additionally contains its own Public ($PK_{ATM}$) and Private Key ($SK_{ATM}$).

**Step 1**

The ATM PIN sends its Public Key to the Host in a secure structure:

The ATM PIN sends its ATM Public Key with its associated Signature. When the Host receives this information it will use the Signature Issuer’s Public Key to validate the signature and obtain the ATM Public Key.

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

WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM.

**Step 2 (Optional)**

The Host verifies that the key it has just received is from a valid sender.

It does this by obtaining the PIN device unique identifier. The ATM PIN sends its Unique Identifier with its associated Signature. When the Host receives this information it will use the Signature Issuer’s Public Key to validate the signature and retrieve the PIN Unique Identifier. It can then check this against the list it received from the Signature Issuer.

The XFS command used to export the PIN Unique Identifier is

WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM.

**Step 3 (Enhanced Remote Key Loading only)**

The Host sends its root public key to the ATM PIN:

The Host sends its Root Public Key ($PK_{ROOT}$) and associated Signature. The ATM PIN verifies the signature using $PK_{SI}$ and stores the key.

The XFS command used to import the host root public key securely as described above is

WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY.

**Step 4**

The Host sends its public key to the ATM PIN:

The Host sends its Public Key ($PK_{HOST}$) and associated Signature. The ATM PIN verifies the signature using $PK_{SI}$ (or $PK_{ROOT}$ in the Enhanced Remote Key Loading Scheme) and stores the key.

The XFS command used to import the host public key securely as described above is

WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY.

**Step 5**

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

The XFS commands used to exchange master symmetric keys as described above are:

- WFS_CMD_PIN_START_KEY_EXCHANGE
- WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY

**Step 6 – Alternative including random number**

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

The Host encrypts the Master Key (KM) with PK_{ATM}. A signature for the random number and encrypted key is then created using SK_{HOST}.

The ATM PIN will then validate the signature using PK_{HOST}, verify the random number and then obtain the master key by decrypting using SK_{ATM}.

The XFS commands used to exchange master symmetric keys as described above are:

- WFS_CMD_PIN_START_KEY_EXCHANGE
- WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY

The following diagrams summaries the key exchange process described above:
8.1.3 Initialization Phase – Signature Issuer and ATM PIN

This would typically occur in a secure manufacturing environment.

```
Signature Issuer

PK_{ATM}
UI_{ATM}
PK_{SI}

Sign(SK_{SI})[PK_{ATM}]
Sign(SK_{SI})[UI_{ATM}]

PIN

PK_{ATM}
PK_{SI}
Sign(SK_{SI})[PK_{ATM}]
SK_{ATM}
UI_{ATM}
Sign(SK_{SI})[UI_{ATM}]
```
8.1.4 Initialization Phase – Signature Issuer and Host

This would typically occur in a secure offline environment.
8.1.5 Key Exchange – Host and ATM PIN

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

**Host**
- Host validates signature with PKSI:
  - PKATM obtained
- Host validates signature with PKSI:
  - UIATM obtained & verified against list
- Encrypt KM with PKATM and generate Signature for result using SKHOST.

**PIN**
- Optionally send PIN
  - Unique Identifier
- PIN validates signature with PKSI:
  - PKHOST obtained
- PIN validates signature with PKHOST and obtains KM by decrypting with SKATM:
  - KM obtained
8.1.6 Key Exchange (with random number) – Host and ATM PIN

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

**Host**
- Host validates signature with PKSI
- PKATM obtained
- Host validates signature with PKSI
- UIATM obtained & verified against list
- Host requests random number \( R_{ATM} \)
- Encrypt \( K_M \) with PKATM and generate Signature for \( R_{ATM} \) and encryption result using SKHOST.

**PIN**
- Optionally send PIN Unique Identifier
- PIN validates signature with PKSI
- PKHOST obtained
- PIN generates random number, \( R_{ATM} \), and starts key exchange
- PIN validates signature with PKHOST, validates RATM and obtains KM by decrypting with SKATM
- KM obtained
8.1.7 Enhanced RKL, Key Exchange (with random number) – Host and ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key when the PIN device and Service Provider supports the Enhanced Signature Remote Key Loading scheme.
8.1.8 Default Keys and Security Item loaded during manufacture

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

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Item Type</th>
<th>Signed by</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;_SigIssuerVendor&quot;</td>
<td>Public Key</td>
<td>N/A</td>
<td>The public key of the signature issuer, i.e. PKSig</td>
</tr>
<tr>
<td>&quot;_EPPCryptKey&quot;</td>
<td>Public/Private key-pair</td>
<td>The private key associated with _SigIssuerVendor</td>
<td>The key-pair used to encrypt and decrypt the symmetric key, i.e. SKATM and PKATM. The public key is used for encryption by the host and the private for decryption by the EPP.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Item Type</th>
<th>Signed by</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;_EPPSignKey&quot;</td>
<td>Public/Private key-pair</td>
<td>The private key associated with _SigIssuerVendor</td>
<td>A key-pair where the private key is used to sign data, e.g. other generated key pairs.</td>
</tr>
</tbody>
</table>
8.2 Remote Key Loading Using Certificates

8.2.1 Certificate Exchange and Authentication

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

The following flow is how the exchange authentication takes place:

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

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

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

```
Host
The Host sends its CertHost to the ATM.
The Host Requests the ATM Keys.
The Host verifies the message. If it verifies then it stores the key.

CertHOST

TPATM

Request for CertATM

CertATM

PIN
The PIN verifies the message. If it verifies then it stores the key and returns the thumbprint.
The PIN sends the keys inside of a certificate.
```
8.2.2 Remote Key Exchange

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

1. First, the WFS_CMD_PIN_START_KEY_EXCHANGE is called. This returns $R_{ATM}$ from the encryptor to be used in the authenticating the WFS_CMD_PIN_IMPORT_RSA_ENCHIPERED_PKCS7_KEY message.

2. Next, WFS_CMD_PIN_IMPORT_RSA_ENCHIPERED_PKCS7_KEY is called. This command sends down the KTK to the encryptor. The following items below show how this is accomplished.

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

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

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

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

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

The following diagram below shows how the Host and ATM transmit the Key Transport Key.
8.2.3 Replace Certificate

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

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

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

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

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

For anyone attempting to write an application that handles this type of chip card, it is essential to read and understand the ZKA specifications see [Ref 17], [Ref 6] and [Ref 7].

8.3.1 How to use the SECURE_MSG commands

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

- Applications must call SECURE_MSG_SEND for every command they send to the chip or to a host system, including those commands that do not actually require secure messaging. This enables the Service Provider to remember security-relevant data that may be needed or checked later in the transaction.

- Applications must pass a complete message as input to SECURE_MSG_SEND, with all fields - including those that will be filled by the Service Provider - being present in the correct length. All fields that are not filled by the Service Provider must be filled with the ultimate values in order to enable MACing by the Service Provider.

- Every command SECURE_MSG_SEND that an application issues must be followed by exactly one command SECURE_MSG_RECEIVE that informs the Service Provider about the response from the chip or host. If no response is received (timeout or communication failure) the application must issue a SECURE_MSG_RECEIVE command with lpSecMsgIn->lpbMsg = NULL to inform the Service Provider about this fact.

- If a system is restarted after a SECURE_MSG_SEND was issued to the Service Provider but before the SECURE_MSG_RECEIVE was issued, the restart has the same effect as a SECURE_MSG_RECEIVE command with lpSecMsgIn->lpbMsg = NULL.

- Between a SECURE_MSG_SEND and the corresponding SECURE_MSG_RECEIVE no SECURE_MSG_SEND with the same lpSecMsgIn->wProtocol must be issued. Other WFS_CMD_PIN... commands – including SECURE_MSG_SEND / RECEIVE with different wProtocol – may be used.
8.3.2 Protocol WFS_PIN_PROTISOAS

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

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

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

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

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

The following bitmap positions are filled by the Service Provider:

- BMP1 - Trace-Nummer
- BMP52 - PAC
- BMP57 - Verschlüsselungsparameter (only the challenge values RNDMES and RNDPAC)
- BMP64 - MAC

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

The following bitmap positions are checked by the Service Provider and have to be filled by the application:

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

For additional documentation of authorization messages see [Ref. 27] – [Ref. 30].
8.3.3 Protocol WFS_PIN_PROTISOLZ

This protocol handles ISO8583 messages between a „Ladeterminal“ and a „Ladezentrale“ (LZ).

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

Both types of GeldKarte chip (type 0 = DEM, type 1 = EUR) are supported.

The following bitmap positions are filled by the Service Provider:

- BMP11: Trace-Nummer
- BMP57: Verschlüsselungsparameter (only the challenge value RNDMES)
- BMP64: MAC

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

The following bitmap positions are checked by the Service Provider and have to be filled by the application:

- Nachrichtentyp
- BMP3: Abwicklungskennzeichen
- BMP4: Transaktionsbetrag
- BMP12: Uhrzeit
- BMP13: Datum
- BMP25: Konditionscode
- BMP41: Terminal-ID
- BMP42: Betreiber-BLZ (caution: "Ladeentgelt“ also in BMP42 is not set by the EPP)
- BMP61: Online-Zeitpunkt
- BMP62: Chipdaten

The following bitmap positions are only checked if they are available:

- BMP43: Standort
- BMP60: Kontodaten Ladeterminal

For a documentation of the Ladezentrale interface see [Ref. 31].
8.3.4 Protocol WFS_PIN_PROTISOPS

This protocol handles ISO8583 messages between a terminal and a "Personalisierungsstelle" (PS). These messages are about OPT.

The Service Provider creates the whole message with WFS_CMD_PIN_SECURE_MSG_SEND, including message type and bitmap.

For a documentation of the Personalisierungsstelle interface see [Ref. 7].
8.3.5 Protocol WFS_PIN_PROTCHIPZKA

This protocol is intended to handle messages between the application and a GeldKarte. Both types of GeldKarte are supported. Both types of load transactions ("Laden vom Kartenkonto" and "Laden gegen andere Zahlungsmittel") are supported.

See the chapter "Command Sequence" below for the actions that Service Providers take for the various chip card commands.

Only the command APDUs to and the response APDUs from the chip must be passed to the Service Provider, the ATR (answer to reset) data from the chip is not passed to the Service Provider.

For a documentation of the chip commands used to load a GeldKarte see [Ref. 31].
8.3.6 Protocol WFS_PIN_PROTRAWDATA

This protocol is intended for vendor-specific purposes. Generally the use of this protocol is not recommended and should be restricted to issues that are impossible to handle otherwise.

For example a HSM that requires vendor-specific, cryptographically secured data formats for importing keys or terminal data may use this protocol.

Application programmers should be aware that the use of this command may prevent their applications from running on different hardware.
8.3.7 Protocol WFS_PIN_PROTPBM

This protocol handles host messages between a terminal and a host system, as specified by PBM protocol.

For documentation of this protocol see [Ref. 8] – [Ref. 13].

Some additions are defined to the PBM protocol in order to satisfy the German ZKA 3.0 PAC/MAC standard. See [Ref. 14].

The commands WFS_CMD_PIN_SECURE_MSG_SEND and WFS_CMD_PIN_SECURE_MSG_RECEIVE handle the PAC and MAC in the VARDATA ‘K’ or ‘Q’ subfield of transactions records and responses. The MAC in the traditional MACODE field is not affected.

In order to enable the Service Provider to understand the messages, the application must provide the messages according to the following rules:

- All alphanumeric fields must be coded in EBCDIC.
- Pre-Edit (padding and blank compression) must not be done by the application. The Service Provider will check the MACMODE field and will perform the pre-edit according to what the MACMODE field intends.
- In order to enable the Service Provider to find the vardata subfield ‘K’ or ‘Q’, it must be included in the message by the application, with the indicator ‘K’ or ‘Q’ and its length set.
- Because CARDDATA (track 2) and T3DATA (track 3) fields always take part in the MAC computation for a transaction record, these fields must be included in the message, even if they already have been sent to the host in a previous transaction record and the CI-Option SHORTREC prevents them from being sent again.
8.3.8 Protocol WFS_PIN_PROTHSMLDI

With this protocol an application can request information about the personalized OPT groups. The information returned consists of personalization record like in BMP62 of an OPT response but without MAC. Data format:

- XX XX VV - group ID and version number (BCD format)
- XX - number of LDIs within the group (BCD format)
- first LDI of the group
- last LDI of the group
- XX XX VV - group ID and version number (BCD format)
- etc. for several groups

Each LDI consists of:

- NN Number of the LDI
- 00 Alg. Code
- LL Length of the following data
- XX...XX data of the LDI

For each group ID the Service Provider must always return the standard LDI. LDI 01 must also be returned for groups AF XX VV. Further LDIs can be returned optionally.
8.3.9 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 does not 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 \textit{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:

<table>
<thead>
<tr>
<th>Command</th>
<th>lpbMsg in lpbSecMsgIn</th>
<th>lpbMsg in lpbSecMsgOut</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_SEND</td>
<td>Byte 0: 0x01 (Generate PAC)</td>
<td>Byte 0: key generation</td>
<td>Generates a session key for PAC generation and finally the PAC itself.</td>
</tr>
<tr>
<td></td>
<td>Byte 1: format (0 or 1)</td>
<td>Byte 1: key version</td>
<td>Determine generation and version values of Master-Key and return them along with the random value.</td>
</tr>
<tr>
<td></td>
<td>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)</td>
<td>Byte 18-25: PAC value (all values are binary values)</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>Byte 0: 0x02 (Get MAC Random)</td>
<td>Byte 0: key generation</td>
<td>Generates a session key for MAC generation (see next step below)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 1: key version</td>
<td>Determine generation and version values of Master-Key and return them along with the random value</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>Byte 0: 0x03 (Generate MAC)</td>
<td>Byte 0-7: generated MAC (binary value)</td>
<td>Generates MAC over bytes 1-n of the inbound message using the session key created in the previous step.</td>
</tr>
<tr>
<td></td>
<td>Byte 1-n: Message to be mac’ed (all values are binary values)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>Byte 0: 0x04 (Verify MAC)</td>
<td>N/a</td>
<td>Generates a session key using the Master key identified by key generation and version by using the random value passed in.</td>
</tr>
<tr>
<td></td>
<td>Byte 1: key generation</td>
<td></td>
<td>Generates a MAC for the message data passed in and compare the resulting MAC with the MAC passed in.</td>
</tr>
<tr>
<td></td>
<td>Byte 2: key version</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte 3-18: MAC random</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte 19-26: MAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte 27-n: Message to be verified (all values are binary values)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note: If no message has been received, this function must be called by omitting Bytes 1-n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 03 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 did not 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 does not 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 does not 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 does not 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.
8.3.10 Protocol WFS_PIN_PROTCIPINCHG

This protocol is intended to handle messages exchanged between the PIN pad and a GeldKarte, which are all related to the PIN change transaction.

Only Type-1-GeldKarte is supported, because the former Type-0-GeldKarte will no longer be used as it was a dedicated Deutsche Mark electronic purse only. The Type-1-GeldKarte is used for Euro currency.

The transaction types supported are:

- PIN-Activation ("PIN-Aktivierung")
- PIN-Activation after Failure ("PIN-Aktivierung nach Fehlerfall")
- PIN-Change ("PIN-Änderung")

See the command sequence section below for the actions that Service Providers take for the various chip card commands.

Only the command APDUs to and the response APDUs from the chip must be passed to the Service Provider, the ATR (answer to reset) data from the chip is not passed to the Service Provider.

For the complete documentation of the chip commands used for PIN-Change see [Ref. 34].
8.3.11 Protocol WFS_PIN_PROTPINCMP

This simple protocol is used to perform a comparison of two PINs entered into the PIN Pad. In order to be able to compare the PINs, the first value must be temporary stored while the second value is entered. The user will be prompted to enter the PIN twice. After the PIN has been entered for the first time, the PIN pad needs to store the PIN value into a temporary location. After the user has entered the PIN for the second time, the PIN pad has to compare both values.

This protocol consists of two subcommands. The first subcommand requests the PIN pad to save the PIN value entered by the WFS_CMD_PIN_GET_PIN command for subsequent comparison. The second subcommand forces the PIN pad to compare the PIN stored with the second value entered by the WFS_CMD_PIN_GET_PIN command. The status of the PIN comparison is returned in the output data.

See the command sequence section below for the actions that Service Providers take for this protocol.

8.3.11.1 Use of WFS_PIN_PROTPINCMP with non-GeldKarte ZKA PIN Management

For use with the non-GeldKarte ZKA PIN compare function (see [Ref 37]) there are two more subcommands “start PIN compare” and “end PIN compare”. These have to be called before entry of the first PIN and after querying of the PAC to signal the end of the PIN comparison, respectively.

This is the command sequence for the non-GeldKarte transaction:

<table>
<thead>
<tr>
<th>Flow</th>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpbMsg in lpbSecMsgIn</th>
<th>lpbMsg in lpbSecMsgOut</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN Compare</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start PIN comparison</td>
<td>SECURE_MSG_SEND</td>
<td>PINCMP</td>
<td>Byte 0: 0x00 (Start PIN compare)</td>
<td></td>
<td>Prepare EPP for PIN comparison. Output data buffer length is zero.</td>
</tr>
<tr>
<td>Let the user enter the new PIN for the first time.</td>
<td>GET_PIN</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>PIN entry.</td>
</tr>
<tr>
<td>Let the user enter the new PIN for the second time</td>
<td>GET_PIN</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>PIN entry.</td>
</tr>
<tr>
<td>Get the PAC of the new PIN via WFS_PIN_PROTGENAS or WFS_PIN_PROTISOAS (as usual).</td>
<td>SECURE_MSG_SEND</td>
<td>PINCMP</td>
<td>Byte 0: 0x02 (Compare PINs)</td>
<td>Byte 0: 0x00 when PIN does not match, and 0x01 when PIN does match.</td>
<td>Compare PIN values.</td>
</tr>
<tr>
<td>End PIN comparison.</td>
<td>SECURE_MSG_SEND</td>
<td>PINCMP</td>
<td>Byte 0: 0xFF (End PIN compare)</td>
<td></td>
<td>All PIN buffers are cleared. Output data buffer length is zero.</td>
</tr>
</tbody>
</table>
Please note that no other PIN commands apart from WFS_CMD_PIN_GET_PIN and WFS_CMD_PIN_SECURE_MSG_SEND as specified above are allowed inside a start / end PIN compare flow, with the exception of creating the PAC for the old PIN. While the old PIN always has to be entered (using WFS_CMD_PIN_GET_PIN) before the “Start PIN Compare”, the PAC for the old PIN may be created (using WFS_CMD_PIN_SECURE_MSG_SEND with wProtocol=WFS_PIN_PROTGENAS) after the “Start PIN Compare” if (enforced by the host protocol) the same session key SK_PAC has to be used for encrypting both the old and the new PIN.
8.3.12 Protocol WFS_PIN_PROTISOPINCHG

This protocol handles ISO8583 messages between an ATM and an authorization system (AS) related to the transactions:

- PIN-Activation ("PIN-Aktivierung")
- PIN-Activation after Failure ("PIN-Aktivierung nach Fehlerfall")
- PIN-Change ("PIN-Änderung")

The message types supported are:

- 0640 (PIN Change / PIN Activation Request)
- 0642 (Confirmation / Reversal Request for PIN Change / PIN Activation)
- 0643 (Confirmation Repeat Request for PIN Change / PIN Activation)
- 0650 (PIN Change / PIN Activation Response)
- 0652 (Confirmation / Reversal Response)

The following bitmap positions are filled by the Service Provider:

- BMP52 PAC
- BMP57 Verschlüsselungsparameter ($K_{Terminal}$ Generation, $K_{Terminal}$ Version, RNDMES and RNDPAC)
- BMP62 (EF_ID, EF_INFO, Record number of PIN, Key Version of $K_{Card}$, EF_FBZ, PAC, Random value returned by GET_CHALLENGE)
- BMP64 MAC

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

See the command sequence section below for the actions that Service Providers take for the various messages.

For the complete documentation of the messages used for PIN-Change see [Ref. 34].
8.3.13 Command Sequence

The following list shows the sequence of actions an application has to take for the various GeldKarte Transactions. Please note that this is a summary and is just intended to clarify the purpose of the chipcard-related WFS_CMD_PIN_... commands. In no way it can replace the ZKA specifications mentioned above.

<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preparation for Load/Unload</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>SELECT FILE DF BÖRSE</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td>recognize type of chip</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>READ RECORD EF ID</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>record EF_ID</td>
<td>store EF ID</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>READ RECORD EF_LLOG</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>record EF_LLOG</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>READ RECORD EF_BÖRSE</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>record EF_BÖRSE</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>READ_RECORD EF_BETRAG</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>record EF_BETRAG</td>
<td></td>
</tr>
</tbody>
</table>

<p>| Load against other ec-Card | | | |
| SECURE_MSG_SEND | CHIPZKA | for type 0 chips only | Command APDU |
| | | READ RECORD EF_KEYD | |
| SECURE_MSG_RECEIVE | CHIPZKA | record EF_KEYD | |
| SECURE_MSG_SEND | CHIPZKA | for type 1 chips only | Command APDU |
| | | GET KEYINFO | |
| SECURE_MSG_RECEIVE | CHIPZKA | Response APDU | |
| SECURE_MSG_SEND | CHIPZKA | Command APDU | GET CHALLENGE |
| SECURE_MSG_RECEIVE | CHIPZKA | Random number RND1 from Chip | store RND1 |
| SECURE_MSG_SEND | CHIPZKA | Command APDU | LADEN EINLEITEN with Secure Msg. |
| | | fill: | Terminal ID |
| | | - Traceno. | |
| | | - RND2 | |
| | | - MAC | |
| SECURE_MSG_RECEIVE | CHIPZKA | Response APDU | store response APDU for later check of ISOLZ message, BMP 62 |
| SECURE_MSG_SEND | ISOAZ | ISO8583 message 0200 | Authorization Request |
| | | Authorization Request | Fill: |
| | | - Traceno. (BMP 11) | - PAC (BMP 52) |
| | | - RND8ES + RND_PAC (BMP 57) | - MAC (BMP 64) |
| | | check other security relevant fields | |
| SECURE_MSG_RECEIVE | ISOAZ | ISO8583 message 0210 | Authorization Response |
| | | Authorization Response | check MAC and other security relevant fields |
| SECURE_MSG_SEND | ISOLZ | ISO8583 message 0200 | Ladeanfrage |
| | | Ladeanfrage | Fill: |
| | | - Traceno. (BMP 11) | - RND8ES (BMP 57) |
| | | - MAC (BMP 64) | check other security relevant fields. |
| | | check other security relevant fields. | |
| SECURE_MSG_RECEIVE | ISOLZ | ISO8583 message 0210 | Ladeantwort |
| | | Ladeantwort | check MAC and other security relevant fields, store BMP62 for later use in LADEN command. |
| SECURE_MSG_SEND | CHIPZKA | Command APDU | GET CHALLENGE |</p>
<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Random number RND3 from chip</td>
<td>store RND3</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU LADEN with Secure Msg.</td>
<td>provide complete command from BMP62 of ISOLZ response, compute command MAC</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td>check response MAC</td>
</tr>
<tr>
<td>GET_JOURNAL</td>
<td>ISOAZ</td>
<td>Vendor specific</td>
<td></td>
</tr>
<tr>
<td>GET_JOURNAL</td>
<td>ISOAZ</td>
<td>Vendor specific</td>
<td></td>
</tr>
<tr>
<td>Reversal of a Load against other ec-Card</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU SELECT FILE DF BÖRSE</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU GET CHALLENGE</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Random number RND5 from chip</td>
<td>store RND5</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU LADEN EINLEITEN with Secure Msg.</td>
<td>Fill: -Terminal ID -Traceno. -RND6 -Keyno. KGKLT -MAC</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td>store response APDU for later check of ISOLZ message, BMP 62</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOAZ</td>
<td>ISO8583 message 0400 Storno</td>
<td>Fill: -Traceno. (BMP 11) -PAC (BMP 52) -RNDMES + RNDPAC (BMP 57) -MAC (BMP 64) check other security relevant fields</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOAZ</td>
<td>ISO8583 message 0410 Storno Response</td>
<td>check MAC and other security relevant fields.</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOAZ</td>
<td>ISO8583 message 0400 Storno</td>
<td>Fill: -Traceno. (BMP 11) -RNDMES (BMP 57) -MAC (BMP 64) check other security relevant fields.</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOAZ</td>
<td>ISO8583 message 0410 Storno Response</td>
<td>check MAC and other security relevant fields.</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU GET CHALLENGE</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Random number RND7 from chip</td>
<td>store RND7</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU LADEN with Secure Msg.</td>
<td>provide complete command from BMP62 of ISOLZ response, compute command MAC</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td>check response MAC</td>
</tr>
<tr>
<td>GET_JOURNAL</td>
<td>ISOAZ</td>
<td>Vendor specific</td>
<td></td>
</tr>
<tr>
<td>GET_JOURNAL</td>
<td>ISOAZ</td>
<td>Vendor specific</td>
<td></td>
</tr>
</tbody>
</table>

PIN Verification Type 0

<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU GET CHALLENGE</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Random number RND0 from chip</td>
<td>store RND0</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td>Command APDU EXTERNAL AUTHENTICATE</td>
<td>fill -Keyno. KINFO -ENCRND</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td>Message Type</td>
<td>Chip</td>
<td>Command APDU</td>
<td>Function</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>PUT DATA</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>READ RECORD EF_INFO with Secure Messaging</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>GET CHALLENGE</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>VERIFY</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>PIN Verification Type 1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>GET KEYINFO</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>GET CHALLENGE</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Random number RND2 from chip</td>
<td>store RND2</td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>VERIFY</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Random number RND0 from chip</td>
<td>store RND0</td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>MUTUAL AUTHENTICATE</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>VERIFY</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>Laden vom Kartenkonto</strong> (both types)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>LADEN EINLEITEN</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>ISOLZ</td>
<td>ISO8583 message 0200 Ladeanfrage</td>
<td>fill</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Terminal ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Trace No.</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>ISOLZ</td>
<td>ISO8583 message 0210 Ladeantwort</td>
<td>check MAC and other security relevant fields.</td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>LADEN</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>GET_JOURNAL</strong></td>
<td>ISOLZ</td>
<td>Vendor specific</td>
<td></td>
</tr>
</tbody>
</table>

**Reversal of a „Laden vom Kartenkonto“**

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Chip</th>
<th>Command APDU</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>SELECT FILE DF_BÖRSE</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>CHIPZKA</td>
<td>Command APDU</td>
<td>LADEN EINLEITEN</td>
</tr>
<tr>
<td><strong>SECURE_MSG_RECEIVE</strong></td>
<td>CHIPZKA</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td><strong>SECURE_MSG_SEND</strong></td>
<td>ISOLZ</td>
<td>ISO8583 message 0400 Storno</td>
<td>fill</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Traceno. (BMP 11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- RNDMES (BMP 57)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- MAC (BMP 64)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>check other security relevant fields.</td>
</tr>
<tr>
<td>Message Type</td>
<td>Receiver</td>
<td>Sender</td>
<td>Action</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOLZ</td>
<td>CHIPZKA</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET_JOURNAL</td>
<td>ISOLZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unload</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOLZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOLZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOLZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOLZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOLZ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Repeated Messages (Stornowiederholung / Entladequittungswiederholung)**

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Receiver</th>
<th>Sender</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOLZ</td>
<td></td>
<td></td>
<td>ISO8583 message 0401 or Entladequittung 0201 Stornowiederholung</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOLZ</td>
<td></td>
<td></td>
<td>ISO8583 message 0410 or Entladequittung 0202 Stornoantwort 410</td>
</tr>
<tr>
<td>GET_JOURNAL</td>
<td>ISOLZ</td>
<td></td>
<td></td>
<td>Vendor specific</td>
</tr>
</tbody>
</table>

**Command WFS_CMD_PIN_ wProtocol WFS_PIN_P ROT**

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Receiver</th>
<th>Sender</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation for PIN Change</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
<td>Command APDU READ RECORD EF_ID</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
<td>Command APDU READ RECORD EF_ID</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
<td>Command APDU Record EF_ID</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPZKA</td>
<td></td>
<td></td>
<td>Command APDU GET CHALLENGE</td>
</tr>
<tr>
<td>Command</td>
<td>wProtocol</td>
<td>lpbMsg</td>
<td>Service Provider’s actions</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Random number RND0 from Chip</td>
<td>Store RND0</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPPINCHG</td>
<td>Command APDU READ RECORD EF_INFO</td>
<td>Fill RND1</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Response APDU Record EF_INFO</td>
<td>Check MAC, Store EF_INFO Will be inserted into BMP62 of a PIN Change request</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPPINCHG</td>
<td>Command APDU GET KEYINFO</td>
<td>Store version byte Will be inserted into BMP62 of a PIN Change request</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Response APDU Record EF_INFO</td>
<td>Store version byte Will be inserted into BMP62 of a PIN Change request</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPPINCHG</td>
<td>Command APDU SEARCH RECORD ‘01’ of EF_PWDD</td>
<td>Store record number Will be inserted into BMP62 of a PIN Change request</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Response APDU</td>
<td>Store record number Will be inserted into BMP62 of a PIN Change request</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPPINCHG</td>
<td>Command APDU READ RECORD EF_FBZ</td>
<td>Store record number Will be inserted into BMP62 of a PIN Change request</td>
<td></td>
</tr>
<tr>
<td>PIN Verification</td>
<td>CHIPPINCHG</td>
<td>Random number RND0 from chip</td>
<td>Store RND0</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>CHIPPINCHG</td>
<td>Command APDU GET KEYINFO</td>
<td>Fill ENC0</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>CHIPPINCHG</td>
<td>Response APDU</td>
<td>Check ENC1</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>CHIPPINCHG</td>
<td>Command APDU GET CHALLENGE</td>
<td>Provide complete command APDU</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>CHIPPINCHG</td>
<td>Response APDU</td>
<td>Check MAC</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>HSMPINCMP</td>
<td>Byte 0: 0x01 (Save PIN)</td>
<td>Save the PIN value entered for subsequent compare. Output data buffer length is zero.</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>HSMPINCMP</td>
<td>Byte 0: 0x02 (Compare PINs)</td>
<td>Compare PIN values. Returns Byte 0: as 0x00 when PIN does not match, and 0x01 when PIN does match. Create PAC for new PIN if values match</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>CHIPPINCHG</td>
<td>Command APDU MANAGE SECURITY ENVIRONMENT</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td>PIN Change</td>
<td>CHIPPINCHG</td>
<td>Command APDU GET CHALLENGE</td>
<td>Hold the user enter the PIN for the second time, by invoking the command WFS_CMD_PIN_GET_PIN</td>
<td></td>
</tr>
</tbody>
</table>

Let the user enter the PIN for the second time, by invoking the command WFS_CMD_PIN_GET_PIN.
<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpbMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Random number RND0 from Chip</td>
<td>Store RND0 Will be inserted into BMP62 of a PIN Change request</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOPINCHG</td>
<td>ISO8583 Message 0640</td>
<td>Fill - PAC old PIN (BMP52) - K_Terminal generation + K_Terminal version + RND_MES (BMP57) - Chip Data (BMP62) with PAC of new PIN - MAC (BMP64)</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0650</td>
<td>Check MAC</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPPINCHG</td>
<td>Command APDU from BMP62</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Response APDU</td>
<td></td>
</tr>
</tbody>
</table>

**PIN Change Confirmation/Repeated Confirmation**

<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpbMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0642 or 0643 BMP25 = 00</td>
<td>Fill - K_Terminal generation + K_Terminal version + RND_MES (BMP57) - Chip Data (BMP62) with PAC of new PIN - MAC (BMP64)</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0652</td>
<td>Check MAC</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0642 or 0643 BMP25 ≠ 00</td>
<td>Fill - K_Terminal generation + K_Terminal version + RND_MES (BMP57) - Chip Data (BMP62) with PAC of old PIN - MAC (BMP64)</td>
</tr>
</tbody>
</table>

**PIN Change Reversal/Repeated Reversal**

<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpbMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0640</td>
<td>Fill - PAC entered PIN (BMP52) - K_Terminal generation + K_Terminal version + RND_MES + RND_PAC (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0650</td>
<td>Check MAC</td>
</tr>
</tbody>
</table>

**PIN Activation after failure**

<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpbMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0640</td>
<td>Fill - PAC entered PIN (BMP52) - K_Terminal generation + K_Terminal version + RND_MES + RND_PAC (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)</td>
</tr>
</tbody>
</table>

**PIN Activation**

<table>
<thead>
<tr>
<th>Command WFS_CMD_PIN_</th>
<th>wProtocol WFS_PIN_PROT</th>
<th>lpbMsg</th>
<th>Service Provider’s actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPPINCHG</td>
<td>Command APDU</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Response APDU</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>CHIPPINCHG</td>
<td>Command APDU</td>
<td></td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>CHIPPINCHG</td>
<td>Random number RND0 from Chip</td>
<td>Store RND0 Will be inserted into BMP62 of a PIN Activation request</td>
</tr>
<tr>
<td>SECURE_MSG_SEND</td>
<td>ISOPINCHG</td>
<td>ISO8583 Message 0640</td>
<td>Fill - PAC entered PIN (BMP52) - K_Terminal generation + K_Terminal version + RND_MES + RND_PAC (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)</td>
</tr>
<tr>
<td>SECURE_MSG_RECEIVE</td>
<td>ISOPINCHG</td>
<td>ISO8583 message 0650</td>
<td>Check MAC</td>
</tr>
<tr>
<td>Secure Message Send</td>
<td>Command APDU</td>
<td>Secure Message Receive</td>
<td>Response APDU</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>CHIPPING</td>
<td>Command APDU from BMP62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHIPPING</td>
<td>Response APDU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command APDU</td>
<td>Command APDU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHIPPING</td>
<td>Command APDU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHIPPING</td>
<td>Command APDU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHIPPING</td>
<td>Command APDU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random number RND0</td>
<td>Store RND0 Will be inserted into BMP62 of a PIN Activation confirmation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISO8583 message 0642 or 0643</td>
<td>Fill - KTerminal generation + KTerminal version + RNDMES (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISO8583 message 0652</td>
<td>Check MAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command APDU</td>
<td>Command APDU from BMP62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response APDU</td>
<td>Response APDU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8.4 EMV Support

EMV support by this specification consists in the ability of importing Certification Authority and Chip Card Public Keys, creating the PIN blocks for offline PIN verification and verifying static and dynamic data. This section is used to further explain concepts and functionality that needs further clarification.

The PIN service is able to manage the EMV chip card regarding the card authentication and the RSA local PIN verification. Two steps are mandatory in order to reach these two functions: The loading of the keys which come from the Certification Authorities or from the card itself, and the EMV PIN block management.

The Service Provider is responsible for all key validation during the import process. The application is responsible for management of the key lifetime and expiry after the key is successfully imported.

8.4.1 Keys loading

The final goal of an application is to retrieve the keys located on card to perform the operations of authentication or local PIN check (RSA encrypted). These keys are provided by the card using EMV certificates and can be retrieved using a Public Key provided by a Certification Authority. The application should first load the keys issued by the Certification Authority. At transaction time the application will use these keys to load the keys that the application has retrieved from the chip card.

Certification Authority keys

These keys are provided in the following formats:

- Plain text.
- Plain Text with EMV 2000 Verification Data (See [Ref. 4] under the reference section for this document).
- EPI CA (or self signed) format as specified in the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5] under the reference section for this document).
- PKCSV1_5 encrypted (as used by GIECB in France) (See [Ref. 15] under the reference section for this document).

EPI CA format

The following table corresponds to table 4 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5]) and identifies the Europay Public Key (self-certified) and the associated data:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID of Certificate Subject</td>
<td>5</td>
<td>RID for Europay</td>
<td>Binary</td>
</tr>
<tr>
<td>Europay public key Index</td>
<td>1</td>
<td>Europay public key Index</td>
<td>Binary</td>
</tr>
<tr>
<td>Subject public key Algorithm</td>
<td>1</td>
<td>Algorithm to be used with the Europay public key</td>
<td>Binary</td>
</tr>
<tr>
<td>Indicator</td>
<td></td>
<td>Index set to 0x01</td>
<td></td>
</tr>
<tr>
<td>Subject public key Length</td>
<td>1</td>
<td>Length of the Europay public key Modulus</td>
<td>Binary</td>
</tr>
<tr>
<td>Subject public key Exponent</td>
<td>1</td>
<td>Length of the Europay public key Exponent</td>
<td>Binary</td>
</tr>
<tr>
<td>Leftmost Digits of Subject</td>
<td>Nca-37</td>
<td>Nca-37 most significant bytes of the Europay</td>
<td>Binary</td>
</tr>
<tr>
<td>public key</td>
<td></td>
<td>public key Modulus</td>
<td></td>
</tr>
<tr>
<td>Subject public key Remainder</td>
<td>37</td>
<td>37 least significant bytes of the Europay public</td>
<td>Binary</td>
</tr>
<tr>
<td>key Modulus</td>
<td></td>
<td>key Modulus</td>
<td></td>
</tr>
<tr>
<td>Subject public key Exponent</td>
<td>1</td>
<td>Exponent for Europay key</td>
<td>Binary</td>
</tr>
<tr>
<td>Subject public key Certificate</td>
<td>Nca</td>
<td>Output of signature algorithm</td>
<td>Binary</td>
</tr>
</tbody>
</table>

Table 1

The following table corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 and identifies the Europay Public Key Hash code and associated data:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID of Certificate Subject</td>
<td>5</td>
<td>RID for Europay</td>
<td>Binary</td>
</tr>
<tr>
<td>Europay public key Index</td>
<td>1</td>
<td></td>
<td>Europay public key Index</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---</td>
<td>---</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Subject public key Algorithm Indicator</td>
<td>1</td>
<td>Algorithm to be used with the Europay public key Index, set to 0x01</td>
<td>Binary</td>
</tr>
<tr>
<td>Certification Authority public key Check Sum</td>
<td>20</td>
<td>Hash-code for Europay public key</td>
<td>Binary</td>
</tr>
</tbody>
</table>

Table 2

Table 2 corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5]).

**Chip card keys**

These keys are provided as EMV certificates which come from the chip card in a multiple layer structure (issuer key first, then the ICC keys). Two kinds of algorithm are used with these certificates in order to retrieve the keys: One for the issuer key and the other for the ICC keys (ICC Public Key and ICC PIN encipherment key). The associated data with these algorithms – The PAN (Primary Account Number) and the SDA (Static Data to be Authenticated) - come also from the chip card.
8.4.2 PIN Block Management

The PIN block management is done through the command WFS_CMD_PIN_GET_PINBLOCK. A new format WFS_PIN_FORMEMV has been added to indicate to the PIN service that the PIN block must follow the requirements of the EMVco, Book2 – Security & Key management Version 4.0 document. The parameter lpsCustomerData is used in this case to transfer to the PIN service the challenge number coming from the chip card. The final encryption must be done using a RSA Public Key. Please note that the application is responsible to send the PIN block to the chip card inside the right APDU.
8.4.3 SHA-1 Digest

The SHA-1 Digest is a hash algorithm used by EMV in validating ICC static and dynamic data item. The SHA-1 Digest is supported through the WFS_CMD_PIN_DIGEST command. The application will pass the data to be hashed to the Service Provider. Once the encryptor completes the SHA-1 hash code, the Service Provider will return the 20-byte hash value back to the application.
8.5 French Cartes Bancaires

“Groupement des Cartes Bancaires” from France has specified a cryptographic architecture for ATM networks. See the document [Ref. 15] for details.

The XFS command WFS_CMD_PIN_ENC_IO with the protocol WFS_PIN_ENC_PROT_GIECB is used for:

- ATM initialization
- Renewal of ATM master key
- Renewal of HOST master key
- Generation and loading of key transport key

Keys loaded or generated with WFS_CMD_PIN_ENC_IO get names like any other keys in a XFS PIN service. WFS_INF_PIN_KEY_DETAIL[_EX] shows the key with this name and the name may be used with WFS_CMD_PIN_IMPORT_KEY[_EX] to delete a key.

8.5.1 Data Structure for WFS_CMD_PIN_ENC_IO

Data will be transferred as tag-length-value (TLV) structure, encoded according to the distinguished encoding rules (DER) defined in [Ref. 16].

The following is a list of top level tags defined for the use with WFS_PIN_ENC_PROT_GIECB. All these tags have the APPLICATION class, therefore the Identifier Octets are (binary):

- 0 1 0 n n n n - for the primitive types
- 0 1 1 n n n n - for the constructed types

<table>
<thead>
<tr>
<th>Tag Number</th>
<th>Primitive / Constructed</th>
<th>Identifier Octet</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P</td>
<td>0x40</td>
<td><strong>Protocol Version</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The INTEGER value zero for this version of the protocol</td>
</tr>
<tr>
<td>1</td>
<td>P</td>
<td>0x41</td>
<td><strong>Interchange Code</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>An ASCII string holding one of the interchange codes defined in [Ref. 15], e.g. “HRN-H1”</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0x62</td>
<td><strong>Interchange Data</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The data items as defined by [Ref.15], see table below for details</td>
</tr>
<tr>
<td>3</td>
<td>P</td>
<td>0x43</td>
<td><strong>Key Name</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>An ASCII string holding the name for the key being loaded or generated.</td>
</tr>
</tbody>
</table>

The Interchange Data (Tag 2) is constructed from data items where tag numbers of the sub-tags from 1 to 23 correspond to the data item numbers (“N° donnée”) as defined in section 3.1 of [Ref. 15]. Some of the data items consist of data elements, for these the constructed encoding will be used. For data items with no data elements the primitive encoding will be used.

All Tags have the CONTEXT class, therefore the Identifier Octets are (binary):

- 1 0 0 n n n n - for the primitive types
- 1 0 1 n n n n - for the constructed types
<table>
<thead>
<tr>
<th>Tag (=Data Item No)</th>
<th>Primitive / Constructed</th>
<th>Identifier Octet</th>
<th>Data Item Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>0xA1</td>
<td>IdKG</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0xA2</td>
<td>KTK-encrypted</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0xA3</td>
<td>KGp</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>0xA4</td>
<td>KDp</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>0xA5</td>
<td>SnSCD</td>
</tr>
<tr>
<td>6</td>
<td>P</td>
<td>0x86</td>
<td>Rand</td>
</tr>
<tr>
<td>7</td>
<td>P</td>
<td>0x87</td>
<td>HOST authentication</td>
</tr>
<tr>
<td>8</td>
<td>P</td>
<td>0x88</td>
<td>KDp signature</td>
</tr>
<tr>
<td>9</td>
<td>P</td>
<td>0x89</td>
<td>KGp signature</td>
</tr>
<tr>
<td>10</td>
<td>P</td>
<td>0x8A</td>
<td>KTK signature</td>
</tr>
<tr>
<td>11</td>
<td>P</td>
<td>0x8B</td>
<td>KT-encrypted</td>
</tr>
<tr>
<td>12</td>
<td>P</td>
<td>0x8C</td>
<td>Ksc-encrypted</td>
</tr>
<tr>
<td>13</td>
<td>P</td>
<td>0x8D</td>
<td>PIN cryptogram</td>
</tr>
<tr>
<td>14</td>
<td>P</td>
<td>0x8E</td>
<td>Seal</td>
</tr>
<tr>
<td>15</td>
<td>P</td>
<td>0x8F</td>
<td>Thumbprint of KDp</td>
</tr>
<tr>
<td>16</td>
<td>P</td>
<td>0x90</td>
<td>Thumbprint of KGp</td>
</tr>
<tr>
<td>17</td>
<td>C</td>
<td>0xB1</td>
<td>IdKD</td>
</tr>
<tr>
<td>18</td>
<td>C</td>
<td>0xB2</td>
<td>IdKTK</td>
</tr>
<tr>
<td>19</td>
<td>C</td>
<td>0xB3</td>
<td>IdKT</td>
</tr>
<tr>
<td>20</td>
<td>C</td>
<td>0xB4</td>
<td>IdKSC</td>
</tr>
<tr>
<td>21</td>
<td>P</td>
<td>0x95</td>
<td>Manufacturer</td>
</tr>
<tr>
<td>22</td>
<td>C</td>
<td>0xB6</td>
<td>SCD type</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>0xB7</td>
<td>Firmware version</td>
</tr>
</tbody>
</table>

Inside the constructed data items, primitive encoding is used for the data elements, all tags having CONTEXT class with tag numbers corresponding to the data element numbers ("N° d’élément de donnée") as defined in section 3.1 of [Ref. 15].

Example:
The example shows the DER encoding of the input for a WFS_CMD_PIN_ENCIO command, for the interchange “GIN-H5”. All data except the 128 byte content of data item 7 is shown in hexadecimal (0x omitted for the sake of readability).

```
40 01 00      (tag / length / value for Protocol Version 0)
41 06 47 49 4E 2D 47 34  (tag / length / value for Interchange Code “GIN-H5"
62 81 B5      (tag / length for Interchange Data)
A1 14         (tag / length for data item 1)
   81 01 00      (data element 1)
   82 0C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (data element 2)
   83 01 00      (data element 3)
A5 10         (tag / length for data item 5)
   81 03 00 00 00      (data element 1)
   82 09 00 00 00 00 00 00 00 00 00 00 00 (data element 2)
   86 08 00 00 00 00 00 00 00 00 00 (tag / length / value for data item 6)
87 81 80 <128 bytes> (tag / length / value for data item 7)
43 05 4D 59 4B 45 59 (tag / length / value for Key Name “MYKEY”)```
### 8.5.2 Command Sequence

The following list shows the sequence of actions an application has to take for the various *Cartes Bancaires* interchanges.

- **GIN (ATM initialization)**

<table>
<thead>
<tr>
<th>Action</th>
<th>Interchange Code</th>
<th>Key Name</th>
<th>Input Data Items</th>
<th>Output Data Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thumbprint supplied by host via external channel (GIN-H1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GIN-G2</td>
<td></td>
<td></td>
<td>21,22,23</td>
</tr>
<tr>
<td>Host Communication (GIN-G2 / GIN-H3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GIN-H3</td>
<td>Key Name for KG</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GIN-G4</td>
<td></td>
<td></td>
<td>5,6,1</td>
</tr>
<tr>
<td>Host Communication (GIN-G4 / GIN-H5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GIN-H5</td>
<td>Key Name for KD</td>
<td>5,6,1,7</td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GIN-G6</td>
<td></td>
<td></td>
<td>5,4,8</td>
</tr>
<tr>
<td>Host Communication (GIN-G6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GIN-G7</td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>Send thumbprint to host via external channel (GIN-G7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **GRN (Renewal of ATM Master Key)**

<table>
<thead>
<tr>
<th>Action</th>
<th>Interchange Code</th>
<th>Key Name</th>
<th>Input Data Items</th>
<th>Output Data Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GRN-G1</td>
<td></td>
<td></td>
<td>5,6,1</td>
</tr>
<tr>
<td>Host Communication (GRN-G1 / GRN-H2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GRN-H2</td>
<td>Key Name for KD</td>
<td>5,6,1,7</td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GRN-G3</td>
<td></td>
<td></td>
<td>5,4,8,17</td>
</tr>
<tr>
<td>Host Communication (GRN-G3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>GRN-C or GRN-R</td>
<td></td>
<td></td>
<td>17</td>
</tr>
</tbody>
</table>

The Interchange codes “GRN-C” to commit the transaction resp. “GRN-R” to roll back the transactions are an addition to those defined in [Ref. 15].

- **HRN (Renewal of HOST Master Key)**

<table>
<thead>
<tr>
<th>Action</th>
<th>Interchange Code</th>
<th>Key Name</th>
<th>Input Data Items</th>
<th>Output Data Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Communication (HRN-H1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>HRN-H1</td>
<td>Key Name for KG</td>
<td>3,9,1</td>
<td></td>
</tr>
</tbody>
</table>
- **DKT (Generation and Loading of KTK)**

<table>
<thead>
<tr>
<th>Action</th>
<th>Interchange Code</th>
<th>Key Name</th>
<th>Input Data Items</th>
<th>Output Data Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>DKT-G1</td>
<td></td>
<td></td>
<td>5,6</td>
</tr>
<tr>
<td>Host Communication (DKT-G1 / DKT-H2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WFS_CMD_PIN_ENCIO</td>
<td>DKT-H2</td>
<td>Key Name for KTK</td>
<td>5,6,2,10,1,17</td>
<td></td>
</tr>
</tbody>
</table>
8.6 Secure Key Entry

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

8.6.1 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 `fwKeyEntryMode` 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 PIN pad (i.e. the product of `wRows` by `wColumns`). On a regular shaped PIN pad 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.

8.6.1.1 `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 `lpFuncKeyDetail` parameter. Any positions on the PIN pad that are not used must be defined as a WFS_PIN_FK_UNUSED in the `ulFK` and `ulShiftFK` field of the `lppHexKeys` structure.

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

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.

8.6.1.2 `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 `lpFuncKeyDetail` parameter. Other function keys as defined by the `lpFuncKeyDetail` parameter that can be enabled must also be reported. Any positions on the PIN pad 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 to F are accessed by using the shift key in combination with another function key, e.g. shift-0 (zero) is hex digit 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.

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

### 8.6.1.3 *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 *lpFuncKeyDetail* parameter. Other function keys as defined by the *lpFuncKeyDetail* parameter that can be enabled must also be reported. Any positions on the PIN pad 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.

<table>
<thead>
<tr>
<th>Index</th>
<th>usXPos</th>
<th>usYPos</th>
<th>usxSize</th>
<th>usYSize</th>
<th>ulFK</th>
<th>ulShiftFK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>250</td>
<td>200</td>
<td>FK_1</td>
<td>FK_B</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>0</td>
<td>250</td>
<td>200</td>
<td>FK_2</td>
<td>FK_C</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>0</td>
<td>250</td>
<td>200</td>
<td>FK_3</td>
<td>FK_D</td>
</tr>
<tr>
<td>3</td>
<td>750</td>
<td>0</td>
<td>250</td>
<td>200</td>
<td>FK_CLEAR</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>200</td>
<td>250</td>
<td>200</td>
<td>FK_4</td>
<td>FK_E</td>
</tr>
<tr>
<td>5</td>
<td>250</td>
<td>200</td>
<td>250</td>
<td>200</td>
<td>FK_5</td>
<td>FK_F</td>
</tr>
<tr>
<td>6</td>
<td>500</td>
<td>200</td>
<td>250</td>
<td>200</td>
<td>FK_6</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>7</td>
<td>750</td>
<td>200</td>
<td>250</td>
<td>200</td>
<td>FK_CANCEL</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>400</td>
<td>250</td>
<td>200</td>
<td>FK_7</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>9</td>
<td>250</td>
<td>400</td>
<td>250</td>
<td>200</td>
<td>FK_8</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>400</td>
<td>250</td>
<td>200</td>
<td>FK_9</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>11</td>
<td>750</td>
<td>400</td>
<td>250</td>
<td>200</td>
<td>FK_ENTER</td>
<td>FK_UNUSED</td>
</tr>
</tbody>
</table>
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 `lpFuncKeyDetail` parameter. 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.

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 PIN pad 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.
<table>
<thead>
<tr>
<th>Index</th>
<th>usXPos</th>
<th>usYPos</th>
<th>usxSize</th>
<th>usYSize</th>
<th>ulFK</th>
<th>ulShiftFK</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>930</td>
<td>414</td>
<td>60</td>
<td>180</td>
<td>FK_CLEAR</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>12</td>
<td>780</td>
<td>612</td>
<td>40</td>
<td>180</td>
<td>FK_UNUSED</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>13</td>
<td>830</td>
<td>612</td>
<td>40</td>
<td>180</td>
<td>FK_0</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>14</td>
<td>880</td>
<td>612</td>
<td>40</td>
<td>180</td>
<td>FK_UNUSED</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>15</td>
<td>930</td>
<td>612</td>
<td>60</td>
<td>180</td>
<td>FK_UNUSED</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>16</td>
<td>680</td>
<td>810</td>
<td>40</td>
<td>180</td>
<td>FK_A</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>17</td>
<td>730</td>
<td>810</td>
<td>40</td>
<td>180</td>
<td>FK_B</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>18</td>
<td>780</td>
<td>810</td>
<td>40</td>
<td>180</td>
<td>FK_C</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>19</td>
<td>830</td>
<td>810</td>
<td>40</td>
<td>180</td>
<td>FK_D</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>20</td>
<td>880</td>
<td>810</td>
<td>40</td>
<td>180</td>
<td>FK_E</td>
<td>FK_UNUSED</td>
</tr>
<tr>
<td>21</td>
<td>930</td>
<td>810</td>
<td>60</td>
<td>180</td>
<td>FK_F</td>
<td>FK_UNUSED</td>
</tr>
</tbody>
</table>
8.6.2 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.
9. Appendix-B (Country Specific WFS_CMD_PIN_ENC_IO protocols)

This section is used for country-specific extensions to the WFS_CMD_PIN_ENC_IO command.

9.1 Luxemburg Protocol

The general XFS command WFS_CMD_PIN_ENC_IO is used to communicate transparently with the security module (see also command specifications).

In particular, to access the Luxembourg encryption commands defined in the following paragraphs, the input structure WFSPINENCIO of the WFS_CMD_PIN_ENC_IO command has to be defined as follows:

**Input Param**

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

- **wProtocol**
  - Must be set to the constant WFS_PIN_ENC_PROT_LUX.

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

- **lpvData**
  - Points to an input structure that contains the data specific to the Luxemburg protocol that has to be sent to the encryption module. This input structure is specific for each command defined in the protocol (see following paragraphs), but has following general form:

```c
LPPROTLUXIN    lpvData;
```

```c
typedef struct _prot_lux_in
{
  WORD     wCommand;
  ... Command Input Data ... 
} PROTLUXIN, *LPPROTLUXIN;
```

- **wCommand**
  - Specifies the command that has to be executed in the security module.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_CMD_ENC_IO_LUX_LOAD_APPKEY</td>
<td>Load an Application Key.</td>
</tr>
<tr>
<td>WFS_CMD_ENC_IO_LUX_GENERATE_MAC</td>
<td>Generate the CBC-MAC.</td>
</tr>
<tr>
<td>WFS_CMD_ENC_IO_LUX_CHECK_MAC</td>
<td>Check the CBC-MAC.</td>
</tr>
<tr>
<td>WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK</td>
<td>Build the PIN block.</td>
</tr>
<tr>
<td>WFS_CMD_ENC_IO_LUX_DECRYPT_TDES</td>
<td>Decrypt data.</td>
</tr>
<tr>
<td>WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES</td>
<td>Encrypt data.</td>
</tr>
</tbody>
</table>

- **... Command Input Data ...**
  - Specifies the command input data. This field is specific for each command defined in the protocol (see following paragraphs).

In the same way, to access the results of the private Luxembourg encryption commands, the output structure LPWFSPINENCIO of the WFS_CMD_PIN_ENC_IO command will be as follows:

**Output Param**

```c
LPWFSPINENCIO lpEncIoOut;
```

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

- **wProtocol**
  - Is set to the constant WFS_PIN_ENC_PROT_LUX.
ulDataLength
Specifies the length in bytes of the structure pointed to by the following field lpvData.

lpvData
Points to a PROTLUXOUT structure that contains the reply data specific to the Luxembourg protocol. This output structure is specific for each command defined in the protocol (see following paragraphs), but has following general form:

```c
typedef struct _prot_lux_out
{
    WORD     wCommand;
    WORD     wResult;
    ... Command Output Data ...
} PROTLUXOUT, *LPPROTLUXOUT;
```

wCommand
Specifies the command that has to be executed in the encryption module. This field contains the same value as the corresponding field in the input structure.

wResult
Specifies the command reply codes specific for this protocol. Possible general values for the Luxembourg protocol are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROT_LUX_SUCCESS</td>
<td>Command terminated correctly.</td>
</tr>
<tr>
<td>PROT_LUX_ERR_INVALID_CMD</td>
<td>Invalid command. The wCommand issued is not valid or not supported.</td>
</tr>
<tr>
<td>PROT_LUX_ERR_INVALID_DATA</td>
<td>The data structure passed as input parameter for the command contains invalid or incoherent data.</td>
</tr>
<tr>
<td>PROT_LUX_ERR_INVALID_KEY</td>
<td>The key needed for the operation was not loaded or is invalid. This operation failed.</td>
</tr>
</tbody>
</table>

... Command Output Data ...

Specifies the command output data. This field is specific for each command defined in the protocol (see following paragraphs). In the case of an error, the command specific structure is returned, but only the wCommand and the wResult fields are valid.

Comments
Luxembourg encryption commands defined in the following paragraphs will return the generic error PROT_LUX_ERR_INVALID_DATA when the input data is invalid.
9.1.1 WFS_CMD_ENC_IO_LUX_LOAD_APPKEY

Description

This command can be used to load an Application Key and to replace the Transport Key. Once the keys are loaded the encryptor will use the keys to do the other commands.

The encryptor will use the Application Key to obtain a random encrypted session key needed for the PIN Encryption, the MAC Computation and the Data Encryption/Decryption.

The application will use the Transport Key for loading the other keys (MK_MAC, MK_PAC and MK_ENC) into the encryptor.

When this command is used for replacing the Transport Key, the new Transport key is provided encrypted by the existing Transport Key.

The generation of the first Transport Key is the responsibility of the Authorization Center in Luxemburg (CETREL). The loading method of the first Transport Key into the encryptor is vendor dependent.

Keys loaded through this command are reported through the WFS_INF_PIN_KEY_DETAIL and WFS_INF_PIN_KEY_DETAIL_EX commands.

Keys loaded through this command do not require to be deleted before the application can replace them.

To access this command, the structure WFSPINENCIO of the WFS_CMD_PIN_ENC_IO command has to be defined as required by the Luxembourg protocol (see general definition in the first paragraph). The only definitions specific to this command are the input and output structures pointed to by the lpvData fields. They are defined as follows:

Input Param

LPPROT_TLUXLOADAPPKEY lpvData;

typedef struct _prot_lux_load_app_key_in
{
    WORD wCommand;
    LPSTR lpsKeyName;
    LPSTR lpsSequenceNumber;
    LPWFSXDATA lpxKeyData;
} PROTLUXLOADAPPKEYIN, *LP PROTLUXLOADAPPKEYIN;

wCommand
Is set to WFS_CMD_ENC_IO_LUX_LOAD_APPKEY.

lpsKeyName
This field contains the name of the key to be loaded. The Service Provider will right pad the lpsKeyName to 20 bytes with char 0x20.

Allowed values are:

- “MK_MAC” for the MAC key. Used for MAC calculation only.
- “MK_PAC” for the PIN block key. Used for PIN block construction only.
- “MK_ENC” for the ENC/DEC key. Used for data encryption/decryption only.
- “BANK_TRANS_KEY” for the Transport Key. It can only be used for loading the other keys (MK_MAC, MK_PAC and MK_ENC) into the encryptor.

lpsSequenceNumber
This field is defined by the Authorization Center in Luxemburg (CETREL) and contains a 4 bytes key logic number as follows:

- Least significant 2 bytes represent the Key Generation
- Most significant 2 bytes represent the Key Version

The key logic number will contribute in the MAC calculation, in the PIN block construction and in the Data Encryption/Decryption.

Allowed values are:

- “2001” for the MK_MAC key
- “2002” for the MK_PAC key
- “2003” for the MK_ENC key
- “2004” for the BANK_TRANS_KEY encrypted by the existing BANK_TRANS_KEY

`lpxKeyData`

`lpxKeyData` contains the 40 bytes of the Key data in ZKA key-file format (encrypted key of 16 bytes, HASH of 16 bytes and MAC of 8 bytes).

The MAC in the `lpxKeyData` is calculated with the contribution of the values from the `lpsKeyName` (20 bytes), `lpsSequenceNumber` (4 bytes) and the key data itself (16 bytes) in the following order:

- `lpsKeyName`
- `lpsSequenceNumber`
- Key data

**Output Param**  

LPPROTLUXLOADAPPKEYOUT lpvData;

```c
typedef struct _prot_lux_load_app_key_out {
    WORD wCommand;
    WORD wResult;
} PROTLUXLOADAPPKEYOUT, *LPPROTLUXLOADAPPKEYOUT;
```

- `wCommand`  
  Is set to WFS_CMD_ENC_IO_LUX_LOAD_APPKEY.

- `wResult`  
  The command reply codes (see general definition in the first paragraph). The following specific error codes are possible:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROT_LUX_ERR_VERIFICATION_FAILED</td>
<td>Verification failed. The supplied MAC does not match with the calculated MAC.</td>
</tr>
</tbody>
</table>

**Comments**  

This command will return generic error PROT_LUX_ERR_INVALID_KEY when Key Transport Key is not loaded.
9.1.2 WFS_CMD_ENC_IO_LUX_GENERATE_MAC

Description
This command is used to generate the CBC-MAC (Message Authentication Code ISO9797-1:1999, Padding Method 1, MAC Algorithm 3).

This command returns the generated MAC for the data passed in.

To access the WFS_CMD_ENC_IO_LUX_GENERATE_MAC command, the structure WFSPINENCIO of the WFS_CMD_PIN_ENC_IO command has to be defined as required by the Luxembourg protocol (see general definition in the first paragraph). The only definitions specific to this command are the input and output structures pointed by the lpvData fields. Those are defined as follows:

Input Param
LPPROTLUXGENERATEMACIN lpvData;

typedef struct _prot_lux_generate_mac_in
{
  WORD     wCommand;
  LPWFSXDATA    lpxData;
  WORD     wMacLength;
} PROTLUXGENERATEMACIN, *LPPROTLUXGENERATEMACIN;

wCommand
Is set to WFS_CMD_ENC_IO_LUX_GENERATE_MAC.

lpxData
The lpxData parameter contains the data whose MAC is to be generated. Data will be padded according to ISO9797-1:1999, Padding Method 1 if it is not passed in as multiple of 8 bytes.

wMacLength
Specifies the MAC length. Legal values are: 2, 4, 6 or 8.

Output Param
LPPROTLUXGENERATEMACOUT lpvData;

typedef struct _prot_lux_generate_mac_out
{
  WORD     wCommand;
  WORD     wResult;
  LPWFSXDATA    lpxMac;
  LPWFSXDATA    lpxRandom;
} PROTLUXGENERATEMACOUT, *LPPROTLUXGENERATEMACOUT;

wCommand
Is set to WFS_CMD_ENC_IO_LUX_GENERATE_MAC.

wResult
The command reply codes (see general definition in the first paragraph).

lpxMac
The lpxMac parameter contains the generated MAC.

lpxRandom
The lpxRandom parameter contains the random value used to work out the session key.

Comments
The MAC is in ISO9797-1 format and is obtained from a random session key. The generated MAC is returned with the lpxRandom value that was used to obtain the random session key. This command will return generic error PROT_LUX_ERR_INVALID_KEY when MK_MAC key is not loaded.
9.1.3 WFS_CMD_ENC_IO_LUX_CHECK_MAC

**Description**
This command verifies the CBC-MAC (Message Authentication Code ISO9797-1:1999, Padding Method 1, MAC Algorithm 3).

This command generates a MAC for the data passed in and compares it with the provided MAC value.

To access the WFS_CMD_ENC_IO_LUX_CHECK_MAC command, the structure WFSPINENCIO of the WFS_CMD_PIN_ENC_IO command has to be defined as required by the Luxembourg protocol (see general definition in the first paragraph). The only definitions specific to this command are the input and output structures pointed by the lpvData fields. Those are defined as follows:

**Input Param**

```c
LPPROTLUXCHECKMACIN lpvData;
typedef struct _prot_lux_check_mac_in 
{
    WORD     wCommand;
    LPWFSXDATA    lpxData;
    LPWFSXDATA    lpxMac;
    LPWFSXDATA    lpxRandom;
} PROTLUXCHECKMACIN, *LPPROTLUXCHECKMACIN;
```

* **wCommand**
  Is set to WFS_CMD_ENC_IO_LUX_CHECK_MAC.

* **lpxData**
  The lpxData parameter contains the data whose MAC is to be checked. Data will be padded according to ISO9797-1:1999, Padding Method 1 if it is not passed in as multiple of 8 bytes.

* **lpxMac**
  The lpxMac parameter contains the MAC that is to be checked.

Legal values for the MAC length are: 2, 4, 6 or 8.

* **lpxRandom**
  The lpxRandom parameter contains the random value used to work out the session key.

**Output Param**

```c
LPPROTLUXCHECKMACOUT lpvData;
typedef struct _prot_lux_check_mac_out 
{
    WORD     wCommand;
    WORD     wResult;
} PROTLUXCHECKMACOUT, *LPPROTLUXCHECKMACOUT;
```

* **wCommand**
  Is set to WFS_CMD_ENC_IO_LUX_CHECK_MAC.

* **wResult**
  The command reply codes (see general definition in the first paragraph). The following specific error codes can be returned:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROT_LUX_ERR_VERIFICATION_FAILED</td>
<td>Verification Failed. The MAC generated by this command does not compare with the MAC passed in by the application.</td>
</tr>
</tbody>
</table>

**Comments**
If the value of wResult is PROT_LUX_SUCCESS, then the MAC check was successful. This command will return generic error PROT_LUX_ERR_INVALID_KEY when MK_MAC key is not loaded.
9.1.4 WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK

Description

This command is used to construct the PIN blocks described below for remote PIN check. For PIN block format see comment section below.

To access the WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK command, the structure WFSPINENCIO of the WFS_CMD_PIN_ENC_IO command has to be defined as required by the Luxembourg protocol (see general definition in the first paragraph). The only definitions specific to this command are the input and output structures pointed by the lpvData fields. Those are defined as follows:

Input Param

LPPROTLUXPINBLOCKIN lpvData;

typedef struct _prot_lux_pinblock_in
{
  WORD wCommand;
  WORD wFormat;
} PROTLUXPINBLOCKIN, *LPPROTLUXPINBLOCKIN;

wCommand

Is set to WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK.

wFormat

Specifies the format of the PIN block. Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROT_LUXFORMISO1</td>
<td>ISO-1 PIN Block</td>
</tr>
</tbody>
</table>

Output Param

PROTLUXPINBLOCKOUT lpvData;

typedef struct _prot_lux_pinblock_out
{
  WORD wCommand;
  WORD wResult;
  LPWFSXDATA lpxPinBlock;
  LPWFSXDATA lpxRandom;
} PROTLUXPINBLOCKOUT, *LPPROTLUXPINBLOCKOUT;

wCommand

Is set to WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK.

wResult

The command reply codes (see general definition in the first paragraph). The following specific error can be returned:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROT_LUX_ERR_PIN_FORMAT_LENGTH</td>
<td>The PIN block could not be constructed because PIN was not entered or the PIN length was invalid.</td>
</tr>
</tbody>
</table>

lpxPinBlock

The lpxPinBlock parameter contains the constructed PIN block.

lpxRandom

The lpxRandom parameter contains the random value used to calculate the session key.

Comments

The PIN block is constructed in an ISO-1 format with random number padding and then Triple DES encrypted using a random session key. The encrypted PIN block is returned with the lpxRandom value that was used to obtain the random session key. This command will return generic error PROT_LUX_ERR_INVALID_KEY when MK_PAC key is not loaded.
9.1.5 WFS_CMD_ENC_IO_LUX_DECRYPT_TDES

Description
This command is used to decrypt the data according to triple DES algorithm.

To access the WFS_CMD_ENC_IO_LUX_DECRYPT_TDES command, the structure
WFS_PINENCIO of the WFS_CMD_PIN_ENC_IO command has to be defined as required by the
Luxembourg protocol (see general definition in the first paragraph). The only definitions specific
to this command are the input and output structures pointed by the lpvData fields. Those are
defined as follows:

Input Param
LPPROTLUXDECRYPTTDESIN lpvData;

t typedef struct __prot_lux_decrypt_tdes_in
{  
    WORD wCommand;
    WORD wType;
    LPWFSXDATA lpxData;
    LPWFSXDATA lpxIV;
    LPWFSXDATA lpxRandom;
} PROTLUXDECRYPTTDESIN, *LPPROTLUXDECRYPTTDESIN;

wCommand
Is set to WFS_CMD_ENC_IO_LUX_DECRYPT_TDES.

wType
An integer word specifying the type of triple DES decryption to be used as one of the following
flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROT_LUXTRIDESECB</td>
<td>Triple DES with Electronic Code Book.</td>
</tr>
<tr>
<td>PROT_LUXTRIDESCBC</td>
<td>Triple DES with Cipher Block Chaining.</td>
</tr>
</tbody>
</table>

lpxData
The lpxData parameter contains the data to be decrypted. Data must be multiple of 8-byte blocks.

lpxIV
If wType is WFS_PIN_LUXTRIDESCBC then this field contains the 8 bytes of data containing
the Initial Value needed for decryption in CBC mode. Otherwise this field is ignored.

lpxRandom
The lpxRandom parameter contains the random value used to calculate the session key.

Output Param
LPPROTLUXDECRYPTTDESOUT lpvData;

t typedef struct __prot_lux_decrypt_tdes_out
{  
    WORD wCommand;
    WORD wResult;
    LPWFSXDATA lpxData;
} PROTLUXDECRYPTTDESOUT, *LPPROTLUXDECRYPTTDESOUT;

wCommand
Is set to WFS_CMD_ENC_IO_LUX_DECRYPT_TDES.

wResult
The command reply codes (see general definition in the first paragraph).

lpxData
The lpxData parameter contains the decrypted data.

Comments
The Triple-DES decryption uses a random session key. The session key is derived from a random
number that is provided in lpxRandom. This command will return generic error
PROT_LUX_ERR_INVALID_KEY when MK_ENC key is not loaded.
9.1.6  WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES

**Description**

This command is used to encrypt the data according to triple DES algorithm.

To access the WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES command, the structure WFSPINENCIO of the WFS_CMD_PIN_ENC_IO command has to be defined as required by the Luxembourg protocol (see general definition in the first paragraph). The only definitions specific to this command are the input and output structures pointed by the `lpvData` fields. Those are defined as follows:

**Input Param**

```c
typedef struct _prot_lux_encrypt_tdes_in
{
    WORD     wCommand;
    WORD     wType;
    LPWFSXDATA    lpxData;
    LPWFSXDATA    lpxIV;
} PROTLUXENCRYPTTDESIN, *LPProTluxEncryptTDESin;
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_LUXTRIDESCBC</td>
<td>Triple DES with Electronic Code Book.</td>
</tr>
<tr>
<td>WFS_PIN_LUXTRIDESCBC</td>
<td>Triple DES with Cipher Block Chaining.</td>
</tr>
</tbody>
</table>

- **wCommand**
  Is set to WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES.

- **wType**
  An integer word specifying the type of triple DES encryption to be used as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_LUXTRIDESCBC</td>
<td>Triple DES with Electronic Code Book.</td>
</tr>
<tr>
<td>WFS_PIN_LUXTRIDESCBC</td>
<td>Triple DES with Cipher Block Chaining.</td>
</tr>
</tbody>
</table>

- **lpxData**
  The `lpxData` parameter contains the data to be encrypted. Data must be multiple of 8-byte blocks. Application must fill the end of the data with 0x00 if the data does not contain a multiple of 8-byte blocks.

- **lpxIV**
  If `wType` is WFS_PIN_LUXTRIDESCBC then this field contains the 8 bytes of data containing the Initial Value needed for encryption in CBC mode. Otherwise this field is ignored.

**Output Param**

```c
typedef struct _prot_lux_encrypt_tdes_out
{
    WORD     wCommand;
    WORD     wResult;
    LPWFSXDATA    lpxData;
    LPWFSXDATA    lpxRandom;
} PROTLUXENCRYPTTDESOUT, *LPProTluxEncrytTDESOut;
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES.</td>
<td></td>
</tr>
</tbody>
</table>

- **wCommand**
  Is set to WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES.

- **wResult**
  The command reply codes (see general definition in the first paragraph).

- **lpxData**
  The `lpxData` parameter contains the encrypted data.

- **lpxRandom**
  The `lpxRandom` parameter contains the random value used to calculate the session key.

**Comments**

The Triple-DES encryption uses a random session key. The session key is derived from a random number that is returned in `lpxRandom`. This command will return generic error.
9.1.7 Luxemburg-specific Header File

This header section is to be created into a separate file from the standard xfspin.h and identifies the definitions for the Luxemburg Protocol only.

/*******************************************************************************
*                                                                           *
*xfspinlux.h XFS - Personal Identification Number Keypad (PIN) Luxemburg    *
*Protocol definitions                                                        *
*                                                                           *
*******************************************************************************/
#ifndef __INC_XFSPINLUX__H
#define __INC_XFSPINLUX__H

#ifdef __cplusplus
extern "C" {
#endif
/*   be aware of alignment   */
#pragma pack(push,1)
/* values of PROTLUXIN.wCommand */
#define WFS_CMD_ENC_IO_LUX_LOAD_APPKEY              (0x0001)
#define WFS_CMD_ENC_IO_LUX_GENERATE_MAC             (0x0002)
#define WFS_CMD_ENC_IO_LUX_CHECK_MAC                (0x0003)
#define WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK           (0x0004)
#define WFS_CMD_ENC_IO_LUX_DECRYPT_TDES             (0x0005)
#define WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES             (0x0006)
#define PROT_LUX_RESULT_OFFSET (0)
/* values of PROTLUXOUT.wResult */
#define PROT_LUX_SUCCESS                            (0)
#define PROT_LUX_ERR_INVALID_CMD                    (*((PROT_LUX_RESULT_OFFSET + 1)))
#define PROT_LUX_ERR_INVALID_DATA                   (*((PROT_LUX_RESULT_OFFSET + 2)))
#define PROT_LUX_ERR_INVALID_KEY                    (*((PROT_LUX_RESULT_OFFSET + 3)))
/* values of PROTLUXLOADAPPKEYOUT.wResult */
/* values of PROTLUXCHECKMACOUT.wResult */
#define PROT_LUX_ERR_VERIFICATION_FAILED            (*((PROT_LUX_RESULT_OFFSET + 4)))
/* values of PROTLUXPINBLOCKOUT.wResult */
#define PROT_LUX_ERR_PIN_FORMAT_LENGTH              (*((PROT_LUX_RESULT_OFFSET + 5)))
/* values of PROTLUXDECRYPTTDESIN.wType and PROTLUXENCRYPTTDESIN.wType*/
#define PROT_LUXTRIDESECB                           (0x0000)
#define PROT_LUXTRIDESCBC                           (0x0001)
/* values of PROTLUXPINBLOCKIN.fwFormat */
#define PROT_LUXFORMISO1                            (0x0001)

// Used to type-cast specific command to access common fields
typedef struct _prot_lux_in
{
    WORD                   wCommand;
} PROTLUXIN, *LPPROTLUXIN;

// Used to type-cast specific response to access common fields
typedef struct _prot_lux_out
{
    WORD wCommand;
    WORD wResult;
} PROTDLUXOUT, *LPPROTDLUXOUT;

typedef struct _prot_lux_load_app_key_in
{
    WORD wCommand;
    LPSTR lpsKeyName;
    LPSTR lpsSequenceNumber;
    LPWFSXDATA lpxKeyData;
} PROTDLUXLOADAPPKEYIN, *LPPROTDLUXLOADAPPKEYIN;

typedef struct _prot_lux_load_app_key_out
{
    WORD wCommand;
    WORD wResult;
} PROTDLUXLOADAPPKEYOUT, *LPPROTDLUXLOADAPPKEYOUT;

typedef struct _prot_lux_generate_mac_in
{
    WORD wCommand;
    LPWFSXDATA lpxData;
    WORD wMacLength;
} PROTDLUXGENERATEMACIN, *LPPROTDLUXGENERATEMACIN;

typedef struct _prot_lux_generate_mac_out
{
    WORD wCommand;
    WORD wResult;
    LPWFSXDATA lpxMac;
    LPWFSXDATA lpxRandom;
} PROTDLUXGENERATEMACOUT, *LPPROTDLUXGENERATEMACOUT;

typedef struct _prot_lux_check_mac_in
{
    WORD wCommand;
    LPWFSXDATA lpxData;
    LPWFSXDATA lpxMac;
    LPWFSXDATA lpxRandom;
} PROTDLUXCHECKMACIN, *LPPROTDLUXCHECKMACIN;

typedef struct _prot_lux_check_mac_out
{
    WORD wCommand;
    WORD wResult;
} PROTDLUXCHECKMACOUT, *LPPROTDLUXCHECKMACOUT;

typedef struct _prot_lux_pinblock_in
{
    WORD wCommand;
    WORD wFormat;
} PROTDLUXPINBLOCKIN, *LPPROTDLUXPINBLOCKIN;

typedef struct _prot_lux_pinblock_out
{
    WORD wCommand;
    WORD wResult;
    LPWFSXDATA lpxPinBlock;
    LPWFSXDATA lpxRandom;
} PROTDLUXPINBLOCKOUT, *LPPROTDLUXPINBLOCKOUT;

typedef struct _prot_lux_decrypt_tdes_in
{
    WORD wCommand;
    WORD wType;
    LPWFSXDATA lpxData;
    LPWFSXDATA lpxIV;
    LPWFSXDATA lpxRandom;
typedef struct _prot_lux_decrypt_tdes_out
{
    WORD wCommand;
    WORD wResult;
    LPWFSXDATA lpxData;
} PROTLUXDECRYPTTDESOUT, *LPPROTLUXDECRYPTTDESOUT;

typedef struct _prot_lux_encrypt_tdes_in
{
    WORD wCommand;
    WORD wType;
    LPWFSXDATA lpxData;
    LPWFSXDATA lpxIV;
} PROTLUXENCRYPTTDESIN, *LPPROTLUXENCRYPTTDESIN;

typedef struct _prot_lux_encrypt_tdes_out
{
    WORD wCommand;
    WORD wResult;
    LPWFSXDATA lpxData;
    LPWFSXDATA lpxRandom;
} PROTLUXENCRYPTTDESOUT, *LPPROTLUXENCRYPTTDESOUT;

/* restore alignment */
#pragma pack(pop)
#endif /* __cplusplus */
#endif    /* __INC_XFSPINLUX__H */
10. Appendix–C (Standardized *lpszExtra* fields)

This section contains the values that have been standardized for the *lpszExtra* fields within previous releases of the PIN specification. These values are still applicable to this version of the standard and must be supported if the functionality is supported.

10.1 **WFS_INF_PIN_STATUS**

The following standardized *lpszExtra* values have been defined for the WFS_INF_PIN_STATUS command.

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

\[
\text{CERTIFICATESTATE} = \langle 0\text{nnnnnnnn} \rangle, \text{ where nnnnnnnn is the ASCII representation of a hexadecimal value. This state determines which public verification or encryption key should be read out of the device. For example CERTIFICATESTATE = 0x00000001 indicates that the state of the Encryptor is Primary. The possible values are the following:}
\]

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

The following standardized *lpszExtra* values have been defined for the WFS_INF_PIN_CAPABILITIES command.

For German HSMs this parameter will contain the following information:

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

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

**REMOTE_KEY_SCHEME=<0xnnnnnnnn>,** where nnnnnnnnn is the ASCII representation of a hexadecimal value. **REMOTE_KEY_SCHEME** will specify to the user which type(s) of Remote Key Loading/Authentication is supported. For example, “**REMOTE_KEY_SCHEME=0x00000002**” indicates that three-party certificates are supported.

The support level is defined as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_RSA_AUTH_2PARTY_SIG</td>
<td>Two-party Signature based authentication.</td>
</tr>
<tr>
<td>WFS_PIN_RSA_AUTH_3PARTY_CERT</td>
<td>Three-party Certificate based authentication.</td>
</tr>
</tbody>
</table>

**RSA_SIGN_ALGORITHM=<0xnnnnnnnn>,** where nnnnnnnnn is the ASCII representation of a hexadecimal value. **RSA_SIGN_ALGORITHM** will specify what type(s) of RSA Signature Algorithms is supported. For example, “**RSA_SIGN_ALGORITHM=0x00000001**” indicates that RSA_PKCS_V1_5 is supported. The support level is defined as a combination of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PKCS1_V1_5</td>
<td>SSA_PKCS_V1_5 Signatures supported.</td>
</tr>
<tr>
<td>WFS_PIN_SIGN_RSASSA_PSS</td>
<td>SSA_PSS Signatures supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_CRYPT_RSAES_PKCS1_V1_5</td>
<td>AES_PKCS_V1_5 algorithm supported.</td>
</tr>
<tr>
<td>WFS_PIN_CRYPT_RSAES_OAEP</td>
<td>AES_OAEP algorithm supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_RSA_KCV_SHA1</td>
<td>The key check value contains a SHA 1 of the</td>
</tr>
<tr>
<td></td>
<td>public key as defined in Ref. 3.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_SIG_GEN_RSA_KEYPAIR</td>
<td>Specifies if the Service Provider supports the</td>
</tr>
<tr>
<td></td>
<td>RSA Signature Scheme</td>
</tr>
<tr>
<td></td>
<td>WFS_CMD_PIN_GENERATE_RSA_KEYPAIR and</td>
</tr>
<tr>
<td></td>
<td>WFS_CMD_PIN_EXPORT_RSA_EPP_SIG NED commands.</td>
</tr>
</tbody>
</table>
WFS_PIN_SIG_RANDOM_NUMBER Specifies if the Service Provider returns a random number from the WFS_CMD_PIN_START_KEY_EXCHANGE command within the RSA Signature Scheme.

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_PIN_HASH_SHA1_DIGEST</td>
<td>The SHA 1 digest algorithm is supported by the WFS_CMD_PIN_DIGEST command.</td>
</tr>
</tbody>
</table>

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

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

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

ENCIOPROTOCOLS=0xnnnn where nnnn is the ASCII representation of the combination of the values supported for the wProtocol parameter.

A Service Provider may automatically generate a beep on key presses, this is reported by the following key=value pair:

AUTOBEEP=<0/1> - (0 means no beeps are generated automatically, 1 means beeps are generated automatically)

For devices where the secure PIN keypad is integrated within a generic Win32 keyboard then, if the following pair is present:
“KYBD=COMBINED_WIN32” - then standard Win32 key events will be generated for any key when there is no ‘active’ GET_DATA or GET_PIN command.

Note that XFS continues to support PIN keys define only, and is not extended to support new alphanumeric keys.

This feature assists in developing generic browser based applications which need to access both PIN and generic keyboards.

When an application wishes to receive XFS-based key information then he can use the XFS GET_DATA and GET_PIN functions.

No Win32 keystrokes are generated for any key (active or not) in a combined device when GET_DATA or GET_PIN are ‘active’.

When no GET_DATA or GET_PIN function is ‘active’ then any key press will result in a Win32 key event. These events can be ignored by the application, if required.

Note that this does not compromise secure PIN entry – there will be no Win32 keyboard events during PIN collection.

On terminals and kiosks with separate PIN and Win32 keyboards, the Win32 keyboard behaves purely as a PC keyboard and the PIN device behaves only as an XFS device.