CEN

WORKSHOP

AGREEMENT

CWA 16374-6

December 2011

ICS 35.240.40

English version

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.

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

Management Centre: Avenue Marnix 17, B-1000 Brussels

Ref. No.:CWA 16374-6:2011 D/E/F

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_KEY_DETAIL 25

4.4 WFS_INF_PIN_FUNCKEY_DETAIL 27

4.5 WFS_INF_PIN_HSM_TDATA 30

4.6 WFS_INF_PIN_KEY_DETAIL_EX 31

4.7 WFS_INF_PIN_SECUREKEY_DETAIL 33

4.8 WFS_INF_PIN_QUERY_LOGICAL_HSM_DETAIL 37

4.9 WFS_INF_PIN_QUERY_PCIPTS_DEVICE_ID 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_BANKSYS_IO 68
5.1.15 WFS_CMD_PIN_RESET 69
5.1.16 WFS_CMD_PIN_HSM_SET_TDATA 70
5.1.17 WFS_CMD_PIN_SECURE_MSG_SEND 72
5.1.18 WFS_CMD_PIN_SECURE_MSG_RECEIVE 74
5.1.19 WFS_CMD_PIN_GET_JOURNAL 76
5.1.20 WFS_CMD_PIN_IMPORT_KEY_EX 77
5.1.21 WFS_CMD_PIN_ENC_IO 80
5.1.22 WFS_CMD_PIN_HSM_INIT 82
5.1.23 WFS_CMD_PIN_SECUREKEY_ENTRY 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_KEYPRESS_BEEP 90
5.1.28 WFS_CMD_PIN_SET_PINBLOCK_DATA 91

5.1.29 WFS_CMD_PIN_SET_LOGICAL_HSM 92
5.1.30 WFS_CMD_PIN_IMPORT_KEYBLOCK 94
5.1.31 WFS_CMD_PIN_POWER_SAVE_CONTROL 95

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_ENCIPHERED_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

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 Luxemburg 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 Luxemburg-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:

3.00	October 18, 2000	Initial release.
3.02	May 21, 2003	For a description of changes from version 3.00 to version 3.02 see the PIN 3.02 Migration document.
3.03	September 24, 2004	For a description of changes from version 3.02 to version 3.03 see the PIN 3.03 Migration document.
3.10	November 29, 2007	For a description of changes from version 3.03 to version 3.10 see the PIN 3.10 Migration document.
3.20	March 2nd, 2011	For a description of changes from version 3.10 to version 3.20 see the PIN 3.20 Migration document.

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**, **WFSExecute**, **WFSGetInfo** and **WFSAsyncGetInfo** functions.

This section describes the general interface for the following functions:

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

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

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

Important Notes:

- This revision of this specification does not define 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
3. SHA-1 Hash algorithm ANSI 9:30:2-1993: *Public Key Cryptography for Financial Services Industry Part2*
4. EMVCo, EMV2000 Integrated Circuit Card Specification for Payment Systems, Book 2 – Security and Key Management, Version 4.0, December 2000
5. Europay International, EPI CA Module Technical – Interface specification Version 1.4
6. ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ec-Karte mit Chip, Online-Personalisierung von Terminal-HSMs, Version 3.0, 2. 4. 1998
7. ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ZKA-Chipkarte, Online-Vor-Initialisierung und Online-Anzeige einer Außerbetriebnahme von Terminal-HSMs, Version 1.0, 04.08.2000
8. 473x Programmers Reference Volume 1 - TP-820399-001A
9. 473x Programmers Reference Volume 2 - TP-820403-001A
10. 473x Programmers Reference Volume 3 - TP-820400-001A
11. 473x Programmers Reference Volume 4 - TP-820404-001A
12. 473x P-Model Programmers Reference - TP-820397-001A
13. 473x Log Reference Guide - TP-820398-001A
14. Diebold's Specification for support of Online Preinitialization and Personalization of Terminal HSMs (OPT) and support for the PAC/MAC standards for the 473x Protocol, Diebold USA, Revision 1.10, revised on May 2002
15. Groupement des Cartes Bancaires "CB", Description du format et du contenu des données cryprographiques échangées entre GAB et GDG, Version 1.3 / Octobre 2002
16. ITU-T Recommendation X.690 – ASN.1 encoding rules (also published as ISO/IEC International Standard 8825-1), 1997
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
21. ANSI X9.8-1995, Banking – Personal Identification Number Management and Security, Part 1 + 2, American National Standards Institute
22. ISO 9564-1, Banking – Personal Identification Number management and security, Part 1, First Edition 1991-12-15, International Organization for Standardization
23. ISO 9564-2, Banking – Personal Identification Number management and security, Part 2, First Edition 1991-12-15, International Organization for Standardization
24. IBM, Common Cryptographic Architecture: Cryptographic Application Programming Interface, SC40-1675-1, IBM Corp., Nov 1990
25. R:L: Rivest, A. Shamir, and L.M. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, v. 21, n.2, Feb 1978, pp. 120-126
26. Security for Computer Networks by Donald W. Davies & William L. Price, Second Edition, John Wiley & Sons, 1989
27. Regelwerk für das deutsche ec-Geldautomaten-System, Stand: 22. Nov. 1999
28. Bank-Verlag, Köln, Autorisierungszentrale GA/POS der privaten Banken, Spezifikation für GA-Betreiber, Version 3.12, 31. Mai 2000
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. ANS X9 TR-31 2005, Interoperable Secure Key Exchange Key Block Specification for Symmetric Algorithms

36. Oliself2 Specifiche Tecniche, PIN Block Detail for WFS_PIN_FORMAP
37. PCI Security Standards Council PCI PTS approval list https://www.pcisecuritystandards.org/security_standards/ped/pedapprovallist.html

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:

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

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

Value	Meaning
WFS_PIN_ENCREADY	The encryption module is initialized and ready (at least one key is imported into the encryption module).
WFS_PIN_ENCNOTREADY	The encryption module is not available or not ready due to hardware error or communication error.
WFS_PIN_ENCNOTINITIALIZED	The encryption module is not initialized (no master key loaded).

WFS_PIN_ENCBUSY	The encryption module is busy (implies that the device is busy).
WFS_PIN_ENCUNDEFINED	The encryption module state is undefined.
WFS_PIN_ENCINITIALIZED	The encryption module is initialized and master key (where required) and any other initial keys are loaded; ready to import other keys.

lpszExtra
Specifies a list of vendor-specific, or any other extended, information. The information is returned as a series of "*key=value*" strings so that it is easily extendable by Service Providers. Each string will be null-terminated, 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.

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

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):

Value	Meaning
WFS_PIN_BEEP_ON_ACTIVE	An automatic tone will be generated for all active keys.
WFS_PIN_BEEP_ON_INACTIVE	An automatic tone will be generated for all in-active keys.

dwCertificateState
Specifies the state of the public verification or encryption key in the PIN certificate modules as one of the following flags:

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

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_DEVICENOTINPOSITION, *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:

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

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:

Value	Meaning
WFS_PIN_AFMNOTSUPP	No anti-fraud module is available.
WFS_PIN_AFMOK	Anti-fraud module is in a good state and no foreign device is detected.
WFS_PIN_AFMINOP	Anti-fraud module is inoperable.
WFS_PIN_AFMDEVICEDETECTED	Anti-fraud module detected the presence of a foreign device.
WFS_PIN_AFMUNKNOWN	The state of the anti-fraud module cannot be determined.

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                   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;
```

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.

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

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

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

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

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

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

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

WFS_PIN_FORMDIEBOLDCO	PIN with the length of 4 to 12 digits, each one with a value of 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:

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

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

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

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

Value	Meaning
WFS_PIN_DISPNONE	No display unit.
WFS_PIN_DISPLEDTHROUGH	Lights next to text guide user.
WFS_PIN_DISPDISPLAY	A real display is available (this doesn't apply for self-service).

bIDConnect
Specifies whether the PIN pad is directly physically connected to the ID card unit. 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:

Value	Meaning
WFS_PIN_IDKEYINITIALIZATION	ID key is returned by the WFS_CMD_PIN_INITIALIZATION command.
WFS_PIN_IDKEYIMPORT	ID key is required as input for the WFS_CMD_PIN_IMPORT_KEY and WFS_CMD_PIN_DERIVE_KEY command.

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

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

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

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

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.

Value	Meaning	Type
WFS_PIN_GUIDANCE_NOT_AVAILABLE	There is no guidance light control available at this position.	A
WFS_PIN_GUIDANCE_OFF	The light can be off.	B
WFS_PIN_GUIDANCE_SLOW_FLASH	The light can blink slowly.	B
WFS_PIN_GUIDANCE_MEDIUM_FLASH	The light can blink medium frequency.	B

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:

Value	Meaning
TRUE	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.
FALSE	The PIN will always be cleared by the device after processing. The WFS_CMD_PIN_MAINTAIN_PIN is not supported.

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

Value	Meaning
WFS_PIN_BEEP_ACTIVE_AVAILABLE	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.
WFS_PIN_BEEP_ACTIVE_SELECTABLE	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.
WFS_PIN_BEEP_INACTIVE_AVAILABLE	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.
WFS_PIN_BEEP_INACTIVE_SELECTABLE	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.

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:

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

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

Value	Meaning
WFS_PIN_SIGN_RSASSA_PKCS1_V1_5	SSA_PKCS_V1_5 Signatures supported.
WFS_PIN_SIGN_RSASSA_PSS	SSA_PSS Signatures supported.

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

Value	Meaning
WFS_PIN_CRYPT_RSAES_PKCS1_V1_5	AES_PKCS_V1_5 algorithm supported.
WFS_PIN_CRYPT_RSAES_OAEP	AES_OAEP algorithm supported.

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

Value	Meaning
WFS_PIN_RSA_KCV_SHA1	SHA-1 is supported as defined in Ref. 3.

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

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

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:

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

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

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

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:

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

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:

Value	Meaning
WFS_PIN_ANSTR31KEYBLOCK	Supports ANS TR-31 Keyblock format key import.

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 LPWFSPINKEYDETAIL *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:

Value	Meaning
WFS_PIN_USECRYPT	Key can be used for encryption/decryption.
WFS_PIN_USEFUNCTION	Key can be used for PIN functions.
WFS_PIN_USEMACING	Key can be used for MACing.
WFS_PIN_USEKEYENCKEY	Key is used as key encryption key.
WFS_PIN_USENODUPLICATE	Key can be imported only once.
WFS_PIN_USESVENCKEY	Key is used as CBC Start Value encryption key.
WFS_PIN_USECONSTRUCT	Key is under construction through the import of multiple parts. This value can be returned in combination with any of the other key usage flags (other than WFS_PIN_USESECURECONSTRUCT).
WFS_PIN_USESECURECONSTRUCT	Key is under construction through the import of multiple parts from a secure encryption key entry buffer. This value can be returned in combination with any of the other key usage flags (other than WFS_PIN_USECONSTRUCT).
WFS_PIN_USEANSTR31MASTER	Key is an ANS X9 TR-31 key block master key (see reference 35).

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:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key name is not found.

Comments None.

4.4 WFS_INF_PIN_FUNCKEY_DETAIL

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

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

Input Param LPULONG lpulFDKMask;

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

Output Param LPWFSPINFUNCKEYDETAIL lpFuncKeyDetail;

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

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

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

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

WFS_PIN_FK_OEM1	
WFS_PIN_FK_OEM2	
WFS_PIN_FK_OEM3	
WFS_PIN_FK_OEM4	
WFS_PIN_FK_OEM5	
WFS_PIN_FK_OEM6	

usNumberFDKs
This value indicates the number of FDK structures returned. Only supported FDKs are returned.

lppFDKs
Pointer to an array of pointers to WFSPINFDK 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;
     } WFSPINFDK, *LPWFSPINFDK;
```

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

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

usXPosition
For FDKs, specifies the 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).

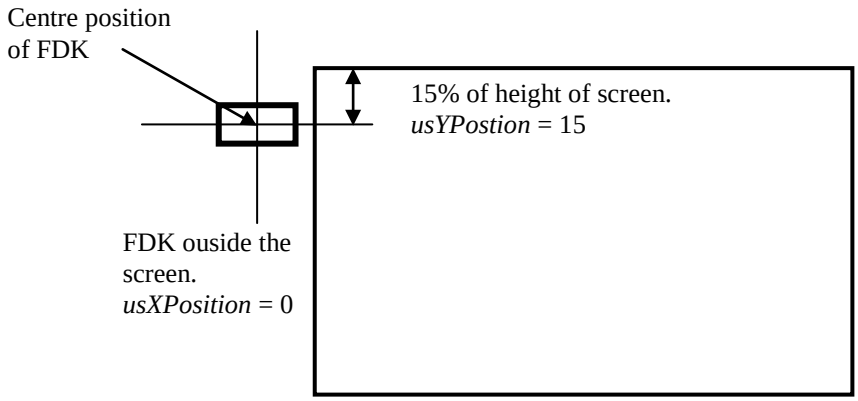


Diagram: Shows how *usXPosition* and *usYPosition* are set.

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;

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

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

Comments None.

4.6 WFS_INF_PIN_KEY_DETAIL_EX

Description This command returns extended detailed information about the keys in the encryption module, including DES, private and public keys. 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 LPWFSPINKEYDETAILEX *lppKeyDetailEx;

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

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

lpsKeyName
Specifies the name of the key.

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

Value	Meaning
WFS_PIN_USECRYPT	Key can be used for encryption/decryption.
WFS_PIN_USEFUNCTION	Key can be used for PIN functions.
WFS_PIN_USEMACING	Key can be used for MACing.
WFS_PIN_USEKEYENCKEY	Key is used as key encryption key.
WFS_PIN_USENODUPLICATE	Key can be imported only once.
WFS_PIN_USESVENCKEY	Key is used as CBC Start Value encryption key.
WFS_PIN_USEPINLOCAL	Key is used for local PIN check.
WFS_PIN_USERSAPUBLIC	Key is used as a public key for RSA encryption including EMV PIN block creation.
WFS_PIN_USERSAPRIVATE	Key is used as a private key for RSA decryption.
WFS_PIN_USERSAPRIVATESIGN	Key is used as a private key for RSA Signature generation. Only data generated within the device can be signed.
WFS_PIN_USECHIPINFO	Key is used as KGK_{INFO} key (only ZKA standard).
WFS_PIN_USECHIPPIN	Key is used as KGK_{PIN} key (only ZKA standard).
WFS_PIN_USECHIPPS	Key is used as K_{PS} key (only ZKA standard).
WFS_PIN_USECHIPMAC	Key is used as K_{MAC} key (only ZKA standard).
WFS_PIN_USECHIPLT	Key is used as KGK_{LT} key (only ZKA standard).
WFS_PIN_USECHIPMACLZ	Key is used as K_{PACMAC} key (only ZKA standard).

WFS_PIN_USECHIPMACAZ	Key is used as K_{MASTER} key (only ZKA standard).
WFS_PIN_USERSAPUBLICVERIFY	Key is used as a public key for RSA signature verification and/or data decryption.
WFS_PIN_USECONSTRUCT	Key is under construction through the import of multiple parts. This value can be returned in combination with any one of the other key usage flags (other than WFS_PIN_USESECURECONSTRUCT).
WFS_PIN_USESECURECONSTRUCT	Key is under construction through the import of multiple parts from a secure encryption key entry buffer. This value can be returned in combination with any of the other key usage flags (other than WFS_PIN_USECONSTRUCT).
WFS_PIN_USEANSTR31MASTER	Key is an ANS X9 TR-31 key block master key (see reference 35).

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:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key name is not found.

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.

Value	Meaning
WFS_PIN_SECUREKEY_NOTSUPP	Secure key entry is not supported, all other parameters are undefined.

WFS_PIN_SECUREKEY_REG_SHIFT	Secure key hex digits 'A' - 'F' are accessed through the shift key. Digits 'A' - 'F' are accessed through the shift key followed by one of the other function keys. The keys associated with 'A' to 'F' are defined within the *lppHexKeys* parameter. The keyboard has a regular shaped key layout where all rows have the same number of keys and all columns have the same number of keys, e.g. 5x4. The *lppHexKeys* parameter must contain one entry for each key on the 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.

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

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

On a 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:

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

WFS_PIN_FK_F	(hex digit F)
WFS_PIN_FK_SHIFT	(Shift key used during hex entry)

ulClearFDK
The FDK code mask reporting any FDKs associated with Clear. If this field is 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 1and 1000, where 1 is the smallest possible size and 1000 is the full height of the keyboard.

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

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

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

Comments Examples keyboard layouts are provided in section 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;

```
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 values 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.

```
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 EBCIDIC 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 LPWFSPINPCIPTSDEVICEID lpPCIPTSDeviceId;

```
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 LPWFSPINCRYPT lpCrypt;

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

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

Value	Meaning
WFS_PIN_MODEENCRYPT	Encrypt with key.
WFS_PIN_MODEDECRYPT	Decrypt with key.
WFS_PIN_MODERANDOM	An 8 byte random value shall be returned (in this case all the other input parameters are ignored).

This parameter does not apply to MACing.

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

lpxKeyEncKey
If NULL, *lpsKey* is used directly for encryption/decryption. Otherwise, *lpsKey* is used to decrypt (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:

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments 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 *lpxValue* parameter, If WFS_PIN_USESECURECONSTRUCT is used then the encryption key part in the secure key buffer previously populated with the WFS_CMD_PIN_SECUREKEY_ENTRY command is used and *lpxValue* is ignored. Key parts loaded with the WFS_PIN_USESECURECONSTRUCT flag can only be stored once as the encryption key in the secure key buffer is no longer available after this command has been executed. The WFS_PIN_USECONSTRUCT and WFS_PIN_USESECURECONSTRUCT construction flags cannot be used in combination.

Input Param LPWFSPINIMPORT lpImport;

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

lpsKey
Specifies the name of key being loaded.

lpsEncKey
lpsEncKey specifies a key name or a format name which 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:

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

WFS_PIN_USESECURECONSTRUCT	Key is under construction through the import of multiple parts. This value is used in combination with the actual usage flags for the key. *lpxValue* is ignored as the encryption key part is taken from the secure key buffer.
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 *fwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* are ignored.

Output Param LPWFSXDATA lpxKVC;

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

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

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments When keys are loaded in multiple parts, all parts of the key loaded must set the relevant construction value in the *fwUse* field along with any 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;

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

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

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

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

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

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

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

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

WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxStartValue* is not supported 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:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.4 WFS_CMD_PIN_GET_PIN

Description This function stores the PIN entry via the PIN pad. From the point this function is invoked, PIN digit entries are *not* passed to the application. For each PIN digit, or any other active key entered, an execute notification event 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:

Value	Meaning
WFS_PIN_COMPAUTO	The command terminated automatically, because maximum length was reached.
WFS_PIN_COMPENTER	The ENTER Function Key was pressed as terminating key.
WFS_PIN_COMPCANCEL	The CANCEL Function Key was pressed as terminating key.

WFS_PIN_COMPCONTINUE	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_COMPHELP	The HELP Function Key was pressed as terminating key.
WFS_PIN_COMPFK	A Function Key (FK) other than ENTER, CLEAR, CANCEL, BACKSPACE, HELP was pressed as terminating key.
WFS_PIN_COMPCONTFDK	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:

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

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

Value	Meaning
WFS_EXEE_PIN_KEY	A key has been pressed at the PIN pad.
WFS_EXEE_PIN_ENTERDATA	The PIN pad is ready for the user to start entering data.

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:

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

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 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Output Param LPSTR lpsOffset;

lpsOffset
Computed PIN Offset.

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

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

WFS_ERR_PIN_NOTALLOWED	PIN entered by the user is not allowed.
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:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

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

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

WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* is not supported 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:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.8 WFS_CMD_PIN_LOCAL_VISA

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.9 WFS_CMD_PIN_PRESENT_IDC

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

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

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

Comments Example for the use of the algorithm WFS_PIN_PRESENT_CLEAR:

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

Bytes 0 to 4					Bytes 5 to 8
CLA	INS	A1	A2	Lc	PIN block
0xBC	0x20	0x00	0x00	0x04	0xXX 0xXX 0xXX 0xXX

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:

Byte 5								Byte 6								Byte 7							Byte 8							
0	0	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	Digit 1				Digit 2			Digit3				Digit 4																			

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

ulChipDataLength	9
lpbChipData	0xBC2000000400003FFF
ulPINPointer	5
usPINOffset	2

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

Byte 5								Byte 6								Byte 7							Byte 8							
0	0	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	Digit 1				Digit 2			Digit3				Digit 4																			

Resulting in a chip card command of:

Bytes 0 to 4					Bytes 5 to 8
CLA	INS	A1	A2	Lc	PIN block
0xBC	0x20	0x00	0x00	0x04	0x04 0x8D 0x3F 0xFF

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 LPWFSPINBLOCK lpPinBlock;

```
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. 0x0123456789ABCDEF 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 "0123456789ABCDEF\0". The hex digits 0xA to 0xF 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;

lpxPinBlock
Pointer to the encrypted PIN block.

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

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.11 WFS_CMD_PIN_GET_DATA

Description This function is used to return keystrokes entered by the user. It will automatically set the PIN pad to echo characters on the display if there is a display. For each keystroke an execute notification event 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_FK_CANCEL
WFS_PIN_FK_CLEAR
WFS_PIN_FK_BACKSPACE

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

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.

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.

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 LPWFSPINDATA lpPinData;

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

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

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

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

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

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

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

Value	Meaning
WFS_EXEE_PIN_KEY	A key has been pressed at the PIN pad.
WFS_EXEE_PIN_ENTERDATA	The PIN pad is ready for the user to start entering data.

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 *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;
```

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

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

Output Param LPWFSXDATA lpxIdentification;

lpxIdentification
Pointer to the value of the ID key encrypted by the encryption key. 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:

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

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

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

Comments None.

5.1.13 WFS_CMD_PIN_LOCAL_BANKSYS

Description The PIN block previously built by the WFS_CMD_PIN_GET_PINBLOCK 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:

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.14 WFS_CMD_PIN_BANKSYS_IO

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

Input Param LPWFSPINBANKSYSIO lpBanksysIoIn;

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

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

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

Output Param LPWFSPINBANKSYSIO lpBanksysIoOut;

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

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

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

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

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

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

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

5.1.15 WFS_CMD_PIN_RESET

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

5.1.16 WFS_CMD_PIN_HSM_SET_TDATA

Description This function allows 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:

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

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:

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

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

Value	Meaning
WFS_SRVE_PIN_HSM_TDATA_CHANGED	The terminal data has changed.

Comments None.

5.1.17 WFS_CMD_PIN_SECURE_MSG_SEND

Description This command handles all messages that should be 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:

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

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

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

Output Param LPWFSPINSECMSG lpSecMsgOut;

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

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

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

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

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

Comments None.

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:

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

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:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this message.
WFS_ERR_PIN_MACINVALID	The MAC of the message is not correct.
WFS_ERR_PIN_PROTINVALID	The specified protocol is invalid.
WFS_ERR_PIN_FORMATINVALID	The format of the message is invalid.
WFS_ERR_PIN_CONTENTINVALID	The contents of one of the security relevant fields are invalid.
WFS_ERR_PIN_KEYNOTFOUND	No key was found for MAC verification.

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

Value	Meaning
WFS_SRVE_PIN_HSM_TDATA_CHANGED	The terminal data has changed.

Comments None.

5.1.19 WFS_CMD_PIN_GET_JOURNAL

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

Input Param LPWORD lpwProtocol;

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

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

Output Param LPWFSXDATA lpxJournalData;

lpxJournalData
Pointer to the journal data.

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

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

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

Comments None.

5.1.20 WFS_CMD_PIN_IMPORT_KEY_EX

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

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

Input Param LPWFSPINIMPORTKEYEX lpImportKeyEx;

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

lpsKey
Specifies the name of key being loaded.

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

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

lpxControlVector
Specifies the control vector of the key to be loaded. It contains the attributes of the key. If this parameter is NULL the keys is only specified by *dwUse*. See also [Ref. 26].

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

Value	Meaning
WFS_PIN_USECRYPT	Key is used for encryption and decryption.
WFS_PIN_USEFUNCTION	Key is used for PIN block creation.
WFS_PIN_USEMACING	Key is used for MACing.
WFS_PIN_USEKEYENCKEY	Key is used as key encryption key.
WFS_PIN_USEPINLOCAL	Key is used for local PIN check.
WFS_PIN_USERSAPUBLIC	Key is used as a public key for RSA encryption including EMV PIN block creation.
WFS_PIN_USERSAPRIVATE	Key is used as a private key for RSA decryption (it is not recommend that private keys are imported with this function).

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

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

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

Output Param None.

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

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments

When keys are loaded in multiple parts, all parts of the key loaded must set the relevant construction value in the *dwUse* field along with any 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:

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

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:

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

Events None.

Comments 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:

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

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

Output Param None.

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

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

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

Value	Meaning
WFS_SRVE_PIN_HSM_TDATA_CHANGED	The terminal data has changed.

Comments None.

5.1.23 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 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 LPWFSPINSECUREKEYENTRY lpSecureKeyEntry;

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

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

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

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

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

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

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

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

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

Output Param LPWFSPINSECUREKEYENTRYOUT lpSecureKeyEntryOut;

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

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

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

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

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

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

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

Value	Meaning
WFS_EXEE_PIN_KEY	A key has been pressed at the 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 LPWFSPINGENERATEKCV lpGenerateKCV;

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

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

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

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

Output Param LPWFSPINKCV lpKCV;

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

lpxKCV
Contains the key check value data that can be used for verification of the key.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key encryption key was not found.
WFS_ERR_PIN_KEYNOVALUE	The specified key exists but has no value loaded.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_MODENOTSUPPORTED	The KCV mode is not supported.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

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:

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.

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

Output Param None.

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

Value	Meaning
WFS_ERR_PIN_INVALID_PORT	An attempt to set a guidance light to a new value was invalid because the guidance light does not exist.

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.

Value	Meaning
TRUE	The PIN should be maintained after PIN processing commands for multiple uses.
FALSE	The PIN will be cleared and subsequent PINs will not be maintained for multiple uses.

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:

Value	Meaning
WFS_PIN_BEEP_ON_ACTIVE	Specifies that beeping should be enabled for active keys. If this flag is not present then beeping is disabled for active keys.
WFS_PIN_BEEP_ON_INACTIVE	Specifies that beeping should be enabled for in-active keys. If this flag is not present then beeping is disabled for in-active keys.

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:

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.1.29 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:

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_INVALIDHSM	The logical HSM serial number specified is not valid.

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

Value	Meaning
WFS_SRVE_PIN_HSM_CHANGED	Indicates that the current logical HSM has changed to the HSM identified within the event.

Comments None.

5.1.30 WFS_CMD_PIN_IMPORT_KEYBLOCK

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 LPWFSPINIMPORTKEYBLOCK lpImportKeyBlock;

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

lpsKey
Specifies the name of key being loaded.

lpsEncKey
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:

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments 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:

Value	Meaning
WFS_ERR_PIN_POWERSAVETOOSHORT	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.

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

Value	Meaning
WFS_SRVE_PIN_POWER_SAVE_CHANGE	The power save recovery time has changed.

Comments None.

5.2 Common commands for Remote Key Loading Schemes

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

5.2.1 WFS_CMD_PIN_START_KEY_EXCHANGE

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

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

The WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY and WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY commands end the key exchange process.

Input Param None.

Output Param LPWFSPINSTARTKEYEXCHANGE lpStartKeyExchange;

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

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

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

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

Events None.

Comments None.

5.3 Remote Key Loading Using Signatures

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

5.3.1 WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY

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

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

Input Param LPWFSPINIMPORTRSAPUBLICKEY lpImportRSAPublicKey;

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

lpsKey
Specifies the name of key being loaded.

lpxValue
Contains the PKCS #1 formatted RSA Public Key to be loaded, represented in DER encoded ASN.1.

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

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

If *dwUse* equals zero the specified key is deleted.

When no signature is required to authenticate the deletion of a public key, all parameters but *lpsKey* are ignored. In addition, WFS_CMD_PIN_IMPORT_KEY, WFS_CMD_PIN_IMPORT_KEY_EX, WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY and WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY can be used to delete a key that has been imported with this command.

When a signature is required to authenticate the deletion of the public key, all parameters in the command are used. *lpxValue* must contain the concatenation of the 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. UI_{ATM}|| $PK_{TO\ DELETE}$. *lpxSignature* contains the signature generated from *lpxValue* using the private key component of the public key being deleted.

The equivalent commands in the certificate scheme must not be used to delete a key imported through the signature scheme.

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

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

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

lpxSignature
Contains the Signature associated with the key being imported 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 lpImportRSAPublicKeyOutput;

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

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

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

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

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOTFOUND	The key name supplied in *lpsSigKey* was not found.
WFS_ERR_PIN_USEVIOLATION	An invalid use was specified for the key being imported.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.
WFS_ERR_PIN_SIG_NOT_SUPP	The Service Provider does not support the Signature Algorithm requested. The key was discarded.
WFS_ERR_PIN_SIGNATUREINVALID	The signature verification failed. The key has not been stored or deleted.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3.2 WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM

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

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

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

See section 8.1.7 (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 LPWFSPINEXPORTRSAISSUERSIGNEDITEM lpExportRSAIssuerSignedItem;

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

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

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

lpsName
Specifies the name of the public key to be exported. The private/public key pair was installed during manufacture; see section 8.1.7 (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 LPWFSPINEXPORTRSAISSUERSIGNEDITEMOUTPUT lpExportRSAIssuerSignedItemOutput;

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

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

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

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

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

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

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3.3 WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY

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

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

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

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

Input Param LPWFSPINIMPORTRSASIGNEDDESKEY lpImportRSASignedDESKey;

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

lpsKey
Specifies the name of key being loaded.

lpsDecryptKey
Specifies the name of the RSA private key used to decrypt the symmetric key. See section 8.1.7 (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:

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

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

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

Value	Meaning
WFS_PIN_USECRYPT	Key is used for encryption and decryption.
WFS_PIN_USEFUNCTION	Key is used for PIN block creation.
WFS_PIN_USEMACING	Key is used for MACing.
WFS_PIN_USEKEYENCKEY	Key is used as key encryption key.
WFS_PIN_USEPINLOCAL	Key is used for local PIN check.
WFS_PIN_USENODUPLICATE	Key can be imported only once.
WFS_PIN_USESVENCKEY	Key is used as CBC Start Value encryption key.
WFS_PIN_USEANSTR31MASTER	Key can be used for importing keys packaged within an ANS TR-31 key block.

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

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

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

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

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

Output Param LPWFSPINIMPORTRSASIGNEDDESKEYOUTPUT lpImportRSASignedDESKeyOutput;

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

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

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

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

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

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

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

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3.4 WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR

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

The newly generated key pair can only be used for the use defined in the *dwUse* flag. This flag defines the use of the private key; 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:

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

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

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

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

Output Param None.

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

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

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

command:

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3.5 WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED_ITEM

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

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

See section 8.1.7 (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 LPWFSPINEXPORTRSAEPPSIGNEDITEM lpExportRSAEPPSignedItem;

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

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

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

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

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

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

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

Output Param LPWFSPINEXPORTRSAEPPSIGNEDITEMOUTPUT lpExportRSAEPPSignedItemOutput;

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

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

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

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

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

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.4 Remote Key Loading with Certificates

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

5.4.1 WFS_CMD_PIN_LOAD_CERTIFICATE

Description This command is used to load a host certificate or to load a new encryptor certificate from a Certificate Authority to make remote key loading possible. This command can be called only once if there are no plans for a new CA to take over the duties. If a new CA does take over the duties, then this command should be called after the WFS_CMD_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:

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

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

Value	Meaning
WFS_SRVE_PIN_CERTIFICATE_CHANGE	The certificate module state has changed.

Comments None.

5.4.2 WFS_CMD_PIN_GET_CERTIFICATE

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

Input Param LPWFSPINGETCERTIFICATE lpGetCertificate;

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

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

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

Output Param LPWFSPINGETCERTIFICATEOUPUT lpGetCertificateOutput;

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

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

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

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

Events None.

Comments None.

5.4.3 WFS_CMD_PIN_REPLACE_CERTIFICATE

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

Input Param LPWFSPINREPLACECERTIFICATE lpReplaceCertificate;

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

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

Output Param LPWFSPINREPLACECERTIFICATEOUTPUT lpReplaceCertificateOuput

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

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

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

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

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

Value	Meaning
WFS_SRVE_PIN_CERTIFICATE_CHANGE	The certificate module state has changed.

Comments None.

5.4.4 WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY

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

Input Param LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEY lpImportRSAEncipheredPKCS7Key;

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

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

lpsKey
Specifies the name of the key to be stored.

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

Value	Meaning
WFS_PIN_USECRYPT	Key can be used for encryption/decryption.
WFS_PIN_USEFUNCTION	Key can be used for PIN functions.
WFS_PIN_USEMACING	Key can be used for MACing.
WFS_PIN_USEKEYENCKEY	Key is used as key encryption key.
WFS_PIN_USENODUPLICATE	Key can be imported only once.
WFS_PIN_USESVENCKEY	Key is used as CBC Start Value encryption key.
WFS_PIN_USEANSTR31MASTER	Key can be used for importing keys packaged within an ANS TR-31 key block.

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

Output Param LPWFSPINIMPORTRSAENCIPHEREDPKCS7KEYOUTPUT lpImportRSAEncipheredKeyOut;

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

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

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

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

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

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments 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:

```
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:

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

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

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

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

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

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

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

If *wImportScheme* is WFS_PIN_EMV_IMPORT_EPI_CA then *lpxImportData* contains the concatenation of tables 4 and 13, as specified in 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 *lpxImportData* contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the Service Provider. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value: '9F32'), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value: '90'), the remainder length (1 byte). The remainder value (variable length – EMV Tag value: '92'), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value: '5A'). The Service Provider will compare the leftmost three 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, Book2 – Security & Key Management Version 4.0, Table 4 (See [Ref. 4] under the reference section for this document). *lpsSigKey* defines the previously loaded key used to verify the signature.

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

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

If *wImportScheme* is WFS_PIN_EMV_IMPORT_PKCSV1_5_CA then *lpxImportData* contains the CA public key signed with the previously loaded public key specified in *lpsSigKey*. *lpxImportData* consists of the concatenation of EMV 2000 Book II Table 23(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:

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

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

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

5.5.2 WFS_CMD_PIN_DIGEST

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

Input Param LPWFSPINDIGEST lpDigest;

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

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

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

Output Param LPWFSPINDIGESTOUTPUT lpDigestOutput;

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

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

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

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

Events None.

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 LPWFSPINKEY lpKey;

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

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

ulDigit
Specifies the digit entered by the user. When working in encryption mode or secure key entry mode (WFS_CMD_PIN_GET_PIN and WFS_CMD_PIN_SECUREKEY_ENTRY), the value of this field is 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;

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:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not loaded or attempting to delete a non-existent key.
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_ALGORITHMNOTSUPP	The specified algorithm is not supported by this key.

Comments None.

6.4 WFS_SRVE_PIN_OPT_REQUIRED

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

Event Param 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:

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

Comments None.

6.6 WFS_SRVE_PIN_HSM_TDATA_CHANGED

Description This event indicates that one of the values of the terminal data has changed (these are the data that can be set using WFS_CMD_PIN_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_HSM_INIT 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 LPWFSPINHSMIDENTIFIER lpHSMChanged;

```
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

Description This mandatory event notifies the application when the device is ready for the user to start entering data.

Event Param None.

Comments None.

6.9 WFS_SRVE_PIN_DEVICEPOSITION

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

Event Param LPWFSPINDEVICEPOSITION lpDevicePosition;

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

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

Value	Meaning
WFS_PIN_DEVICEINPOSITION	The device is in its normal operating position.
WFS_PIN_DEVICENOTINPOSITION	The device has been removed from its normal operating position.
WFS_PIN_DEVICEPOSUNKNOWN	The position of the device cannot be determined.

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;

```
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_IO                      (PIN_SERVICE_OFFSET + 17)
#define WFS_CMD_PIN_RESET                           (PIN_SERVICE_OFFSET + 18)
#define WFS_CMD_PIN_HSM_SET_TDATA                   (PIN_SERVICE_OFFSET + 19)
#define WFS_CMD_PIN_SECURE_MSG_SEND                 (PIN_SERVICE_OFFSET + 20)
#define WFS_CMD_PIN_SECURE_MSG_RECEIVE              (PIN_SERVICE_OFFSET + 21)
#define WFS_CMD_PIN_GET_JOURNAL                     (PIN_SERVICE_OFFSET + 22)
#define WFS_CMD_PIN_IMPORT_KEY_EX                   (PIN_SERVICE_OFFSET + 23)
#define WFS_CMD_PIN_ENC_IO                          (PIN_SERVICE_OFFSET + 24)
#define WFS_CMD_PIN_HSM_INIT                        (PIN_SERVICE_OFFSET + 25)
#define WFS_CMD_PIN_IMPORT_RSA_PUBLIC_KEY           (PIN_SERVICE_OFFSET + 26)
```

```
#define WFS_CMD_PIN_EXPORT_RSA_ISSUER_SIGNED_ITEM   (PIN_SERVICE_OFFSET + 27)
#define WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY       (PIN_SERVICE_OFFSET + 28)
#define WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR           (PIN_SERVICE_OFFSET + 29)
#define WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED_ITEM      (PIN_SERVICE_OFFSET + 30)
#define WFS_CMD_PIN_LOAD_CERTIFICATE                (PIN_SERVICE_OFFSET + 31)
#define WFS_CMD_PIN_GET_CERTIFICATE                 (PIN_SERVICE_OFFSET + 32)
#define WFS_CMD_PIN_REPLACE_CERTIFICATE             (PIN_SERVICE_OFFSET + 33)
#define WFS_CMD_PIN_START_KEY_EXCHANGE              (PIN_SERVICE_OFFSET + 34)
#define WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY (PIN_SERVICE_OFFSET + 35)
#define WFS_CMD_PIN_EMV_IMPORT_PUBLIC_KEY           (PIN_SERVICE_OFFSET + 36)
#define WFS_CMD_PIN_DIGEST                          (PIN_SERVICE_OFFSET + 37)
#define WFS_CMD_PIN_SECUREKEY_ENTRY                 (PIN_SERVICE_OFFSET + 38)
#define WFS_CMD_PIN_GENERATE_KCV                    (PIN_SERVICE_OFFSET + 39)
#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 WFSPINSTATUS.fwDevice */

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

/* values of WFSPINSTATUS.fwEncStat */

#define WFS_PIN_ENCREADY                            (0)
#define WFS_PIN_ENCNOTREADY                         (1)
#define WFS_PIN_ENCNOTINITIALIZED                   (2)
#define WFS_PIN_ENCBUSY                             (3)
#define WFS_PIN_ENCUNDEFINED                        (4)
#define WFS_PIN_ENCINITIALIZED                      (5)
#define WFS_PIN_ENCPINTAMPERED                      (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 WFSPINSTATUS.dwGuidLights [...]
              WFSPINCAPS.dwGuidLights [...]
*/

#define     WFS_PIN_GUIDANCE_PINPAD                 (0)

/* Values of WFSPINSTATUS.dwGuidLights [...]
             WFSPINCAPS.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_PIN_DEVICEPOSUNKNOWN                (2)
#define     WFS_PIN_DEVICEPOSNOTSUPP                (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_PIN_FORMISO0                            (0x0004)
#define WFS_PIN_FORMISO1                            (0x0008)
#define WFS_PIN_FORMECI2                            (0x0010)
#define WFS_PIN_FORMECI3                            (0x0020)
#define WFS_PIN_FORMVISA                            (0x0040)
#define WFS_PIN_FORMDIEBOLD                         (0x0080)
#define WFS_PIN_FORMDIEBOLDCO                       (0x0100)
#define WFS_PIN_FORMVISA3                           (0x0200)
#define WFS_PIN_FORMBANKSYS                         (0x0400)
#define WFS_PIN_FORMEMV                             (0x0800)
#define WFS_PIN_FORMISO3                            (0x2000)
#define WFS_PIN_FORMAP                              (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)

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

#define WFS_PIN_USECRYPT                            (0x0001)
#define WFS_PIN_USEFUNCTION                         (0x0002)
#define WFS_PIN_USEMACING                           (0x0004)
#define WFS_PIN_USEKEYENCKEY                        (0x0020)
#define WFS_PIN_USENODUPLICATE                      (0x0040)
#define WFS_PIN_USESVENCKEY                         (0x0080)
#define WFS_PIN_USECONSTRUCT                        (0x0100)
#define WFS_PIN_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_USECHIPPIN                          (0x00200000)
#define WFS_PIN_USECHIPPS                           (0x00400000)
#define WFS_PIN_USECHIPMAC                          (0x00800000)
#define WFS_PIN_USECHIPLT                           (0x01000000)
#define WFS_PIN_USECHIPMACLZ                        (0x02000000)
#define WFS_PIN_USECHIPMACAZ                        (0x04000000)
#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_RES8                                 (0x02000000)
#define WFS_PIN_FK_OEM1                                 (0x04000000)
#define WFS_PIN_FK_OEM2                                 (0x08000000)
#define WFS_PIN_FK_OEM3                                 (0x10000000)
#define WFS_PIN_FK_OEM4                                 (0x20000000)
#define WFS_PIN_FK_OEM5                                 (0x40000000)
#define WFS_PIN_FK_OEM6                                 (0x80000000)

/* additional values of WFSPINFUNCKEYDETAIL.ulFuncMask */

#define WFS_PIN_FK_UNUSED                               (0x00000000)

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

/* values of 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                        (0x00200000)
#define WFS_PIN_FK_FDK23                        (0x00400000)
#define WFS_PIN_FK_FDK24                        (0x00800000)
#define WFS_PIN_FK_FDK25                        (0x01000000)
#define WFS_PIN_FK_FDK26                        (0x02000000)
#define WFS_PIN_FK_FDK27                        (0x04000000)
#define WFS_PIN_FK_FDK28                        (0x08000000)
#define WFS_PIN_FK_FDK29                        (0x10000000)
#define WFS_PIN_FK_FDK30                        (0x20000000)
#define WFS_PIN_FK_FDK31                        (0x40000000)
#define WFS_PIN_FK_FDK32                        (0x80000000)

/* values of WFSPINCRYPT.wMode */

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

/* values of WFSPINENTRY.wCompletion */

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

/* values of WFSPINSECMSG.wProtocol */

#define WFS_PIN_PROTISOAS                       (1)
#define WFS_PIN_PROTISOLZ                       (2)
#define WFS_PIN_PROTISOPS                       (3)
#define WFS_PIN_PROTCHIPZKA                     (4)
#define WFS_PIN_PROTRAWDATA                     (5)
#define WFS_PIN_PROTPBM                         (6)
#define WFS_PIN_PROTHSMLDI                      (7)
#define WFS_PIN_PROTGENAS                       (8)
#define WFS_PIN_PROTCHIPINCHG                   (9)
#define WFS_PIN_PROTPINCMP                      (10)
#define WFS_PIN_PROTISOPINCHG                   (11)

/* values of WFSPINHSMINIT.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
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_EXPORT_EPP_ID                       (0x0001)
#define WFS_PIN_EXPORT_PUBLIC_KEY                   (0x0002)

/* values of WFSPINIMPORTRSASIGNEDDESKEY.dwRSAEncipherAlgorithm and
WFSPINCAPS.dwRSACryptAlgorithm */

#define WFS_PIN_CRYPT_RSAES_PKCS1_V1_5              (0x00000001)
#define WFS_PIN_CRYPT_RSAES_OAEP                    (0x00000002)

/* values of WFSPINGENERATERSAKEYPAIR.wExponentValue */

#define WFS_PIN_DEFAULT                             (0)
#define WFS_PIN_EXPONENT_1                          (1)
#define WFS_PIN_EXPONENT_4                          (2)
#define WFS_PIN_EXPONENT_16                         (3)

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

#define WFS_PIN_KEYSINGLE                           (0x0001)
#define WFS_PIN_KEYDOUBLE                           (0x0002)

/* values of WFSPINGETCERTIFICATE.wGetCertificate  */

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

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

#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.fwKeyEntryMode */
```

```
#define WFS_PIN_SECUREKEY_NOTSUPP                   (0x0000)
#define WFS_PIN_SECUREKEY_REG_SHIFT                 (0x0001)
#define WFS_PIN_SECUREKEY_REG_UNIQUE                (0x0002)
#define WFS_PIN_SECUREKEY_IRREG_SHIFT               (0x0004)
#define WFS_PIN_SECUREKEY_IRREG_UNIQUE              (0x0008)

/* 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_HSMSTATEINVALID                 (-(PIN_SERVICE_OFFSET + 23))
#define WFS_ERR_PIN_MACINVALID                      (-(PIN_SERVICE_OFFSET + 24))
#define WFS_ERR_PIN_PROTINVALID                     (-(PIN_SERVICE_OFFSET + 25))
#define WFS_ERR_PIN_FORMATINVALID                   (-(PIN_SERVICE_OFFSET + 26))
#define WFS_ERR_PIN_CONTENTINVALID                  (-(PIN_SERVICE_OFFSET + 27))
#define WFS_ERR_PIN_SIG_NOT_SUPP                    (-(PIN_SERVICE_OFFSET + 29))
#define WFS_ERR_PIN_INVALID_MOD_LEN                 (-(PIN_SERVICE_OFFSET + 31))
#define WFS_ERR_PIN_INVALIDCERTSTATE                (-(PIN_SERVICE_OFFSET + 32))
#define WFS_ERR_PIN_KEY_GENERATION_ERROR            (-(PIN_SERVICE_OFFSET + 33))
#define WFS_ERR_PIN_EMV_VERIFY_FAILED               (-(PIN_SERVICE_OFFSET + 34))
#define WFS_ERR_PIN_RANDOMINVALID                   (-(PIN_SERVICE_OFFSET + 35))
#define WFS_ERR_PIN_SIGNATUREINVALID                (-(PIN_SERVICE_OFFSET + 36))
#define WFS_ERR_PIN_SNSCDINVALID                    (-(PIN_SERVICE_OFFSET + 37))
#define WFS_ERR_PIN_NORSAKEYPAIR                    (-(PIN_SERVICE_OFFSET + 38))
#define WFS_ERR_PIN_INVALID_PORT                    (-(PIN_SERVICE_OFFSET + 39))
#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                  usNumberFDKs;
    LPWFSPINFDK             *lppFDKs;
} WFSPINFUNCKEYDETAIL, *LPWFSPINFUNCKEYDETAIL;

typedef struct _wfs_pin_key_detail_ex
{
```

```
    LPSTR                   lpsKeyName;
    DWORD                   dwUse;
    BYTE                    bGeneration;
    BYTE                    bVersion;
    BYTE                    bActivatingDate[4];
    BYTE                    bExpiryDate[4];
    BOOL                    bLoaded;
    LPWFSXDATA              lpxKeyBlockHeader;
} WFSPINKEYDETAILEX, *LPWFSPINKEYDETAILEX;

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

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

/* WFS_INF_PIN_PCIPTS_DEVICE_ID command output structure */
typedef struct _wfs_pin_pcipts_deviceid
{
    LPSTR                   lpszManufacturerIdentifier;
    LPSTR                   lpszModelIdentifier;
    LPSTR                   lpszHardwareIdentifier;
    LPSTR                   lpszFirmwareIdentifier;
    LPSTR                   lpszApplicationIdentifier;
} WFSPINPCIPTSDEVICEID, *LPWFSPINPCIPTSDEVICEID;

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

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

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

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

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

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

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

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

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

typedef struct _wfs_pin_local_visa
{
```

```
    LPSTR                   lpsPAN;
    LPSTR                   lpsPVV;
    WORD                    wPVVDigits;
    LPSTR                   lpsKey;
    LPWFSXDATA              lpxKeyEncKey;
} WFSPINLOCALVISA, *LPWFSPINLOCALVISA;

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

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

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

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

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

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

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

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

typedef struct _wfs_pin_local_banksys
{
```

```
    LPWFSXDATA              lpxATMVAC;
} WFSPINLOCALBANKSYS, *LPWFSPINLOCALBANKSYS;

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

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

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

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

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

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

/* 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;
```

```
} WFSPINIMPORTRSAPUBLICKEY, *LPWFSPINIMPORTRSAPUBLICKEY;

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

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

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

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

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

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

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

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

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

typedef struct _wfs_pin_load_certificate_output
{
```

```
    LPWFSXDATA              lpxCertificateData;
} WFSPINLOADCERTIFICATEOUTPUT, *LPWFSPINLOADCERTIFICATEOUTPUT;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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;
    LPWFSPINHSMINFO         *lppHSMInfo;
} WFSPINHSMDETAIL, *LPWFSPINHSMDETAIL;


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

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


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

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

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

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



/* restore alignment */
```

```
#pragma pack(pop)

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

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

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

8.1 Remote Key Loading Using Signatures

8.1.1 RSA Data 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 $digest_2$;
2. ATM uses the Host's public key to decrypt the digital signature. The digital signature was produced using the Host's private key to encrypt the data digest; therefore, when decrypted with the Host's public key it produces the same digest. Consider this $digest_1$. Incidentally, no other public key in the world would work to decrypt $digest_1$ – only the public key corresponding to the signing private key.
3. ATM compares $digest_1$ with $digest_2$.

If $digest_1$ matches $digest_2$ exactly, the ATM has confirmed the following:

- Data was not tampered with in transit. Changing a single bit in the data sent from the Host to the ATM would cause $digest_2$ to be different than $digest_1$. Every data block has a unique digest; therefore, an altered data block is detected by the ATM.
- Public key used to decrypt the digital signature corresponds to the private key used to create it. No other public key could possibly work to decrypt the digital signature, so the ATM was not handed someone else's public key.

This gives an overview of how Digital Signatures can be used in Data Authentication. In particular, Signatures can be used to validate 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 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 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 Sign(SK_{SI})[UI_{ATM}] for the Unique Id (UI_{ATM}) on the ATM PIN. The Signature Issuer uses SK_{SI} to do this.
5. The Signature Issuer installs its Public Key (PK_{SI}) on the ATM PIN. It also derives and installs the Signature Sign(SK_{SI})[PK_{ATM}] of the ATM PIN's Public Key (PK_{ATM}) on the ATM PIN. The Signature Issuer uses SK_{SI} to do this.
6. The ATM PIN device additionally contains its own Public (PK_{ATM}) 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 (K_M) with PK_{ATM}. A signature for this is then created using SK_{HOST}. The ATM PIN will then validate the signature using PK_{HOST} and then obtain the master key by decrypting using SK_{ATM}.

The XFS 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 (K_M) with PK_{ATM}. A signature for the random number and encrypted key is then created using SK_{HOST}.

The ATM PIN will then validate the signature using $PK_{HOST,}$ verify the random number and then obtain the master key by decrypting using SK_{ATM}.

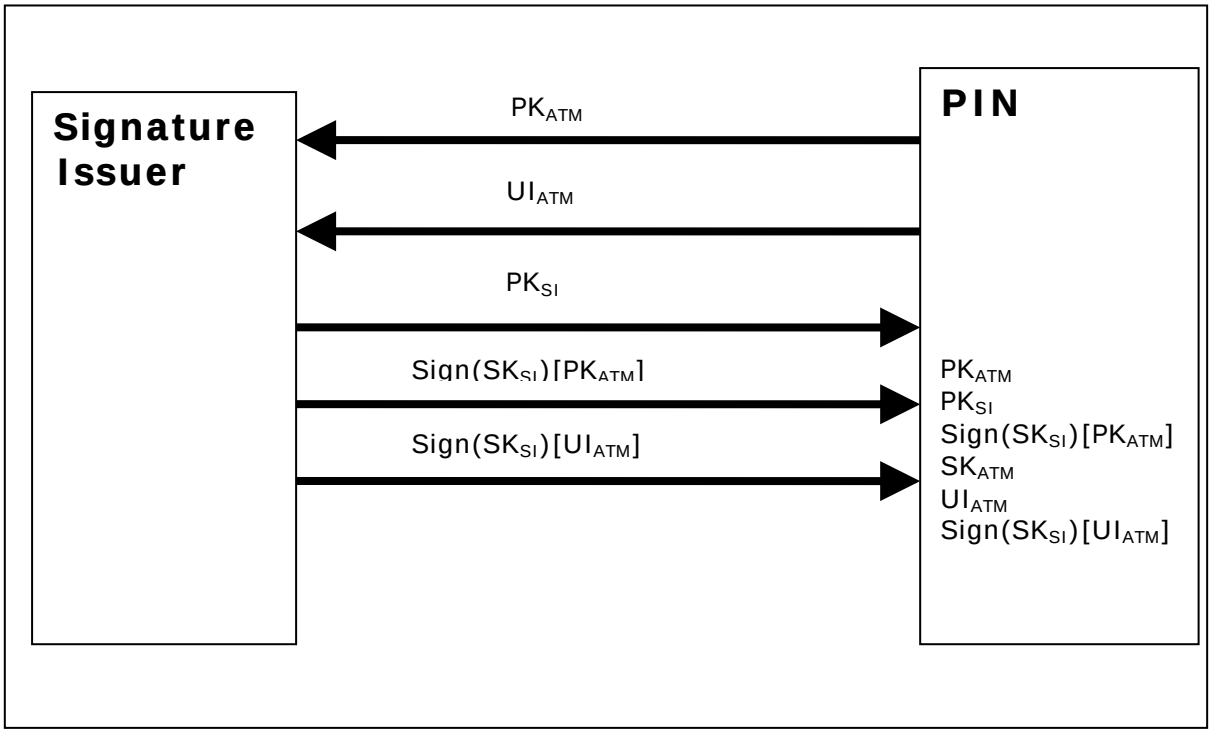
The XFS commands used to exchange master symmetric keys as described above are:

- WFS_CMD_PIN_START_KEY_EXCHANGE
- WFS_CMD_PIN_IMPORT_RSA_SIGNED_DES_KEY

The following diagrams summaries the key exchange process described above:

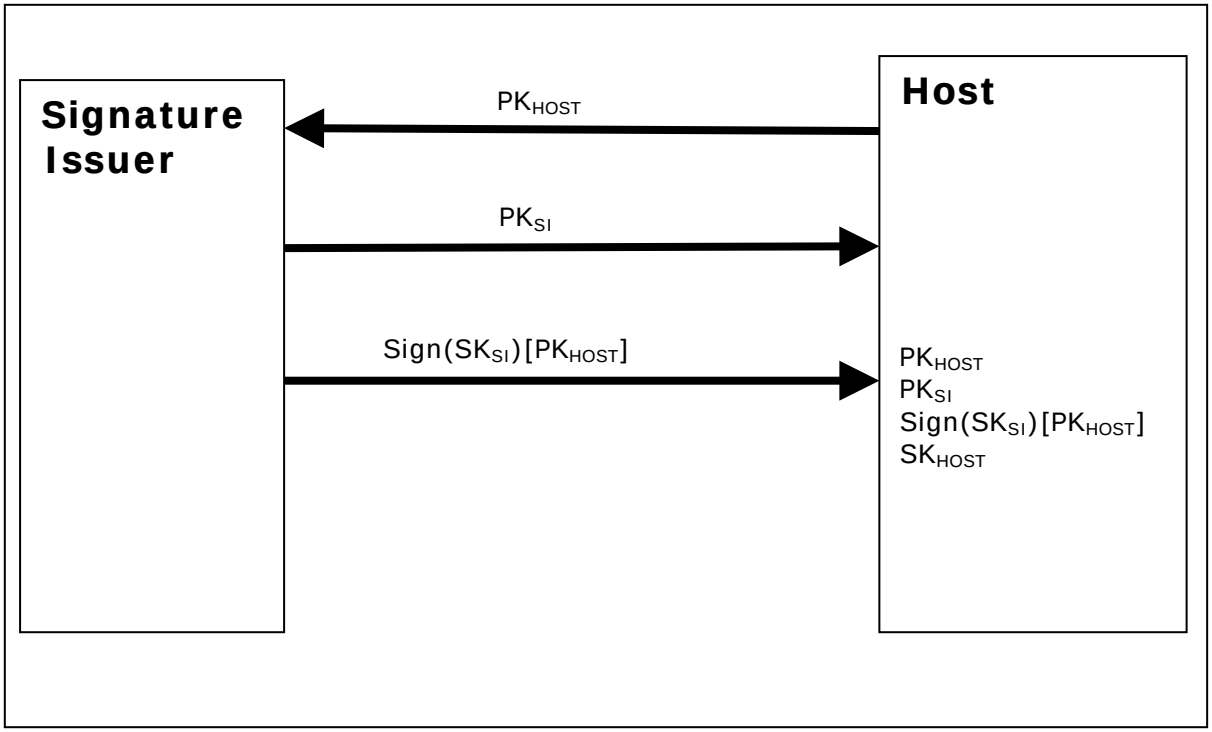
8.1.3 Initialization Phase – Signature Issuer and ATM PIN

This would typically occur in a secure manufacturing environment.



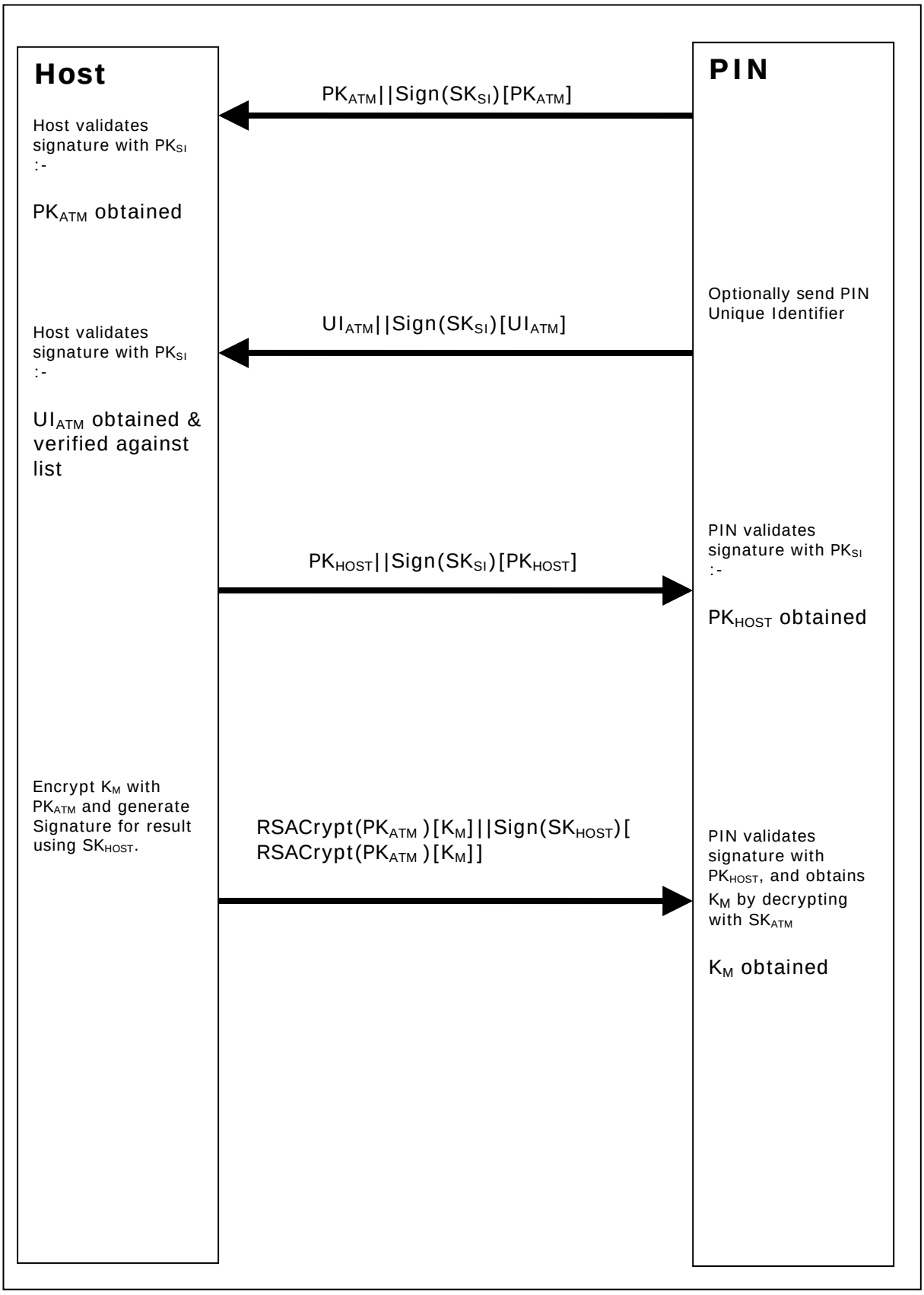
8.1.4 Initialization Phase – Signature Issuer and Host

This would typically occur in a secure offline environment.



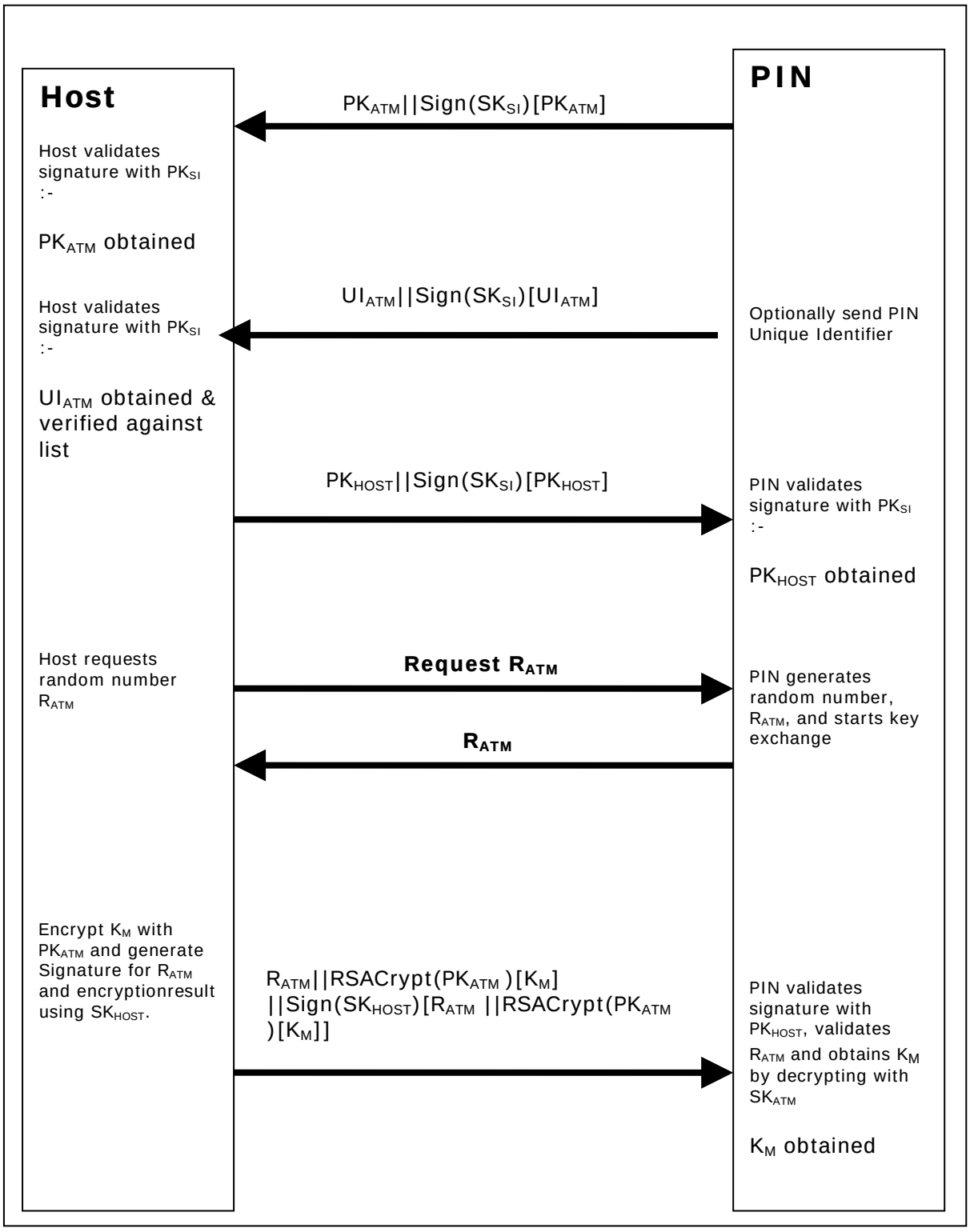
8.1.5 Key Exchange – Host and ATM PIN

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.



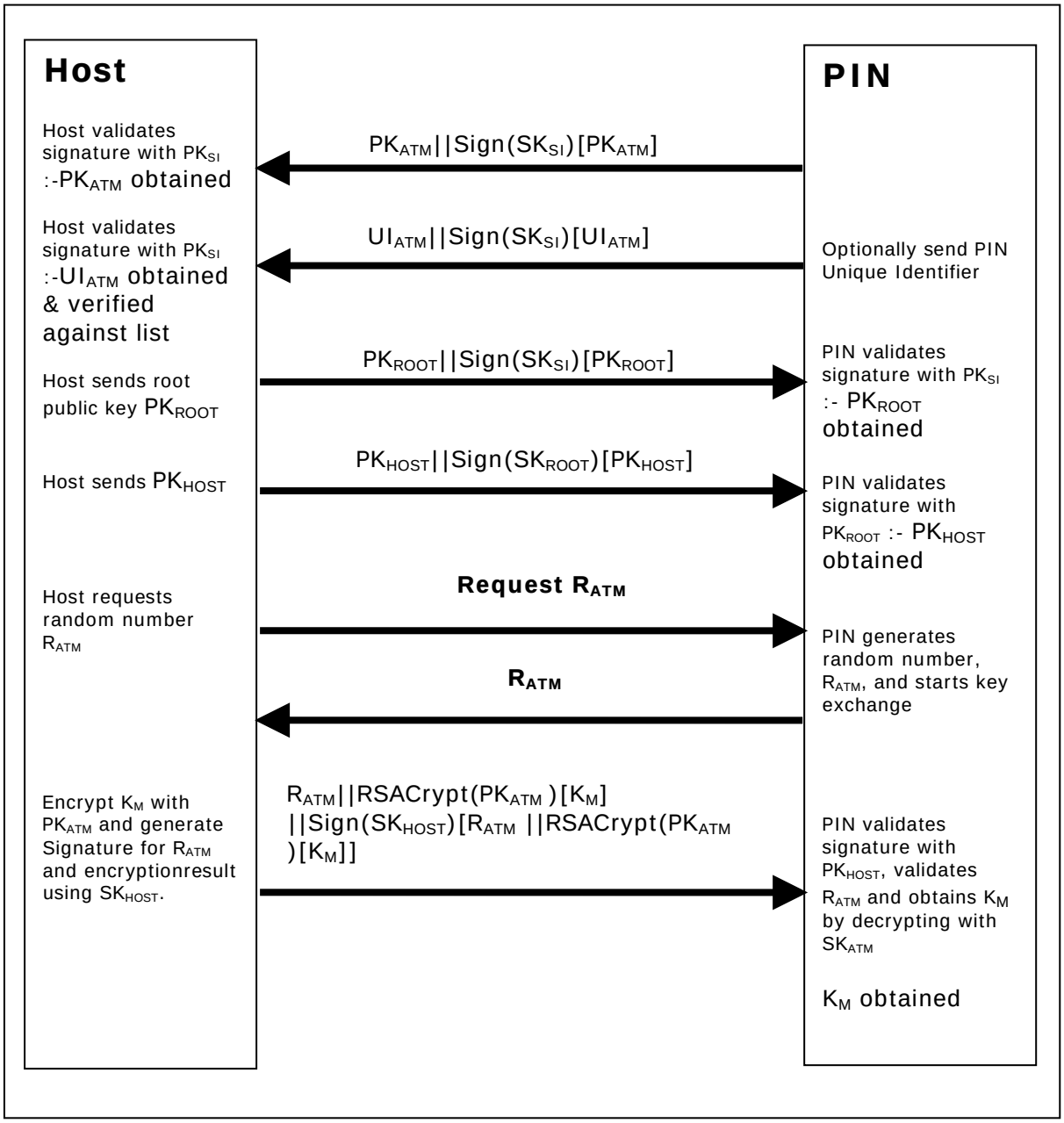
8.1.6 Key Exchange (with random number) – Host and ATM PIN

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.



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.

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

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

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

8.2 Remote Key Loading Using Certificates

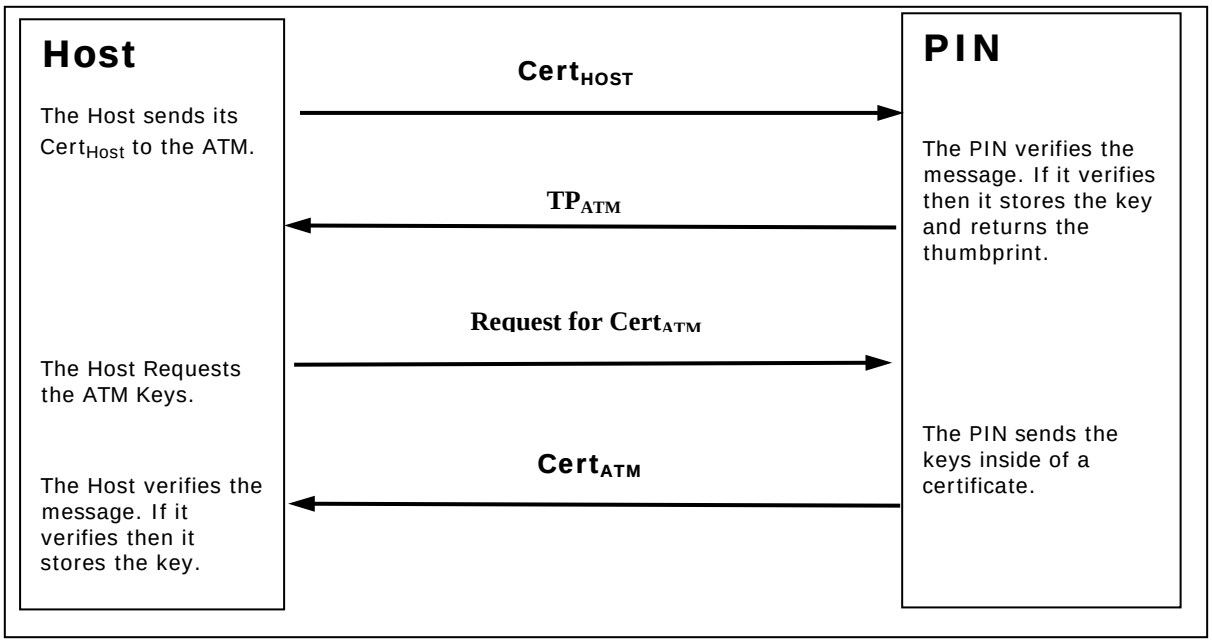
8.2.1 Certificate Exchange and Authentication

In summary, both end points, the ATM 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:



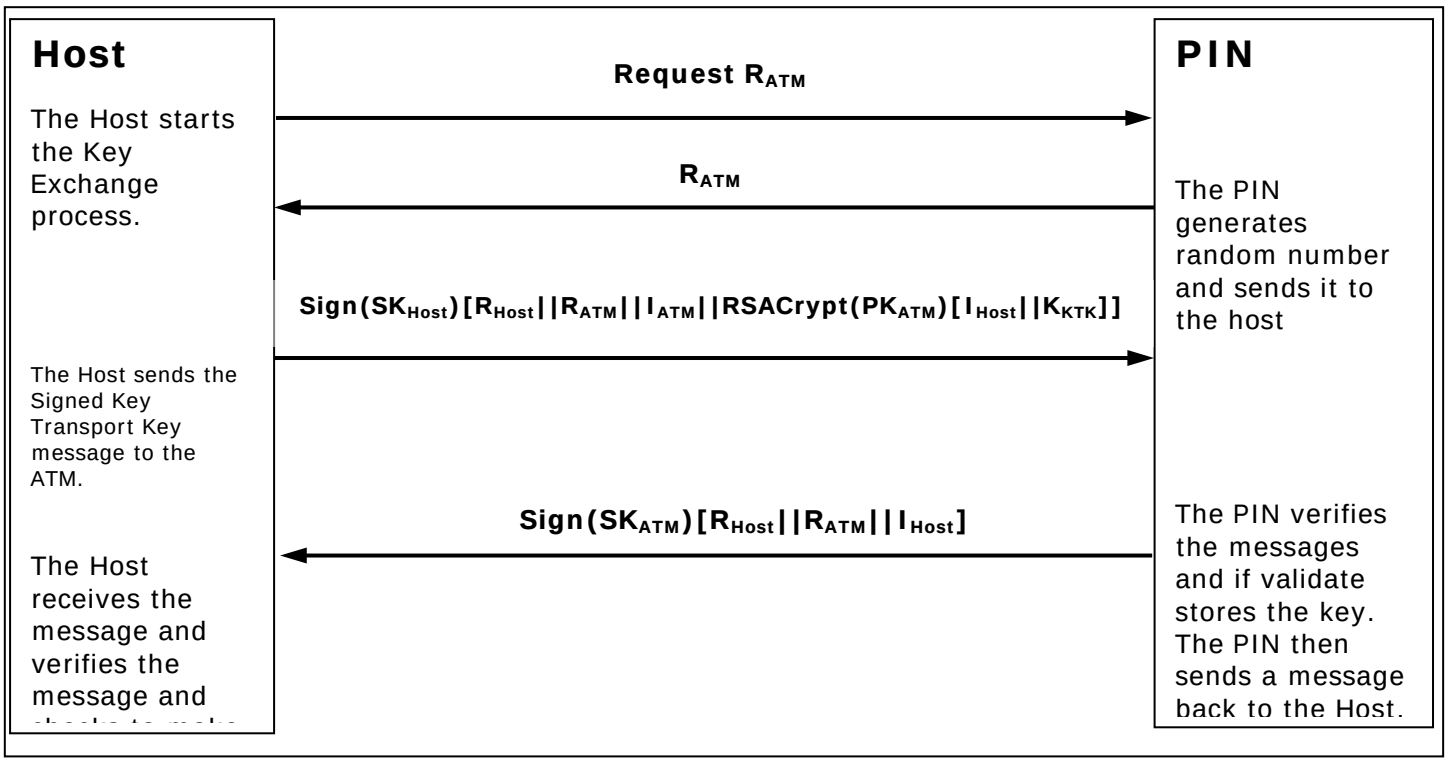
8.2.2 Remote Key Exchange

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

1. First, the WFS_CMD_PIN_START_KEY_EXCHANGE is called. This returns R_{ATM} from the encryptor to be used in the authenticating the WFS_CMD_PIN_IMPORT_RSA_ENCHIPERED_PKCS7_KEY message.
2. Next, WFS_CMD_PIN_IMPORT_RSA_ENCIPHERED_PKCS7_KEY is called. This 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 encyrptor. This message is sent to the host.
   d) The HOST verifies the message sent from the encryptor by using the ATM's public verification key. The HOST then checks the identifier of the host and then compares the identifier in the message with the one stored in the HOST. Then checks the random number sent in the message and to the one stored in the HOST. The HOST finally checks the encryptor's random number with the one received in received in the WFS_CMD_PIN_START_KEY_EXCHANGE command.

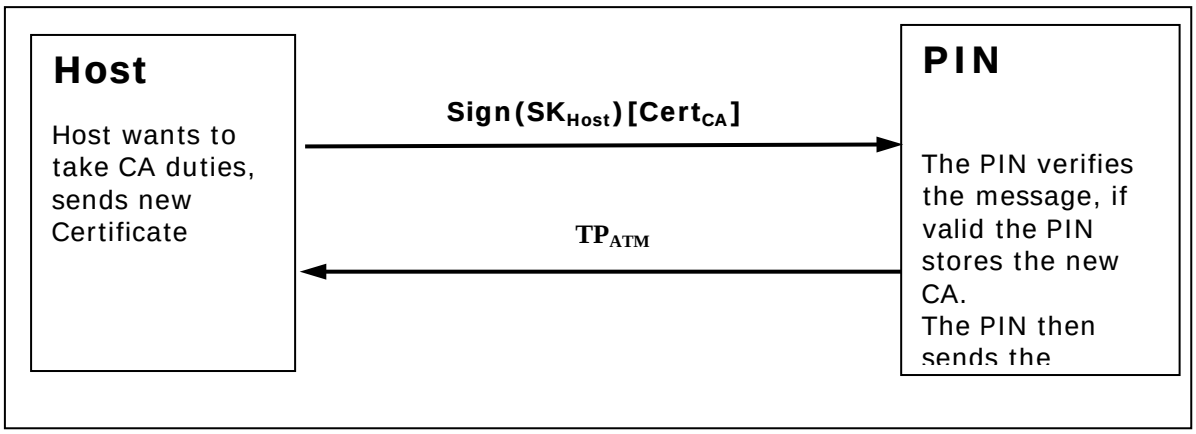
The following diagram below shows how the Host and ATM transmit the Key Transport Key.



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:

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

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

The following bitmap positions are checked by the Service Provider and have to be filled by the application:

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

For 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 *lpbMsg* of WFSPINSECMSG contains a subcommand, which is used to qualify the type of operation. The remaining bytes of the command data are depending on the value of the subcommand.

The following sub-commands are defined:

- GeneratePAC (Code 0x01)
  Returns the encrypted PIN block together with generation and version values of the Master Key and the PAC random value.
- GetMACRandom (Code 0x02)
  Returns the generation and version values of the Master Key and the MAC random value.
- GenerateMAC (Code 0x03)
  Returns the generated MAC for the message data passed in. Note that the MAC is generated for exactly the data that is presented (contents and sequence). Data that should not go into MAC calculation must not be passed in.
- VerifyMAC (Code 0x04)
  Generates a MAC for the data passed in and compares it with the provided MAC value. MAC random value, key generation and key version must be passed in separately.

Command/Message sequence:

Command WFS_CMD_PIN_	lpbMsg in lpbSecMsgIn	lpbMsg in lpbSecMsgOut	Service Provider´s actions
SECURE_MSG_SEND	Byte 0: 0x01 (Generate PAC) Byte 1: format (0 or 1) Byte 2-9: ANF (Primary Account Number, if length is less than 12 digits, value must be left padded with binary 0, only applicable for format 0)	Byte 0: key generation Byte 1: key version Byte 2-17: PAC random Byte 18-25: PAC value (all values are binary values)	Generates a session key for PAC generation and finally the PAC itself. Determine generation and version values of Master-Key and return them along with the random value.
SECURE_MSG_SEND	Byte 0: 0x02 (Get MAC Random)	Byte 0: key generation Byte 1: key version Byte 2-17: MAC random (all values are binary values)	Generates a session key for MAC generation (see next step below) Determine generation and version values of Master-Key and return them along with the random value
SECURE_MSG_SEND	Byte 0: 0x03 (Generate MAC) Byte 1-n: Message to be mac'ed (all values are binary values)	Byte 0-7: generated MAC (binary value)	Generates MAC over bytes 1-n of the inbound message using the session key created in the previous step.
SECURE_MSG_RECEIVE	Byte 0: 0x04 (Verify MAC) Byte 1: key generation Byte 2: key version Byte 3-18: MAC random Byte 19-26: MAC Byte 27-n: Message to be verified (all values are binary values) Note: If no message has been received, this function must be called by omitting Bytes 1-n	N/a	Generates a session key using the Master key identified by key generation and version by using the random value passed in. Generates a MAC for the message data passed in and compare the resulting MAC with the MAC passed in.

Returns:

The error code WFS_ERR_PIN_FORMATINVALID is returned when:

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

The error code WFS_ERR_PIN_HSMSTATEINVALID is returned when:

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

The error code WFS_ERR_PIN_MACINVALID is returned when:

- The subcommand in Byte 0 of *lpbMsg* for Execute Command WFS_CMD_PIN_SECURE_MSG_RECEIVE with protocol WFS_PIN_PROTGENAS is 04 (Verify MAC) and the MACs 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_PROTCHIPINCHG

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:

Flow	**Command WFS_CMD_PIN_**	**wProtocol WFS_PIN_PROT**	lpbMsg in lpbSecMsgIn	lpbMsg in lpbSecMsgOut	**Service Provider’s actions**
PIN Compare					
Start PIN comparison	SECURE_MSG_SEND	PINCMP	Byte 0: 0x00 (Start PIN compare)		Prepare EPP for PIN comparison. Output data buffer length is zero.
Let the user enter the new PIN for the first time.	GET_PIN	n/a	n/a	n/a	PIN entry.
	SECURE_MSG_SEND	PINCMP	Byte 0: 0x01 (Save PIN)		Save the PIN value entered for subsequent compare. Output data buffer length is zero.
Let the user enter the new PIN for the second time	GET_PIN	n/a	n/a	n/a	PIN entry.
	SECURE_MSG_SEND	PINCMP	Byte 0: 0x02 (Compare PINs)	Byte 0: 0x00 when PIN does not match, and 0x01 when PIN does match.	Compare PIN values.
Get the PAC of the new PIN via WFS_PIN_PROTGENAS or WFS_PIN_PROTISOAS (as usual).					
End PIN comparison.	SECURE_MSG_SEND	PINCMP	Byte 0: 0xFF (End PIN compare)		All PIN buffers are cleared. Output data buffer length is zero.

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, RND_{MES} and RND_{PAC})
- 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.

Command WFS_CMD_PIN_	**wProtocol WFS_PIN_ PROT**	**lpbMsg**	**Service Provider's actions**
Preparation for Load/Unload			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	recognize type of chip
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_ID	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_ID	store EF_ID
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_LLOG	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_LLOG	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ_RECORD EF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_BÖRSE	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ_RECORD EF_BETRAG	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_BETRAG	
Load against other ec-Card			
SECURE_MSG_SEND	CHIPZKA	**for type 0 chips only** Command APDU READ RECORD EF_KEYD	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_KEYD	
SECURE_MSG_SEND	CHIPZKA	**for type 1 chips only** Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND1 from Chip	store RND1
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN with Secure Msg.	fill: -Terminal ID -Traceno. -RND2 -MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	store response APDU for later check of ISOLZ message, BMP 62
SECURE_MSG_SEND	ISOAZ	ISO8583 message 0200 Authorization Request	Fill: - Traceno. (BMP 11) - PAC (BMP 52) - RND_{MES} + RND_{PAC} (BMP 57) - MAC (BMP 64) check other security relevant fields
SECURE_MSG_RECEIVE	ISOAZ	ISO8583 message 0210 Authorization Response	check MAC and other security relevant fields
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0200 Ladeanfrage	Fill: - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	

Command WFS_CMD_PIN_	wProtocol WFS_PIN_ PROT	lpbMsg	Service Provider's actions
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND3 from chip	store RND3
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN with Secure Msg.	provide complete command from BMP62 of ISOLZ response , compute command MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check response MAC
GET_JOURNAL	ISOLZ	Vendor specific	
GET_JOURNAL	ISOAZ	Vendor specific	
Reversal of a Load against other ec-Card			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND5 from chip	store RND5
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN with Secure Msg.	Fill: -Terminal ID -Traceno. -RND6 -Keyno. KGK_{LT} -MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	store response APDU for later check of ISOLZ message, BMP 62
SECURE_MSG_SEND	ISOAZ	ISO8583 message 0400 Storno	Fill: - Traceno. (BMP 11) - PAC (BMP 52) - RND_{MES} + RND_{PAC} (BMP 57) - MAC (BMP 64) check other security relevant fields
SECURE_MSG_RECEIVE	ISOAZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields.
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0400 Storno	Fill: - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND7 from chip	store RND7
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN with Secure Msg.	provide complete command from BMP62 of ISOLZ response , compute command MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check response MAC
GET_JOURNAL	ISOLZ	Vendor specific	
GET_JOURNAL	ISOAZ	Vendor specific	

PIN Verification Type 0			
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND0 from chip	store RND0
SECURE_MSG_SEND	CHIPZKA	Command APDU EXTERNAL AUTHENTICATE	fill -Keyno. K_{INFO} -ENCRND
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	

SECURE_MSG_SEND	CHIPZKA	Command APDU PUT DATA	fill RND1
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_INFO with Secure Messaging	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_INFO	check MAC
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND2 from chip	store RND2
SECURE_MSG_SEND	CHIPZKA	Command APDU VERIFY	provide complete command APDU
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
PIN Verification Type 1			
SECURE_MSG_SEND	CHIPZKA	Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND0 from chip	store RND0
SECURE_MSG_SEND	CHIPZKA	Command APDU MUTUAL AUTHENTICATE	fill ENC0
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check ENC1
SECURE_MSG_SEND	CHIPZKA	Command APDU VERIFY	provide complete command APDU
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check MAC
„Laden vom Kartenkonto“ (both types)			
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0200 Ladeanfrage	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields.
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	

Reversal of a „Laden vom Kartenkonto“			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN	fill -Terminal ID -Traceno.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.

SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	
Unload			
SECURE_MSG_SEND	CHIPZKA	ENTLADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message Entladeanfrage 0200	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Entladeantwort 0210	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	ENTLADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	ENTLADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message Entladequittung 0202	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Entladebestätigung 0212	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	Command APDU ENTLADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	

Repeated Messages (Stornowiederholung / Entladequittungswiederholung)			
SECURE_MSG_SEND	ISOLZ	ISO8583 message Stornowiederholung 0401 or Entladequittungswiederholung 0203	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Stornoantwort 410 or Entladebestätigung 0212	check MAC and other security relevant fields
GET_JOURNAL	ISOLZ	Vendor specific	

Command WFS_CMD_PIN_	**wProtocol WFS_PIN_PROT**	**lpbMsg**	**Service Provider's actions**
Preparation for PIN Change			
SECURE_MSG_SEND	CHIPPINCHG	Command APDU READ RECORD EF_ID	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU Record EF_ID	Store EF_ID Will be inserted into BMP62 of a PIN Change request
SECURE_MSG_SEND	CHIPPINCHG	Command APDU GET CHALLENGE	

Command WFS_CMD_PIN_	wProtocol WFS_PIN_PROT	lpbMsg	Service Provider's actions
SECURE_MSG_RECEIVE	CHIPPINCHG	Random number RND0 from Chip	Store RND0
SECURE_MSG_SEND	CHIPPINCHG	Command APDU READ RECORD EF_INFO	Fill RND1
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU Record EF_INFO	Check MAC, Store EF_INO Will be inserted into BMP62 of a PIN Change request
SECURE_MSG_SEND	CHIPPINCHG	Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU Version of KCard	Store version byte Will be inserted into BMP62 of a PIN Change request
SECURE_MSG_SEND	CHIPPINCHG	Command APDU SEARCH RECORD '01' of EF_PWDD	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	Store record number Will be inserted into BMP62 of a PIN Change request
SECURE_MSG_SEND	CHIPPINCHG	Command APDU READ RECORD EF_FBZ	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU Initial value FBZ Actual value FBZ	
PIN Verification			
SECURE_MSG_SEND	CHIPPINCHG	Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	
SECURE_MSG_SEND	CHIPPINCHG	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPPINCHG	Random number RND0 from chip	Store RND0
SECURE_MSG_SEND	CHIPPINCHG	Command APDU MUTUAL AUTHENTICATE	Fill ENC0
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	Check ENC1
SECURE_MSG_SEND	CHIPPINCHG	Command APDU VERIFY	Provide complete command APDU
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	Check MAC Create PAC for old PIN
PIN Change			
Let the user enter the PIN for the first time, by invoking the command WFS_CMD_PIN_GET_PIN			
SECURE_MSG_SEND	HSMPINCMP	Byte 0: 0x01 (Save PIN)	Save the PIN value entered for subsequent compare. Output data buffer length is zero.
Let the user enter the PIN for the second time, by invoking the command WFS_CMD_PIN_GET_PIN			
SECURE_MSG_SEND	HSMPINCMP	Byte 0: 0x02 (Compare PINs)	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
SECURE_MSG_SEND	CHIPPINCHG	Command APDU MANAGE SECURITY ENVIRONMENT	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	
SECURE_MSG_SEND	CHIPPINCHG	Command APDU GET CHALLENGE	

Command WFS_CMD_PIN_	wProtocol WFS_PIN_PROT	lpbMsg	Service Provider's actions
SECURE_MSG_RECEIVE	CHIPPINCHG	Random number RND0 from Chip	Store RND0 Will be inserted into BMP62 of a PIN Change request
SECURE_MSG_SEND	ISOPINCHG	ISO8583 Message 0640	Fill - PAC old PIN (BMP52) - $K_{Terminal}$ generation + $K_{Terminal}$ version + RND_{MES} + RND_{PAC} (BMP57) - Chip Data (BMP62) with PAC of new PIN - MAC (BMP64)
SECURE_MSG_RECEIVE	ISOPINCHG	ISO8583 message 0650	Check MAC
SECURE_MSG_SEND	CHIPPINCHG	Command APDU from BMP62	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	
PIN Change Confirmation/ Repeated Confirmation			
SECURE_MSG_SEND	ISOPINCHG	ISO8583 message 0642 or 0643 BMP25 = 00	Fill - $K_{Terminal}$ generation + $K_{Terminal}$ version + RND_{MES} (BMP57) - Chip Data (BMP62) with PAC of **new** PIN - MAC (BMP64)
SECURE_MSG_RECEIVE	ISOPINCHG	ISO8583 message 0652	Check MAC
PIN Change Reversal/ Repeated Reversal			
SECURE_MSG_SEND	ISOPINCHG	ISO8583 message 0642 or 0643 BMP25 ≠ 00	Fill - $K_{Terminal}$ generation + $K_{Terminal}$ version + RND_{MES} (BMP57) - Chip Data (BMP62) with PAC of **old** PIN - MAC (BMP64)
SECURE_MSG_RECEIVE	ISOPINCHG	ISO8583 message 0652	Check MAC

PIN Activation after failure			
SECURE_MSG_SEND	ISOPINCHG	ISO8583 message 0640	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)
SECURE_MSG_RECEIVE	ISOPINCHG	ISO8583 message 0650	Check MAC

PIN Activation			
SECURE_MSG_SEND	CHIPPINCHG	Command APDU MANAGE SECURITY ENVIRONMENT	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	
SECURE_MSG_SEND	CHIPPINCHG	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPPINCHG	Random number RND0 from Chip	Store RND0 Will be inserted into BMP62 of a PIN Activation request
SECURE_MSG_SEND	ISOPINCHG	ISO8583 Message 0640	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)
SECURE_MSG_RECEIVE	ISOPINCHG	ISO8583 message 0650	Check MAC

SECURE_MSG_SEND	CHIPPINCHG	Command APDU from BMP62	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	
PIN Activation Confirmation/ Repeated Confirmation			
SECURE_MSG_SEND	CHIPPINCHG	Command APDU MANAGE SECURITY ENVIRONMENT	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	
SECURE_MSG_SEND	CHIPPINCHG	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPPINCHG	Random number RND0 from Chip	Store RND0 Will be inserted into BMP62 of a PIN Activation confirmation
SECURE_MSG_SEND	ISOPINCHG	ISO8583 message 0642 or 0643 BMP25 = 00	Fill - $K_{Terminal}$ generation + $K_{Terminal}$ version + RND_{MES} (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)
SECURE_MSG_RECEIVE	ISOPINCHG	ISO8583 message 0652	Check MAC
SECURE_MSG_SEND	CHIPPINCHG	Command APDU from BMP62	
SECURE_MSG_RECEIVE	CHIPPINCHG	Response APDU	

8.4 EMV Support

EMV support by this specification consists in the ability of importing Certification Authority and Chip Card Public Keys, creating the PIN blocks for offline PIN verification and verifying static and dynamic data. This section is used to further explain concepts and functionality that needs further clarification.

The PIN service is able to manage the EMV chip card regarding the card authentication and the RSA local PIN verification. Two steps are mandatory in order to reach these two functions: The loading of the keys which come from the Certification Authorities or from the card itself, and the EMV PIN block management.

The Service Provider is responsible for all key validation during the import process. The application is responsible for management of the key lifetime and expiry after the key is successfully imported.

8.4.1 Keys loading

The final goal of an application is to retrieve the keys located on card to perform the operations of authentication or local PIN check (RSA encrypted). These keys are provided by the card using EMV certificates and can be retrieved using a Public Key provided by a Certification Authority. The application should first load the keys issued by the Certification Authority. At transaction time the application will use these keys to load the keys that the application has retrieved from the chip card.

Certification Authority keys
These keys are provided in the following formats:

- Plain text.
- Plain Text with EMV 2000 Verification Data (See [Ref. 4] under the reference section for this document).
- EPI CA (or self signed) format as specified in the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5] under the reference section for this document).
- PKCSV1_5 encrypted (as used by GIECB in France) (See [Ref. 15] under the reference section for this document).

EPI CA format
The following table corresponds to table 4 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5]) and identifies the Europay Public Key (self-certified) and the associated data:

Field name	Length	Description	Format
ID of Certificate Subject	5	RID for Europay	Binary
Europay public key Index	1	Europay public key Index	Binary
Subject public key Algorithm Indicator	1	Algorithm to be used with the Europay public key Index, set to 0x01	Binary
Subject public key Length	1	Length of the Europay public key Modulus (equal to *Nca)*	Binary
Subject public key Exponent Length	1	Length of the Europay public key Exponent	Binary
Leftmost Digits of Subject public key	*Nca*-37	*Nca*-37 most significant bytes of the Europay public key Modulus	Binary
Subject public key Remainder	37	37 least significant bytes of the Europay public key Modulus	Binary
Subject public key Exponent	1	Exponent for Europay public key	Binary
Subject public key Certificate	*Nca*	Output of signature algorithm	Binary

Table 1

The following table corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 and identifies the Europay Public Key Hash code and associated data.

Field name	Length	Description	Format
ID of Certificate Subject	5	RID for Europay	Binary

Europay public key Index	1	Europay public key Index	Binary
Subject public key Algorithm Indicator	1	Algorithm to be used with the Europay public key Index, set to 0x01	Binary
Certification Authority public key Check Sum	20	Hash-code for Europay public key	Binary

Table 2

Table 2 corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. 5]).

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

Tag Number	Primitive / Constructed	Identifier Octet	Contents
0	P	0x40	**Protocol Version** The INTEGER value zero for this version of the protocol
1	P	0x41	**Interchange Code** An ASCII string holding one of the interchange codes defined in [Ref. 15], e.g. "HRN-H1"
2	C	0x62	**Interchange Data** The data items as defined by [Ref.15], see table below for details
3	P	0x43	**Key Name** An ASCII string holding the name for the key being loaded or generated.

The Interchange Data (Tag 2) is constructed from data items where tag numbers of the sub-tags from 1 to 23 correspond to the data item numbers ("N° 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

Tag (=Data Item No)	Primitive / Constructed	Identifier Octet	Data Item Label
1	C	0xA1	IdKG
2	C	0xA2	KTK-encrypted
3	C	0xA3	KGp
4	C	0xA4	KDp
5	C	0xA5	SnSCD
6	P	0x86	Rand
7	P	0x87	HOST authentication
8	P	0x88	KDp signature
9	P	0x89	KGp signature
10	P	0x8A	KTK signature
11	P	0x8B	KT-encrypted
12	P	0x8C	Ksc-encrypted
13	P	0x8D	PIN cryptogram
14	P	0x8E	Seal
15	P	0x8F	Thumbprint of KDp
16	P	0x90	Thumbprint of KGp
17	C	0xB1	IdKD
18	C	0xB2	IdKTK
19	C	0xB3	IdKT
20	C	0xB4	IdKSC
21	P	0x95	Manufacturer
22	C	0xB6	SCD type
23	C	0xB7	Firmware version

Inside the constructed data items, primitive encoding is used for the data elements, all tags having CONTEXT class with tag numbers corresponding to the data element numbers (“N° 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 (data element 2)
      83 01 00                                  (data element 3)
   A5 10                                  (tag / length for data item 5)
      81 03 00 00 00                            (data element 1)
      82 09 00 00 00 00 00 00 00 00 00        (data element 2)
   86 08 00 00 00 00 00 00 00 00           (tag / length / value for data item 6)
   87 81 80 <128 bytes>                 (tag / length / value for data item 7)
43 05 4D 59 4B 45 59                    (tag / length / value for Key Name “MYKEY”)
```

8.5.2 Command Sequence

The following list shows the sequence of actions an application has to take for the various *Cartes Bancaires* interchanges.

- GIN (ATM initialization)

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
Thumbprint supplied by host via external channel (GIN-H1)				
WFS_CMD_PIN_ENCIO	GIN-G2			21,22,23
Host Communication (GIN-G2 / GIN-H3)				
WFS_CMD_PIN_ENCIO	GIN-H3	Key Name for KG	3	16
WFS_CMD_PIN_ENCIO	GIN-G4			5,6,1
Host Communication (GIN-G4 / GIN-H5)				
WFS_CMD_PIN_ENCIO	GIN-H5	Key Name for KD	5,6,1,7	
WFS_CMD_PIN_ENCIO	GIN-G6			5,4,8
Host Communication (GIN-G6)				
WFS_CMD_PIN_ENCIO	GIN-G7			15
Send thumbprint to host via external channel (GIN-G7)				

- GRN (Renewal of ATM Master Key)

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
WFS_CMD_PIN_ENCIO	GRN-G1			5,6,1
Host Communication (GRN-G1 / GRN-H2)				
WFS_CMD_PIN_ENCIO	GRN-H2	Key Name for KD	5,6,1,7	
WFS_CMD_PIN_ENCIO	GRN-G3			5,4,8,17
Host Communication (GRN-G3)				
WFS_CMD_PIN_ENCIO	GRN-C or GRN-R		17	

The Interchange codes “GRN-C” to commit the transaction resp. “GRN-R” to roll back the transactions are an addition to those defined in [Ref. 15].

- HRN (Renewal of HOST Master Key)

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
Host Communication (HRN-H1)				
WFS_CMD_PIN_ENCIO	HRN-H1	Key Name for KG	3,9,1	

- DKT (Generation and Loading of KTK)

Action	Interchange Code	Key Name	Input Data Items	Output Data Items
WFS_CMD_PIN_ENCIO	DKT-G1			5,6
Host Communication (DKT-G1 / DKT-H2)				
WFS_CMD_PIN_ENCIO	DKT-H2	Key Name for KTK	5,6,2,10,1,17	

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 f*wKeyEntryMode* represents a regular shaped PIN pad (WFS_PIN_SECUREKEY_REG_UNIQUE or WFS_PIN_SECUREKEY_REG_SHIFT) then *lppHexKeys* must contain one entry for each physical key on the 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.

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

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

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

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.

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

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

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

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.

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

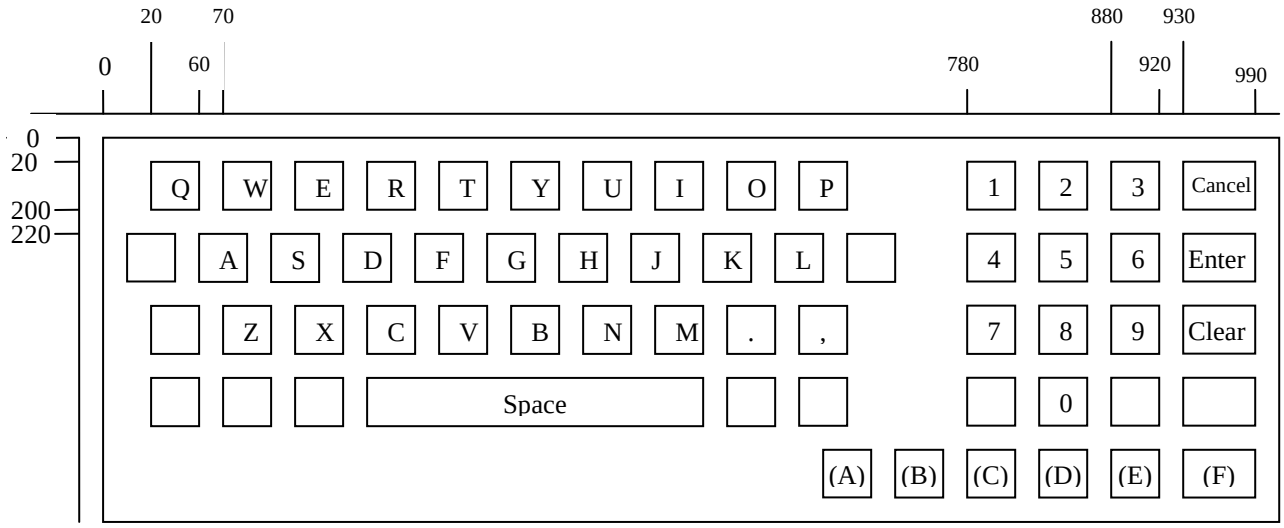
In the above example, where the hex digits 'A' to 'F' are accessed through shift '0' to ‘5’, *wColumns* will be 4, *wRows* will be 5 and the *lppHexKeys* will contain the entries in the array as defined in the following table.

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
0	0	0	250	200	FK_1	FK_B
1	250	0	250	200	FK_2	FK_C
2	500	0	250	200	FK_3	FK_D
3	750	0	250	200	FK_CLEAR	FK_UNUSED
4	0	200	250	200	FK_4	FK_E
5	250	200	250	200	FK_5	FK_F
6	500	200	250	200	FK_6	FK_UNUSED
7	750	200	250	200	FK_CANCEL	FK_UNUSED
8	0	400	250	200	FK_7	FK_UNUSED
9	250	400	250	200	FK_8	FK_UNUSED
10	500	400	250	200	FK_9	FK_UNUSED
11	750	400	250	200	FK_ENTER	FK_UNUSED

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
12	0	600	250	200	FK_UNUSED	FK_UNUSED
13	250	600	250	200	FK_0	FK_A
14	500	600	250	200	FK_UNUSED	FK_UNUSED
15	750	600	250	200	FK_UNUSED	FK_UNUSED
16	0	800	1000	200	FK_SHIFT	FK_UNUSED

8.6.1.4 *fwKeyEntryMode* == WFS_PIN_SECUREKEY_IRREG_UNIQUE

When *fwKeyEntryMode* is WFS_PIN_SECUREKEY_REG_UNIQUE then the values in the array report which physical keys are associated with the function keys 0-9, A-F and any other function keys that can be enabled as defined in the *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 if *wColumns* is larger than *wRows*, etc. A Service Provider must return the position and size data for each key.



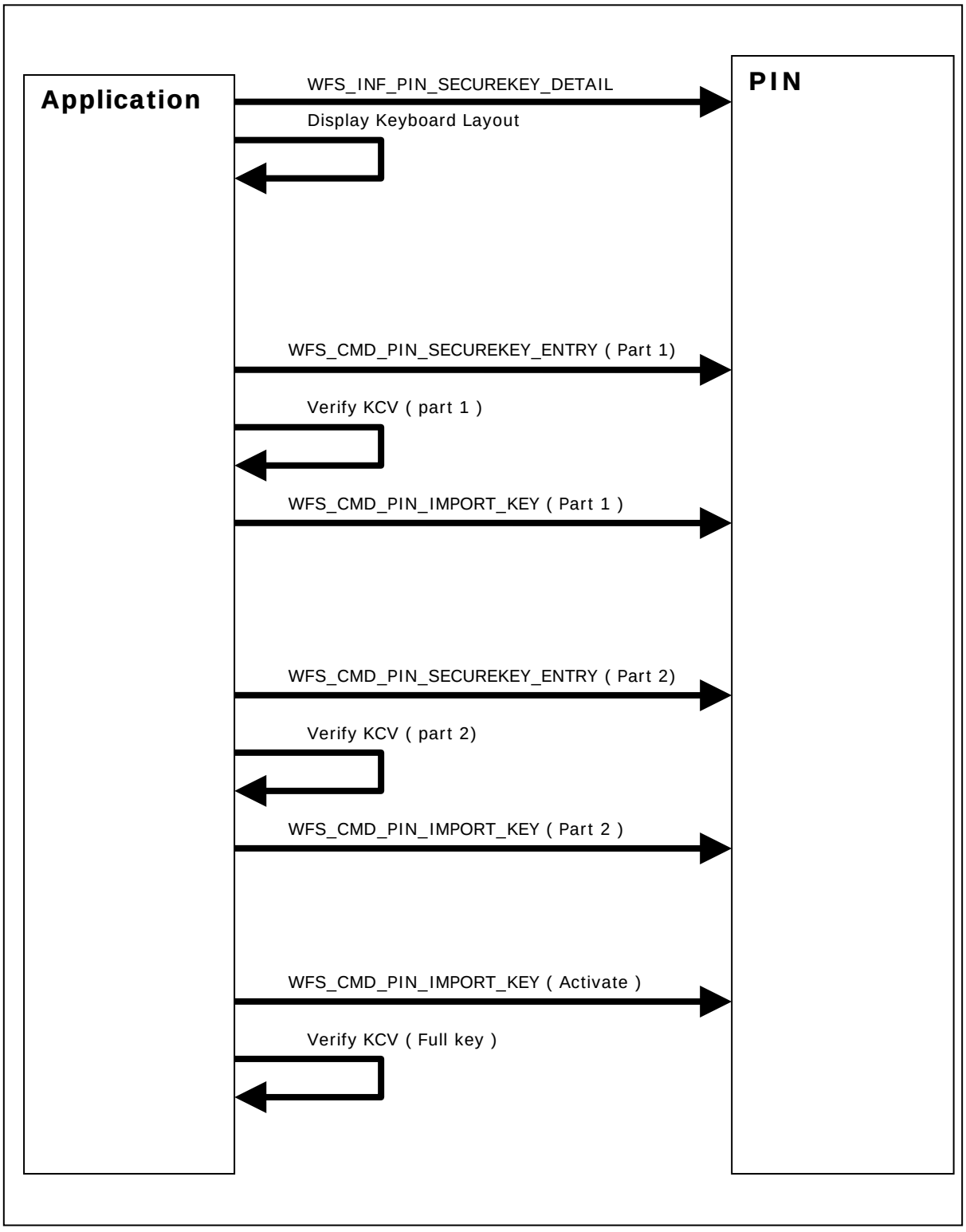
In the above example, where an alphanumeric keyboard supports secure key entry and the hex digits are located as shown, the *lppHexKeys* will contain the entries in the array as defined in the following table. All the hex digits and function keys that can be enabled must be included in the array; in 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.

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
0	780	18	40	180	FK_1	FK_UNUSED
1	830	18	40	180	FK_2	FK_UNUSED
2	880	18	40	180	FK_3	FK_UNUSED
3	930	18	60	180	FK_CANCEL	FK_UNUSED
4	780	216	40	180	FK_4	FK_UNUSED
5	830	216	40	180	FK_5	FK_UNUSED
6	880	216	40	180	FK_6	FK_UNUSED
7	930	216	60	180	FK_ENTER	FK_UNUSED
8	780	414	40	180	FK_7	FK_UNUSED
9	830	414	40	180	FK_8	FK_UNUSED
10	880	414	40	180	FK_9	FK_UNUSED

Index	usXPos	usYPos	usxSize	usYSize	ulFK	ulShiftFK
11	930	414	60	180	FK_CLEAR	FK_UNUSED
12	780	612	40	180	FK_UNUSED	FK_UNUSED
13	830	612	40	180	FK_0	FK_UNUSED
14	880	612	40	180	FK_UNUSED	FK_UNUSED
15	930	612	60	180	FK_UNUSED	FK_UNUSED
16	680	810	40	180	FK_A	FK_UNUSED
17	730	810	40	180	FK_B	FK_UNUSED
18	780	810	40	180	FK_C	FK_UNUSED
19	830	810	40	180	FK_D	FK_UNUSED
20	880	810	40	180	FK_E	FK_UNUSED
21	930	810	60	180	FK_F	FK_UNUSED

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 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:

```
LPPROTLUXIN                    lpvData;

   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.

Value	Meaning
WFS_CMD_ENC_IO_LUX_LOAD_APPKEY	Load an Application Key.
WFS_CMD_ENC_IO_LUX_GENERATE_MAC	Generate the CBC-MAC.
WFS_CMD_ENC_IO_LUX_CHECK_MAC	Check the CBC-MAC.
WFS_CMD_ENC_IO_LUX_BUILD_PINBLOCK	Build the PIN block.
WFS_CMD_ENC_IO_LUX_DECRYPT_TDES	Decrypt data.
WFS_CMD_ENC_IO_LUX_ENCRYPT_TDES	Encrypt data.

... 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 LPWFSPINENCIO lpEncIoOut;

```
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:

```
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 Luxemburg protocol are:

Value	Meaning
PROT_LUX_SUCCESS	Command terminated correctly.
PROT_LUX_ERR_INVALID_CMD	Invalid command. The *wCommand* issued is not valid or not supported.
PROT_LUX_ERR_INVALID_DATA	The data structure passed as input parameter for the command contains invalid or incoherent data.
PROT_LUX_ERR_INVALID_KEY	The key needed for the operation was not loaded or is invalid. This operation failed.

... 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 LPPROTLUXLOADAPPKEY lpvData;

```
typedef struct _prot_lux_load_app_key_in
     {
     WORD                      wCommand;
     LPSTR                     lpsKeyName;
     LPSTR                     lpsSequenceNumber;
     LPWFSXDATA                lpxKeyData;
     } PROTLUXLOADAPPKEYIN, *LPPROTLUXLOADAPPKEYIN;
```

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;

```
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:

Value	Meaning
PROT_LUX_ERR_VERIFICATION_FAILED	Verification failed. The supplied MAC does not match with the calculated MAC.

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 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 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:

Value	Meaning
PROT_LUX_ERR_VERIFICATION_FAILED	Verification Failed. The MAC generated by this command does not compare with the MAC passed in by the application.

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:

Value	Meaning
PROT_LUXFORMISO1	ISO-1 PIN Block

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:

Value	Meaning
PROT_LUX_ERR_PIN_FORMAT_LENGTH	The PIN block could not be constructed because PIN was not entered or the PIN length was invalid.

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 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 LPPROTLUXDECRYPTTDESIN lpvData;

```
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:

Value	Meaning
PROT_LUXTRIDESECB	Triple DES with Electronic Code Book.
PROT_LUXTRIDESCBC	Triple DES with Cipher Block Chaining.

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;

```
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 LPPROTLUXENCRYPTTDESIN lpvData;

```
typedef struct _prot_lux_encrypt_tdes_in
     {
     WORD                      wCommand;
     WORD                      wType;
     LPWFSXDATA                lpxData;
     LPWFSXDATA                lpxIV;
     } PROTLUXENCRYPTTDESIN, *LPPROTLUXENCRYPTTDESIN;
```

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:

Value	Meaning
WFS_PIN_LUXTRIDESECB	Triple DES with Electronic Code Book.
WFS_PIN_LUXTRIDESCBC	Triple DES with Cipher Block Chaining.

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 LPPROTLUXENCRYPTTDESOUT lpvData;

```
typedef struct _prot_lux_encrypt_tdes_out
     {
     WORD                      wCommand;
     WORD                      wResult;
     LPWFSXDATA                lpxData;
     LPWFSXDATA                lpxRandom;
     } PROTLUXENCRYPTTDESOUT, *LPPROTLUXENCRYPTTDESOUT;
```

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;
} PROTLUXOUT, *LPPROTLUXOUT;

typedef struct _prot_lux_load_app_key_in
{
    WORD                    wCommand;
    LPSTR                   lpsKeyName;
    LPSTR                   lpsSequenceNumber;
    LPWFSXDATA              lpxKeyData;
} PROTLUXLOADAPPKEYIN, *LPPROTLUXLOADAPPKEYIN;

typedef struct _prot_lux_load_app_key_out
{
    WORD                    wCommand;
    WORD                    wResult;
} PROTLUXLOADAPPKEYOUT, *LPPROTLUXLOADAPPKEYOUT;

typedef struct _prot_lux_generate_mac_in
{
    WORD                    wCommand;
    LPWFSXDATA              lpxData;
    WORD                    wMacLength;
} PROTLUXGENERATEMACIN, *LPPROTLUXGENERATEMACIN;

typedef struct _prot_lux_generate_mac_out
{
    WORD                    wCommand;
    WORD                    wResult;
    LPWFSXDATA              lpxMac;
    LPWFSXDATA              lpxRandom;
} PROTLUXGENERATEMACOUT, *LPPROTLUXGENERATEMACOUT;

typedef struct _prot_lux_check_mac_in
{
    WORD                    wCommand;
    LPWFSXDATA              lpxData;
    LPWFSXDATA              lpxMac;
    LPWFSXDATA              lpxRandom;
} PROTLUXCHECKMACIN, *LPPROTLUXCHECKMACIN;

typedef struct _prot_lux_check_mac_out
{
    WORD                    wCommand;
    WORD                    wResult;
} PROTLUXCHECKMACOUT, *LPPROTLUXCHECKMACOUT;

typedef struct _prot_lux_pinblock_in
{
    WORD                    wCommand;
    WORD                    wFormat;
}PROTLUXPINBLOCKIN, *LPPROTLUXPINBLOCKIN;

typedef struct _prot_lux_pinblock_out
{
    WORD                    wCommand;
    WORD                    wResult;
    LPWFSXDATA              lpxPinBlock;
    LPWFSXDATA              lpxRandom;
} PROTLUXPINBLOCKOUT, *LPPROTLUXPINBLOCKOUT;

typedef struct _prot_lux_decrypt_tdes_in
{
    WORD                    wCommand;
    WORD                    wType;
    LPWFSXDATA              lpxData;
    LPWFSXDATA              lpxIV;
    LPWFSXDATA              lpxRandom;
```

```
} PROTLUXDECRYPTTDESIN, *LPPROTLUXDECRYPTTDESIN;

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)

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

#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:

CERTIFICATESTATE=<0xnnnnnnnn>, 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:

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

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

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

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

Value	Meaning
WFS_PIN_SIGN_RSASSA_PKCS1_V1_5	SSA_PKCS_V1_5 Signatures supported.
WFS_PIN_SIGN_RSASSA_PSS	SSA_PSS Signatures supported.

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

Value	Meaning
WFS_PIN_CRYPT_RSAES_PKCS1_V1_5	AES_PKCS_V1_5 algorithm supported.
WFS_PIN_CRYPT_RSAES_OAEP	AES_OAEP algorithm supported.

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

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

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

Value	Meaning
WFS_PIN_SIG_GEN_RSA_KEY_PAIR	Specifies if the Service Provider supports the RSA Signature Scheme WFS_CMD_PIN_GENERATE_RSA_KEY_PAIR and WFS_CMD_PIN_EXPORT_RSA_EPP_SIGNED commands.

WFS_PIN_SIG_RANDOM_NUMBER	Specifies if the Service Provider returns a random number from the WFS_CMD_PIN_START_KEY_EXCHANGE command within the RSA Signature Scheme.
WFS_PIN_SIG_EXPORT_EPP_ID	Specifies if the Service Provider supports exporting the EPP Security Item within the RSA Signature Scheme.

For EMV support the following key/value pairs indicate the level of support of the Service Provider. Note that a series of this key/value pairs may occur that lists all import schemes supported by the PIN 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.

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

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

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

The capabilities associated with key loading in multiple 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.