

payShield® 10K

Host Programmer's Manual

PUGD0541-005



Date: June 2021

Doc. Number: PUGD0541-005

All information herein is either public information or is the property of and owned solely by Thales DIS France S.A. and/or its subsidiaries or affiliates who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Thales DIS France S.A. and any of its subsidiaries and affiliates (collectively referred to herein after as "Thales") information.

This document can be used for informational, non-commercial, internal and personal use only provided that:

- The copyright notice below, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any network computer or broadcast in any media and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Thales makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Thales reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Thales hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Thales be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Thales does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Thales be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Thales products. Thales disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service or loss of privacy.

Copyright © 2018-2021 Thales Group. All rights reserved. Thales and the Thales logo are trademarks and service marks of Thales and/or its subsidiaries and affiliates and are registered in certain countries. All other trademarks and service marks, whether registered or not in specific countries, are the properties of their respective owners.

Follow this link to find the End User Licensing Agreement: <https://cpl.thalesgroup.com/legal>

Contents

Contents	3
Revision Status	7
References	8
1 Introduction	9
1.1 General Information	9
1.2 Command Message Format	10
1.3 Response Message Format	10
1.4 Data Representation	11
1.5 Input/Output Flow Control	15
1.6 Error Handling	15
1.7 Error Logs	16
1.8 Multiple HSMs	17
1.9 User Storage	17
2 Host Connections	18
2.1 TCP/IP Protocol	18
2.2 UDP Protocol	20
2.3 payShield 10K PS10-F (FICON)	21
3 PIN Printing and Solicitation	23
3.1 PIN Mailers	23
3.2 PIN Solicitation	27
3.3 Printer Support	32
4 Transaction Key Schemes	36
4.1 Racal Transaction Key Scheme (RTKS)	36
4.2 Simultaneous use of both Racal and Australian Transaction Key Schemes	37
4.3 Derived Unique Key Per Transaction (DUKPT)	39
4.4 German Banking Industry Committee (GBIC)	52
5 RSA and ECC Cryptosystem	53
5.1 RSA Cryptosystem	53
5.2 ECC Cryptosystem	54
5.3 General Information	56
5.4 Common Parameters	56
5.5 Worked Examples	65
6 Local Master Keys (LMKs)	78
6.1 Introduction	78
6.2 Multiple LMKs	78
6.3 Key Block & Variant Key Comparison Table	78
6.4 Converting Key Names	80

7	Variant LMK Key Scheme.....	81
7.1	How the Variant scheme works	81
7.2	Local Master Key (LMK) Variants.....	82
7.3	Test Variant LMK	87
7.4	Variant Key Type Codes.....	90
7.5	Key Type Table.....	92
8	Key Block LMK Key Scheme	96
8.1	Introduction	96
8.2	Relationship between Thales Key Blocks and TR-31 Key Blocks.....	96
8.3	Support for Thales Key Blocks	96
8.4	Key Block Format	96
8.5	Encrypted key data	106
8.6	Authenticator.....	107
8.7	Key Block Local Master Keys (LMKs)	107
8.8	Console Command examples.....	109
8.9	Host Commands	109
9	Multiple LMKs.....	111
9.1	Introduction	111
9.2	The Need for Multiple LMKs	111
9.3	Multiple LMK Licensing.....	111
9.4	Managing Multiple LMKs	112
9.5	Authorization	113
9.6	LMK table.....	114
9.7	Deleting LMKs.....	114
9.8	Identifying the Required LMK	115
10	Migrating LMKs	118
10.1	Introduction	118
10.2	Multiple LMKs	118
10.3	Overview of the LMK Migration Process	118
10.4	Formatting LMK smart cards	120
10.5	Generating LMK component cards.....	120
10.6	Creating Copies of LMK Component Cards	120
10.7	Loading the new LMK.....	121
10.8	Loading the old LMK.....	122
10.9	Migrating keys between Variant LMKs	122
10.10	Migrating keys from Variant to Key Block LMKs.....	125
10.11	Migrating keys between Key Block LMKs.....	129
10.12	Migrating keys from Key Block to Variant LMKs.....	130
10.13	Migrating keys for PCI HSM compliance	130
10.14	Re-encrypting PINs.....	131
10.15	Re-encrypting decimalization tables.....	132
10.16	Switching to the new LMK	134
10.17	Taking advantage of Multiple LMKs	134
10.18	Tidying up after migration to a new LMK.....	135

11	TR-31 Key Blocks	138
11.1	TR-31 Key Block Structure	138
11.2	Key Block Header	139
11.3	Optional Header	142
11.4	Using TR-31 Key Blocks	143
11.5	GISKE Key Block	143
12	DES Key Support	144
12.1	DES Keys	144
12.2	Key Usage	144
12.3	Key Encryption Schemes	144
12.4	Key Generate, Import and Export	145
12.5	Rejection of Weak, Semi-Weak, & Possibly Weak Keys	145
13	AES Key Support	148
13.1	Overview	148
13.2	Support for AES	149
13.3	Implementing AES on an existing payShield 10K	151
13.4	AES in Host commands	152
13.5	Support for AES in console commands	160
13.6	Support for AES in payShield Manager	161
14	User Storage	163
14.1	Introduction	163
14.2	User Storage Area	163
14.3	Defining Block Size	164
14.4	User Storage when using Fixed Block Sizes	164
14.5	User Storage when using Variable Block Sizes	169
14.6	Storing RSA and ECC Keys Using Key Indexes	170
14.7	Host Command Summary	171
15	SNMP	172
15.1	Introduction	172
15.2	Network Connectivity	172
15.3	SNMP Version	173
15.4	Configuring Traps	173
15.5	Information provided by the payShield 10K through SNMP	173
16	PIN Block Formats	176
16.1	General	176
16.2	Thales PIN Block Formats	176
17	Key Component Printing	183
17.1	Introduction	183
17.2	Process Description	183
18	Moving to PCI HSM Compliance	189
18.1	Introduction	189

18.2	PIN Blocks	189
18.3	Updating Key Type 002 in Host Commands	192
18.4	Migrating keys from Key Type 002	193
18.5	Diebold Table re-encryption.....	197
Appendix A - Key Scheme Table		198
Appendix B - Reduced Character Sets		199
Appendix C - Thales Key Block / TR 31 Key Usage Conversion.....		200
Appendix E - Print Formatting Symbols		201
Appendix F - Example laser printer formatting control codes		204
Appendix G - SNMP MIB.....		206
Technical Support Contacts		243

Revision Status

Revision	Date	Changes
001	April 2019	Initial issue
002	July 2019	Editorial updates
003	August 2020	Updates for Software Version v1.1a: Appendix G, "List of Authorized Activities"
004	January 2021	Updates for Software Version v1.2a including: Details on the support provided for Elliptic Curve Cryptography (ECC) in v1.2a included References to the Trusted Management Device (TMD) added Section on User Storage updated A number of other updates and corrections have been included
005	June 2021	Updates Editorial

References

The following documents are referenced within this manual.

Reference Number	Title
1	payShield 10K Installation and User Guide
2	payShield 10K Host Command Reference Manual
3	PKCS#1: RSA Cryptography Standard – Version 2.2 October 2012
4	Visa Integrated Circuit Card Specification, Version 1.5 May 2009
5	MasterCard M/Chip 4 Security and Key Management 1.0 October 2002
6	ASC X9 TR-31, Interoperable Secure Key Exchange Key Block Specification for Symmetric Algorithms, 2018.
7	Global Interoperable Secure Key Exchange Key Block Specification, version 2.3, written by ACI Worldwide, HP Atalla, Diebold, Thales e-Security and VeriFone Inc. 2002.
8	payShield Trusted Management Device (TMD) User Guide

1 Introduction

The payShield 10K hardware security module (HSM) acts as a peripheral to the Host computer. It performs cryptographic processing in a physically secure environment on behalf of the Host. The processing is performed by a payShield 10K HSM in response to commands, which it receives via a data link.

Typically, an HSM is used in a real-time, online environment performing key management, PIN and MAC related functions, as required by the system.

This manual contains programming notes to assist the application programmer. A complete command reference can be found in the *payShield 10K Host Command Reference Manual*.

Note: All Host Commands are disabled by default.

1.1 General Information

The application program sends commands to an HSM, and receives responses from it. Each command and response consists of a variable number of fields.

Versions of the payShield 10K HSM can be configured to support TCP/IP and UDP communications protocols. The HSM has no flow control support so the programmer must ensure that the HSM input buffer is not exceeded.

The HSM returns an error code to the Host as part of the response message. The programmer must ensure that a suitable response is made to each type of error.

In a typical system, a minimum of two HSMs are connected to the Host via separate Host ports. The HSM units are independent, and the programmer should make maximum use of all the HSM units to increase throughput, using one HSM if another is already processing data or is faulty. Also, it is useful to ensure that the program allows for additional HSM units to be subsequently added as throughput requirements increase.

Each HSM has a user storage area reserved for use by the programmer to store data required during processing. Typically, it is used to store keys and tables. Instructing an HSM to access data from user storage reduces the amount of data necessary in each command, and thus reduces the communications time.

There is a facility to print data (e.g., account holder PINs) at a printer connected to a payShield 10K HSM. The HSM must have format information for the data before sending it to the printer. The program must send a print format command to an HSM before print commands can be issued.

1.2 Command Message Format

To give an HSM an instruction, the Host application must assemble a message containing all the necessary information and send it to an HSM as a sequence of characters on the communications link. In general, each command consists of the following fields:

- Message Header

The message header field can be any length from 1 to 255 characters and it is configured at installation. It can contain any printable characters and an HSM returns them unmodified in the response message.

It can be used to label commands and their responses for systems that implement batch queues or which multi-thread commands.

- Command Code

Every command has a unique two-character command code. The command codes are detailed in the *payShield 10K Host Command Reference Manual*.

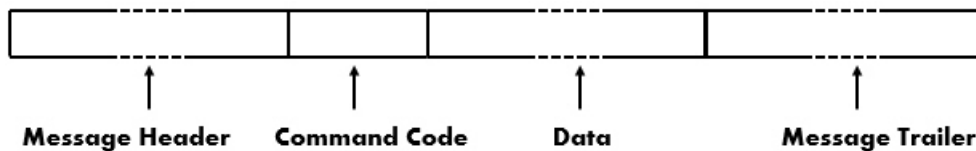
- Data

Most HSM commands require data, often including cryptographic keys. Details of the data for each command can be found in the *payShield 10K Host Command Reference Manual*.

- Message Trailer

The message trailer is an additional variable-length field (to a maximum of 32 characters), which can be used to pass additional details required by the Host for further processing. The field should always be preceded by the EM control character; ASCII and EBCDIC value is X'19.

The data in this field can be any printable character, and it is returned in the response message unchanged for error codes 00 and 02. (The trailer is not returned for other error codes.)



1.3 Response Message Format

To inform the Host of the results of processing, an HSM sends a message containing all the necessary information as a sequence of characters on the communications link. A response message is generated for each of the following:

- In response to a command
- As a second response to a print command after an HSM has finished sending the print data to the printer.
- In response to the entry of PIN solicitation data at the console (but only after the Host has enabled this function).

Each response from the HSM consists of the following:

- Message Header

The message header field is a copy of the field received in the command message from the Host. The data is returned to the Host unchanged.

It can be used to label commands and their responses for systems that implement batch queues or which multi-thread commands.

- Response Code

Every response has a unique two-character code. Normally this code has the same first character as the command to which it is a response, and the second character is one greater than the second character of the command (e.g., if the command code is AA, the response code is AB). The value of each code is detailed in the *payShield 10K Host Command Reference Manual*.

- Error Code

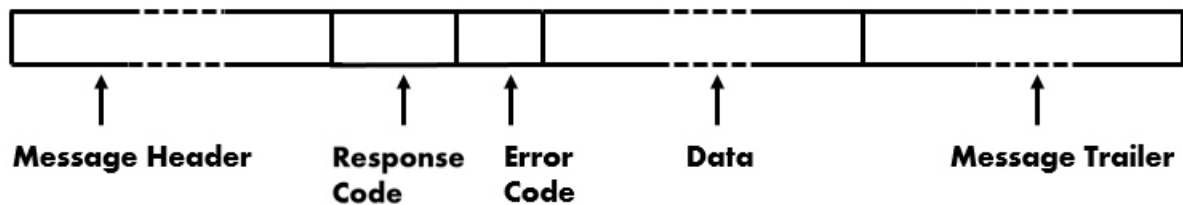
The two-character error code field is used by an HSM to report errors detected during processing. The values are alphanumeric and the value 00 indicates that no errors have been found. If an error (other than 00) is returned, subsequent fields, with the exception of the end of text character, are not returned by an HSM. Error codes specific to a command or frequently returned with a command are listed with the command code in the *payShield 10K Host Command Reference Manual*. A list of global errors is also included in the *payShield 10K Host Command Reference Manual*.

- Data

Many payShield 10K HSM commands return data as a result of the processing. Details of the contents of the returned data are given in the *payShield 10K Host Command Reference Manual*. Generally, data is not returned for error codes other than 00. There are some exceptions to this rule, for example the Key Import command (A6) returns error code 01 to advise that the key being imported does not have odd parity.

- Message Trailer

The message trailer field is present only if it was present in the command message, and it is returned unchanged. It is not returned for error codes other than 00 or 02.



1.4 Data Representation

When sending data to an HSM, other than data that is already in character format, encode each digit (0-9, A-F) as a character (e.g., to send the hexadecimal value 1234ABCD to an HSM requires 8 characters).

For Ethernet communications, the HSM accepts certain fields in binary format. Refer to individual host commands for full details.

Note: The payShield 10K HSM automatically detects whether an incoming command message uses ASCII or EBCDIC characters, and process the command accordingly, returning the result in the same format.

1.4.1 ASCII Character Codes

The table shows the ASCII characters and their hexadecimal values.

ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX
NUL	00	SP	20	@	40	`	60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	\$	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
BEL	07	'	27	G	47	g	67
BS	08	(28	H	48	h	68
HT	09)	29	I	49	i	69
LF	0A	*	2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	-	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F
DLE	10	0	30	P	50	p	70
DC1	11	1	31	Q	51	q	71
DC2	12	2	32	R	52	r	72
DC3	13	3	33	S	53	s	73
DC4	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[5B	{	7B
FS	1C	<	3C	\	5C		7C
GS	1D	=	3D]	5D	}	7D
RS	1E	>	3E	^	5E	~	7E
US	1F	?	3F	=	5F	DEL	7F

1.4.2 EBCDIC Character Codes

The table on the following page shows the EBCDIC characters and their hexadecimal values.

EBCDIC	HEX	EBCDIC	HEX	EBCDIC	HEX	EBCDIC	HEX
NUL	00	SP	40		80		C0
SOH	01		41	a	81	A	C1
STX	02		42	b	82	B	C2
ETX	03		43	c	83	C	C3
HT	04		44	d	84	D	C4
	05		45	e	85	E	C5
	06		46	f	86	F	C6
DEL	07		47	g	87	G	C7
	08		48	h	88	H	C8
	09		49	i	89	I	C9
VT	0A	.(period)	4A		8A		CA
	0B		4B	{	8B		CB
	0C						
FF	0C	<	4C		8C		CC
CR	0D	(4D		8D		CD
SO	0E	+	4E		8E		CE
SI	0F		4F		8F		CF
DLE	10	&	50		90		D0
DC1	11		51	j	91	J	D1
DC2	12		52	k	92	K	D2
DC3	13		53	l	93	L	D3
BS	14		54	m	94	M	D4
	15		55	n	95	N	D5
	16		56	o	96	O	D6
	17		57	p	97	P	D7
CAN	18	!	58	q	98	Q	D8
EM	19		59	r	99	R	D9
	1A		5A		9A		DA
	1B		5B	}	9B		DB
	1C	*	5C		9C		DC
	1D)	5D		9D		DD
	1E	;	5E		9E		DE
	1F		5F		9F		DF
FS	20	- (minus)	60	~ (tilde)	A0	\	E0
	21	/	61		A1		E1
	22		62		A2	S	E2
	23		63		A3	T	E3
LF	24		64	u	A4	U	E4
ETB	25		65	v	A5	V	E5
ESC	26		66	w	A6	W	E6
	27		67	x	A7	X	E7
	28		68	y	A8	Y	E8
	29		69	z	A9	Z	E9
	2A		6A		AA		EA
	2B		6B		AB		EB
ENQ	2C	%	6C	[AC		EC
	2D	_(underscore)	6D		AD		ED
ACK	2E	>	6E		AE		EE
BEL	2F	?	6F		AF		EF
SYN	30		70		B0	0	F0
	31		71		B1	1	F1
	32		72		B2	2	F2
	33		73		B3	3	F3
EOT	34		74		B4	4	F4
	35		75		B5	5	F5
	36		76		B6	6	F6
	37		77		B7	7	F7
	38		78		B8	8	F8

EBCDIC	HEX	EBCDIC	HEX	EBCDIC	HEX	EBCDIC	HEX
	39	`(grave)	79		B9	9	F9
	3A	:	7A		BA		FA
	3B	#	7B		BB		FB
DC4	3C	@	7C		BC		FC
NAK	3D	'	7D]	BD		FD
	3E	=	7E		BE		FE
SUB	3F	"	7F		BF		FF

1.5 Input/Output Flow Control

There is no flow control provided by a payShield 10K HSM. It is the responsibility of the application to ensure that the input buffer in the HSM, which is 32K bytes per connection, is not exceeded.

1.6 Error Handling

There are four types of errors generated by a payShield 10K HSM:

- Fatal errors
- Non-recoverable errors
- Recoverable errors
- Programming errors

Fatal errors indicate a hardware fault in the equipment. Such an error should be logged and reported for user action to be taken (e.g., report to supervisor). Fatal errors are normally reported on the console and are not seen by the host application. The host application usually times out if a fatal error occurs.

Non-recoverable errors cannot be rectified by the program and need user intervention (e.g., with an HSM set into the Authorized state). Such errors should also be logged and reported for user action to be taken (e.g., report to supervisor). This type of error does not mean that an HSM cannot action other types of commands.

Recoverable errors may be the result of data corruption or indicate that an HSM cannot process a command because some other action is required first. The application should attempt to recover by re-issuing the command, attempting to clear the corruption or by implementing the missing action (e.g., an HSM reports that the print format definition is not loaded, so the program should load it and re-issue the failed command).

Programming errors are normally found during testing, but if they occur at other times, they are probably non-recoverable.

Additionally, the application should monitor an HSM for timeouts on the interface.

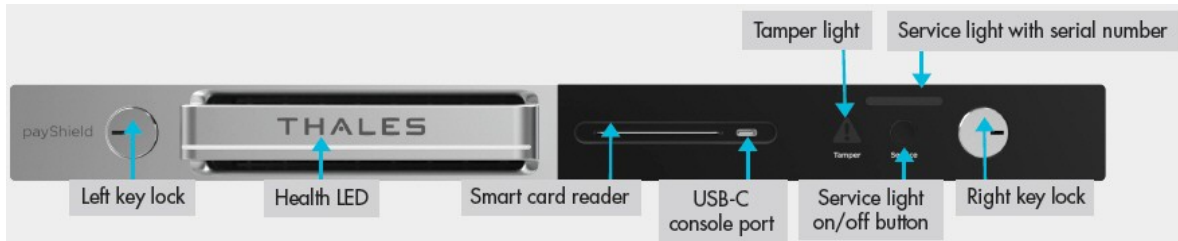
In any of the above events, the application should try to continue processing by using another HSM to action the command. Continued failure may indicate a catastrophic failure of all HSM units (unlikely), a power failure or a program error.

The application should monitor usage of all HSM units and mark any unit as "out of service" if it has given a fatal error, or where a unit repeatedly reports non-recoverable errors.

1.7 Error Logs

Hardware failures, software errors and alarm events are recorded in the Error Log. This has 100 slots, with fields for error code, sub-code, date, time and severity level. When an error is recorded for a particular error code, any subsequent error with the same code updates the date and time for that code, thus each error type remains in the log until it is cleared.

The status of the payShield 10K is displayed via the Health LED on the front of the unit.



- Solid white indicates no errors or no unread errors in the error log
- Solid red indicates unread error(s) in the error log

Error log maintenance can be performed two ways: via payShield Manager and via console command.

- Refer to the *payShield 10K Installation and User Guide*, section titled “Using payShield Manager.
- Refer to the *payShield 10K Console Guide* for discussion of the ERRLOG and CLEARERR commands.

An example of the ERRLOG console command follows:

Entries in the HSM error log have a hash-based integrity check using HMAC. In this example the verification of integrity of the entry failed. A message indicates that an error happened during the verification process and the entry is shown as Unparsed.

```
Offline> ERRLOG <Return>
Error Log (3 entries)
-----
973: May 31 15:17:35 ERROR: [FAN 1 is now present] (Severity: 3, Code = 0x00000003, Sub-Code = 0x00000018)
Error hmac mismatch - Unable to verify text integrity
 974: UNPARSED [[FAN1 is missing, setting FAN??? speed to 16000 RPM] (Severity: 3, Code = 0x00000003, Sub-Code = 0x00000018]
 975: May 31 17:33:14 ERROR: [FAN 1 is now NOT present] (Severity: 3, Code = 0x00000003, Sub-Code = 0x00000018)
```

Please copy this log to a text file and send it to your regional Thales E-Security Support center.

Confirm error log has been read and error light should be extinguished? [Y/N]: Y <Return>

Offline>

1.8 Multiple HSMs

A typical system has two or more HSM units connected as 'live' units. This provides increased capability where the processing requires more than one HSM, and provision for backup in the event of an HSM failure.

Each HSM is normally connected to the Host via a separate Host port, although a port-sharing unit can be used if the number of Host ports available is limited. The sharing configuration is not capable of providing backup if the port or the port-sharing unit becomes faulty.

Optionally it is possible to have a backup unit not connected to the Host but ready for connection in place of a faulty unit. This is not the preferred practice because the unit may remain idle for a long time and may itself have developed a fault.

In addition to the 'live' units, a typical system contains at least one HSM connected to a test or development computer system. This allows changes in the environment to be tested, without disturbing the live system.

1.9 User Storage

It is possible to store keys and other data securely inside the payShield 10K using the User Storage facility. Follow this link to: [Chapter 14 "User Storage"](#).

2 Host Connections

The following communication connections are possible between the payShield 10K HSM and the Host computer:

- Ethernet TCP/IP
- Ethernet UDP

2.1 TCP/IP Protocol

IP addresses can be configured manually or obtained automatically by using DHCP. DNS is also supported to allow the HSM to be addressed by name rather than by its IP address.

The HSM employs TCP for the transfer of data. It acts as a TCP server supporting multiple TCP clients configurable via the CH command. The maximum number of TCP sockets that can be supported is 64. If a TCP client attempts to establish a connection with an HSM that already has the maximum number of configured sockets active, the TCP client's request is rejected.

The HSM supports the TCP Push function. To improve the efficiency of data transfer the TCP protocol software can buffer data into larger blocks, or divide the data into smaller blocks. This is useful for time-critical applications, such as transaction processing systems, where response time is more important than Ethernet utilization efficiency.

The HSM always returns a response to a command using the Push function.

The payShield 10K Secure Host Communications facility supports the use of TLS to protect the TCP/IP link between the HSM and the host. The payShield 10K also allows whitelists of acceptable host IP addresses to be configured using its Access Control Lists (ACL) facility.

Refer to the *payShield 10K Security Manual* for additional information regarding TLS. For to the *payShield 10K Console Guide* for additional information regarding the CH console command.

2.1.1 Port Addresses

When the port is configured using the Console or payShield Manager, a well-known port address is assigned (default value 1500). Refer to the *payShield 10K Host Command Reference Manual*.

2.1.2 Sending Commands

The HSM expects a command to be sent in the form defined in the following table.

Field	Size	Format	Description
LENGTH COMMAND	2 n	Byte Byte	Length of the COMMAND field HSM command
Note: The field COMMAND should not be bracketed by X'02 - X'03 as used with the Async protocol.			

Multiple commands can be sent to an HSM within one TCP transmission. Each should be of the form defined in the table.

Example:

The command format for a diagnostics command (NC) is:

X'00 X'06 X'31 X'32 X'33 X'34 X'4E X'43

where the HSM message header length is set to 04, a message header of 1234 is used, and character representation is ASCII.

2.1.3 Returning Responses

When an HSM receives a command from a TCP client, the command is processed and the response returned to the TCP client. The response is of the form defined in the table.

Field	Size	Format	Description
LENGTH RESPONSE	2 n	Byte Byte	Length of the RESPONSE field HSM response
Note: The field RESPONSE is not bracketed by X'02 - X'03 (or alternative value) as used with the Async protocol.			

The result of each command sent to an HSM is returned as a separate response to the TCP client. This also operates when multiple commands are sent to an HSM in a single TCP transmission.

All HSM responses are returned to the TCP client using the TCP Push function.

Example:

The response format from a diagnostics command (NC) is:

X'00	X'21	X'31	X'32	X'33	X'34	X'4E	X'43	X'30	X'30	X'32	X'36
X'38	X'36	X'30	X'34	X'37	X'34	X'34	X'34	X'39	X'31	X'32	X'34
X'32	X'32	X'30	X'30	X'30	X'37	X'2D	X'45	X'30	X'30	X'30	

where the HSM message header length is set to 04, a message header of 1234 is used, and the character representation is ASCII.

The example shows the error code returned was 00 and the LMK check value returned was 2686047444912422 and the firmware installed is 0007-E000.

2.2 UDP Protocol

The HSM client expects all UDP connections to be made on the Well-Known-Port at the IP address. The IP address and Well-Known-Port address are defined to the HSM when configuring the software settings with the Console CH command.

All UDP host clients sending data to an HSM send the datagrams to the Well-Known-Port at the IP address. The HSM (UDP server) processes the datagram and returns a datagram response to the originating UDP host client.

UDP is a connection-less protocol. If an HSM detects an error in a received datagram it is discarded. The UDP host client should support a time-out mechanism whereby if a response is not received within the time-out period the original request is re-sent.

2.2.1 Sending Commands

The payShield 10K HSM expects a command to be sent in the form defined in the table.

Field	Size	Format	Description
LENGTH	2	Byte	Length of the COMMAND field
COMMAND	n	Byte	HSM command
Note: The field COMMAND is not bracketed by X'02 - X'03 (or alternative value) as used with the Async protocol.			

Only a single command can be sent to an HSM in one UDP transmission (packet).

Example:

The command format for a diagnostics command (NC) is:

X'00 X'06 X'31 X'32 X'33 X'34 X'4E X'43

where the HSM message header length is set to 04, a message header of 1234 is used, and character representation is ASCII.

2.2.2 Returning Responses

When an HSM receives a command from a UDP client the command is processed and the response returned to the UDP client. The response is of the form defined in the table.

Field	Size	Format	Description
LENGTH	2	Byte	Length of the RESPONSE field
RESPONSE	n	Byte	HSM command
Note: The field RESPONSE is not bracketed by X'02 - X'03 (or alternative value) as used with the Async protocol.			

The result of each command sent to an HSM is returned as a separate response to the UDP client.

Example:

The response format from a diagnostics command (NC) is:

X'00	X'21	X'31	X'32	X'33	X'34	X'4E	X'43	X'30	X'30	X'32	X'36
X'38	X'36	X'30	X'34	X'37	X'34	X'34	X'34	X'39	X'31	X'32	X'34
X'32	X'32	X'30	X'30	X'30	X'37	X'2D	X'45	X'30	X'30	X'30	

where the HSM message header length is set to 04, a message header of 1234 is used, and the character

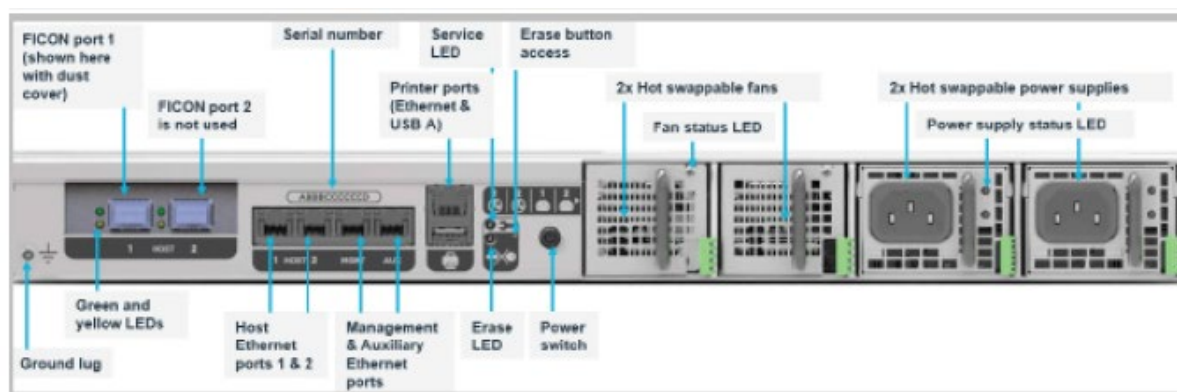
representation is ASCII.

The example shows the error code returned was 00 and the LMK check value returned was 2686047444912422 and the firmware installed is 0007-E000. The *payShield 10K Installation and User Guide* provides extensive information about setting up FICON on the payShield 10K.

2.3 payShield 10K PS10-F (FICON)

FICON is the system for fiber-optic connections to IBM mainframes, and replaces the older ESCON system.

The payShield 10K can be ordered with an auto-sensing factory-fitted FICON (Fiber Connection) interface. This provides a port for connection to an IBM mainframe host computer to allow host commands and responses to be transmitted using a FICON fiber optic interface.



The HSM's FICON interface supports speeds of 32 Gbps, with the option of 8 Gbps or 16 Gbps, if available. If using a switched fabric, the connecting switch must have the connecting port type set to fabric port (F_port).

The FICON interface can be ordered with a choice of transceivers to support. One Transceiver **MUST** be ordered with each payShield 10K FICON Platform:

Part Number	Model Number	Transceiver type
971-000075-001	PS10-F-XCV-S	payShield 10K FICON Short Wave Transceiver

The FICON interface must be specified when the payShield 10K is ordered. It is installed in the factory and cannot be added to an existing payShield 10K.

The only package and performance licence available for the payShield 10K FICON is the following which **MUST** be ordered together with the Hardware Platform, Transceiver as well as any optional licenses and hardware accessories as required:

Part Number	Model Number	Transceiver type
971-701630-001	PS10-PRM-X	Premium package - 2500 cps

Support for this model is provided in base software version v1.3a and above.

Connections

payShield Manager provides a secure GUI interface with an authenticated, encrypted connection allowing a full remote or local management of the payShield 10K. Remote payShield Manager requires an Ethernet cable from the Management Port into your network. If you are not using DHCP, then you may need to use the console to set up the static IP address for payShield Manager.

(Refer to the payShield Manager Quick Start Guide. For the Console, refer to the payShield 10K Console Guide.)

2.3.1 Configuration Settings

The FICON interface is configured using the CH console command or using payShield Manager. Refer to the

payShield 10K Installation and User Guide, section titled *payShield 10K FICON Platform Variant*.

3 PIN Printing and Solicitation

This chapter describes how the HSM can:

- Use directly-attached impact printers to print PIN mailers which are used to notify cardholders of their PINs when new cards are issued, or
- Use directly-attached impact printers to print PIN Solicitation mailers which invite users to submit their desired PINs, and how to process the PIN submitted by the user.
- Use directly-attached laser printers for PIN mailers or PIN Solicitation mailers.
- Support PIN mailer or PIN Solicitation mailer printing in high-volume "print factories".

The focus here is on the role of the payShield 10K in the use of printed matter to engage with cardholders when setting up PINs. Some organizations are moving to using electronic means (in particular, the telephone system or the Internet) to set up PINs.

3.1 PIN Mailers

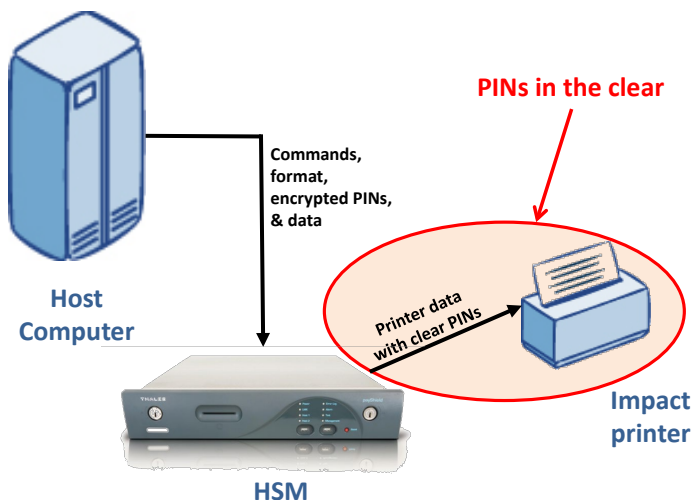
The principle of PIN mailer printing is straightforward - the card issuer's computer systems generate a PIN when a new card is issued or a PIN change is requested, print it in tamper-evident PIN mailer stationery through which the PIN cannot be viewed, and post it to the cardholder separately from the card itself.

The need for security, to prevent the PIN being disclosed within the card issuer's data center, means that an HSM should be used to prevent the PIN being available in the clear. Using the methodology described in this chapter, the PIN is in the clear only on the cable connecting the printer to the HSM (and, of course, inside the PIN mailer).

The stages in providing a secure method of printing PIN mailers are as follows:

- Set up the PIN Mailer form at the HSM
- Generate an Encrypted PIN
- Send PIN Mailer data for printing into the mailer at the HSM
- Verify PIN data cryptography

The following sections assume use of an impact printer directly attached to the HSM and the use of specialized multi- part stationery requiring the use of impact printers.



3.1.1 Setting up the PIN Mailer form

Forms for conveying and protecting PINs can be produced on a printer attached to a payShield 10K HSM. The documents are printed at the terminal in response to a command from the Host.

A form definition needs to be loaded into the payShield 10K so that it can format the output sent to the printer. The HSM can hold a single form definition at a time: this definition formats the printed output. Form definitions may be used for other purposes such as key component printing, and so a PIN Mailer form may need to be loaded before each PIN mailer print run. The form definition data is loaded onto the HSM as text strings. This data consists of:

- Formatting symbols to format the data (See Appendix D, "*Print Formatting Symbols*")
- Constants (text strings)
- Variable print field markers, which will be populated with data when the PIN mailers are to be printed.

The maximum length of a form definition is 299 symbols and characters of constant data.

This form definition is loaded onto the HSM using payShield 10K Host commands. The definition data is stored in an HSM until power is removed, or until a reset is performed.

The form definition allows for PIN digits to be printed as words (using the ^V and ^W symbols). Where words other than the standard English words for numerals ("ZERO", "ONE", "TWO", ...) are to be used, these can also be loaded to the HSM by using a Host command. The required host commands are as follows. (See the *payShield 10K Host Command Reference Manual* for full details.)

payShield 10K Host Command	
Command	Command Description
PA	Load Formatting Data to HSM - to load the first part of the set of symbols and constant characters.
PC	Load Additional Formatting Data to HSM - to load any continuation of the symbols and constant characters. (PC is a continuation of PA, and must be preceded by a PA command.)
LI	Load a PIN Text String (to allow PIN digits to be printed as words in languages other than English).

It is possible to format PIN mailers to be "one-up" (i.e. the PIN mailers are printed one after another on the PIN mailer stationery) or "two-up" (i.e. the PIN mailers are printed in side-by-side pairs).

Example of a two-up formatting string:

Desired result:

1	THOMAS M SMITH	JOHN R JONES
2	APT 4B 1782	427 WEST 9TH ST 3690
3	39 ELM DR	WAYNE PA 19132
4	MEDIA PA 19063	
5		
6	NEVER DISCLOSE YOUR PIN	NEVER DISCLOSE YOUR PIN
7		

The formatting string required to generate the above form is:

```
>L>003^0>033^4>L>003^1>023^P>033^5>053^Q>L>003^2>033^6>L>003^3>033^7>L>L>003NEVER DISCLOSE  
YOUR PIN>033 NEVER DISCLOSE YOUR PIN >L>F
```

In this example, the following print formatting symbols are used:

>L means carriage return + line feed (CR/LF)

>nnn means skip to output column nnn

^n means insert variable print field n - the actual value will be provided at print time

^P means insert clear PIN for mailer 1 of the side-by-side pair

^Q means insert clear PIN for mailer 2 of the side-by-side pair

>F means Form Feed (FF)

The full set of print formatting symbols is defined in Appendix D, "Print Formatting Symbols".

Note that the PIN is not printed on the mailer, although the last 6 digits can be printed for the cardholder's convenience.

Example of a one-up formatting string:

Desired result:

	THOMAS M SMITH	
2	APT 4B	1782
3	39 ELM DR	
4	MEDIA PA 19063	
5	YOUR FULL SERVICE BANK	
6		
1	JOHN R JONES	
2	427 WEST 9TH ST	3690
3	WAYNE PA 19132	
4		
5	YOUR FULL SERVICE BANK	
6		

The formatting string required to generate the above form is:

```
>L>013^0>L>013^1>041^P>L>013^2>L>013^3>L>013 YOUR FULL SERVICE BANK>L>F>
```

See the previous example to understand the meanings of the formatting codes.

3.1.2 Generate an Encrypted PIN

A PIN must be generated at the time that the card data is created in preparation for the issuing of the card. For security, this PIN must be encrypted whenever it is associated with the card data, such that no individual or application has access to both the card's PAN and clear-text PIN.

The payShield 10K offers various host commands for generating a PIN and returning it, encrypted using the HSM's LMK, to the Host application:

payShield 10K Host Command	
ID	Command Description
JA	Generate a random PIN. This command can prevent the generation of PINs considered to be weak.
EE	Derive a PIN from the PAN, using the IBM method.
GA	Derive a PIN from the PAN using the Diebold method.

The PIN is stored encrypted under the LMK, and is passed to the HSM in this encrypted form at print time. Therefore the PIN is never available in plain text to the host computer.

3.1.3 Sending PIN Mailer data for printing at the HSM

When it is required to print the PIN Mailer, the following Host command is used to send the variable PIN mailer data to the HSM.

payShield 10K Host Command	
ID	Command Description
PE	Print PIN/PIN and Solicitation Data

A PE command is issued for each PIN mailer record. The parameters for this command include:

- Whether this record is for one-up printing, or whether it is record 1 or 2 for two-up printing
- The PAN
- The PIN, encrypted under the LMK

The values for the Print Fields identified in the form definition.

The HSM will decrypt the PIN, and then output the record in the defined format to the printer. The only time the PIN is in the clear is on the cable linking the HSM to the printer.

3.1.4 PINs created outside of the payShield 10K

The process described above assumes that the PIN was originally created using the payShield 10K and is encrypted using the payShield 10K LMK.

It may be, however, that the PIN derives from a different source, which cannot encrypt the PIN under the payShield 10K LMK. In this case, the PIN will be available as a PIN Block encrypted using a Zone PIN Key (ZPK) and the following payShield 10K Host command should be used to obtain the LMK-encrypted PIN which is required by the PE command:

payShield 10K Host Command	
ID	Command Description
JE	Translate PIN from ZPK to LMK

3.1.5 Verify PIN data cryptography

The Host application cannot "see" the data that the HSM has sent to the printer, and therefore a mechanism is required to enable the Host application to verify that the cryptographic processing for the PIN mailer printing was performed correctly.

After the HSM has processed the PE Host command, it sends 2 responses to the Host application:

payShield 10K Host Command	
ID	Command Description
PF	Before the PIN mailer is printed. This includes a cryptographic PIN check value calculated using the LMK.
PZ	After the PIN mailer is printed. This confirms that the physical printing activity completed successfully.

The Host application should confirm the check value returned in the PF response. It does this by sending the following Host command - preferably to a different HSM from the one that printed the mailer (but which is using the same LMK).

payShield 10K Host Command	
ID	Command Description
PG	Verify PIN/PIN and Solicitation Mailer Cryptography

This command contains the same PAN and encrypted PIN as was used in the PE Host command, and the check value contained in the PF response.

The PH response to the PG Host command will indicate whether there is a mismatch between the encrypted PIN and the check value.

3.2 PIN Solicitation

The objective of PIN solicitation is to send a mailer to the cardholder to invite them to:

- Select their desired PIN when a new card is about to be issued and return the desired PIN to the card issuer.
- Select a new PIN for a previously issued card.

The PIN may be returned by various methods depending on how the card issuer's applications have been designed, such as:

- Returning all or part of the PIN Solicitation mailer onto which the PIN has been entered
- Entry at an ATM
- Entry via the Internet
- Entry by telephone (Interactive Voice Recognition)

Once the PIN is returned, it is then associated with the card data in the card issuer's application

The crucial factor in making this secure is that the mailer and returned data do not include the PAN. Instead they include a reference number which is a cryptographic representation of the last 10 digits of the account number (excluding the account number check digit). This reference number is a 10-digit number, followed by two check digits.

Knowledge of the PIN and reference number is of no value to a fraudster. The relationship between PAN and reference number is determined only within the HSM, and the PAN and PIN are never presented together.

Because the reference number is the only link to the cardholder's PIN, there must be a means of validating the data that is manually entered. There is no way to validate the PIN except through dual entry procedures or through the visual comparison of the value entered and the value recorded on the mailer form.

The 12-digit reference number, unlike the PIN, can be validated by a Host program - see the next sub-section. The check digits can be validated during or after data entry.

PIN solicitation data is batch processed using Host commands. The number of records entered must be greater than or equal to the minimum batch size specified in the payShield 10K's security settings. Each batch consists of at least one logical record. Each logical record contains the 12-digit reference number in the returned solicitation mailer and the cardholder-selected PIN.

When the batch has been loaded to the payShield 10K internal memory, the HSM encrypts the PINs under LMK pair 02-03, and decrypts the reference numbers, yielding a value which contains the 10 right-most digits of the account number (excluding the check digit). The encrypted PIN and 10 digits of the account number are returned to the Host. The Host can match the account number digits and store the encrypted PIN for subsequent processing (for verification purposes or the creation of PIN offsets etc.).

3.2.1 Validation Algorithm

The algorithm for validating the two check digits of a reference number is as follows:

The first of the two check digits is calculated as:

$$\text{MOD } 10 [10 - \text{MOD } 10 (Y)]$$

where Y is the sum of the products obtained by multiplying the 3rd to the 10th digits of the reference number by the following weights:

Digit	Weight
3	9
4	7
5	8
6	6
7	7
8	9
9	6
10	8

Second Check Digit:

The second check digit is calculated as:

$$\text{MOD } 10 [10 - \text{MOD } 10 (Z)]$$

where Z is the sum of the following:

$$f(\text{digit } 1) + \text{digit } 2 + f(\text{digit } 3) + \text{digit } 4 + f(\text{digit } 5) + \text{digit } 6 + f(\text{digit } 7) + \text{digit } 8 + f(\text{digit } 9) + \text{digit } 10 + f(\text{first check digit})$$

The value of $f(\text{digit } n)$ is determined as follows:

Digit	$f(\text{digit } n)$
0	0
1	2
2	4
3	6
4	8
5	1
6	3
7	5
8	7
9	9

The MOD 10 (n) operation yields a value that is the remainder after dividing n by 10. This remainder is the same as the low-order digit on n.

Example:

The following example illustrates the validation of the reference number 936125183702, where 0 is the first check digit and 2 is the second check digit.

10-digit reference										check digits	
9	3	6	1	2	5	1	8	3	7	0	2

CHECK DIGIT VALIDATION EXAMPLE

FIRST CHECK DIGIT

6	1	2	5	1	8	3	7								
x 9	x 7	x 8	x 6	x 7	x 9	x 6	x 8								
5	7	1	3	7	7	1	5								
4	+	7	+	6	+	0	+	7	+	2	+	8	+	6	= 260

MOD 10 [10 – MOD 10 (260)] = 0

MOD 10 [10-0] = 0

MOD 10 [10] = 0

SECOND CHECK DIGIT

f(9) + 3 + f(6) + 1 + f(2) + 5 + f(1) + 8 + f(3) + 7 + f(0) =

9 + 3 + 3 + 1 + 4 + 5 + 2 + 8 + 6 + 7 + 0 = 48

mod 10 [10 – mod 10 (48)] = 8

mod 10 [10 – 8] = 2

mod 10 [2] = 2

3.2.2 Stages in PIN Solicitation

The stages in providing a secure method of PIN Solicitation are as follows:

The stages in providing a secure method of PIN Solicitation are as follows:

- Set up PIN Solicitation Mailer form at the HSM
- Generate an Encrypted PIN (optional)
- Send PIN Solicitation Mailer data to the HSM for printing on a printer directly attached to the HSM.
- Verify the PIN data cryptography
- Processing the selected PIN returned by the cardholder.

As for PIN printing, this section assumes use of an impact printer directly attached to the HSM and the use of specialized multi-part stationery requiring the use of impact printers.

3.2.2.1 Setting up the PIN Solicitation Mailer

This is the same process as described earlier for setting up the PIN Mailer form.

The available symbols to control printing (sent to the HSM using the PA and PC Host commands) include:

^R for Reference Number 1

^Q for Reference Number 2.

See Appendix D: "Print Formatting Symbols" for the full set of printing control symbols.

The mailer can be designed to print the Reference Number but not the PIN in situations where the cardholder is required to provide a PIN. Alternatively, both the reference Number and the PIN can be printed for situations where the cardholder is invited to change the default PIN that the issuer has provided and will use in the absence of a customer-specified PIN.

Note that the PAN is not printed on the mailer, although the last 6 digits can be printed for the card holder's convenience.

3.2.2.2 Generating an Encrypted PIN

An encrypted PIN is required only where the card issuer is providing a default PIN which the cardholder is invited to change. This is not necessary where the card issuer always requires the cardholder to select their own PIN.

Where an encrypted PIN is required, the same Host commands are used as described previously.

3.2.2.3 Sending PIN Solicitation Mailer data for printing

Where the mailer is to include both the PIN and the Reference Number, the same PE Host command as described earlier should be used.

Where only the Reference Number (without the PIN) is to be printed on the mailer, the OA Host command should be used.

payShield 10K Host Command	
ID	Command Description
OA	Print a PIN Solicitation Mailer

This provides the same information as the PE Host command, except that it does not provide an encrypted PIN.

The Reference Number is not part of the PE or OA command, but is calculated by the HSM from the last 10 digits of the PAN (excluding the check digit) prior to printing the mailer.

3.2.2.4 Verify PIN Solicitation data cryptography

The Host application cannot "see" the data that the HSM has sent to the printer, and therefore a mechanism is required to enable the Host application to verify that the cryptographic processing for the PIN Solicitation mailer printing was performed correctly.

After the HSM has processed the PE or OA Host command, it sends 2 responses to the Host application:

payShield 10K Responses to PE Command	
ID	Command Description
PF	Before the PIN mailer is printed. This includes a cryptographic PIN check value calculated using the LMK.
PZ	After the PIN mailer is printed. This confirms that the physical printing activity completed successfully.

payShield 10K Responses to OA Command	
ID	Command Description
OB	Before the PIN Solicitation mailer is printed. This includes a cryptographic Reference Number check value calculated using the LMK.
OZ	After the PIN Solicitation mailer is printed. This confirms that the physical printing activity completed successfully.

The Host application should confirm the check value returned in the PF or OB response. It does this by sending the following Host commands - preferably to a different HSM (but with the same LMK).

payShield 10K Host Command	
ID	Command Description
PG	Verify PIN/PIN and Solicitation Mailer Cryptography (for a PF response)
RC	Verify Solicitation Mailer Cryptography (for an OB response)

These commands contain the same PAN and encrypted PIN as was used in the PE or OA Host command, and the check value contained in the PF or OB response.

The PH or RD response to the PG or RC Host command will indicate whether there is a mismatch between the encrypted PIN and the check value.

3.2.2.5 Processing the selected PIN returned by the cardholder

The cardholder will return their selected PIN together with the Reference Number. This data needs to be processed as follows prior to the card being issued.

3.2.2.5.1 Entry of PIN and Reference Number into a Host application

The PIN and Reference Number need to be entered into a host application. The reference Number consists of 10 digits of data plus 2 check digits. The Host application should confirm that the Reference Number has been entered correctly by validating the check digits. This should be done in the host application, using the algorithm documented elsewhere in this manual.

3.2.2.5.2 Recovery of PAN and Encrypted PIN

The Reference Number and plain-text selected PIN must now be converted into a plain-text PAN and encrypted PIN, which can be passed to the card issuing system. This is accomplished using the HSM.

Up to 2,520 records (consisting of Reference Number + Selected PIN) can be loaded into the HSM at a time. This

data goes into the HSM's user storage area, and overwrites any data that is already there: any user storage area data that is overwritten in this way will need to be reloaded after this process is completed.

These records are loaded using the following Host commands:

payShield 10K Host Command	
ID	Command Description
QC	Final Load of Solicitation Data to User Storage - one QC command is required for the batch, and can contain up to 1,260 records.
QA	Load Solicitation Data to User Storage - if more than 1,260 records is needed for the batch. Each QA command can contain 25 records. Any QA command(s) must precede the QC command.

The QB responses to the QA command are simple acknowledgements. The actual returned data for all the QA and QC commands is returned in one or two QD responses to the QC command. (If it is required to ensure that only a single QD response is returned, the batch size must not exceed 1,260 records.)

For each of these records, the QD response(s) provide(s) the 10 rightmost digits of the PAN in plain-text and the PIN encrypted under the LMK. This data will then be used by the host applications to generate the data for the card to be issued.

In order to prevent the plain text PIN in the QA/QC commands from being matched with the account number in the QD response(s), the order of the records in the QD response(s) is randomized. To provide additional security, the minimum batch size (as defined in the CS Console command or in the payShield Manager Configuration / Security Settings / Initial dialogue box) should be made as large as possible.

3.3 Printer Support

3.3.1 Impact Printers

The payShield 10K allows direct attachment of printers with the following interfaces:

- Serial
- Parallel - D25 connectors
- Parallel - Centronics
- USB

These printers are connected to one of the USB ports on the payShield 10K. Serial and parallel printers are attached using the appropriate serial or parallel adapter cable from Thales: these adapter cables are intelligent and include microprocessors, the drivers for which are included with the HSM software. (USB adapters from other sources should not be used as these may not work with the drivers installed on the HSM.)

Cable length limitations conform to appropriate standards. (Note that for serial printers, the maximum cable lengths are dependent on the baud rate of the interface and the physical characteristics of the type of cable used.)

Printers must support the ASCII character set.

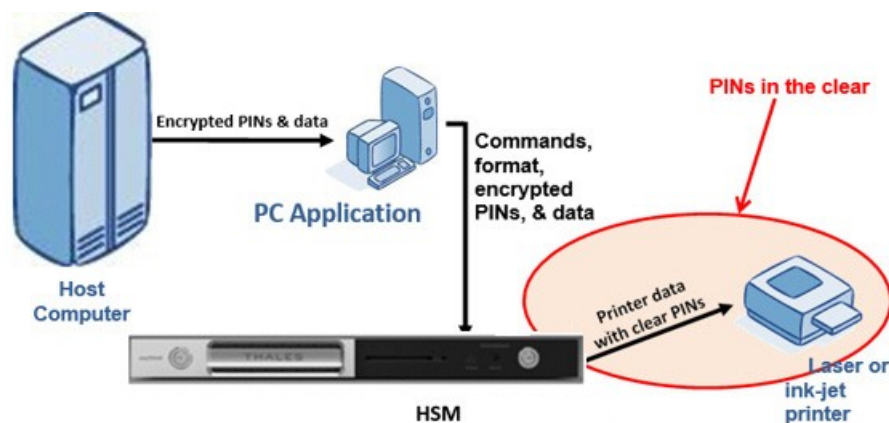
3.3.2 Directly-attached Laser or Ink Jet

There is a gradual move away from character-based impact printers to laser (or ink-jet) printers for the production of PIN or PIN Solicitation mailers. These offer benefits in terms of performance, quality of output, and use of a wider range of stationery.

A PC application (e.g., Thales PINman) is required to manage and drive the laser printer, providing very accurate positioning of printing. Additional fonts are required, and the laser printer may require additional memory to accommodate these.

As with the use of directly-attached impact printers, this arrangement offers a high degree of security because the clear PINs are only present on the printer cable and in the printer.

For convenience, this document will refer just to laser printers. However, ink-jet printers can also be used.



3.3.2.1 Stationery for Laser-printed mailers

Mailer printing using laser printers cannot use the traditional multi-part stationery, which requires an impact printer to cause the secret data to be printed inside the envelope.

Instead specialized laser printer stationery is used, with a plastic panel where the PIN (or secret data) will be printed. The panel has a printed mask on it such that the PIN printed on the panel cannot be read unless the panel is removed and held up to the light or the mask is rubbed off: if these actions are performed, it is evident that the panel has been tampered with.

Special fonts, provided by the stationery manufacturer, are required to allow the PIN to be printed over the mask. Accurate positioning is essential to align the mask and printed PIN.

Examples of stationery made for this purpose are:

- Bastione PIN-TAB
- Hyadalam Laser pin
- Page International ICS V3

3.3.2.2 Security Concerns

The use of laser printers for printing of mailers is growing in popularity because of the benefits in terms of performance, reduced noise, and quality of output. However, some concern has been expressed about the security of this approach - for example at:

<http://www.cl.cam.ac.uk/~mkb23/research/PIN-Mailer.pdf>

Anyone contemplating implementing laser printing technology should research the concerns and implement any additional security measures that they feel are appropriate.

3.3.2.3 Modifications for this solution

The principles described previously relating to the use of character impact printers apply equally to the use of laser printers, but there are some additional considerations.

Fonts

The laser printer must be capable of allowing the specialized fonts needed for the stationery to be loaded. This may require additional memory to be installed in the printer.

PC Application

A PC application is necessary to enable this approach to work. This must be able to:

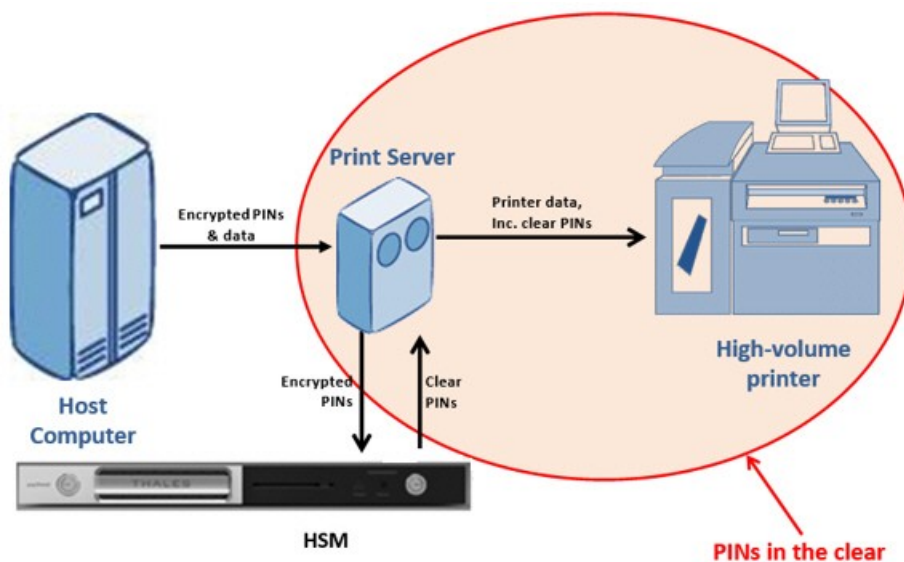
- Communicate with the payShield 10K - sending the appropriate host commands and data (including encrypted keys) and processing responses.
- Design output forms.
- Pass pri
- Enter control data to the HSM, using the PA and PC host commands as described previously. Particularly important is the ability to accurately position the (notional) print head - the standard column/row positioning provided by the standard PA/PC formatting commands is not precise enough, and the laser printer chosen must allow print head positioning by user-defined data. In the PA/PC command the control string required by the printer to do this would be passed by the application to the HSM using the "I" Symbol ('6A' in EBCDIC, '7C' in ASCII). See Appendix E, "Example laser printer formatting control codes" for an example.

3.3.3 High-volume "Print Factories"

3.3.3.1 Overview

Where high-volume PIN Mailer printing is required, greater throughput can be achieved by using a print server driving one or more high-volume laser printers, possibly providing other functions such as collating, stapling, stuffing into envelopes, and so on.

In this case, the printer is not directly connected to the HSM but is connected to the print server. The print server can make use of one or more HSMs to decrypt the PINs provided by the host application. It then incorporates the clear PINs returned by the HSM in the data sent to the printer.



Commercial products implementing this architecture are available from various vendors such as Red Titan and Phalanx e-Solutions.

3.3.3.2 Security Considerations

Whereas in the previously discussed cases of directly attached printers the clear PINs are only present in the printer and the cable connecting the printer to the HSM, in the print factory scenario the PINs are also in the clear in the print server and on the network connecting the print server to the HSM. Furthermore, the print server is also connected to the host system and potentially to the institution's network.

There is therefore a wider range of attack opportunities than with directly attached printers. Organizations implementing this printing architecture should satisfy themselves that they have implemented suitable security and system design measures - which might include:

- Physical access controls
- Network isolation
- Software design preventing PINs returned by the HSM from being accessed more than once
- Use of different print engines used for sensitive and non-sensitive data
- Automatic memory flushing.

Solutions of this type will use similar stationery to that used with directly attached laser or ink-jet printers, and so the Caution described in that section also applies here.

However, the Phalanx PIN Production System using PIN-TAB mailers meets the UK Cards Association Standard 71, Security level 4 (Formerly APACS PIN standard).

3.3.3.3 Relevant payShield 10K Commands

The following payShield 10K Host commands allow:

- a PIN encrypted under the LMK to be decrypted, and
- a PIN encrypted under a ZPK to be translated to encryption under the LMK (so that it can then be decrypted).

payShield 10K Host Command	
ID	Command Description
NG	Decrypt a PIN encrypted with LMK.
JE	Translate a PIN from ZPK to LMK Encryption

The response to the NG command includes both the clear PIN for PIN Mailer printing and a Reference Number that can be used for PIN Solicitation Mailer printing.

4 Transaction Key Schemes

The transaction key scheme is a technique in which data-encrypting keys change with each transaction in a manner that cannot be followed by a third party. This is typically of use in Electronic Fund Transfer at Point of Sale (EFTPOS) systems where fund transfer requests and responses are exchanged between a retailer (EFTPOS terminal) and an acquirer, and then, optionally, between the acquirer and the card issuer.

The payShield 10K HSM supports three techniques:

- Racal (or APACS) Transaction Key Scheme (RTKS).
- Derived Unique Key Per Transaction (DUKPT).
- Australian (AS 2805) Transaction Key Scheme.

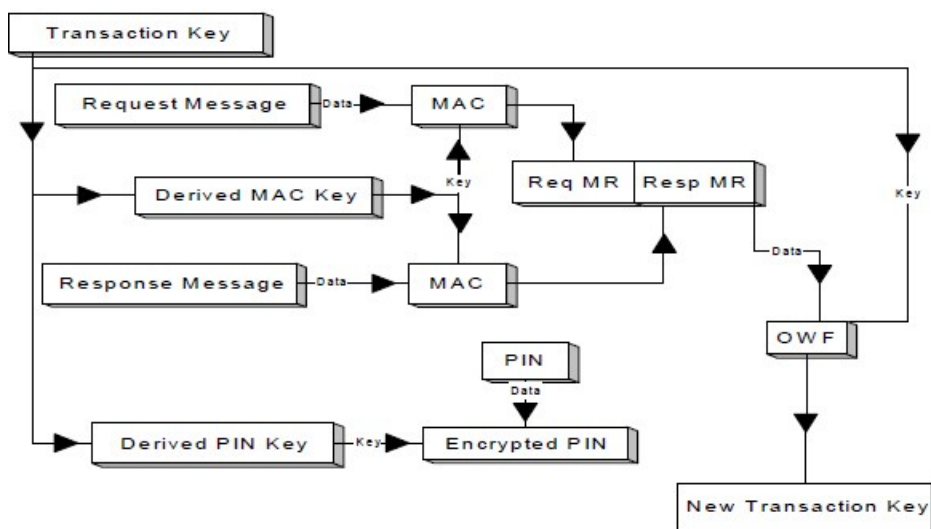
The payShield 10K also provides functionality to enable it to participate in systems meeting the requirements of GBIC/ZKA: this is discussed later in this chapter.

4.1 Racal Transaction Key Scheme (RTKS)

The Racal Transaction Key Scheme (RTKS) is a key management technique that is closely coupled with message authentication. The functions provided by an HSM include key management in addition to MAC generation and verification.

In the Racal Transaction Key Scheme the TPK and the TAK are updated after each EFT transaction using an algorithm that depends on the current key and the details of the transaction (which are known to both communicating parties but do not form part of the transmitted message, and so would not be known to a third party).

This affords a very high degree of protection for the cryptographic keys. Even if a third party were able to discover the value of the cryptographic key in use at a particular time, this would not facilitate discovery of the keys used on previous transactions (i.e., the scheme has good break-backward protection). Also, if some card data were not transmitted, the third party would not be able to discover the new value of the keys for the subsequent transaction (break-forward protection).



The key update algorithm used in the Racal Transaction Key Scheme is based on a One- Way Function (OWF) involving the current key value and the Message Authentication Code (MAC) residues from the request and response messages from the transaction. The use of the MAC residues is important, as they are not part of the transmitted data.

In this scheme, the MACs are calculated using a key derived from the transaction key, and not the transaction key itself. This also applies to the PIN encrypting key.

For more details of the Racal Transaction Key Scheme see Racal-Transcom Publication RRL4 Secure Key Management for Pin Encryption and Message Authentication.

4.1.1 RTKS Functions

The following functions are all for use at an acquirer site:

- Transaction request with PIN (T/AQ key). Used to receive a cardholder request message from a terminal with a PIN encrypted under the T/AQ key.
- Transaction request without PIN. Used to receive a cardholder request message from a terminal with no PIN.
- Transaction request with PIN (T/CI key). Used to receive the request from the terminal when the PIN key cannot be determined by the acquirer.
- KEYVAL translation. Used to pass KEYVAL to the card issuer (required to derive the PIN key) when the PIN key cannot be determined by the acquirer.
- Administration request. Used to receive an administration request message (such as a reconciliation request).
- Transaction response originating at the card issuer. Used when authorization is generated at the card issuer.
- Transaction response originating at the acquirer. Used when authorization is generated by the acquirer.
- Verify confirmation message from terminal. Used to verify the MAC on a confirmation message from the terminal.

The host commands RI, RK, RM, RO, RQ, RS and RU are only available when Racal Transaction Key Scheme is selected in the payShield 10K security settings.

The existing Racal Transaction key commands have been modified to support longer messages. The new commands are backward compatible with existing systems.

The details of the modifications are as follows:

Old style:	Pointer (not all functions)	2 H	
	Message Length	2 H	
	Message Text	n A	
New style:	Pointer (if required)	2 H	
	Message Length	2 H	
	Message Text	n A	
	Delimiter	1 C	
	Extended Message Pointer(s)	4 H	
	Extended Message Length	4 H	
	Extended Message	n A	Optional, only if above field is non zero

To use the extended message length option, the calling application has to set the Message Length field to zero, whereupon the Message Text field will be of zero length, i.e. not present. The zero Message Length enables an HSM to check for the optional Delimiter, any Extended Message Pointer(s), and the Extended Message Length field which defines the length of the Extended Message.

Some of the functions do not include a pointer to items included in the message, While other functions include either one or two pointers. If a function does include one or two pointers, one or two Extended Message Pointers are included after the Delimiter as appropriate. The original pointer(s) in the function are ignored when extended messages are used, however the 2 hex digit placeholder(s) for the original pointer(s) must still be supplied.

While the extended commands allow for message sizes up to 65537 characters long (hex FFFF), in practice the limit is imposed by the maximum size of the HSM input buffer. The HSM has a much larger input buffer (32K) per connection although the interface option in use may impose limits which are smaller than this. The HSM will check that the message lengths (and the pointers) are within sensible limits for an HSM platform executing the function.

Users may, if they wish, use the Extended Message Length scheme for small messages (i.e. less than 160 bytes).

4.2 Simultaneous use of both Racal and Australian

Transaction Key Schemes

The original design of the payShield assumed that only one (or none) of these schemes would be used on any one HSM. The desired TKS was selected using a security setting, e.g., in the CS Console command:

Transaction Key Scheme: Racal, Australian or None [R/A/N]:

Because the two TKSs were mutually exclusive, the same Host command code was used to identify two different commands, one applicable to the Racal TKS and the other applicable to the Australian TKS. The command codes affected by this are:

Code	Racal TKS ∅	Australian TKS ∅
RI	Transaction Request With a PIN (T/AQ Key)	Verify a Transaction Request, with PIN, when CD Field not Available
RK	Transaction Request Without a PIN	Generate Transaction Response, with Auth Para Generated by Acquirer
RM	Administration Request Message	Generate Transaction Response with Auth Para Generated by Card Issuer
RO	Transaction Response with Auth Para from Card Issuer	Translate a PIN from PEK to ZPK Encryption
RQ	Generate Auth Para and Transaction Response	Verify a Transaction Completion Confirmation
RS	Confirmation	Generate a Transaction Completion Response
RU	Transaction Request With a PIN (T/CI Key)	Generate Auth Para at the Card Issuer
RW	Translate KEYVAL	Generate an Initial Terminal Key

∅ Racal TKS

∅ Australian TKS

The functionality delivered when any of these Host commands was used was dependent on the security setting mentioned above.

More recently, the need has arisen for both TKSs to be used on the same payShield 10K, and therefore to have access to both versions of these R* commands. In order to allow users to have access to both versions of the commands, the R* commands have been "aliased" as H* commands - e.g., for each of the two RI commands there is an HI command which is identical to the RI command except for the Command code (HI instead of RI) and Response code (HJ instead of RJ).

Where the user wishes to use the version of the command that is appropriate to the TKS selected in the security setting they should use the R* variant of the command.

For example:

- if the TKS is set to "Racal" and the user wants the "Transaction Request Without a PIN" functionality then they would use the RK command.
- if the TKS is set to "Australian" and the user wants the "Generate Transaction Response, with Auth Para Generated by Acquirer" functionality then they would use the RK command.

This is exactly as for earlier versions of payShield 10K software, and so provides backwards compatibility for existing applications.

On the other hand, if the user wishes to use the version of the command that is not appropriate to the selected TKS, they should use the H* variant of the command. For example:

- if the TKS is set to "Australian" and the user wants the "Transaction Request Without a PIN" functionality then they would use the HK command.
- if the TKS is set to "Racal" and the user wants the "Generate Transaction Response, with Auth Para Generated by Acquirer" functionality then they would use the HK command.

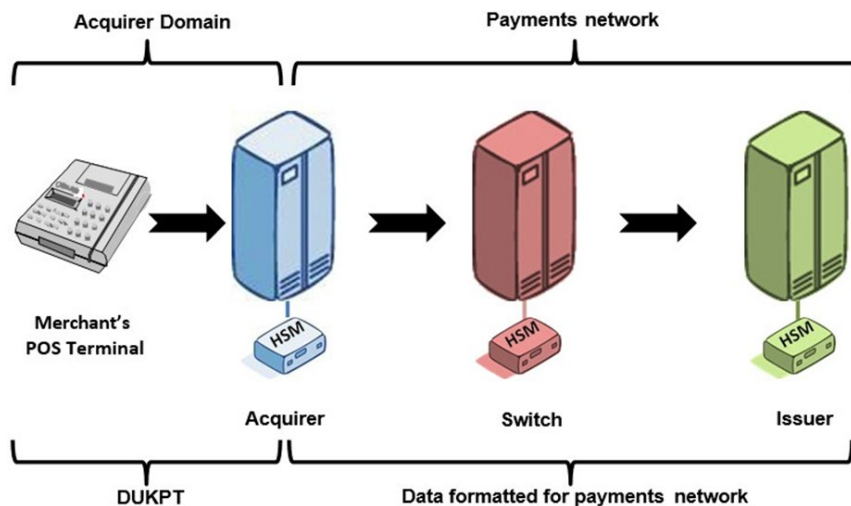
In summary:

	If Transaction Key Scheme = Racal	If Transaction Key Scheme = Australian
You want to process Racal Transaction Key commands	Use the R* variant of the command	Use the H* variant of the command
You want to process Australian Transaction Key commands	Use the H* variant of the command	Use the R* variant of the command

4.3 Derived Unique Key Per Transaction (DUKPT)

DUKPT (Derived Unique Key Per Transaction) is a scheme to manage TDES (Tripe-DES) encryption keys in a card payment environment. It has traditionally been used with POS (Point of Sale) terminals in North America, but adoption is growing in other regions and for other applications - and so DUKPT is becoming increasingly important in the payments space. The DUKPT scheme is specified in ANSI X9.24-1:2009.

DUKPT is used for encrypting PIN blocks, encrypting other data, and message authentication (MACing). DUKPT is typically used between merchants and their Acquirer. The Acquirer will repackage the transaction data for passage through the payments network (which uses Master/Session key management) before passing it to a payments switch.



The strength of DUKPT lies in the fact that a new, unique key is generated for each transaction, such that if a

transaction key is compromised this cannot be used to attack previous at that terminal or transactions at any other terminal. In addition, key management in the DUKPT environment is simplified by having a single master key which can manage an entire estate of terminals.

This technique involves the use of a non-secret "key serial number" and a secret "Base Derivation Key", or BDK. On each transaction, the PIN pad uses a unique key based on a previous key and the key serial number, which contains a transaction counter. It encrypts the PIN with this key, then returns both the encrypted PIN and the key serial number to the acquirer. In an HSM, the key generated by the PIN pad is derived dynamically and independently of the PIN pad, using the original BDK together with the key serial number supplied by the PIN pad.

The same BDK can be used by thousands of PIN pads because each PIN pad has a unique serial number. Therefore, each PIN pad produces a unique key for every transaction and a successful cryptographic attack on one PIN pad will have no effect on any other. The acquirer only has to manage a relatively small number of BDKs, and the algorithm to derive a given transaction key is designed in such a way as to require very little overhead in an HSM.

The Host has the responsibility for maintaining the BDKs. For each transaction, the Host verifies that the serial number supplied by the PIN pad is valid and extracts from internal storage the appropriate encrypted BDK identified by the left-most portion of the serial number. The Host controls BDK generation.

4.3.1 Single-DES and Triple-DES variants of DUKPT

The payShield 10K HSM supports both single DES and Triple-DES variants of DUKPT.

4.3.2 DUKPT Keys

Three levels of keys are employed in DUKPT:

- Base Derivation Key (BDK) - a TDES master key owned by the acquirer. The BDK is used for a large number of terminals - perhaps all the terminals that the provider ships, or to a model of terminals, or to a serial number range.
- Initial Key (IKEY, IK, or IPEK) - a TDES key that is unique to a terminal. The IKEY is used to initiate the sequence of transaction keys, and is then discarded by the terminal.
- Transaction key - generated within the terminal. Keys for PIN encryption, data encryption, and MACing are derived from the transaction key. Each transaction is provided with a unique key to protect its data. When the encrypted data is received by the Acquirer, the Acquirer will derive the same transaction key using the same process that the terminal used to derive the encryption key.

These are described in the following sections.

It can be seen from this process that there is no requirement for the Acquirer and terminal to exchange keys - except in the unlikely event of a terminal generating a million transaction keys and therefore requiring a new IKEY.

It is expected that DUKPT will be able to manage AES keys in the future.

4.3.3 The Base Derivation Key (BDK)

The BDK is a double-length TDES key which is usually generated and owned by the acquirer, and is used for a large number of terminals. (If the BDK is owned by an organization other than the Acquirer, it will need to be distributed to the Acquirer to enable them to process transactions. It will also need to be distributed to any other organization involved in generating IKEYs.) Multiple BDKs will generally be held to allow for different terminal families or groups. BDK distribution can be done:

- electronically, with the BDK protected by a Zone Master Key (ZMK), or
- in the form of printed components, with separate component holders coming together to enable the BDK to be formed from their components.

A stored BDK must be protected by encryption using an appropriate Key Encryption Key (KEK) whenever it exists outside of a Tamper Resistant Security Module (TRSM). Where Thales payment HSMs are used, BDKs are protected by the HSM's Local Master Key (LMK).

4.3.3.1 Base Derivation Key (BDK) Support in the payShield 10K

The following host commands are available to manage BDKs:

Code	Description
A0	Generate a random BDK and return it to the Host encrypted under the Local Master Key (LMK)
A6	Accept a BDK encrypted under a Zone Master Key (ZMK) and translate it to encryption under the LMK
A8	Translate a BDK from LMK to ZMK encryption

The latest revision of the DUKPT standard (X9.24-1:2009) defines two different methods for deriving data authentication and encryption keys:

- The bidirectional method uses a single key to protect terminal-to-host data and host-to-terminal data. BDK types 1 and 3 support the bidirectional method. (Note that a BDK type 3 uses the 'PIN encryption' variant to derive the data encryption key, and therefore can only be used to perform data encryption operations. A BDK-3 cannot be used to perform PIN-related operations.)
- The unidirectional method uses two keys: one key to protect terminal-to-host data, and another key to protect host-to-terminal data. BDK types 2 and 4 support the unidirectional method.

Further information about the usage of the various BDK types is provided later in this chapter.

4.3.3.2 Zone Master Key (ZMK) Support

The payShield 10K HSM supports single-length Zone Master Keys (ZMKs), 16 hexadecimal characters (64 bits); and double-length Zone Master Keys (ZMKs), 32 hexadecimal characters (128 bits). The DUKPT command set ignores the S/D (single/double length) parameter set in the payShield 10K security settings (e.g., by the CS (Configure Security) command).

4.3.4 The Initial Key (IKDY or IK)

The IKEY (originally referred to as an IPEK - Initial PIN Encryption Key) is unique to its terminal. The IKEY is calculated from:

- the BDK
- the Key Serial Number (KSN) which is unique to the terminal - see below.

The payShield 10K A0 host command can generate an IKEY and export it under a TMK (Terminal Master Key) or ZMK (Zone Master Key).

Once created, the IKEY is installed into the terminal. (The IKEY is also recreated transiently by the Acquirer when processing transactions from the terminal in order to derive the same transaction key that the terminal used to encrypt its data.)

4.3.5 Key Serial Number

The KSN has a maximum length of 80 bits, and has the following structure:

Element	Length	Description
Key Set Identifier	5-9 Hex chars. (20-36 bits)	Identifies the BDK to be used for this terminal.
Sub-key Identifier	1 Hex char. (4 bits)	Currently set to 0
Device Identifier	2-5 Hex chars. (8-20 bits)	Unique identifier (i.e. serial number) for this terminal (always even).
Transaction Counter	1 bit + 5 Hex chars. (21 bits)	Counter of the number of PIN encryptions since terminal was initialized.

The first 3 elements in the table above form the Initial Key Serial Number, and do not change during the life of the terminal (unless a new IKEY is loaded for any reason).

Often only 64 bits of the KSN are used, with the KSN padded with "F" Hex characters to the left. In this scheme, the KSN would have the following structure:

- Padding - 4 "F" Hex characters, 16 bits.
- Key set identifier - 6 Hex characters, 24 bits. This allows for about 16 million different BDKs.
- Device Identifier - 5 Hex characters, 20 bits. This includes one bit of the Transaction Counter, leaving 19 bits for the actual Device Identifier. This means that about half a million different devices can be managed by the Device Identifier. No two terminals with the same base derivation key and sub- key identifiers may be given the same device identifier. Because the terminal packs the left-most bit of the transaction counter as the right-most bit of the device identifier, this field is always even (the right-most bit is set to zero).
- Transaction Counter - 5 Hex characters, 20 bits (plus the bit included in the Device Identifier). The transaction counter is supplied by the terminal to identify a particular transaction. It is used by an HSM to compute the actual PIN key. The left-most bit is supplied as the right-most bit of the device identifier, so the length of this field is 20 bits (5 hex digits). This allows for about 1 million transactions before a new IKEY would be required - a limit which is unlikely to be reached.

The terminal cannot accept a serial number longer than 20 characters, so the Host ensures that the total length of the first three fields does not exceed 15 characters.

The Host also supplies to an HSM a three-character KSN descriptor, which defines the length (in characters) of each of the first 3 fields. It is included with the KSN in Host storage and is used by the Host to identify the base derivation key. The KSN descriptor consists of:

- Left character:
base derivation key identifier length.
- Middle character:
sub-key identifier key length (currently always 0).
- Right character:
device identifier length.

Transaction Keys

When the IKEY is installed in the terminal, it calculates up to 21 "Future Keys". These Transaction Keys are the keys that will be used in the encryption of future transactions. The calculation of these keys involves the value of the transaction counter, which increments for each transaction.

When the initial batch of Future Keys has been derived, the IKEY is no longer required, and is deleted by the terminal.

When a transaction is being processed, the next available Transaction Key is used. The keys used for PIN block encryption, MACing, and data encryption are derived from this transaction key.

The KSN is also modified by incrementing the Transaction Counter.

The DUKPT terminal sends its encrypted data and the KSN, together with other transaction data, to the Acquirer.

As each Transaction Key is used, it is deleted by the terminal and replaced by a new Future Transaction Key. This means that even if the security of the terminal is compromised in any way and its keys extracted, they cannot be used to attack a previous transaction from this or any other terminal because the key for that transaction has already been deleted and each terminal generates different keys.

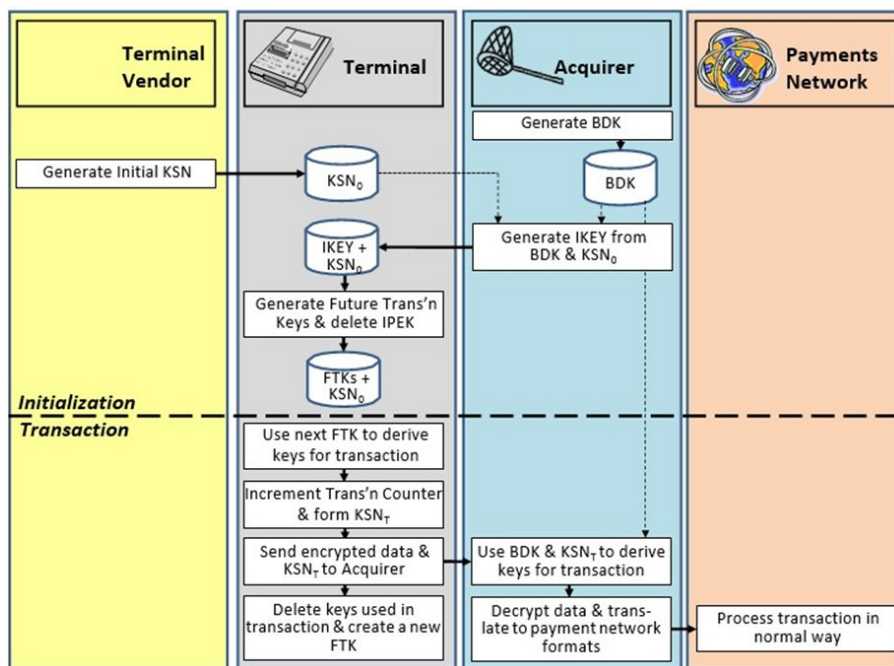
4.3.6 Processing by the Acquirer

When the encrypted data is sent by the terminal to the Acquirer, the KSN (including the transaction counter) is also sent. The Acquirer can reconstruct the Transaction Key used by the terminal from the KSN and the appropriate BDK (as identified in the KSN's Key Set Identifier).

The Acquirer needs to re-package the data received from the terminal into the standard formats used by the payments network. This will include actions such as:

- Translating a DUKPT PIN Block encrypted with the DUKPT transaction key into one of the standard PIN Block formats that the payments network uses encrypted with a Zone PIN Key (ZPK).
- Verifying and translating MACs.

4.3.7 Summary of DUKPT Operations



Notes:

- KSN_0 = Initial Key Serial Number (with Transaction Counter = 0). May be modified by Acquirer before generating IKEY.
- KSN_T = Key Serial Number for the transaction, with Transaction Counter incremented)
- The BDKs held by the acquirer will be protected using an HSM.
- The Acquirer operations shown here will involve the use of an HSM for various cryptographic functions.

AES DUKPT

ANSI X9.24-3:2017 introduces support for deriving a key encryption key (known as the DUKPT Update Key) from a BDK, specifically in order to securely transport a new Initial Key (IKEY) to the terminal. The general key management commands have been extended to support this new KEK.

In support of 3DES DUKPT, the payShield 10K supports five different types of DUKPT Base Derivation Keys (BDKs):

BDK-1 implements ANSI X9.24-1:2009, and derives bi-directional data and MAC keys:

- When the HSM encrypts/decrypts data or generates/verifies a MAC using a BDK-1, it derives the “both ways” data key or MAC key respectively

BDK-2 implements ANSI X9.24-1:2009, and derives uni-directional data and MAC keys:

- When the HSM encrypts data or generates a MAC using a BDK-2, it derives the “response” data key or MAC key respectively
- When the HSM decrypts data or verifies a MAC using a BDK-2, it derives the “request” data key or MAC key respectively

BDK-3 implements ANSI X9.24-1:2009, and derives a single data key, but uses the same techniques that other BDK types use to derive the PIN key. BDK-3 is only supported by the message encryption commands (M0, M2 and M4), and for the sake of clarity, should be ignored in the remainder of this specification!

BDK-4 implements ANSI X9.24-1:2009, and derives uni-directional data and MAC keys:

- When the HSM encrypts data or generates a MAC using a BDK-4, it derives the “request” data key or MAC key respectively
- When the HSM decrypts data or verifies a MAC using a BDK-4, it derives the “response” data key or MAC key respectively

In summary, three different types of BDK (BDK-1, BDK-2 and BDK-4) are required in order to fully support the standard 3DES DUKPT functionality.

Each of these different types of BDKs allow the use of AES BDKs (in addition to supporting 3DES BDKs).

BDK-5 provides support for the Italian Standard Key Derivation Method SPE-DEF-041-112.

- A BDK-5 is identical to a BDK-1 apart from the derivation of the IKEY, which uses a proprietary method.

The table below describes all the different types of BDKs to be supported by the HSM:

BDK Name	Use Case	Key Algorithms			Key Usage	Derived Keys			
		BDK	IKEY	UK		Alg	PIN key	Data Key	MAC key
BDK-1	Acquirer/Terminal	3DES	3DES	-	B0	3DES	One	Same key used in both directions	Same key used in both directions
		AES	AES	AES		AES			
BDK-2	Acquirer	3DES	3DES	-	41	3DES	One	Different key used for sending/receiving	Different key used for sending/receiving
		AES	AES	AES		AES			
BDK-3	Acquirer/Terminal	3DES	3DES		42	3DES	-	Same key used in both directions	-

BDK Name	Use Case	Key Algorithms			Key Usage	Derived Keys			
		BDK	IKEY	UK		Alg	PIN key	Data Key	MAC key
BDK-4	Terminal	3DES	3DES	-	43	3DES	One	Different key used for sending/receiving	Different key used for sending/receiving
		AES	AES	AES		AES			
BDK-5	Acquirer/Terminal (Italian key deviation)	3DES	3DES	-	44		One	-	-

The “use cases” are defined as follows:

- “Acquirer” means the HSM is behaving like a traditional acquirer, receiving “request” messages from the terminal (which need to be decrypted and validated), and generating “response” messages for the terminal (which need to be encrypted and MAC'd)
- “Terminal” means the HSM is behaving like a terminal, generating “request” messages for an acquirer (which need to be encrypted and MAC'd), and receiving “response” messages from an acquirer (which need to be decrypted and validated).

4.3.7.1 Implementation Notes for AES DUKPT

- The BDK (like all other AES keys) must be protected by an AES Key Block LMK.
- The BDK will always be a 128/192/256-bit AES key.
- The Initial Key (IKEY) - derived from the AES BDK - will always be a 128/192/256-bit AES key.
- The Update Key (UK) - derived from the AES BDK - will always be a 128/192/256-bit AES key.
- The derived working keys (PIN, Data, MAC keys) will always be AES keys (same size as the AES BDK).
- Note that X9.24-3 also allows for the derivation of 3DES and HMAC keys, but this is not supported at this time.
- All PIN blocks protected by an AES PIN key will use ISO format 4 (HSM format 48).
- The KSN is 24 bytes.

4.3.8 Variations on DUKPT

4.3.8.1 Data Encryption using a PIN Encryption Key Variant

The traditional method of implementing DUKPT X9.24-1:2004 is where the terminal is initially loaded with an IKEY, and thereafter derives a new PIN encryption key (and optionally a new MAC key) for each transaction it performs. Since this method does not support the derivation of data encryption keys, some terminal vendors have implemented the following hybrid method, using two BDKs: one of which is used to derive the PIN encryption keys and MAC keys, and a second BDK which is used to derive the data encryption keys.

In this case, the terminal is initially loaded with IKEYa and IKEYb (which have been derived from BDKa and BDKb respectively). During each transaction, the terminal derives two keys, TransactionKeya and TransactionKeyb. The PIN encryption key (and optionally the MAC key) is derived from TransactionKeya using X9.24-1:2004 compliant methods. The data encryption key is derived from TransactionKeyb using the PIN encryption key derivation method described in

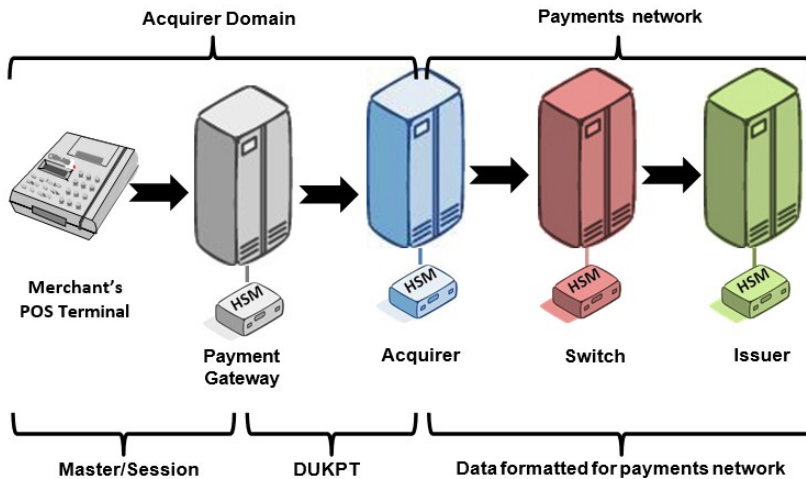
X9.24-1:2004.

BDKa is a BDK Type 1 and BDKb is a BDK Type 3.

4.3.8.2 Data Translation to DUKPT

In the diagram above, the data flow is from the POS terminal to the Acquirer and then into the standard payments network. As mentioned already, this will involve translating DUKPT data to formats used by the payments network.

However, consider the following environment:



Here we have an additional link in the chain - the Payment Gateway. This is an intermediary between the merchant and the Acquirer. The Payment Gateway provides various services to the Merchant, and might be needed because the merchant is too small to have a direct relationship with an Acquirer. For the present discussion we shall assume that the Payment Gateway is required because the Acquirer provides support for only DUKPT terminals, but the merchant does not support DUKPT.

In such an environment, the Payment Gateway will implement master/session key management with the merchant POS terminals, using the same data and PIN Block formats as the payments network. However, because the Acquirer supports only DUKPT key management and data formats on the terminal side, the Payment Gateway must have the capability to translate data to DUKPT formats.

4.3.9 DUKPT for Point-to-Point Encryption and Mobile Acceptance

4.3.9.1 Point-to-Point Encryption (P2PE)

Traditionally in card payment processing, only the PIN is encrypted (in the form of a PIN Block) - other data such as the PAN has generally not been encrypted: PCI DSS covers the Acquirer domain and requires encryption of cardholder data such as the PAN when the data passes over a public network, but not where the data is on a secure private network.

Encrypting cardholder data at the merchant, at the Acquirer, and on the network connecting them - whether this is a private network or not - takes the data out of scope of PCI DSS compliance and audit. This is very attractive to merchants and Acquirers because of the cost of PCI DSS compliance and audits, and so there is a growing interest in encrypting all cardholder data in the Acquirer domain - this is referred to as P2PE.

A wide variety of P2PE implementations are available, with proprietary designs by their vendors. These implementations may involve a Payment Gateway.

DUKPT is often used as the key management method for a P2PE solution, because it provides a standardised way of exchanging encrypted data (not just PINs) and high levels of security.

4.3.9.2 Mobile Acceptance (or Mobile Point of Sale - mPOS)

Traditional card payment terminals are beyond the reach of small merchants or mobile merchants (such as market stall operators or plumbers) because of the cost of the terminals and the fixed-line communications infrastructure

needed to support them.

Mobile Acceptance is a technology that addresses this issue, and is enjoying a rapid rate of adoption.

With Mobile Acceptance, the terminal consists of low-cost, secure card readers and PIN entry devices attached to intelligent mobile communications devices (such as tablets or standard smartphones). This brings the technology within financial reach of any merchant or service provider and removes the dependence on a fixed communications infrastructure.

P2PE must be employed between the mobile terminal and the Acquirer (or a Payment Gateway providing a mobile transaction service to merchants and passing the transactions to an Acquirer).

Again, DUKPT is often employed as the key management scheme in this environment.

4.3.10 The role of the payShield 10K in DUKPT

The payShield 10K provides a secure platform for performing cryptographic functions, protecting secret data against attackers either within or outside of the organization. It can undertake tasks such as:

- Generating keys
- Protecting keys by encrypting with an LMK
- Encrypting/decrypting data
- Importing/exporting keys
- Translating PINs from encryption under the base derivation key to encryption under the appropriate interchange key shared between the acquirer and the issuer or switch.
- Verifying the PINs received from a terminal using base derivation keys. All four HSM verification methods (IBM, Diebold, Visa PVV and Encrypted PIN) are supported.
- Calculating and verifying various types of check values.
- Verifying MACs on messages received from a POS terminal, and generates MACs on messages to be sent to a POS terminal.

The card schemes require that organizations (Acquirers, Switches, and Issuers) who are processing their cards' transactions use HSMs such as the payShield 10K for certain cryptographic functions. Even for functions not covered by these mandates, use of HSMs is best security practice for these organizations.

Systems and products used by these organizations must comply with standards developed by PCI - a body set up by the card schemes. Acquirers' systems are audited against the PCI DSS standard. Payments products must comply with a number of standards and are certified against various PCI standards. For instance, payment terminals, including DUKPT-capable devices, must be certified against the PCI PED (PIN Entry Device) standard. In the case of HSMs, PCI introduced the PCI HSM standard fairly recently. Although this standard is not yet mandated by the card schemes, PCI in their standard for P2PE specify that the HSMs used by the acquirer are certified to either FIPS 140-2 or the PCI HSM standard.

4.3.11 payShield 10K Host Commands for DUKPT

4.3.11.1 Thales Key Types for DUKPT

Thales Key Types for DUKPT

The host commands in the next sub-section use the following Thales key types relating to DUKPT:

Thales Key Type	Key Type Code (Varian LMKs)	Key Usage (Key Block LMKs)	Description
BDK Type 1 ("BDK-1")	009	B0	<p>This type of BDK is used to secure transactions sent between a terminal and an acquirer.</p> <p>Three types of bidirectional keys may be derived from a BDK-1:</p> <ul style="list-style-type: none"> PIN encryption keys Data authentication (MAC) keys Data encryption keys <p>The same key is used in both terminal-to-acquirer messages and acquirer-to-terminal messages.</p> <p>This type of BDK meets the requirements of ANSI X9.24- 1:2009.</p>
BDK Type 2 ("BDK-2")	609	41	<p>This type of BDK is used to secure transactions sent between a terminal and an acquirer.</p> <p>Five types of unidirectional keys may be derived from a BDK-2:</p> <ul style="list-style-type: none"> PIN encryption keys (for terminal-to-acquirer messages) Data authentication (MAC) keys (for terminal-to-acquirer messages) Data authentication (MAC) keys (for acquirer-to-terminal messages) Data encryption keys (for terminal-to-acquirer messages) Data encryption keys (for acquirer-to-terminal messages) <p>Different keys are used for terminal-to-acquirer messages and acquirer-to-terminal messages.</p> <p>This type of BDK meets the requirements of ANSI X9.24- 1:2009.</p>
BDK Type 3 ("BDK-3")	809	42	<p>This type of BDK is used to secure data portions (not PINs or MACs) of transactions sent between a terminal and an acquirer.</p> <p>One type of bidirectional key may be derived from a BDK-3:</p> <ul style="list-style-type: none"> Data encryption key <p>Note: The data encryption key is derived from the BDK-3 using the "PIN verification" variant,</p>

Thales Key Type	Key Type Code (Varian LMKs)	Key Usage (Key Block LMKs)	Description
			<p>but can be used only for data encryption / decryption purposes.</p> <p>The same key is used in both terminal-to-acquirer messages and acquirer-to-terminal messages.</p> <p>This type of BDK does not meet the requirements of ANSI X9.24-1, but is used in some proprietary terminal solutions. Note: this method cannot be used to derive a PIN encryption key, or a data authentication (MAC) key.</p>
BDK Type 4 ("BDK-4")	909	43	<p>This type of BDK is used to secure transactions between a Payment Service Provider (PSP) emulating a DUKPT terminal, and an upstream acquirer.</p> <p>Five types of unidirectional keys may be derived from a BDK-4:</p> <ul style="list-style-type: none"> • PIN encryption keys (for PSP-to-acquirer messages) • Data authentication (MAC) keys (for PSP-to-acquirer messages) • Data authentication (MAC) keys (for acquirer-to-PSP messages) • Data encryption keys (for PSP-to-acquirer messages) • Data encryption keys (for acquirer-to-PSP messages) Different keys are used for PSP-to-acquirer messages and acquirer-to-PSP messages. <p>This type of BDK meets the requirements of ANSI X9.24- 1:2009.</p>
BDK Type 5 ("BDK-5")	-	44	<p>This type of BDK is identical to a BDK-1, apart from the derivation of the IKEY, which uses a proprietary method implementing the Italian Standard Key Derivation Method SPE-DEF-041-112.</p>
IKEY	302	B1	<p>Initial key injected into POS terminal from which future transaction keys are generated.</p>

4.3.11.2 DUKPT Key Management

payShield 10K Host Command	
ID	Command Use
A0	Generate a BDK. Use Mode = 0 where the key is to be used only by the HSM at this time, or Mode = 1 if it is also required to be exported to another device, encrypted under a ZMK. The key type is specified in the command as being the appropriate type of BDK.
A0	Generate an IKEY. Use Mode = A where the key is to be used only by the HSM at this time, or Mode = B if it is also required to be exported to another device, encrypted under a ZMK or TMK. The key type is specified in the command as being the appropriate type of BDK.
A6	Import a BDK encrypted under a ZMK. This command would only be used by the Acquirer if for any reason the BDK was generated by a third party and passed to the Acquirer.
A8	Export a BDK or IKEY encrypted under a ZMK. This command may be used by a third party that generates the BDK/IKEY and passes it to the Acquirer.
BW	Translate a BDK or IKEY from an 'old' LMK to a 'new' LMK.
GK	Export a BDK or IKEY encrypted under an RSA public key. This command may be used by a third party that generates the BDK/IKEY and passes it to the Acquirer.

4.3.11.3 PIN Processing

payShield 10K Host Command	
ID	Command Use
CA	The CA command can translate a PIN Block from encryption under a Terminal PIN Key (TPK) to encryption under DUKPT. This allows non-DUKPT terminals to work with a DUKPT-enabled Acquirer.
G0	Translate a DUKPT PIN Block to encryption under a ZPK. This allows the PIN to be passed into the payments network. The host provides the appropriate BDK-1, BDK-2, BDK-4 or BDK-5 as identified in the KSN sent by the terminal. G0 can also translate PINs between 2 DUKPT schemes – for example, where a DUKPT-enabled service provider exists between DUKPT terminals and a DUKPT-enabled Acquirer.
GO	Verify a PIN Using the IBM Method, with optional MAC verification. The host provides the appropriate BDK-1, BDK-2 or BDK-5 as identified in the KSN sent by the terminal. This command would only be used where the Acquirer is the issuing bank, and the transaction is not being passed through the payments network.
GQ	Verify a PIN Using the Visa PVV Method, with optional MAC verification. The host provides the appropriate BDK-1, BDK-2 or BDK-5 as identified in the KSN sent by the terminal. This command would only be used where the Acquirer is the issuing bank, and the transaction is not being passed through the payments network.
GS	Verify a PIN Using the Diebold Method, with optional MAC verification. The host provides the appropriate BDK-1, BDK-2 or BDK-5 as identified in the KSN sent by the terminal. This command would only be used where the Acquirer is the issuing bank, and the transaction is not being passed through the payments network.
GU	Verify a PIN Using the Encrypted PIN Method, with optional MAC verification. The host provides the appropriate BDK-1, BDK-2 or BDK-5 as identified in the KSN sent by the terminal. This command would only be used where the Acquirer is the issuing bank, and the transaction is not being passed through the payments network.

4.3.11.4 MACing

payShield 10K Host Command	
ID	Command Use
GW	Generate a MAC on a message to be sent to a DUKPT terminal. The host provides the appropriate BDK- 1, BDK-2, or BDK-4 and the KSN for the terminal. Use modes 4-6 (for BDK-1), D-F (for BDK-2), or J-L (for BDK-4): <ul style="list-style-type: none"> Mode 4 or D: Generate an 8-byte MAC Mode 5 or E: Generate Approval MAC (4 leftmost bytes of MAC) Mode 6 or F: Generate Decline MAC (4 rightmost bytes of MAC)
GW	Verify a MAC on a message sent from a DUKPT terminal. The host provides the appropriate BDK-1, BDK- 2, or BDK-4 and the KSN for the terminal. Use modes 1-3 (for BDK-1), A-C (for BDK-2), or G-I (for BDK-4): <ul style="list-style-type: none"> Mode 1 or A: Verify an 8-byte MAC Mode 2 or B: Verify Approval MAC (4 leftmost bytes of MAC) Mode 3 or C: Verify Decline MAC (4 rightmost bytes of MAC)

4.3.11.5 Data Encryption and Decryption

payShield 10K Host Command	
ID	Command Use
M0	Encrypt a block of data to be sent to a DUKPT terminal. The host provides to the HSM the appropriate BDK-1, BDK-2, BDK-3, or BDK-4 and the KSN for the terminal.
M2	Decrypt a block of data sent from a DUKPT terminal. The host provides to the HSM the appropriate BDK- 1, BDK-2, BDK-3, or BDK-4 and the KSN for the terminal.
M4	Translate a block of data from encryption under a BDK-1, BDK-2, or BDK-3 to encryption under a ZEK, DEK, or TEK. The host provides to the HSM the appropriate BDK-1, BDK-2, or BDK-3 and the KSN for the terminal. This allows data from a DUKPT terminal to be passed into the payments network or used for other purposes.
M4	Translate a block of data from encryption under a ZEK, DEK, or TEK to encryption under a BDK-1, BDK- 2, or BDK-3. The host provides to the HSM the appropriate BDK-1, BDK-2, or BDK-3 and the KSN for the terminal. This allows data from payments network or other sources to be passed to a DUKPT terminal.

4.4 German Banking Industry Committee (GBIC)

GBIC (or DK - Die Deutsche Kreditwirtschaft - in German) sets security standards for German domestic card payments. Until 2011, GBIC was known as ZKA - Zentraler Kreditausschuss - and this name is still in widespread use.

Authorized evaluation labs. evaluate complete systems against the security standard and recommend approval by GBIC. Each system has to be individually approved and there is no product certification for HSMs. However, the HSMs used in a system are evaluated as part of the approval process for that system: this evaluation is specific to the use of the HSM in that particular system, and the HSM has to undergo evaluation again when used as part of another system going through the approval process. The payShield 10K has been successfully evaluated in several system approvals by GBIC.

GBIC/ZKA PIN Authentication, Message Authentication, and Data Encryption keys are derived from ZKA Master Keys. The payShield 10K A0 host command supports the generation of these working keys: the required random number input can be provided as one of the A0 command input parameters or generated by the command.

5 RSA and ECC Cryptosystem

This chapter provides information on the functionality provided to support both the RSA and ECC cryptographic algorithms.

RSA functionality has been supported in payShield for a number of years. Support for Elliptic Curve Cryptography (ECC) is introduced for the payShield 10K with the release of software version v1.2a.

Both RSA and ECC are "asymmetric" cryptographic algorithms, where the encryption and decryption keys are different. With both algorithms, it is computationally infeasible to deduce the decryption key from the encryption key. Therefore, the encryption key may be made public and distributed in clear without compromising the security of the decryption key and both algorithms support Public Key Cryptography. In contrast, both the DES and AES cryptographic algorithms are "symmetric" where the keys used by each party to secure a message are the same and therefore must remain secret.

Both RSA and ECC algorithms are usually used in two ways:

- To digitally sign electronic messages to provide proof of the identity of the sender, and to protect the integrity of the contents of the messages.
- To automate and simplify the difficult problem of secret key distribution and management in large distributed networks, such as the Internet.

5.1 RSA Cryptosystem

The RSA public key algorithm was devised in 1979 by Rivest, Shamir and Adleman (hence the name). As noted above, it is an asymmetric cryptographic algorithm, which means that the encryption and decryption keys are different, and that it is computationally infeasible to deduce the decryption key from the encryption key. The RSA algorithm is implemented in all variants of the payShield 10K.

The length of the RSA keys used can be selected from 320 to 4096 bits.

Note: RSA keys longer than 2048 bits can only be used with AES key block LMKs: TDES LMKs have insufficient strength to protect these keys.

5.1.1 RSA Cryptosystem Functions

The RSA cryptosystem provides the following functions:

- Generation of variable-length RSA keys.
- Validation of public key certificates.
- Generation and validation of digital signatures.
- Secure DES key management using RSA public master keys
- Generation of hash values

These functions are implemented by the following host commands:

- Generate an RSA Key Set (EI)
- Load a Secret Key (EK)
- Translate a Secret Key from the Old LMK to a New LMK (EM)
- Generate a MAC on a Public Key (EO)
- Verify a MAC on a Public Key (EQ)
- Validate a Certificate and Generate a MAC on its Public Key (ES)
- Translate a MAC on a Public Key (EU)
- Generate a Signature (EW)
- Validate a Signature (EY)
- Import a DES Key (GI)
- Export a DES Key (GK)

- Hash a Block of Data (GM)

Note: Details of these commands can be found in the *payShield 10K Host Command Reference Manual*.

5.1.2 Even Public Exponents

There is a variant of RSA (known as the "Rabin" variant) which utilizes an even Public Exponent. This variant cannot be used for unique encryption/decryption unless the associated data contains some redundant information which can be used to determine the correct result. Although the commands specified in this document, which use a Public Key, could be used with an even Exponent, there is no guarantee that the results produced by these commands will be correct. It is strongly recommended that the commands in this document are used only with odd Public Exponents. Note that it is not possible to use an HSM to generate an RSA Key Set that has an even Public Exponent.

5.2 ECC Cryptosystem

As noted in the introduction to this chapter, like RSA, Elliptic Curve Cryptography (ECC) is an asymmetric algorithm that supports Public Key cryptography. It is based on a branch of mathematics called elliptic curve and is an alternative technique to RSA.

Some of the advantages over RSA are the smaller key sizes required to provide an equivalent level of security, and an increase in speed of key generation. Therefore, it is considered the next generation implementation of public key cryptography and is becoming more widely used in the payments industry.

5.2.1 Functionality Supported

Functions are provided for:

- ECC Key management:
 - Key pair generation
 - Generation certificate signing request
 - Load public key certificate
 - Translate private and public keys when LMK changed
- ECC Signature Functions:
 - Generate and validate signatures on a message using ECDSA
- ECC Key Derivation Functions:
 - Key Derivation Using Key Agreement: Derives keys using an Elliptic Curve Key Agreement Algorithm (ECKA).
 - Either ECKA-EG (El-Gamal) using ephemeral/static keys
 - Or ECKA-DH (Diffie-Hellman) using ephemeral keys only

5.2.2 Host Commands Supporting ECC

5.2.2.1 ECC Key Management:

- 'FY' – Generate ECC Key Pair (new host command for v1.2a)
The Private key is returned encrypted under the LMK in key block format.
- 'QE' – Generate Certificate Request
Generates a certificate signing request CSR by signing the subject and public key information with the private key to create a self-signed certificate in PKCS#10 format
- 'EO' – Import Public Key Certificate
Imports a Public Key by creating the public key in key block format
- 'EK' – Load Private Key
Load a private key (encrypted under the LMK) into the HSM's tamper-protected memory
- 'EM' – Translate a Private Key
Translate a private key from encryption under an 'old' LMK to encryption under the 'new' LMK
- 'EU' – Translate a Public Key
Translate a public key from protection under an 'old' LMK to protection under the 'new' LMK

5.2.2.2 ECC Signature Functions:

- 'EW' – Generate Signature
Generate a signature on a message using a private key using ECDSA
- 'EY' – Validate Signature
Validate signature on a message using a private key using ECDSA

5.2.2.3 ECC Key Derivation Functions:

- 'IG' – Key Derivation Using Key Agreement.
Derives keys using an Elliptic Curve Key Agreement Algorithm (ECKA). The command supports either ECKA- EG using ephemeral/static keys or ECKA-DH using ephemeral keys only (new host command for v1.2a).

5.2.3 ECC General Information

This section includes some general information on the support for ECC provided in payShield 10K.

It is important to note that for security reasons functions using ECC public and private keys are only available when used with an AES Key Block LMK.

The ECC Prime Curves currently supported for payments are defined in FIPS 186-3 and are as follows:

- '00' – P-256
- '01' – P-384
- '02' – P-521and

ECC functionality is only supported on payShield 10K – it is not available on payShield 9000.

5.3 General Information

5.3.1 HSM Buffer Sizes

The payShield 10K HSM has a 32K-byte input buffer per connection and it is the responsibility of the host application to ensure that the total amount of data sent in an HSM command does not cause a buffer overflow.

5.3.2 Data Formats

Certificates, signatures, encrypted key blocks and message data supplied in commands specified in this document are binary fields, with the leftmost byte being the most significant and the rightmost byte being the least significant. Note that the binary data may be right justified and padded to the left with zeros, if necessary, in order to make the data length (in bits) an exact multiple of eight.

5.4 Common Parameters

Within these functions certain common parameters are defined.

5.4.1 DES Key Type

The DES Key Type field is 4 digits. The first two digits indicate the LMK pair used to encrypt the key, the last two digits indicate the LMK variant.

For example:

- If the DES Key Type is 0600, LMK pair 06-07 is used (no variant).
- If the DES Key Type is 3007, variant 7 of LMK pair 30-31 is used.

5.4.2 Signature Algorithm

01 = RSA

02 = ECC

5.4.3 ECC Curve Reference

Prime Curves defined in FIPS 186-3:

00 = P-256

01 = P-384

02 = P-521

5.4.4 Encryption Identifier

01 = RSA

5.4.5 Hash Identifier

01 = SHA-1, produces a 20 byte result. 02 = MD5, produces a 16 byte result.
03 = ISO 10118-2, produces a 16 byte result.
04 = No hash.
05 = SHA-224
06 = SHA-256
07 = SHA-384
08 = SHA-512

5.4.5.1 01 = SHA-1 hashing algorithm

The ASN.1 DER object identifier for this hashing function is:

{iso(1) identified-organization(3) oiw(14) secsig(3) 2 26}

which encodes as:

2B 0E 03 02 1A

5.4.5.2 02 = MD5 hashing algorithm

The ASN.1 DER object identifier for this hashing function is:

{iso(1) member-body(2) US(840) rsadsi(113549) digest Algorithm(2) 5 }

which encodes as:

2A 86 48 86 F7 0D 02 05

5.4.5.3 03 = ISO 10118-2 hashing algorithm

The ASN.1 DER object identifier for this hashing function is:

2 10 67 4}

which encodes as:

5A 43 04

5.4.5.4 04 = No hash

The no-hash option can be used when an HSM provides signature generation or validation, or certificate validation, on data that is hashed outside an HSM.

If the no-hash option is chosen, the data that is provided in the Validate a Certificate, Generate a Signature and Validate a Signature commands is not modified in any way by an HSM, so it must be precisely the data in the plain signature block (which depends on the pad mode selected by the Pad Mode Identifier). It is the responsibility of the Host application to ensure that the precise data to be included in the signature block is supplied in the command.

5.4.5.5 05 = SHA-224 hashing algorithm

The ASN1.DER object identifier for this hashing function is:

id-SHA224 OBJECT IDENTIFIER ::=

{joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101) csor(3)nistalgorithm(4) hashalgs(2) sha224(4) }

which encodes as:

60 86 48 01 65 03 04 02 04

5.4.5.6 06 = SHA-256 hashing algorithm

The ASN1.DER object identifier for this hashing function is:

id-SHA256 OBJECT IDENTIFIER ::=

{joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) sha256(1) } which encodes as:

60 86 48 01 65 03 04 02 01

5.4.5.7 07 = SHA-384 hashing algorithm

The ASN1.DER object identifier for this hashing function is:

id-SHA384 OBJECT IDENTIFIER ::=

{joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) sha384(2) } which encodes as:

60 86 48 01 65 03 04 02 02

5.4.5.8 08 = SHA-512 hashing algorithm

The ASN1.DER object identifier for this hashing function is:

id-SHA512 OBJECT IDENTIFIER ::=

{joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) sha512(3) } which encodes as:

60 86 48 01 65 03 04 02 03

Example:

If the SHA-1 algorithm is used to hash the data and the resultant hash value is:

0123456789ABCDEF0123456789ABCDEF01234567

and if the PKCS#1 pad mode is used, the data to be provided must be the complete ASN.1 DER encoded DigestInfo, which is:

30 21 300906052B0E03021A0500 04140123456789ABCDEF0123456789ABCDEF01234567.

Note that when using the no-hash mode, the HSM checks that the DER encoded DigestInfo syntax is correct. If there is a digest info syntax error, the HSM returns error code 74.

5.4.6 Pad Mode Identifier

01 = PKCS#1 v1.5

02 = OAEP

04 = PKCS#1 v2.1 method (EMSA-PSS)

Note for host command QE only: 04 = RSASSA-PSS

The PKCS#1 standard (see Reference 3 at the beginning of this manual) defines the padding method to be used before operating with a public or secret RSA key.

5.4.6.1 01 = PKCS#1 v1.5

This simple padding scheme was introduced in the original PKCS#1 specification. The data to be encrypted or decrypted is padded as follows:

00 BT PS 00 D

where:

- BT is a single byte indicating the block type. BT is 01 for a secret key operation; 02 for a public key operation.
- PS is a padding string consisting of bytes FFFF for block type 01, random non-zero bytes for block type 02. PS must contain at least 8 bytes.
- D is the data block.
- The total length of the padded block is equal to the length (in bytes) of the RSA key modulus

The data block D is the ASN.1 encoded message digest, or AES/DES key (depending on the command used), as follows:

DigestInfo ::	SEQUENCE {
digestAlgorithm	DigestAlgorithmIdentifier,
digest	OCTET STRING
}	
DigestAlgorithmIdentifier ::	SEQUENCE { algorithm
	OBJECT IDENTIFIER,
parameters	NULL
}	
Key block ::	SEQUENCE {
key	OCTET STRING,
iv	OCTET STRING
}	

Example 1:

Assume that the SHA-1 algorithm is used to produce the 20-byte digest:

0123456789ABCDEF0123456789ABCDEF01234567.

The DigestAlgorithmIdentifier for SHA-1 is:

30 09 06 05 2B0E03021A 05 00.

Thus, the ASN.1 DER encoded DigestInfo is:

30 21 300906052B0E03021A0500

04140123456789ABCDEF0123456789ABCDEF01234567

Example 2:

If a single-length DES key 0123456789ABCDEF and IV = 9999999999999999 are used, the ASN.1 DER encoding of Key block is:

30 14 04080123456789ABCDEF 04089999999999999999.

When the PKCS#1 pad mode is used, the following validity checks are carried out: For a validation operation (Validate a Certificate, Validate a Signature):

- The length of the data to be validated is equal to the length (in bytes) of the modulus of the key to be used for the validation. If not, error code 76 is returned.
- The first byte of the clear data block is 00. If not, error code 77 is returned.
- The second byte of the clear data block is 01. If not, error code 77 is returned.
- Subsequent bytes consist of at least 8 bytes of binary 1s, followed by a zero byte. If not, error code 77 is returned.
- The hash algorithm object identifier corresponds to that of the identifier of the hash algorithm supplied in the command message. If not, error code 79 is returned.
- The digest is compared with the hash of the supplied data. If the two values are not equal, error code 02 is returned

For a generation operation (Generate a Signature):

- The length (in bytes) of the data block D is at most m-11 (where m is the length, in bytes, of the modulus of the key to be used). If not, error code 76 is returned.

For an import key operation (Import a DES Key):

- The length of the imported key block is equal to the length (in bytes) of the modulus of the secret key to be used to decrypt the block. If not, error code 76 is returned.
- The first byte of the clear data block is 00 and the second byte is 02. If not, error code 77 is returned.
- Subsequent bytes consist of at least 8 bytes of random non-zero bytes, followed by a zero byte. If not, error code 77 is returned.
- The data block D conforms to the ASN.1 encoding rules. If not, error code 77 is returned.

For an export key operation (Export a DES Key):

- The length (in bytes) of the data block D is at most m-11 (where m is the length, in bytes, of the modulus of the key to be used). If not, error code 76 is returned.

5.4.6.2 02 = OAEP

Optimal Asymmetric Encryption Padding (OAEP) was introduced in PKCS#1 v2.0, as an improvement on the original, simple PKCS#1 v 1.5 method described above. OAEP requires four additional parameters:

- Mask Generation Function
01 = MGF
- MGF Hash Function
01 = SHA1
- OAEP Encoding Parameters Length
Specifies the length of the encoding parameters.
- OAEP Encoding Parameters
The host may optionally supply a set of OAEP encoding parameters. If OAEP padding is used, but no Encoding Parameters are required, then OAEP Encoding Parameters Length should be "00", and this field will be empty.

The OAEP fields are encoded according to PKCS#1 version 2.2 (see Reference 3 at the beginning of this manual).

The HSM does not interpret or validate the contents of this field, it applies the Hash Algorithm to it and feeds the result into the OAEP process.

5.4.7 Key Block Type

The Key Block Type field is used with the host commands to import and export keys encrypted under an RSA public key. The values supported are:

- 01 = Standard Key Block Type
- 02 = Key Block Template
- 03 = Unformatted Key Block
- 04 = ASN.1 Encoded Key Block.

Key Block Types '01', '02', and '03' may be used for importing and exporting DES/AES keys. Key Block Types '02', '03', and '04' may be used for importing and exporting HMAC keys.

5.4.7.1 01 = Standard Key Block Type

This is the standard key block format supported in earlier versions of payShield. The format is as shown in the PKCS#1 v1.5 padding scheme above, i.e.:

Key block ::	SEQUENCE
{ key	OCTET STRING,
iv	octet string
}	

Note: For a DES/3DES key, the 'key' may be 8, 16 or 24 bytes, and the 'iv' is 8 bytes; for an AES key, the 'key' may be 16, 24 or 32 bytes, and the 'iv' is 16 bytes.

5.4.7.2 02 = Key Block Template

This method supports any valid ASN.1 DER encoded Key Block format, which may consist of arbitrary encoded data with a Key Block field containing a plain-text DES Key of single, double or triple length.

The Host must supply a block of data, which conforms to ASN.1 DER encoding, with an indication of the position in which the key is located (DES Key Offset). The key data area of the template must be zero filled.

For key export, an HSM overlays the zero filled data with a DES or Triple DES key as appropriate.

For key import, an HSM verifies that the decrypted data conforms to the specified padding, then checks that the supplied template matches the decoded data. It then extracts the data at the position indicated by the DES Key Offset and use this as the key for import.

An example Key Block structure and template is shown below. This structure is used for Diebold Remote Key Transport.

Example Key Block Structure

RecipientInfo ::=	SEQUENCE
{ version	Version,
issuerAndSerialNumber	IssuerAndSerialNumber,
keyEncryptionAlgorithm	KeyEncryptionAlgorithmIdentifier, keyOrKey
block	KeyOrKey block
}	
KeyOrKey block ::=	CHOICE {
encryptedKey	EncryptedKey,
EncryptedKey block	encryptedKey block
}	
EncryptedKey ::=	OCTET STRING
EncryptedKey block ::=	ENCRYPTED Key block – a BIT STRING
Key block ::=	SEQUENCE {
version	Version, -- 0
originatorIssuerAndSerialNumber	IssuerAndSerialNumber,
keyId	KeyId,
key	Key,
keyUsage [0]	KeyUsage OPTIONAL
}	

Example Key Block Template

A key block template corresponding to the above structure is shown below:

30 61	Key block
02 01 00	version = 0
30 47	originatorIssuerAndSerialNumber
30 42	issuer
31 10	
30 0E	
06 03 55 04 03	attributeType = commonName
13 07 52 6F 6F 74 20 43 41	attributeValue = "Root CA"
31 2E	
30 2C	
06 03 55 04 0A	attributeType = organizationName
13 25	attributeValue = "Initial Certificate Authority Company"
49 6E 69 74 69 61 6C 20 43 65 72 74 69	
66 69 63 61 74 65 20 41 75 74 68 6F 72	
69 74 79 20 43 6F 6D 70 61 6E 79	
02 01 02	serialNumber = 2
02 01 00	keyIdentifier = 0, A key
04 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	key

The Key Block Template requires four additional parameters:

- **Key Block Template Length**
The length of the key block data
- **Key Block Template**
The actual template, as shown in the example above
- **DES Key Length**
The length of the DES key within the key block.
- **DES Key Offset**
Offset to the location of the DES key within the key block.
In the example above this points to the beginning of the block of zeros shown in bold italics and the offset is 83 (decimal) bytes.

Another two optional parameters support a check value. The Check Value is not required for the Diebold implementation, but provides flexibility to support applications that require a check value in the key block.

- **Check Value Length**
Length in bytes of the check value field. This field should be 0 if no check value is used.

- Check Value Offset
Offset to the location of the check value within the key block.

5.4.7.3 03 = Unformatted Key Block

This is the format required for remote ATM key loading for NCR ATMs. It consists of only 8, 16 or 24 bytes of key data (for a single, double or triple length DES key), with no encoding or additional information.

5.4.8 Public Key Encoding

- 01 = DER encoding for ASN.1 public key (INTEGER uses unsigned representation)
- 02 = DER encoding for ASN.1 public key (INTEGER uses 2's complement representation)
- 03 = DER encoding for ASN.1 ECC public key in X9.62 format
- 04 = Uncompressed encoding 05 = ISO 7816 format
- 06 = Public key in key block format

An ASN.1 RSAPublicKey has the following definition:

RSAPublicKey ::=	SEQUENCE {
modulus	INTEGER, -- n
publicExponent	INTEGER -- e
	}

payShield 10K HSM base software supports two different representations for INTEGER values in the RSAPublicKey- unsigned INTEGER (Public Key Encoding 01) and 2's complement INTEGER (Public Key Encoding 02).

A public key Modulus represented in 2's complement form will always have a leading 00 byte, the most significant bit of the second byte will always be '1'. A public key modulus represented in unsigned form will never begin with a 00 byte, the most significant bit of the modulus will always be '1'.

To avoid interoperability problems with non-Thales host security modules, it is recommended to use Public Key Encoding 02.

Examples:

1024 bit modulus with an exponent of 03 using Public Key Encoding 01:

Sequence Identifier	Byte Length	Integer Identifier	Modulus Length	Modulus	Integer Identifier	Exponent Length	Exponent
X'30	X'81 X'86	X'02	X'81 X'80	128	X'02	X'01	X'03

Where:

- X'30 is the identifier specifying the start of a sequence.
- X'81 X'86 specifies the length of the following field in bytes:
If value is between X'01 and X'7F then this directly specifies length of following field in bytes (1byte to 127 bytes).
If value is greater than X'80 it defines the number of bytes to define the length of the next field in the above example X'81 therefore length i.e. 1 byte (X'86 - 134 bytes).
- X'02 is the identifier specifying the start of the integer.
- X'81 X'80 specifies the length of the following field in bytes using the same definition as above (128 Bytes).
- The modulus in this example is 1024 bits. For Public Key Encoding 01 this is represented in unsigned form. The resulting field is 128 bytes.
- X'02 is the identifier specifying the start of the second integer.

- X'01 specifies the length of the following field in bytes using the same definition as above (1 Byte).
- X'03 is the value of the exponent.

1024 bit modulus with an exponent of 03 using Public Key Encoding 02:

Sequence Identifier	Byte Length	Integer Identifier	Modulus Length	Modulus	Integer Identifier	Exponent Length	Exponent
X'30	X'81 X'87	X'02	X'81 X'81	129	X'02	X'01	X'03

- X'30 is the identifier specifying the start of a sequence.
- X'81 X'87 specifies the length of the following field in bytes. Using the same definition as described in example above, X'87 - 135 bytes.
- X'02 is the identifier specifying the start of the integer.
- X'81 X'81 specifies the length of the following field in bytes using the same definition as above (129 Bytes).
- The modulus in this example is 1024 bits. For Public Key Encoding 02 this is represented in 2's complement form. The resulting field is 129 bytes. The first byte will always be X'00.
- X'02 is the identifier specifying the start of the second integer.
- X'01 specifies the length of the following field in bytes using the same definition as above (1 Byte).
- X'03 is the value of the exponent.

5.4.9 Signature Format & Signature Output Format (ECC Only)

- 0 = Plain format
- 1 = X9.62 ASN 1 encoded

5.5 Worked Examples

This section demonstrates the typical use of the GK (Export Key under RSA Public Key) host command, whereby a ZPK is securely transported from the local system (an HSM) to a trusted external system (X).



Both systems must previously have generated RSA key pairs. Additionally, both systems must trust each other's public keys. For the local system, this can be achieved using the EO (Generate a MAC on a Public Key) command, which requires Authorized State. How the external system achieves this is outside the scope of this example.

The ZPK is encrypted using X's public key, positioned within a block containing additional data, and then signed using an HSM's private key.

The examples provide values for all input and (anticipated) output parameters to the GK (Export Key under RSA Public Key) command when using both 1024-bit and 2048-bit RSA keys.

The first section defines a number of values that are used in producing the encrypted & signed ZPK (using 1024-bit RSA keys).

The second section Sample Data Generation Procedure (1024-bit RSA keys) shows the intermediate steps to producing the encrypted & signed ZPK (using 1024-bit RSA keys).

The third section Sample Data Calculation (1024-bit RSA keys) consists of a complete command message / response message sequence for this example (using 1024-bit RSA keys).

The fourth section Sample Data Definitions (2048-bit RSA keys) defines a number of values that are used in producing the encrypted & signed ZPK (using 2048-bit RSA keys).

The fifth section Sample Data Generation Procedure (2048-bit RSA keys) shows the intermediate steps to producing the encrypted & signed ZPK (using 2048-bit RSA keys).

The sixth and final section Sample Data Calculation (2048-bit RSA keys) consists of a complete command message / response message sequence for this example (using 2048-bit RSA keys).

5.5.1 Sample Data Definitions (1024-bit RSA keys)

```
DES KEY TO EXPORT = 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C
DES KEY TYPE = ZPK
DES KEY CHECK VALUE = 8B 15 36 D4 D6 0F EC 19
LMK ENCRYPTED DES KEY (AS ZPK) IN KEY SCHEME U = U 51 3E CF 7E 7D 0A 55 54 90 0F A7 09 B9 A5 F7 21 ENCRYPTION
PADDING MODE = OAEP (MGF1, SHA-1, OAEP PARAMS = "09 08 07 06")
ENCRYPTION PUBLIC KEY (ASN.1 DER ENCODED) =
0000: 30 81 89 02 81 81 00 40 CE 04 F8 5B 70 30 49 C8 0%.?@î.ø[p0IÈ
0010: 48 6C 84 C8 EB 40 54 EC B2 2A FD 6B 78 96 7D A2 H1,,Èë@Ti²*ýkx-}¢
0020: D8 CD F7 5E 5E B9 63 94 95 43 C6 8F 54 31 88 11 Øí÷^^¹c"•CET1^.
0030: B3 A9 91 27 A8 C1 D0 9C B1 3C CD 70 05 80 8E 91 ³©''"ÁÐæ±<Íp.eŽ`
0040: 80 80 B0 AF 5A 58 C7 11 8B 44 3F C7 CD AE E4 D5 ee°~ZXÇ.<D?ÇÍ@äÖ
0050: A7 F6 3C C0 F3 59 6C 98 E3 7B 9F 97 9D 53 C4 4B Sö<ÀóYl~ä{ÿ-SÄK
0060: E1 0E C4 06 8F DD 7A 38 24 BD 20 34 F0 E5 EA 19 á.Ä.Ýz8$½ 4ðâê.
0070: 8E FF 9B 36 B0 EF 65 90 BF D0 50 99 E2 8A E0 4D Žÿ>6°iežDPmãŠam
0080: 09 96 D2 E1 36 21 E9 02 03 01 00 01 .-Öá6!é.....
ENCRYPTION PUBLIC KEY MAC = 0E FA 9C 3A
ENCRYPTION PUBLIC KEY AUTHENTICATION DATA = "ABCDEFGH" ENCRYPTION PRIVATE KEY (D) =
0000: 03 36 B4 44 64 B4 71 90 97 20 10 51 9D 6D 1D 29 .6´Dd´q¿ .Qñ .)
0010: 98 FB 54 EA 70 53 F0 92 96 6A CD FC 00 70 0E 1D ~ûTêpSð´-jÍù.p..
0020: 84 16 CA DF A3 E7 F6 F4 DA 7B E0 62 D4 66 A8 05 „.Êß£çöðÚ{àbÔf".
0030: E2 5F 5F B6 88 61 9D 78 74 7A BC E7 06 2B 22 CF â_____ſ^ažtz¼ç.+"İ
0040: DF A6 7B 5B A5 4D 72 BD E4 69 10 03 60 D2 06 37 ß|{[¥Mr½äi..`Ò.7
0050: EC E7 09 A6 19 0C BF 33 D3 CE C7 F9 89 94 31 41 ìç.!.:3ÓÎÇù%¹1A
0060: 42 11 2F 5E 48 FA 05 C6 70 22 D6 05 7E 6D 10 78 B./^Hú.Æp"Ö.~m.x
0070: 78 BA 9C D4 F2 A2 63 9B A8 71 A4 4D F7 EE 8A 21 x°æÖðçc>`q=M÷îŠ! SIGNATURE PUBLIC KEY (ASN.1 DER
ENCODED) =
0000: 30 81 89 02 81 81 00 40 85 8A 70 C8 7F 3E AB 9E 0%.?@...ŠpÈ?>«ž
0010: 13 6D 2C 0C A3 B6 CC 47 ED 68 6E 3C 6F 31 46 C5 .m,.£ſİGíhn<olFÄ

0020: FE 05 64 3B 4F EE F9 B6 9A 1C ED 6A EB D3 B9 15 p.d;Oiùſš.ijëÓ¹.
0030: 31 3C 8C D1 5C BF 26 FB AB D4 8C 6E 08 2A D0 F0 1<EN\ç&û«ÖEn.*Ðð
0040: D5 FD 03 64 56 B6 CE A5 91 DF B7 F5 A4 30 B2 6B Öý.dvſſîſ'ß.ð¼0²k
0050: EA 3A 8C E4 15 2C DC 50 DE AF 9D DF EF D8 AC 10 è:Èä.,ÛPB~BîØ-.
0060: DB FF 2B 92 F1 97 C7 D6 D0 CF BC 1A 6D 85 06 CB Ūý+'ñ-ÇÖĐİ¼.m....È
0070: F8 1B 76 F1 5D 32 AC D0 72 ED 34 72 30 A9 24 F0 ø.vñ]2-Đrî4r0©$ð
0080: 6D B2 E3 8D 55 33 33 02 03 01 00 01 m²ãŨ 33.....
SIGNATURE PRIVATE KEY (D) =
```

```

0000: 01 9F 3E 65 05 75 57 C0 B5 08 7F B4 D3 EE 3B 6B .ÿ>e.uWÀµ.?'Óî;k
0010: 39 4C 42 79 B1 D7 89 33 BD DA C9 B1 E9 3C 62 33 9LBy±×%3¼ÚĖ±é<b3
0020: C3 7B 00 A6 4F E6 87 45 15 76 4E 6A 62 26 2D C0 Ā{.|Oæ†E.vNjb&-Ā
0030: 9D 90 86 72 AF 9E 9A 5F 3D 7A 2A 92 53 66 B8 F6 ?r~žš_ =z*'Sf,ö
0040: 0D A9 89 E5 24 1D 38 B8 1B 01 0D 55 A6 59 C4 E5 .@%â$.8,. U|YĀĀ
0050: 53 2F 2A A9 53 CF 68 CF 20 AE 4B CD 9E 26 F1 60 S/*@Sĩhĩ @Kíž&ñ`
0060: 05 4A 58 29 7A C8 43 7F D1 46 C8 E9 E0 66 EE FF .JX)zĖC?ÑFĖéàfiÿ
0070: 64 93 6C 3D 12 CF 3B 78 42 3D 39 77 3E EF 42 E9 d"l=.ĩ;xB=9w>iBé SIGNATURE PRIVATE KEY (ENCRYPTED UNDER
LMK) =
0000: B9 02 C5 03 07 7E E3 DF FF 12 DC 4F 35 A9 96 22 ¹.Ā..~āßÿ.Ü05©-"
0010: 11 58 A5 59 72 A7 CB 93 0C 39 93 C9 FD 70 07 6D .XŸYr$Ė".9"Eýp.m
0020: F6 3D 62 C1 BF 00 65 86 42 84 72 9B DE AD 89 8A ö=bĀ¿.e†B,,r>P-%Š
0030: 3F 58 94 CE B2 7E 66 E5 15 D7 E1 6A 91 6F F1 96 ?X"Î²~fâ.×áj'oñ-
0040: 89 BF AC 5F 97 54 B1 EF 4A 36 2A 1A 53 6F 3F 82 %¿_—T±iJ6*.So?,
0050: AA 31 F9 F7 1A 95 EC E9 DD 70 76 CE 7E E5 1C B4 *lù÷.•iéŸpvĩ~â.´
0060: 70 FE A8 A8 6D CC E8 25 E8 6E 4E D7 7A 0A 71 22 pb""mĭè%ènN×z.q"
0070: 25 5D 11 1C D2 DA EE DA FC 5A 92 93 39 39 2C 77 %]..ÔÚiÚúZ'"99,w
0080: 93 A2 C9 47 ED 0A F1 7C 4F 15 A6 0F C7 F4 36 E4 "¢ĖGí.ñ|O.|.Çô6ă
0090: 60 D0 38 4F 5F B2 43 3C 01 13 57 44 9A BA 8D 94 `Ð8O_²C<..WDš°?
00A0: 95 98 E0 A5 EE D3 8C D8 8F 29 93 AF 62 E0 0B E2 •~àŸiÓĖøŸ""bà.â
00B0: 12 B7 07 76 05 BC BB 0D D2 EB 87 B1 DC D8 B8 12 ..v.¼»».Ôë±±Ůø,.
00C0: 26 B3 EA 58 58 D1 1F 3B C1 66 DC 75 68 7C EF 64 &³éXXN̂.;ÁfŮuh|iđ
00D0: 4D 46 61 F8 C3 74 AC 76 B4 42 D1 9B D7 D4 63 DC MFaøĀt~v'BN̂>×ÔcŮ
00E0: C2 A0 E7 CA 7C 4E 7A 09 57 DA FA 3E A6 C8 50 4E Ā çĖ|Nz.WŮú>|ĖPN
00F0: BC 49 37 97 C1 89 67 26 07 2C 3C 1C 1B 8A 53 A4 ¼I7-Ā%g&.,<..ŠSµ
0100: AB F1 63 F1 9D 9E ED 59 EF 62 E3 75 22 13 BD 46 «ñcñžíYibău".¼F
0110: 10 BC ED CE 38 78 03 72 F5 D1 2D 1C DF 62 42 73 .¼iî8x.rõÑ-.ßbBs
0120: 52 53 2C 67 84 AE 7B A7 3B B3 AD 1C 33 EE 6E E4 RS,g,,@{§;³-.3înä
0130: A0 F2 62 AE 82 DE F1 80 7A 69 8C 68 27 FA 4A 45 òb@,ĐñĖziĖh'úJE
0140: 73 E2 B4 7C 12 13 9C D3 BC 23 72 9A C3 42 91 20 sâ´|.̂œÓ¼#ršĀB`
0150: 8B 1B 9A F3 1F 7D 37 9D <.šó.}??

```

5.5.2 Sample Data eneration Procedure (1024-bit RSA keys)

1. DER encode ASN.1 format DES Key

```

30 12
03 10
7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C

```

2. Pad #1 using PKCS#1 OAEP

```

0000: 00 D8 BF 0E 30 FB 8A E3 98 9E 6D 47 CE A8 05 A9 .Ø¿.0úšă~žmGÎ".©
0010: 22 D9 19 4A 2C 3B AC 74 11 CA DE CD C8 7F 78 A6 "Û.Ÿ,;-t.ĖPÍĖ?x|
0020: 64 E5 DC 33 CE 37 72 85 A7 62 7A 37 FE C8 8E 8C dăŮ3î7r...$bz7pĖžĖ
0030: 21 21 E0 F9 64 73 24 9C B2 07 7A A3 60 B4 DB ED !!àùds$œ².zſ`´Ůi
0040: 52 EF 02 89 3B C8 E9 A9 C3 E6 E8 AB 29 EC 20 D1 Rĭ.%;Ėé@Āæè«)ì Ñ
0050: F0 CA 0B AF 7D 6A D3 C7 90 D3 F3 73 75 63 AC A6 ôĖ.~}jÓÇ?Óósuc~|
0060: D4 D4 91 FA D8 5F 23 6B 7C 2A 07 E1 09 C9 5D AC ÔÔ`úø_#k|*.á.Ė]~
0070: 16 2C C9 70 12 B0 65 F0 40 A1 02 A2 8A 6A 4C 45 .,Ėp.°eð@;_¢šjLE

```

3. Encrypt #2 using Encryption Public Key

```

0000: 0F 7B 55 85 C7 26 FB 5F 81 45 9F CC 7C 2A CD FC .{U...Ç&û_?EŸÎ|*íü
0010: B2 D4 62 8F CB 86 00 5C 62 47 F9 25 B0 7F FF 05 ²Ôb?Ė†.\bGù%°?ÿ.
0020: 2F D0 03 7A B6 8C 24 DF 98 BB 65 78 D3 40 1B 8B /Ð.zŸĖ$ß~»exÓ@.<
0030: 19 17 9F 5E 92 32 BB D0 87 C5 5F 52 A3 BF 3F E8 ..Ÿ^'2»Ð†Ā_Rſ¿?è
0040: 81 6A 75 C7 19 2D 5F 0D 74 39 23 DA 81 0B 96 FC ?juÇ._.t9#Ů.?-ü
0050: 43 B8 55 B2 FB 60 F3 E8 B3 45 3E 89 43 4E 40 17 C_Ů²û`óè³E>%CN@.
0060: DC 03 66 C8 8D C6 9E 17 A4 C5 89 54 0B 91 11 9A Ů.fĖ?Ėž.¼Ā%T.`.š

```


5.5.3 Sample Data Calculation (1024-bit RSA keys)

Field	Length & Type	Value
COMMAND MESSAGE		
Message Header	m A	
Command Code	2 A	"GK" – Export Key under RSA Public Key
Encryption Identifier	2 N	"01" – RSA Encryption
Pad Mode Identifier	2 N	"02" – OAEP (EME-OAEP-ENCODE)
Mask Generation Function	2 N	"01" – MGF1 as defined in PKCS#1 v2.2
MGF Hash Function	2 N	"01" – SHA-1
OAEP Encoding Parameters Length	2 N	"04"
OAEP Encoding Parameters	N B	"09080706"
OAEP Encoding Parameters Delimiter	1 A	","
DES Key Type	4 N	"0600"
Signature Indicator	1 A	"="
Signature Hash Identifier	2 N	"01" – SHA-1
Signature Identifier	2 N	"01" – RSA
Signature Pad Mode Identifier	2 N	"01" – PKCS#1 v1.5 padding
Header Data Block Length	4 N	"0012"
Header Data Block	n B	"YYYYYYYYYYYY"
Delimiter	1 A	","
Footer Data Block Length	4 N	"0015"
Footer Data Block	n B	"ZZZZZZZZZZZZZZ"
Delimiter	1 A	","
Private Key Flag	2 N	"99"
Private Key Length	4 N	"0344"
Private Key	n B	b902c503077ee3dfff12dc4f35a996221158a55972a7cb930c 3993c9fd70076df63d62c1bf0065864284729bdead898a3f58 94ceb27e66e515d7e16a916ff19689bfac5f9754b1ef4a362a1 a536f3f82aa31f9f71a95ece9dd7076ce7ee51cb470fea8a86d cce825e86e4ed77a0a7122255d111cd2daeedafc5a9293393 92c7793a2c947ed0af17c4f15a60fc7f436e460d0384f5fb243 3c011357449aba8d949598e0a5eed38cd88f2993af62e00be 212b7077605bcb0dd2eb87b1dcd8b81226b3ea5858d11f3b c166dc75687cef644d4661f8c374ac76b442d19bd7d463dcc2 a0e7ca7c4e7a0957dafa3ea6c8504ebc493797c1896726072 c3c1c1b8a53a4abf163f19d9eed59ef62e3752213bd4610bce dce38780372f5d12d1cdf62427352532c6784ae7ba73bb3ad 1c33ee6ee4a0f262ae82def1807a698c6827fa4a4573e2b47c 12139cd3bc23729ac34291208b1b9af31f7d379d
DES Key Flag	1 N	"1"
DES Key (LMK)	1A+32H	"U513ECF7E7D0A5554900FA709B9A5F721"
Check Value	16 H	"8B1536D4D60FEC19"
MAC	4 B	0EFA9C3A
Public Key	n B	3081890281810040CE04F85B703049C8486C84C8EB4054 ECB22AFD6B78967DA2D8CDF75E5EB963949543C68F54 318811B3A99127A8C1D09CB13CCD7005808E918080B0A F5A58C7118B443FC7CDAEE4D5A7F63CC0F3596C98E37 B9F979D53C44BE10EC4068FDD7A3824BD2034F0E5EA1 98EFF9B36B0EF6590BFD05099E28AE04D0996D2E13621 E90203010001
Authentication Data	n A	"ABCDEFGH"
Delimiter	1 A	","

5.5.4 Sample Data Definitions (2048-bit RSA keys)

DES Key to export = 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C

DES Key Type = ZPK

DES Key Check Value = 8B 15 36 D4 D6 0F EC 19

LMK Encrypted DES Key (as ZPK) in key scheme U = U 51 3E CF 7E 7D 0A 55 54 90 0F A7 09 B9 A5 F7 21

Encryption Padding mode = OAEP (MGF1, SHA-1, OAEP Params = "09 08 07 06")

Encryption Public Key (ASN.1 DER encoded) =

```
0000: 30 82 01 0a 02 82 01 01 00 40 f5 b5 3a 32 47 50 0, ..., ...@õµ:2GP
0010: 90 ad de 8a 33 ef 6f 9f 95 ee 58 41 84 f1 fe 7d  -PŠ3ioŸ•îXA,,ñp}
0020: 31 ae 58 c6 ca e2 a0 62 da b0 39 e7 45 03 9f 7a 1@XÆÊâ bú°9çE.Ÿz
0030: a4 3a 23 6a 45 40 ad 0a 3f 36 1f 90 5f 29 b1 a0 ¼:#jE@-.?6.□_)±
0040: e9 c1 65 b0 a6 56 d9 f8 18 c5 7e 05 d7 87 f1 f8 éÁe°|VÜø.Å~.x+ñø
0050: ee 0c 6b e9 b4 2a 83 2c fd bd 35 74 a5 e7 ef 86 î.ké´*f,ý½5t¥çit
0060: 3b 11 ff 3f 95 dc ac bb dc fd d9 0e a9 c7 d5 52 ;.ÿ?•Ü-»ÜýÜ.©ÇÖR
0070: cf 0d 1b e4 71 53 2c f6 4e 02 9c cd d7 d9 ae c9 Ĩ..äqS,öN.æÍ×ÜÖÉ
0080: 2f 90 c3 9d 1b e9 63 ae e6 f5 78 e6 a3 d8 3a c6 /□Ä□.éc@æöxæfØ;Æ
0090: 57 2e b5 52 5d 0a 81 79 bc ba 02 63 d8 2e bb 5d W.µR].□y¼°.cØ.»]
00a0: 77 c3 35 56 16 06 b3 27 01 75 3a ed c4 2a aa e1 wÃ5V...³'.u:íÄ*ªá
00b0: 20 d9 23 3c b3 b9 ef 4d de 0c 7b 07 02 25 3e 25 Û#<³¹iMÐ.{..%>%
00c0: 5c 1e de 05 2d 9c 4f 7a 6c 7a fb f4 e4 08 bb 18 \.Ð.-æOz1zûôä.».
00d0: d9 53 54 c4 44 62 ff 35 2b ff 31 1e 8e 07 a3 83 ÛSTÄDbÿ5+ÿ1.Ž.£f
00e0: 49 a7 88 08 16 4a 24 a0 8f d9 d3 54 9b c3 3d 58 I$^..J$ □ÜÓT>Ã=X
00f0: de c5 25 e9 7f fb fb f8 a6 d4 12 05 12 0a e8 d4 ÞÃ%é□ûûø!Ô....èÖ
0100: ac 75 6f 3e 85 80 91 8d cb 02 03 01 00 01 -uo>...€'□Ë.....
```

Encryption Public Key MAC = aa 5e 41 cc

Encryption Public Key Authentication Data = "ABCDEFGF"

Encryption Private Key (d) =

```
0000: 07 b7 88 c7 78 9b 3c 0c c3 ae a4 5b d1 e5 61 .·^Çx~><.Ã@µ[Nãaa
0010: f0 d6 20 36 74 6f 18 9d 51 ca 6f 17 44 13 ec 9a ðÖ 6to.□QÊo.D.iš
0020: 71 2b f7 ca ed 92 c1 05 88 78 93 93 d5 8a 98 f8 q+÷Êí'Á.^x""ÖŠ~ø
0030: 88 6b f8 81 2d 99 51 f5 e3 09 3b 12 8f a7 c6 3e ^kø□-™Qöä.;.□$E>
0040: df 1b 49 03 61 3d 80 26 7b 68 48 73 a4 47 40 0d ß.I.a=€&{hHsmG@.
0050: 86 b0 36 82 cd 0a 59 e6 63 8d 70 96 d3 87 db ab †°6,Í.Yæc□p-Ó+Û«
0060: 75 a6 97 04 d9 5e 00 bf e3 1d 48 a6 a3 cc 68 18 u|-.Ü^.¿ä.H|£Ïh.
0070: 3d 5c 36 61 e9 94 c7 86 b4 8a 60 7c 23 de 39 35 =\6ae"Ç†'Š'|#Þ95
0080: 4d 19 5d da 82 71 06 4f c7 73 82 a0 f9 ab 9f 10 M.]Ú,q.Oçs, ù«Ÿ.
0090: 25 3b 43 74 b6 5b 29 79 0c 34 c2 ff 82 2c d9 62 %;Ct¶[]y.4Ãÿ,,Üb
00a0: 45 1b 66 0e 43 67 56 a1 e1 9e 04 e8 d1 fd 23 72 E.f.CgV;áž.èÑý#r
00b0: 4d 81 1d 61 54 22 95 2b 66 05 a5 a2 cd 20 6f 74 M□.aT"•+f.¥çÍ ot
00c0: 18 18 fa e4 bd 0f 6e d9 24 64 f6 a7 c6 16 5e 89 ..úä½.nÜ$ðö$Æ.^%
00d0: 46 85 00 12 9f 45 50 ca 6f c9 a9 3b ac e4 2f 39 F...ŸEPÊoÉ©;-ä/9
00e0: ba b7 63 7e 8b bb 97 4c d7 a3 90 c8 39 62 85 ae °·c~<»-L×£□Ë9b...©
00f0: b1 ac 82 e0 70 f8 84 05 53 99 f6 7e 33 ea 76 f1 ±¬,àpø,,.S™ö~3êvñ
```

Signature Public Key (ASN.1 DER encoded) =

```
0000: 30 82 01 0a 02 82 01 01 00 40 bd c2 92 8d cf a2 0, ..., ...@½Ä'□İç
0010: 96 a7 b3 f2 fe 3e 78 c7 24 21 f3 b2 89 a0 ca 5d -Š³òp>xÇ$!ó²% Ê]
0020: 6e e5 9f 4f 14 76 59 95 64 32 d2 30 31 42 8d 9b nãŸO.vY•d2Ò01B□>
0030: 83 99 99 1a db e3 b7 10 17 4c 12 87 1f 53 49 05 f™™.Üä...L.‡.SI.
0040: 82 46 f3 94 22 32 d2 2f 02 b5 af d3 c0 e9 1a 28 ,Fó""2ò/.µ¯ÓÀé.(
0050: 51 13 96 3a 66 05 98 d9 00 40 e8 cb 48 71 c4 48 Q.-:f.~Ü.©èEHqÄH
0060: ba fa 17 46 38 db 3c 35 ba f5 ca 5d c5 6d b9 3d °ú.F8Û<5°öÊ]Åm¹=
```



```

0070: e2 5a c2 a5 c3 d7 49 1f 5b a7 02 98 f0 42 3a de âZÂÿÃ×I.[$.~ðB:þ
0080: 55 06 9b 0b 40 b4 38 78 e0 55 76 b0 8f 71 a8 ee U.>.@´8xàUv°□q`i
0090: 10 b1 f7 52 4d a7 df 52 c4 4b 23 0e 31 e0 b1 e4 .±÷RMSßRÄK#.1à±ä
00a0: b5 a1 e4 b4 7c 94 e9 e5 30 f6 db 57 70 7e 53 5a µ;ä´|"éå0öÜWp~SZ
00b0: 5f bf b0 32 21 44 b5 04 b2 8d 22 b7 21 e4 d5 98 _¿°2!Dµ.²□"!äÖ~
00c0: 10 17 3b 2b 71 4a 70 3b b3 6f fc b8 9a d5 33 32 ..;+qJp;³oü,šÖ32
00d0: 60 91 fb 41 a6 d4 fb 71 f4 12 5e 80 33 1d bc e7 `´ûA|ôûqô.^€3.¼ç
00e0: 53 ec 13 4f 9b 3d a9 e7 77 ff 89 54 cc 25 a8 ff Si.O>=©çwÿ%Tİ%`ÿ
00f0: c7 d3 e0 2f c7 03 f0 cf 0f 18 c8 8d 1f 5d 35 32 ÇÓà/Ç.ðİ..È□.]52
0100: bb 17 db 94 63 03 77 34 73 02 03 01 00 01 ».Û"c.w4s.....

```

SIGNATURE PRIVATE KEY (D) =

```

0000: 01 a1 65 cd 90 11 bb 1c 05 34 34 79 ef b3 d5 fc .;eÍ□.»..44yì³Öü
0010: 14 78 d1 35 c3 1d 65 95 fd e5 71 b5 e7 b7 20 da .xÑ5Ã.e•ýâquç· Û
0020: 89 a7 1e 7c 97 1a fe e0 25 15 a4 86 06 29 9d 97 %$.|-.þà%.µ†.)□-
0030: a0 9c 54 d7 d6 9e 9f ab 64 c3 0c a7 81 d5 26 46 æT×ÖžŸ«dÃ.Ş□Ö&F
0040: f0 b1 71 69 49 d5 95 4f 59 69 6e a6 14 1d 01 d6 ð±qiIÖ•OYin|...Ö
0050: 0e 4c 6e 96 2f fb 4c 03 9d 79 c9 94 73 fd 03 b3 .Ln-/ûL.□yÉ"sý.³
0060: 66 2e 47 07 4a 58 a0 74 db 69 4c 88 6e 9b 12 55 f.G.JX tÛiL^n>.U
0070: 9a 12 a8 2c 60 d6 9f b3 cf 7b 47 20 c5 89 28 8e š.´´,`ÖŸ³İ{G Å% (Ž
0080: 23 4c d6 97 21 d0 19 df 7e a3 05 ac f2 2e 0c e9 #LÖ-!Ð.ß~£.-ò..é
0090: 59 7c 98 1b df aa d7 09 69 d3 04 ca 09 87 4e 52 Y|~.ßª×.iÓ.Ê.+NR
00a0: c5 62 ce 85 39 2b 45 8e 6e e3 aa dd e1 85 bd 62 Åbî...9+EŽnăªÝá...¼b
00b0: 4f 0a 0f 75 29 32 30 4a 4b 07 7c aa 4a c8 50 48 O..u)20JK.|ªJÈPH
00c0: 0a ee 65 55 89 6b 2d 98 40 33 61 1d da 24 f4 e8 .îeU%k-~@3a.Ú$ðè
00d0: 08 79 d5 aa 5f 4b f8 eb fb 98 dc b9 cb cd 09 ac .yÖª_Køëû~Û¹ÉÍ.-
00e0: 78 a0 10 a0 96 5a 63 59 d5 03 2c f3 68 64 d8 b0 x . -ZcYÖ.,óhdØ°
00f0: 05 d2 74 6b 18 a5 0a fb f9 af 84 31 db 5b 41 39 .Òtk.¥.ûû~„1Û[A9

```

Signature Private Key (Encrypted under LMK) =

```

0000: 14 f7 b6 45 05 43 83 da 0c 52 c6 5f 91 86 ed 10 .÷¶E.CfÚ.RE_`+í.
0010: f6 88 11 f2 8d 0f 69 98 00 3f a6 48 a7 5c e5 12 ö^.ð□.i~.?!H$\\ä.
0020: 34 59 53 eb 41 5b b1 de c9 21 f5 dd b3 e1 5d 47 4YSéA[±ÞÉ!öŸ³á]G
0030: 5d 8a 27 e4 c8 a9 48 cc 36 f8 c2 5d 84 7b 42 ba ]Š'äÈ©HÌ6øÃ]„[B°
0040: 66 55 1f 66 da 86 02 b1 8c b0 f9 f0 b4 90 0e 1c fU.fÚ†.±€°ùð´□..
0050: 40 db ba 12 d3 e2 e7 d1 44 af 02 97 70 38 8a f9 @Û°.ÓâçÑD~.-p8Šù
0060: bc 88 d9 fc 29 6b 1f a8 88 44 47 6d 5e b8 fc 70 ¼^Ûû)k.´^DGm^,üþ
0070: 90 8d f0 fc b8 18 b7 bb 48 67 67 d1 cb e4 ba 92 □□ðû,.»HggÑÈä°'
0080: 77 d7 50 08 84 4e eb fe 38 e6 39 1c db c8 d4 1c w×P.„Nëþ8æ9.ÛÈÖ.
0090: f2 f6 5c c4 b8 1a bd a0 6d 37 13 75 06 85 a3 0a òö\Ã.½ m7.u...£.
00a0: 24 62 99 72 02 E0 F5 FE D5 BF 2A 3A 0C 25 0E 3D $B™R.ÀÖÞÖ¿*:.%.=
00b0: 64 09 2E 7D 2A 1B 72 3B D1 57 1A 71 92 68 C3 F0 D..}*R;ÑW.Q'HÃÐ
00c0: df e6 24 8a 8f 3d d9 34 43 7b 34 25 3d 06 1c 40 ßæ$Š□=Û4C{4%=..@
00d0: 14 2e b0 f3 83 25 de 14 3c 6f 3a e3 56 15 fc 61 ..°óf%Þ.<o:äV.üa
00e0: a4 a7 a2 0d e4 f6 bf 38 66 cc 6e 80 6b dd 28 c2 µ$ç.äö¿8fÎnEkÝ(Ã
00f0: ad 96 be f3 b3 ae 81 ee 49 5d 0e 25 90 a3 32 34 --¾ó³@□îI].%□£24
0100: 73 5f 48 a8 53 57 00 df 51 71 e3 9b 1d 6b 35 b7 s_H`SW.ßQqã>.k5·
0110: 28 ad fc e9 4d b6 95 dc 8d 26 9b 86 40 82 30 aa (-üéM¶•Û□&>†@,0ª
0120: 26 b6 c1 cc 72 48 b3 36 d5 65 18 a2 af 09 c4 d2 &¶ÁÎrH³6Öe.ç~.ÄÖ
0130: eb 1a fb f5 4b 91 c6 c0 bd 13 ad 0a f8 d4 ea b2 ë.ûðK`ÆÀ½.-.øÔè²
0140: 3f a2 ad 0f 8c ce 1e 4b be 2f d8 52 4e ed e0 76 ?ç-.ÆÎ.K¾/ØRNiàv
0150: 90 d9 b0 c3 67 0f 71 51 3b 78 be 32 b0 dc 28 56 □Û°Ãg.qQ;x¾2°Û(V
0160: 71 70 49 c0 55 1f 4e fd 5f ac 8a 46 f8 e6 02 dc qpIÄU.Ný_→ŠFøæ.Û
0170: a6 55 a8 2a f9 18 2d e3 90 7d f7 6f e4 8b b9 65 ¦U`*ù.-ã□}÷oä<¹e
0180: 85 f6 9b fe b0 d4 37 4a 36 e0 c3 d6 d5 2d 9f 61 ...ö>þ°Ô7J6äÃÖÖ-Ýa
0190: e9 a2 5e 5e 4a a1 e4 c3 cc 74 4c b4 9e 86 4d 8b éç^èJ;äÃîtL´ž†M<

```



```

01a0: 23 6b e5 15 b2 42 5d 6f 28 0c 74 3f 4b cf b0 64 #kå.²B]o(.t?Kİ°d
01b0: 95 82 e9 9a 06 fb 78 ed f7 5d ae 1f d9 d3 fa 66 •,éš.ûxí÷]®.ÛÓúf
01c0: b9 40 7c f2 b4 4a 6f 6e 3f ab 21 49 0a 30 91 92 ¹@|ò´Jon?«!I.O''
01d0: 5b 0d df f0 98 9d 62 58 9a 4f 23 06 1a 2c db 41 [.Bð~□bXšO#..,ÛA
01e0: cf 34 3d 89 46 eb 39 02 81 9a 81 57 ac b2 b2 bf Ĩ4=%Fè9.□š□W¬²²¿
01f0: 58 7f 84 cf d6 7b fe 91 db 9a fb bf a3 01 fc b5 X□„İÖ{p`Ûšû¿£.ûµ
0200: 7b c5 12 64 bc 3a 75 8e df 57 04 b6 a1 69 18 a6 {Å.d¼:užßW.¶;i.¡
0210: 6f e4 e8 67 78 42 b9 08 0e 5c d1 cf 7e e2 fd 7a oäègxB¹..\\Ñĩ~âýz
0220: a7 ac 3b 4e 41 39 3f a2 d5 14 b9 76 1a 22 a5 fd Š¬;NA9?¢Ö.¹v."¥ý
0230: 80 9a 21 06 d9 9b 40 6a 1c 5b e7 a7 f9 97 7a 98 €š!.Û>@j.[ç$ù-z~
0240: 8b aa 1a 37 54 1c d7 d3 8d b0 64 88 ae c8 0e 0a <ª.7T.xó□°d^@È..
0250: 9c d3 18 17 7b c1 09 db 87 68 ea 57 1f 38 4e 45 œÓ..{Á.ÛþhêW.8NE
0260: 7f 80 08 b8 05 5a 6b 4b 82 98 f8 83 b5 1d 72 40 □€..ZkK,~øfµ.r@
0270: 12 a8 f5 dd 87 8d 58 50 40 bf e5 2c 5c 00 61 22 .''öÝþ□XP@¿â,\.a"
0280: b4 8b 14 e9 13 2e b2 61 75 77 7b 9f 74 b8 f7 3f ´<.é...²auw{ÿt,÷?

```

5.5.5 Sample Data Generation Procedure (2048-bit RSA keys)

1. DER encode ASN.1 format DES Key

```
30 12
```

```
03 10
```

```
7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C 7C
```

2. Pad #1 using PKCS#1 OAEP

```

0000: 00 DF F6 83 6C EA D3 47 BE C2 3C 06 6D 66 19 9E .BöflêÓG¾Å<.mf.ž
0010: 14 83 36 95 7B 3B AC 74 11 CA DE CD C8 7F 78 A6 .f6•{;-t.ÊPÍÈ?x|
0020: 64 E5 DC 33 CE 37 72 85 A7 62 7A 37 FE C8 8E 8C dâÛ3î7r...$bz7pÈŽE
0030: 21 21 E0 F9 64 73 24 9C B2 07 7A A3 60 B4 DB ED !!àùds$œ².z£`´Ûí
0040: 52 EF 02 89 3B C8 E9 A9 C3 E6 E8 AB 29 EC 20 D1 Rì.%;Êé©Ãæè«)ì Ñ
0050: F0 CA 0B AF 7D 6A D3 C7 90 D3 F3 73 75 63 AC A6 ðÊ.¬}jÓÇ?Óósuc¬|
0060: D4 D4 91 FA D8 5F 23 6B 7C 2A 07 E0 39 DB 5E BC ÔÔ`úØ_#k|*.à9Û^¼
0070: 6A 50 B5 0C 6E CC 19 8C 3C DD 7E DE F6 16 30 39 jPµ.nİ.Æ<Ý~Pö.09
0080: 25 5A DD 34 C9 BF 2B 53 98 43 9D B8 1E 0C AF E8 %ZY4Ê¿+S~C?_..¬è
0090: E0 38 6F F8 31 CE 18 5B 17 BA F0 47 83 EE D7 7C à8oø1î.[.°ðGfîx|
00A0: F9 CD 37 C9 9A AD D1 FF 12 8D 27 9F 03 7D F4 41 ùí7Éš-Ñý.?`Ý.)ðA
00B0: A8 3B 9C F8 C4 5E 80 F3 2D AF 6F 20 F1 81 7E 07 ``;œðÅ^€ó-¬o ñ~?.
00C0: 74 E4 0C E8 C0 2A FA D7 BD 22 18 FF BA DB F1 E1 tã.èÀ*ú×½".ÿ°Ûñá
00D0: 5D AB 8A 0A CC A1 6C 72 FB 8A 5E 8A 13 5B EC 2F ]«Š.İ;lrûŠ^Š.[i/
00E0: A8 8A 55 93 F4 56 CF A4 2E 84 23 1D 7B 8E 4F 54 ``ŠU"ôVĩ¤.„#{ŽOT
00F0: 4D AF 58 F9 FF FA 1F 9C 9D F4 A8 5D 2B 2E FA 12 M¬Xùýú.œô?``]+.ú.

```

3. Encrypt #2 using Encryption Public Key

```

0000: 32 27 D3 03 4C 27 B6 48 B7 AE 68 53 77 17 50 62 2'Ó.L'¶H·@hSw.Pb
0010: AF 19 22 24 71 72 E1 E7 51 5C 11 43 81 09 54 FC ¬."$qrácQ\C?.Tù
0020: D4 46 38 AC 96 98 DE 90 AC EC 0E 0A 97 77 93 CA ÔF8¬¬~P¬-ì...-w"Ê
0030: 8C 35 41 7E 0C C9 2B 6A 32 AB C6 60 C7 34 AA 7D E5A~.Ê+j2«Æ`Ç4ª}
0040: FA F8 29 91 71 20 4F 13 7C F9 98 10 91 2B 34 44 úø)`q O.|ù~.´+4D
0050: 9D 7A 39 30 E6 04 13 5D 12 64 D7 0E C5 68 78 E1 ?z90æ..].d×.Åhxá
0060: C2 1F 42 C8 EF DE 21 1C CC 78 1D 84 97 96 72 65 Å.BÈİP!.İx.„--re
0070: 85 9C 3B E9 3A DF F0 B5 DC A7 9D 53 EF E8 6E A6 ...æ;é:ßðµÛ$?Sièn|
0080: 14 61 9A FB C0 6A AD FF 66 C5 D6 BD E3 E0 A4 C0 .aşûÄj-ÿfÅÖ¼ääµÅ
0090: B2 08 4D 30 2B 28 96 65 4E F9 36 40 46 45 22 70 ².M0+(-eNù6@FE"p
00A0: C5 11 AE 6B 03 B1 1B 94 4C 8E FC BE 12 40 E5 95 Å.®k.±."Lžü¾.®â•
00B0: E4 32 AE 8C 9B A7 BE 13 5E 6A 29 60 F9 65 77 F9 ä2®E>S¾.^j)`ùewù

```

```

00C0: 76 34 AC B1 C7 42 C6 28 35 58 BB 68 1A 61 2D FF v4~±ÇBÆ(5X»h.a-ÿ
00D0: 90 82 48 F6 C9 D4 E8 04 76 B5 94 C7 2A E1 47 F0  ?,HöÉÔè.vµ"Ç*áGð
00E0: 02 2C CD FB 9B 63 63 1B 74 D1 99 AF B3 85 26 7C  .,íû>cc.tÑ™³...&|
00F0: 22 3C EF FD 7C EA E8 D1 E0 4A 9E A1 27 FD CB 58 "<íý|éèÑàJž; 'ýËX

```

4. Insert #3 into data block for signing

```

0000: 59 59 59 59 59 59 59 59 59 59 59 59 32 27 D3 03 YYYYYYYYYYYY2'Ó.
0010: 4C 27 B6 48 B7 AE 68 53 77 17 50 62 AF 19 22 24 L'qH·@hSw.Pb¯."$
0020: 71 72 E1 E7 51 5C 11 43 81 09 54 FC D4 46 38 AC qráçQ\ .C.?TüÔF8~
0030: 96 98 DE 90 AC EC 0E 0A 97 77 93 CA 8C 35 41 7E  ~þ?~i..-w"ÊE5A~
0040: 0C C9 2B 6A 32 AB C6 60 C7 34 AA 7D FA F8 29 91  .É+j2«Æ`Ç4ª}úø) `
0050: 71 20 4F 13 7C F9 98 10 91 2B 34 44 9D 7A 39 30  q O.|ù~.`+4D?z90
0060: E6 04 13 5D 12 64 D7 0E C5 68 78 E1 C2 1F 42 C8  æ..].d×.ÅhxáÂ.BÈ
0070: EF DE 21 1C CC 78 1D 84 97 96 72 65 85 9C 3B E9  ìþ!.îx.„--re...œ;é
0080: 3A DF F0 B5 DC A7 9D 53 EF E8 6E A6 14 61 9A FB  :ßðµÛ$?Sièn|.ašû
0090: C0 6A AD FF 66 C5 D6 BD E3 E0 A4 C0 B2 08 4D 30  Àj-ýfÅÖ½ääªÀ².M0
00A0: 2B 28 96 65 4E F9 36 40 46 45 22 70 C5 11 AE 6B  +(-eNù6@FE"pÅ.©k
00B0: 03 B1 1B 94 4C 8E FC BE 12 40 E5 95 E4 32 AE 8C  .±."Lžü¾.@a•ä2@E
00C0: 9B A7 BE 13 5E 6A 29 60 F9 65 77 F9 76 34 AC B1  >$¼.^j)`ùewùv4~±
00D0: C7 42 C6 28 35 58 BB 68 1A 61 2D FF 90 82 48 F6  ÇBÆ(5X»h.a-ÿ?,Hö
00E0: C9 D4 E8 04 76 B5 94 C7 2A E1 47 F0 02 2C CD FB  ÉÔè.vµ"Ç*áGð.,íû
00F0: 9B 63 63 1B 74 D1 99 AF B3 85 26 7C 22 3C EF FD  >cc.tÑ™³...&|"<íý
0100: 7C EA E8 D1 E0 4A 9E A1 27 FD CB 58 5A 5A 5A 5A  |éèÑàJž; 'ýËXZZZZ
0110: 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A  ZZZZZZZZZZZZ

```

5. Generate hash over #4

```

0000: 27 E2 34 FF D1 99 2F 8A A8 56 F0 54 B5 A7 6D 81  'â4ÿÑ™/Š`VðTµ$м?
0010: A5 80 FA 42  ¥€úB

```

6. DER encode ASN.1 format #5

```

30 21
30 09
06 05
2B 0E 03 02 1A
05 00
04 14
27 E2 34 FF D1 99 2F 8A A8 56 F0 54 B5 A7 6D 81 A5 80 FA 42

```

7. Pad #6 using PKCS#1 v1.5

```

0000: 00 01 FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ..ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0060: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0070: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0090: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 30 21 30  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ.0!0
00E0: 09 06 05 2B 0E 03 02 1A 05 00 04 14 27 E2 34 FF  ...+. 'â4ÿ
00F0: D1 99 2F 8A A8 56 F0 54 B5 A7 6D 81 A5 80 FA 42  Ñ™/Š`VðTµ$м?¥€úB

```

8. Encrypt #7 using Signature Private Key:

```

0000: 19 5F 27 EF 10 CA F3 8D 81 98 30 06 38 B1 B4 7C  ._'i.Êó?~?0.8±'|
0010: 53 DA 8C E1 96 31 0A A3 0C 43 2A EB 87 F9 4A CE  SÚŒá-1.£.C*ë#ùJî
0020: 67 E1 5B 0F 1E 6D 88 DB 59 05 EC AE F9 A0 35 61  gá[.m^ÛY.i@ù 5a
0030: 8B D8 C4 AC C6 37 36 03 2B 6E C2 5A D7 87 5B 12  <ØÄ~Æ76.+nÂZ×+ [.
0040: 89 B0 35 A5 F9 FD 77 46 BD CB 69 D2 15 4A C5 99  %°5¥ùýwF½ËiÒ.JÅ™
0050: DD A5 EE 91 DB B3 22 CD D2 83 FB 38 E8 F8 0E 37  Ý¥î`Û³"íÒfû8èø.7
0060: 83 C3 BE 3B 90 7B C6 49 59 EC 82 1B D2 9E 63 A8  fÃ¾;:{ÆIYì,.Òžc`
0070: FD 49 0D 4B 1B ED 2E 29 71 C8 EA 87 3D EC 40 3A  ýI.K.í.)qÈê+=i@:
0080: B1 6C 1F 2B 83 AD 54 8E 81 5A CE EA 26 1B AB 6E  ±l.+f-TŽ?ZÎê&.«n
0090: FC FA 8C B0 79 17 6C D7 10 4A 1E 97 11 4C B6 30  ùúŒ°y.l×.J.LŦ0
00A0: 06 E7 1C B2 C0 5C AD B2 7B 8B 79 31 F7 7D 20 75  .ç.²Ä\~²{<y1÷} u
00B0: 6B 8C 85 07 6D E3 D5 66 95 24 18 95 6A D8 62 66  kŒ...mãŒf•$.•jØbf
00C0: 26 68 37 E9 66 8D E1 E6 58 63 BD 95 B1 2F 1D F7  &h7éf?áæXc½•±/.÷
00D0: E9 B9 41 CC DD E6 55 6F 22 DF 21 14 65 5E 29 0E  é¹AìŸæUo"ß!.e^).
00E0: 89 06 54 B1 11 EF 9A D8 8E 80 67 78 73 F5 73 84  %.T±.išØŽĖgxsôs,,
00F0: 5C DE A5 23 52 8A C8 99 97 07 5B 8C 00 68 F7 A1  \P¥#RŠĖ™-. [Œ.h÷;

```

5.5.6 Sample Data Calculation (2048-bit RSA keys)

Field	Length & Type	Value
COMMAND MESSAGE		
Message Header	m A	
Command Code	2 A	"GK" – Export DES Key
Encryption Identifier	2 N	"01" – RSA Encryption
Pad Mode Identifier	2 N	"02" – OAEP (EME-OAEP-ENCODE)
Mask Generation Function	2 N	"01" – MGF1 as defined in PKCS#1 v2.2
MGF Hash Function	2 N	"01" – SHA-1
OAEP Encoding Parameters Length	2 N	"04"
OAEP Encoding Parameters	N B	"09080706"
OAEP Encoding Parameters Delimiter	1 A	","
DES Key Type	4 N	"0600"
Signature Indicator	1 A	"="
Signature Hash Identifier	2 N	"01" – SHA-1
Signature Identifier	2 N	"01" – RSA
Signature Pad Mode Identifier	2 N	"01" – PKCS#1 v1.5 padding
Header Data Block Length	4 N	"0012"
Header Data Block	n B	"YYYYYYYYYYYY"
Delimiter	1 A	","
Footer Data Block Length	4 N	"0015"
Footer Data Block	n B	"ZZZZZZZZZZZZZZZZ"
Delimiter	1 A	","
Private Key Flag	2 N	"99"
Private Key Length	4 N	"0656"
Private Key	n B	14F7B645054383DA0C52C65F9186ED10F68811F28D0F69980 03FA648A75CE512345953EB415BB1DEC921F5DDB3E15D47 5D8A27E4C8A948CC36F8C25D847B42BA66551F66DA8602B 18CB0F9F0B4900E1C40DBBA12D3E2E7D144AF029770388A F9BC88D9FC296B1FA88844476D5EB8FC70908DF0FCB818B 7BB486767D1CBE4BA9277D75008844EEBFE38E6391CDBC8 D41CF2F65CC4B81ABDA06D3713750685A30A2462997202E0 F5FED5BF2A3A0C250E3D64092E7D2A1B723BD1571A71926 8C3F0DFE6248A8F3DD934437B34253D061C40142EB0F3832 5DE143C6F3AE35615FC61A4A7A20DE4F6BF3866CC6E806B DD28C2AD96BEF3B3AE81EE495D0E2590A33234735F48A85

Field	Length & Type	Value
COMMAND MESSAGE		
		35700DF5171E39B1D6B35B728ADFCE94DB695DC8D269B86 408230AA26B6C1CC7248B336D56518A2AF09C4D2EB1AFBF 54B91C6C0BD13AD0AF8D4EAB23FA2AD0F8CCE1E4BBE2F D8524EED07690D9B0C3670F71513B78BE32B0DC28567170 49C0551F4EFD5FAC8A46F8E602DCA655A82AF9182DE3907 DF76FE48BB96585F69BFEB0D4374A36E0C3D6D52D9F61E9 A25E5E4AA1E4C3CC744CB49E864D8B236BE515B2425D6F2 80C743F4BCFB0649582E99A06FB78EDF75DAE1FD9D3FA66 B9407CF2B44A6F6E3FAB21490A3091925B0DDFF0989D6258 9A4F23061A2CDB41CF343D8946EB3902819A8157ACB2B2B F587F84CFD67BFE91DB9AFBBFA301FCB57BC51264BC3A75 8EDF5704B6A16918A66FE4E8677842B9080E5CD1CF7EE2F D7AA7AC3B4E41393FA2D514B9761A22A5FD809A2106D99B 406A1C5BE7A7F9977A988BAA1A37541CD7D38DB06488AEC 80E0A9CD318177BC109DB8768EA571F384E457F8008B8055 A6B4B8298F883B51D724012A8F5DD878D585040BFE52C5C0 06122B48B14E9132EB26175777B9F74B8F73F
DES Key Flag	1 N	"1"
DES Key (LMK)	1A+32H	"U513ECF7E7D0A5554900FA709B9A5F721"
Check Value	16 H	"8B1536D4D60FEC19"
MAC	4 B	AA 5E 41 CC
Public Key	n B	3082010A028201010040F5B53A32475090ADDE8A33EF6F9F9 5EE584184F1FE7D31AE58C6CAE2A062DAB039E745039F7A A43A236A4540AD0A3F361F905F29B1A0E9C165B0A656D9F8 18C57E05D787F1F8EE0C6BE9B42A832CFDBD3574A5E7EF8 63B11FF3F95DCACBBDCFDD90EA9C7D552CF0D1BE471532 CF64E029CCDD7D9AEC92F90C39D1BE963AEE6F578E6A3D 83AC6572EB5525D0A8179BCBA0263D82EBB5D77C3355616 06B32701753AEDC42AAAE120D9233CB3B9EF4DDE0C7B070 2253E255C1EDE052D9C4F7A6C7AFBF4E408BB18D95354C4 4462FF352BFF311E8E07A38349A78808164A24A08FD9D3549 BC33D58DEC525E97FFBFBF8A6D41205120AE8D4AC756F3E 8580918DCB0203010001
Authentication Data	n A	"ABCDEFGH"
Delimiter	1 A	","
Key Block Type	2 N	"02" – Key Block Template
Key Block Template Length	4 N	"0020"
Key Block Template	n H	"3012031000"
Delimiter	1 A	","
DES Key Offset	4 N	"0004"
Check Value Length	2 N	"00"
Check Value Offset	4 N	"0000"
End Message Delimiter	1 C	
Message Trailer	n A	
RESPONSE MESSAGE		
Message Header	m A	
Response Code	2 A	GL
Error Code	2 A	"00" – No error
Initialisation Value	16 H	"????????????????"
DES Key Length	4 N	"0256"
DES Key (PK)	n B	3227D3034C27B648B7AE685377175062AF1922247172E1E75 15C1143810954FCD44638AC9698DE90ACE0E0A977793CA 8C35417E0CC92B6A32ABC660C734AA7DFAF8299171204F13 7CF99810912B34449D7A3930E604135D1264D70EC56878E1 C21F42C8EFDE211CCC781D8497967265859C3BE93ADFF0B 5DCA79D53EFE86EA614619AFBC06AADFF66C5D6BDE3E0A 4C0B2084D302B2896654EF9364046452270C511AE6B03B11B 944C8EFCBE1240E595E432AE8C9BA7BE135E6A2960F96577 F97634ACB1C742C6283558BB681A612DFF908248F6C9D4E8 0476B594C72AE147F0022CCDFB9B63631B74D199AFB38526 7C223CEFFD7CEAE8D1E04A9EA127FDCB58
Signature Length	4 N	"0256"
Signature	n B	195F27EF10CAF38D8198300638B1B47C53DA8CE196310AA3 0C432AEB87F94ACE67E15B0F1E6D88DB5905ECAEF9A0356 18BD8C4ACC63736032B6EC25AD7875B1289B035A5F9FD77

Field	Length & Type	Value
COMMAND MESSAGE		
		46BDCB69D2154AC599DDA5EE91DBB322CDD283FB38E8F8 0E3783C3BE3B907BC64959EC821BD29E63A8FD490D4B1BE D2E2971C8EA873DEC403AB16C1F2B83AD548E815ACEEA2 61BAB6EFCFA8CB079176CD7104A1E97114CB63006E71CB2 C05CADB27B8B7931F77D20756B8C85076DE3D56695241895 6AD86266266837E9668DE1E65863BD95B12F1DF7E9B941CC DDE6556F22DF2114655E29E890654B111EF9AD88E8067787 3F573845CDEA523528AC89997075B8C0068F7A1
End Message Delimiter	1 C	
Message Trailer	n A	

6 Local Master Keys (LMKs)

6.1 Introduction

LMKs are used to encrypt operational keys used for encryption, MACing, digital signing, etc. LMKs are secret, internal to the HSM, and do not exist outside of the HSM except as components or shares held in smart cards. Each HSM can have a unique LMK, or an organization can install the same LMKs on multiple HSMs within a logical system.

LMKs provide separation between different types of keys to ensure that keys can be used only for their intended purpose. Thales payment HSMs support two types of LMK, both of which provide key separation:

- Variant LMKs. These are double- or triple-length Triple-DES keys and provide key separation by encrypting different types of key with different variants of the LMK. Double-length Variant LMKs have been in use for many years, and are the most widely used type of LMK. Triple-length Variant LMKs were introduced for later versions of the payShield 10K. See Chapter 7, “*Variant LMK Key Scheme*”.
- Key Block LMKs. These are either triple-length Triple-DES keys, or 256-bit AES keys, and key separation is provided by parameters in the key block which govern characteristics such as usage and exportability of the protected key. Key Block LMKs are newer technology than Variant LMKs and so are still less widely used, but provide security benefits. Chapter 8, “*Key Block LMK Key Scheme*” describes Key Block LMKs in more detail.
- It is possible to install multiple LMKs within a single payShield 10K. See Chapter 9, “*Multiple LMKs*”.
- Refer to the *payShield 10K Host Command Reference Manual* for information on how the required LMK can be specified in Host Commands.

6.2 Multiple LMKs

The table below shows the possible values in the key block header fields when creating the standard HSM keys. (This table uses the same pink/blue shading as is used in the *payShield 10K Host Command Reference Manual* to distinguish between information relating to key block and variant keys.)

6.3 Key Block & Variant Key Comparison Table

The table below shows the possible values in the key block header fields when creating the standard HSM keys.

Key Name	Variant		Key Block		
	Key Type	LMK	Key Usage	Algorithm	Mode of Use
BDK-1	009	28-29/0	'B0'	'T', 'A'	'X', 'N'
BDK-2	609	28-29/6	'41'	'T', 'A'	'X', 'N'
BDK-3	809	28-29/8	'42'	'T'	'X', 'N'
BDK-4	909	28-29/9	'43'	'T', 'A'	'X', 'N'
BDK-5	-	-	'44'	'T'	'X', 'N'
CK-DEK	50D	36-37/5	'39'	'T', 'A'	'X', 'N'
CK-ENC	30D	36-37/3	'37'	'T', 'A'	'X', 'N'
CK-MAC	40D	36-37/4	'38'	'T', 'A'	'X', 'N'
CSCK	402	14-15/4	'C0', '11'	'D', 'T'	'C', 'G', 'V', 'N'
CTRDEK	-	-	'25'	'A'	'B', 'D', 'E', 'N'
CVK	402	14-15/4	'C0', '12', '13'	'D', 'T'	'C', 'G', 'V', 'N'
DEK	00B	32-33/0	'D0', '21'	'D', 'T', 'A'	'B', 'D', 'E', 'N'
HMAC	10C	34-35/1	'61', '62', '63', '64', '65'	'H'	'C', 'G', 'V', 'N'
IKEY□	302	14-15/3	'B1'	'T', 'A'	'X', 'N'
KEK	107	24-25/1	'54'	'T', 'A'	'B', 'D', 'E', 'N'
KEK (Transport Key)	-	-	'24'	'A'	'B', 'D', 'E', 'N'
KMC	207	24-25/2	'E7'	'T', 'A'	'N'
KML	200	04-05/2	'E6', '31'	'T', 'A'	'N'
MK-AC	109	28-29/1	'E0'	'T', 'A'	'N'
MK-CVC3	709	28-29/7	'E6', '32'	'T', 'A'	'N'
MK-DAC	409	28-29/4	'E3'	'T'	'N'

Key Name	Variant		Key Block		
	Key Type	LMK	Key Usage	Algorithm	Mode of Use
MK-DN	509	28-29/5	'E4'	'T'	'N'
MK-SMC	309	28-29/3	'E1'	'T'	'N'
MK-SMI	209	28-29/2	'E2'	'T', 'A'	'N'
M_KEY_CONF	-	-	'33'	'A'	'X', 'N'
M_KEY_MAC	-	-	'34'	'A'	'X', 'N'
MS_KEY_CONF	-	-	'35'	'A'	'B', 'N'
MS_KEY_MAC	-	-	'36'	'A'	'C', 'N'
PSK	507	24-25/5	'40'	'T'	'N'
PVK	002	14-15/0	'V0', 'V1', 'V2'	'D', 'T'	'C', 'G', 'V', 'N'
RSA Private Key	00C	34-35/0	'03', '04', '05', '06'	'R'	'D', 'N', 'S'
RSA Public Key	00D	36-37/0	'02'	'R'	'E', 'N', 'V'
ECC Private Key	-	-	'03'	'E'	'X', 'N', 'S'
ECC Public Key	-	-	'02'	'E'	'X', 'N', 'V'
SK-DEK	507	24-25/5	'49'	'T'	'B', 'D', 'E', 'N'
SK-ENC	307	24-25/3	'47'	'T', 'A'	'B', 'D', 'E', 'N'
SK-MAC	407	24-25/4	'48'	'T', 'A'	'C', 'G', 'V', 'N'
SK-RMAC	008	26-27/0	'48'	'T', 'A'	'C', 'G', 'V', 'N'
TAK	003	16-17/0	'M0', 'M1', 'M3', 'M5', 'M6'	'D', 'T', 'A'	'C', 'G', 'V', 'N'
TEK	30B	32-33/3	'D0', '23'	'D', 'T', 'A'	'B', 'D', 'E', 'N'
TKR	002 or 90D	14-15/0 or 36-37/9	'P0', '73'	'D', 'T'	'N'
TMK	002 or 80D	14-15/0 or 36-37/8	'K0', '51'	'D', 'T', 'A'	'B', 'D', 'E', 'N'

φ IKEY is also known as IPEK.

6.4 Converting Key Names

The table below shows some of the conversions between Thales and other organizations' key names:

Organization	Key Description	Thales Key Description	Thales Key Name
Mastercard	Issuer MK	Master Key for Authentication Cryptograms	MK-AC
		Master Key for Secure messaging Integrity	MK-SMI
		Master Key for Secure Message Confidentiality	MK-SMC
		Master Key for Data Authentication Codes	MK-DAC
		Master Key for Dynamic Numbers	MK-DN
Mastercard	ICC MK	Derived Key for Authentication Cryptograms	DK-AC
		Derived Key for Secure Messaging Integrity	DK-SMI
		Derived Key for Secure Messaging Confidentiality	DK-SMC
		Derived Key for Dynamic Numbers	DK-DN
Visa	AWK (Acquirer Working Key)	Zone PIN Key	ZPK
Visa	C2KA (Card Verification Key for generation of CVV2)	CVKA (1st half of double-length CVK)	CVK
Visa	C2KB (Card Verification Key for generation of CVV2)	CVKA (2nd half of double-length CVK)	CVK
Visa	CAKA (Card Verification Key (for generation of CAVV))	CVKA (1st half of double-length CVK)	CVK
Visa	CAKB (Card Verification Key (for generation of CAVV))	CVKA (2nd half of double-length CVK)	CVK
Visa	DMK-AC	Master Key for Authentication Cryptograms	MK-AC
Visa	DMK-MAC	Master Key for Secure messaging Integrity	MK-SMI
Visa	DMK-ENC	Master Key for Secure Message Confidentiality	MK-SMC
Visa	IWK (Issuer Working Key)	Zone PIN Key	ZPK

7 Variant LMK Key Scheme

A Variant LMK is a set of 20 double- or triple-length TDES keys, with different "pairs" (and variants of those pairs) being used to encrypt different types of keys. The Double-length Variant LMK is the original LMK format supported in all versions of Racal/Thales payment HSM firmware.

Note: The term "Variant LMK" refers to the 'variant' method of encrypting keys; a Variant LMK is not itself a variant of any other key.

Keys encrypted under a Variant LMK have an associated (3-digit) Key Type Code, which is used to enforce key separation between different types of keys. The different types of symmetric keys available are included in the table below.

Note: Variant LMKs cannot be used to protect AES keys or RSA keys longer than 2048 bits, or ECC keys.

The tables below use the same pink shading as in the *payShield 10K Host Command Reference* Manual to indicate information relating to variant keys.

7.1 How the Variant scheme works

Each key of a double- or triple-length TDES key set is encrypted separately using the ECB mode of encryption. For the second key, a variant is applied to the encryption key. There are five variants to enable the encryption of each key distinctly. This application of variants enforces the key use as a double- or triple-length key and the key order. This scheme is available for encryption of keys under the Local Master Key and for import and export of keys.

Variant Local Master Keys can be either:

- Double-length TDES keys, consisting of a left and right half. Each half consists of 16 hexadecimal characters.
- Triple-length TDES keys, consisting of a left, middle and right part. Each part, as for double-length keys, consists of 16 hexadecimal characters.

Other key encryption keys, such as ZMKs, can be double- or triple-length TDES keys.

The variant is applied to the right half of double-length encrypting keys, and to the middle part of triple-length encrypting keys.

The tags for this scheme are as follows:

- U - Double-length DES keys
- T - Triple-length DES keys

The following variants are used for this purpose:

- Double-length key:
 - Left part - 6A
 - Right part - 5A
- Triple-length key:
 - Left part - 6A
 - Middle part - DE
 - Right part - 2B

Example 1:

Given a double-length encrypting key of: XXXX XXXX XXXX XXXX YYYY YYYY YYYY YYYY, and a double-length key of: AAAA AAAA AAAA AAAA BBBB BBBB BBBB BBBB:

- The variant A6 is applied to the first two hex characters of Y to encrypt A.
- The variant 5A is applied to the first two hex characters of Y to encrypt B.

Example 2:

Given a double-length encrypting key of: XXXX XXXX XXXX XXXX YYYY YYYY YYYY YYYY, and a triple-length key of: AAAA AAAA AAAA AAAA BBBB BBBB BBBB BBBB CCCC CCCC CCCC CCCC:

- The variant 6A is applied to the first two hex characters of Y to encrypt A.
- The variant DE is applied to the first two hex characters of Y to encrypt B
- The variant 2B is applied to the first two hex characters of Y to encrypt C

Variants are applied by "Exclusive ORing" (XOR) the first two characters of Y with the Variant.

7.2 Local Master Key (LMK) Variants

To protect key usage, variants of the Local Master Key are used for encryption of defined keys or key components. The Key Type Table (see later in this chapter) defines the LMK pairs and variants that are used to protect various types of keys. For example, an MK- SMI is encrypted using LMK 28-29 variant 2.

In order to create an LMK variant, a non-secret fixed value (known as a variant) is XOR'ed with the first byte of the LMK. The variants used by the HSM are:

Variant 1 :	A6
Variant 2 :	5A
Variant 3 :	6A
Variant 4 :	DE
Variant 5 :	2B
Variant 6 :	50
Variant 7 :	74
Variant 8 :	9C
Variant 9 :	FA

The example below demonstrates how a Local Master Key variant is calculated for key type MK-SMI:

1. Refer to the *Key Type Table* to select the appropriate LMK pair for the type of key that you wish to encrypt. For example, for key type MK-SMI, LMK 28-29 is used:

Test LMK 28-29: 1A1A 1A1A 1A1A 1A1A 1C1C 1C1C 1C1C 1C1C
2. Identify from the Key Type Table which Variant of the LMK is required. For key type MK-SMI variant 2 is used:

Variant 2: 5A
3. Exclusive-OR the selected variant with the first byte of the LMK pair:

1AXOR 5A = 40
4. Replace the left-most byte of the LMK pair with the result of Step 3 and use the resulting key to encrypt the MK-

SMI:

LMK Variant 2 = 401A 1A1A 1A1A 1A1A 1C1C 1C1C 1C1C 1C1C

When the Variants are applied to the standard Double-length Variant Test LMK or Triple-length Variant Test LMK (see later in this chapter), the left-most bytes of the sets are as follows:

7.2.1 Double-length Variant LMK

LMK Pair	First byte of LMK								
	1	2	3	4	5	6	7	8	9
00-01	A7	5B	6B	DF	2A	51	75	9D	FB
02-03	86	7A	4A	FE	0B	70	54	BC	DA
04-05	E6	1A	2A	9E	6B	10	34	DC	BA
06-07	C7	3B	0B	BF	4A	31	15	FD	9B
08-09	26	DA	EA	5E	AB	D0	F4	1C	7A
10-11	07	FB	CB	7F	8A	F1	D5	3D	5B
12-13	67	9B	AB	1F	EA	91	B5	5D	3B
14-15	46	BA	8A	3E	CB	B0	94	7C	1A
16-17	BA	46	76	C2	37	4C	68	80	E6
18-19	A7	5B	6B	DF	2A	51	75	9D	FB
20-21	A4	58	68	DC	29	52	76	9E	F8
22-23	A1	5D	6D	D9	2C	57	73	9B	FD
24-25	B5	49	79	CD	38	43	67	8F	E9
26-27	B0	4C	7C	C8	3D	46	62	8A	EC
28-29	BC	40	70	C4	31	4A	6E	86	E0
30-31	85	79	49	FD	08	73	57	BF	D9
32-33	80	7C	4C	F8	0D	76	52	BA	DC
34-35	8C	70	40	F4	01	7A	5E	B6	D0
36-37	89	75	45	F1	04	7F	5B	B3	D5
38-39	A7	5B	6B	DF	2A	51	75	9D	FB

7.2.2 Triple-length Variant LMK

LMK Pair	First byte of LMK								
	1	2	3	4	5	6	7	8	9
00-01	75	89	b9	0d	f8	83	a7	4f	29
02-03	2c	d0	e0	54	a1	da	fe	16	70
04-05	9b	67	57	e3	16	6d	49	a1	c7
06-07	a7	5b	6b	df	2a	51	75	9d	fb
08-09	13	ef	df	6b	9e	e5	c1	29	4f
10-11	cd	31	01	b5	40	3b	1f	f7	91
12-13	f7	0b	3b	8f	7a	01	25	cd	ab
14-15	2f	d3	e3	57	a2	d9	fd	15	73
16-17	15	e9	d9	6d	98	e3	c7	2f	49
18-19	6e	92	a2	16	e3	98	bc	54	32
20-21	29	d5	e5	51	a4	df	fb	13	75

LMK Pair	First byte of LMK								
	1	2	3	4	5	6	7	8	9
22-23	6b	97	a7	13	e6	9d	b9	51	37
24-25	bc	40	70	c4	31	4a	6e	86	e0
26-27	c1	3d	0d	b9	4c	37	13	fb	9d
28-29	68	94	a4	10	e5	9e	ba	52	34
30-31	58	a4	94	20	d5	ae	8a	62	04
32-33	32	ce	fe	4a	bf	c4	e0	08	6e
34-35	26	da	ea	5e	ab	d0	f4	1c	7a
36-37	1a	e6	d6	62	97	ec	c8	20	46
38-39	ce	32	02	b6	43	38	1c	f4	92

Note: the term "LMK Pair" and the associated numbering scheme is retained for historical reasons, even though for triple-length LMKs each key is actually a triplet.

7.2.3 Local Master Key Triple DES Variant scheme

The Local Master Key Variants described in the previous section are used only to protect key usage. The HSM can also use the variant technique to provide additional protection to Triple-DES keys:

- To ensure that the Left and Right parts of a double-length Triple-DES key can only be used as such.
- To ensure that the Left, Middle and Right parts of a triple-length Triple-DES key can only be used as such.

The following variants are used for this purpose:

- Double-length key:
 - Left part – A6
 - Right part – 5A
- Triple-length key:
 - Left part – 6A
 - Middle part – DE
 - Right part – 2B

Key Scheme tags are used to identify the technique used to encrypt keys.

- 'U' is used to identify double-length Triple DES keys that are encrypted using the Triple-DES variant Scheme
- 'T' is used to identify triple-length Triple DES keys that are encrypted using the Triple-DES variant scheme.

The example below demonstrates how a double-length MK-SMI is encrypted using this method. The test key to be encrypted is:

Test MK-SMI = F1F1 F1F1 F1F1 F1F1 C1C1 C1C1 C1C1 C1C1

1. Refer to the Key Type Table to select the appropriate LMK pair and variant for the type of key that you wish to encrypt. For example, for key type MK-SMI, LMK 28-29 Variant 2 is used (this is the LMK variant that was calculated in the example in section 15):

LMK 28-29 Variant 2 = 401A 1A1A 1A1A 1A1A 1C1C 1C1C 1C1C 1C1C

2. Select the appropriate variants to be applied to the encrypting key. In this case the MK-SMI is a double-length Triple-DES key, so the following variants should be used:

- To encrypt the left part of the MK-SMI – A6
- To encrypt the right part of the MK-SMI – 5A

3. To create the key with which to encrypt the left part of MK-SMI, Exclusive-OR A6 with the first byte of the right part of the LMK pair:

1C **XOR** A6 = BA

Key with which to encrypt left part of MK-SMI

= 01A 1A1A 1A1A 1A1A **BA**1C 1C1C 1C1C 1C1C

4. Use the key calculated in step 3 to encrypt the left part of the MK-SMI:

Key with which to encrypt left part of MK-SMI

= 401A 1A1A 1A1A 1A1A **BA**1C 1C1C 1C1C 1C1C

Left part of MK-SMI = F1F1 F1F1 F1F1 F1F1

Result of Triple-DES encryption is: 5178 C9D3 D105 2B15

5. To create the key with which to encrypt the right part of MK-SMI, Exclusive-OR 5A with the first byte of the right part of the LMK pair

1C **XOR** 5A = 46

Key with which to encrypt left part of MK-SMI

= 401A 1A1A 1A1A 1A1A **46**1C 1C1C 1C1C 1C1C

6. Use the key calculated in step 5 to encrypt the right part of the MK-SMI:

Key with which to encrypt right part of MK-SMI

= 401A 1A1A 1A1A 1A1A **46**1C 1C1C 1C1C 1C1C

Right part of MK-SMI = C1C1 C1C1 C1C1 C1C1

Result of Triple-DES encryption is:

BF6A EC45 8B4A 4564

The encrypted MK-SMI is the result of step 4 concatenated with the result of step 6:

5178 C9D3 D105 2B15 BF6A EC45 8B4A 4564

The example above can be demonstrated on an HSM by using the FK console command, with inputs as follows:

Offline-AUTH> **FK** <Return>

Enter key length [1,2,3]: **2** <Return>

Enter key type: **209** <Return>

Enter key scheme: **U** <Return>

Enter component type [X,H,T,E,S]: **X** <Return>

Enter number of components [1-9]: **2** <Return>

Enter component 1: **50505050505050505050505050505050** <Return>

Enter component 2: **A1A1A1A1A1A1A1A19090909090909090** <Return>

Encrypted key: U 5178 C9D3 D105 2B15 BF6A EC45 8B4A 4564

Key check value: 8357D9

When the Variants are applied to the standard test LMK set (see next section), the first bytes of the second key are as follows:

7.2.3.1 Double-length Variant LMK

LMK Pair	First byte of second key of the LMK				
	Double-length Key Scheme Tag "U"		Triple-length Key Scheme Tag "T"		
	1 of 2	2 of 2	1 of 3	2 of 3	3 of 3
04-05	F7	0B	3B	8F	7A
06-07	D6	2A	1A	AE	5B
14-15	57	AB	9B	2F	DA
16-17	A7	5B	6B	DF	2A
18-19	A7	5B	6B	DF	2A
20-21	A2	5E	6E	DA	2F
22-23	B6	4A	7A	CE	3B
24-25	B3	4F	7F	CB	3E
26-27	BF	43	73	C7	32
28-29	BA	46	76	C2	37
30-31	83	7F	4F	FB	0E
32-33	8F	73	43	F7	02
34-35	8A	76	46	F2	07
36-37	97	6B	5B	EF	1A
38-39	A7	5B	6B	DF	2A

7.2.3.2 Triple-length Variant LMK

LMK Pair	First byte of second key of the LMK				
	Double-length Key Scheme Tag "U"		Triple-length Key Scheme Tag "T"		
	1 of 2	2 of 2	1 of 3	2 of 3	3 of 3
04-05	CE	32	02	B6	43
06-07	68	94	A4	10	E5
14-15	BC	40	70	C4	31
16-17	94	68	58	EC	19
18-19	DA	26	16	A2	57
20-21	B6	4A	7A	CE	3B
22-23	07	FB	CB	7F	8A
24-25	0E	F2	C2	76	83
26-27	F8	04	34	80	75
28-29	A8	54	64	D0	25
30-31	79	85	B5	01	F4
32-33	6B	97	A7	13	E6
34-35	29	D5	E5	51	A4
36-37	7A	86	B6	02	F7
38-39	43	BF	8F	3B	CE

Note: the term "LMK Pair" and the associated numbering scheme is retained for historical reasons, even though for triple-length LMKs each key is actually a triplet.

When the HSM encrypts a key component under a variant of the LMK, an additional variant (0xFF) is applied to the first byte of the LMK. This is to prevent a single encrypted component from masquerading as an encrypted key.

7.3 Test Variant LMK

The values of the LMK pairs contained in the "Test LMK" Smartcard are shown in the table below. The two Passwords are also held in this device, and their values are also shown below.

The PIN for each Test LMK Smartcard is: **1 2 3 4**

7.3.1 Double-length Variant LMK

LMK		Key
00-01	01 01 01 01 01 01 01 01	79 02 CD 1F D3 6E F8 BA
02-03	20 20 20 20 20 20 20 20	31 31 31 31 31 31 31 31
04-05	40 40 40 40 40 40 40 40	51 51 51 51 51 51 51 51
06-07	61 61 61 61 61 61 61 61	70 70 70 70 70 70 70 70
08-09	80 80 80 80 80 80 80 80	91 91 91 91 91 91 91 91
10-11	A1 A1 A1 A1 A1 A1 A1 A1	B0 B0 B0 B0 B0 B0 B0 B0
12-13	C1 C1 01 01 01 01 01 01	D0 D0 01 01 01 01 01 01
14-15	E0 E0 01 01 01 01 01 01	F1 F1 01 01 01 01 01 01
16-17	1C 58 7F 1C 13 92 4F EF	01 01 01 01 01 01 01 01
18-19	01 01 01 01 01 01 01 01	01 01 01 01 01 01 01 01
20-21	02 02 02 02 02 02 02 02	04 04 04 04 04 04 04 04
22-23	07 07 07 07 07 07 07 07	10 10 10 10 10 10 10 10
24-25	13 13 13 13 13 13 13 13	15 15 15 15 15 15 15 15
26-27	16 16 16 16 16 16 16 16	19 19 19 19 19 19 19 19
28-29	1A 1A 1A 1A 1A 1A 1A 1A	1C 1C 1C 1C 1C 1C 1C 1C
30-31	23 23 23 23 23 23 23 23	25 25 25 25 25 25 25 25
32-33	26 26 26 26 26 26 26 26	29 29 29 29 29 29 29 29
34-35	2A 2A 2A 2A 2A 2A 2A 2A	2C 2C 2C 2C 2C 2C 2C 2C
36-37	2F 2F 2F 2F 2F 2F 2F 2F	31 31 31 31 31 31 31 31
38-39	01 01 01 01 01 01 01 01	01 01 01 01 01 01 01 01
Password 1		01 01 01 01 01 01 01 01
Password 2		NOWISTHETIMEFORA

The check value for the Double-length Variant Test LMK is: 268604

7.3.2 Triple-length Variant Test LMK

LMK		Key																							
00-01	D3 CB 07 68 76 A2 07 04	01 01 01 01 01 01 01 01	32 B9 7F 73 34 AE B6 5E																						
02-03	8A CD 34 CE F4 91 79 9D	F1 19 94 8F E5 E6 B6 9B	61 97 8A 40 D0 83 04 32																						
04-05	3D 80 AD C8 6D 83 97 2F	68 EC 6B 7A 23 25 DA 98	A2 23 6D 1A 89 9B 07 32																						
06-07	01 34 76 B6 F4 08 BA 6B	CE 45 4C 2C 6D A8 B3 5E	BA C2 4A E6 1F 43 70 49																						
08-09	B5 7A E3 58 A2 1A DA 89	19 C2 5E 9E F4 8A B3 01	61 C1 23 1A 8F C4 2A 38																						
10-11	6B F7 10 C1 DF 13 7C EC	7F B3 7F E9 38 F2 A7 3D	BA F2 C4 B5 9B FD 1C 54																						
12-13	51 DC F1 58 D6 CD 0E CE	A2 E9 BC 0B 1F 85 EF 8C	EA C8 10 A8 1A C8 A7 EA																						
14-15	89 B5 CB BA 43 80 C8 91	1A 6E 1A D6 61 1C 1C DA	94 68 E3 CB 1A 26 9E FB																						
16-17	B3 FB D3 4A 5E 51 EC 52	32 AD FE BA 32 0D 68 7C	7A 68 31 EF 25 58 C4 A7																						
18-19	C8 B9 49 D6 2C 57 9E A7	7C CD CE A1 D0 3D 9E 6B	F4 A1 E6 3B E5 85 8A 83																						
20-21	8F 32 B9 10 E9 D5 6E DF	10 02 FB B5 57 AB 8F 73	0D 34 4A B3 89 38 F2 FD																						
22-23	CD 34 89 FB 38 54 97 61	A1 31 1F CB 92 AE 54 B3	16 76 13 16 76 A8 76 DF																						
24-25	1A CB 8C C1 26 1F FE EA	A8 E9 EA 58 80 1A A7 85	46 20 91 85 AB 3D 89 37																						
26-27	67 AB EC 46 16 23 D3 70	5E 85 F8 2F 15 26 62 68	02 15 D3 2F 8A CE D5 91																						
28-29	CE 23 B0 98 B0 34 B6 CD	0E 08 CD FE 3D 08 B5 0D	4F 80 D3 83 9D 73 85 BF																						
30-31	FE 3E 64 3E 92 C1 23 D3	DF 89 B6 83 43 A2 61 6D	AB 61 10 C4 A7 9E EA AB																						
32-33	94 DF 13 58 3B B5 E3 1F	CD B6 B5 32 AE 6D A8 DC	0D A8 6B EF 34 C7 51 8A																						
34-35	80 76 7F FD 76 F1 CE 57	8F 1F EC 15 AE 3E 7F 10	16 38 80 D6 29 58 08 CD																						
36-37	BC FB D6 89 FE 86 15 E0	DC AD 8F 8F 49 F8 0D 61	3D EA 73 F2 EC C2 F2 7C																						
38-39	68 B6 3D 1A F8 73 D5 92	E5 1C 1F D5 80 C1 D3 A1	2C 85 32 2A 1F 07 D9 08																						
Password 1		D3 CB 07 68 76 A2 07 04																							
Password 2		01 01 01 01 01 01 01 01																							

Note: the term "LMK Pair" and the associated numbering scheme is retained for historical reasons, even though for triple-length LMKs each key is actually a triplet.

The check value for the Triple-length Variant Test LMK is: 665641

7.4 Variant Key Type Codes

Each key has a Key Type as defined in the following table. The relationship between Key Type and LMK Variant is shown in the Key Type Table in the next section.

Key Type	Key Type Code ¹	Key Type Code ²	Description
ZMK	000	000	Zone Master Key (also known as ZCMK)
ZMK (Comp)	100	100	Zone Master Key Component (legacy commands only)
KML	200	200	Master Load Key (Visa Cash)
KEKr	300	300	(AS 2805) Key Encryption Key of Recipient
KEKs	400	400	(AS 2805) Key Encryption Key of Sender
ZPK	001	001	Zone PIN Key
PVK	002	002	PIN Verification Key
PVVK	002	002	(OBKM) PVV Key
TPK	002	70D	Terminal PIN Key
PEK	002	70D	(AS 2805) PIN Encipherment Key
PEK	002	70D	(LIC031) PIN Encryption Key
TMK	002	80D	Terminal Master Key
KT	002	80D	(AS 2805) Transaction Key
TK	002	80D	(AS 2805) Terminal Key
KI	002	80D	(AS 2805) Initial Transport Key
KCA	002	80D	(AS 2805) Sponsor Cross Acquirer Key
KMA	002	80D	(AS 2805) Acquirer Master Key Encrypting Key
TKR	002	90D	Terminal Key Register
TMK1	102	102	(AS 2805) Terminal Master Key
TMK2	202	202	(AS 2805) Terminal Master Key
IKEY ^φ	302	302	Initial Key (DUKPT)
CK-ENK	30D	30D	(Issuing) Card Key for Cryptograms
CVK	402	402	Card Verification Key
CSCK	402	402	Card Security Code Key
CK-MAC	40D	40D	(Issuing) Card Key for Authentication
CK-DEK	50D	50D	(Issuing) Card Key for Authentication
KIA	602	602	(AS 2805) Acquirer Initialization Key
TAK	003	003	Terminal Authentication Key
TAKr	203	203	(AS 2805) Terminal Authentication Key of Recipient
TAKs	103	103	(AS 2805) Terminal Authentication Key of Sender
KML	105	105	(OBKM) Master Load Key
KML _{ISS}	105	105	(OBKM) Master Load Key for Issuer
KMX	205	205	(OBKM) Master Currency Exchange Key
KMX _{ISS}	205	205	(OBKM) Master Currency Exchange Key for Issuer
KMP	305	305	(OBKM) Master Purchase Key
KMP _{ISS}	305	305	(OBKM) Master Purchase Key for Issuer
KI _{S.5}	405	405	(OBKM) S5 Issuer Key
KM3L	505	505	(OBKM) Master Key for Load & Unload Verification

Key Type	Key Type Code ¹	Key Type Code ²	Description
KM3L _{ISS}	505	505	(OBKM) Master Key for Load & Unload Verification for Issuer
KM3X	605	605	(OBKM) Master Key for Currency Exchange Verification
KM3X _{ISS}	605	605	(OBKM) Master Key for Currency Exchange Verification for Issuer
KMAC _{S4}	705	705	(OBKM)
KMAC _{S5}	805	805	(OBKM)
KMAC _{ACQ}	905	905	(OBKM)
KMAC _{ACX}	905	905	(OBKM)
WWK	006	006	Watchword Key
KMAC _{UPD}	106	106	(OBKM)
KMAC _{MA}	206	206	(OBKM)
KMAC _{CI}	306	306	(OBKM)
KMAC _{ISS}	306	306	(OBKM)
KMSC _{ISS}	406	406	(OBKM) Secure Messaging Master Key
BKEM	506	506	(OBKM) Transport key for key encryption
BKAM	606	606	(OBKM) Transport key for message authentication
KEK	107	107	(Issuing) Key Encryption Key
KMC	207	207	(Issuing) Master Personalization Key
SK-ENC	307	307	(Issuing) Session Key for cryptograms and encrypting card messages
SK-MAC	407	407	(Issuing) Session Key for authenticating card messages
SK-DEK	507	507	(Issuing) Session Key for encrypting secret card data
KD-PERSO	507	507	(Issuing) KD Personalization Key
ZKA MK	607	607	Master key for GBIC/ZKA key derivation
MK-KE	807	807	(Issuing) Master KTU Encipherment key
MK-AS	907	907	(Issuing) Master Application Signature (MAC) key
ZAK	008	008	Zone Authentication Key
ZAKs	108	108	(AS 2805) Zone Authentication Key of Sender
ZAKr	208	208	(AS 2805) Zone Authentication Key of Recipient
BDK-1	009	009	Base Derivation Key (type 1)
MK-AC	109	109	Master Key for Application Cryptograms
MK-SMI	209	209	Master Key for Secure Messaging (for Integrity)
MK-SMC	309	309	Master Key for Secure Messaging (for Confidentiality)
MK-DAC	409	409	Master Key for Data Authentication Codes
MK-DN	509	509	Master Key for Dynamic Numbers
BDK-2	609	609	Base Derivation Key (type 2)
MK-CVC3	709	709	Master Key for CVC3 (Contactless)
BDK-3	809	809	Base Derivation Key (type 3)
BDK-4	909	909	Base Derivation Key (type 4)
ZEK	00A	00A	Zone Encryption Key
ZEKs	10A	10A	(AS 2805) Zone Encryption Key of Sender
ZEKr	20A	20A	(AS 2805) Zone Encryption Key of Recipient
DEK	00B	00B	Data Encryption Key

Key Type	Key Type Code ¹	Key Type Code ²	Description
TEK	00B	00B	(AS 2805) Terminal Encryption Key
TEKs	10B	10B	(AS 2805) Terminal Encryption Key of Sender
TEKr	10B	10B	(AS 2805) Terminal Encryption Key of recipient
TEK	30B	30B	Terminal Encryption Key
RSA-SK	00C	00C	RSA Private Key
HMAC	10C	10C	HMAC key
RSA-PK	00D	00D	RSA Public Key
TPK	002	70D	Terminal PIN Key
PEK	002	70D	(AS 2805) PIN Encipherment Key
TMK	002	80D	Terminal Master Key
KT	002	80D	(AS 2805) Transaction Key
TK	002	80D	(AS 2805) Terminal Key
KI	002	80D	(AS 2805) Initial Transport Key
KCA	002	80D	(AS 2805) Sponsor Cross Acquirer Key
KMA	002	80D	(AS 2805) Acquirer Master Key Encrypting Key
TKR	002	90D	Terminal Key Register

Notes:

- 1 This Key Type Code column applies when the security setting "Enforce key type 002 separation for PCI HSM compliance" has a value of 'N' – i.e. key separation is not PCI HSM complaint.
- 2 This Key Type Code column applies when the security setting "Enforce key type 002 separation for PCI HSM compliance" has a value of 'Y' – i.e. key separation is PCI HSM complaint.

7.5 Key Type Table

The payShield 10K HSM provides a set of commands for key generation, key export and key import. An export command is one that translates a key from LMK encryption to encryption under a transport key, for sending to another party. Import is the reverse, for receiving keys and translating to local storage. The Key Type Table (see below) controls 'permitted actions' for the console and host commands used to generate, import and export keys when the transport key is a DES/3DES key. When using other types of transport keys (e.g. RSA), consult the appropriate command reference manual to determine 'permitted actions' and authorization requirements.

Errors are reported when an action breaks the rules imposed by the table. For example:

29 : Key function not permitted

The Key Type table is automatically modified dependent on the value of the Security Setting

"Enforce key type 002 separation for PCI HSM compliance": a setting of 'N' indicates non-compliance with the requirements of PCI HSM, and a setting of 'Y' indicates compliance with the key separation requirements of PCI HSM.

7.5.1 Non PCI-HSM Compliant (for backwards compatibility)

Variant ⇨		0			1			2			3			4			5			6			7			8			9		
LMK ↓		G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I
Pair	Code																														
04 – 05	00	ZMK			ZMK (Comp)			KML			KEKr ¹			KEKs ¹																	
		A	A ⁶	A ⁷	U	A	U	U	A	U	U	A	U	U	A	U															

Variant ⇨		0			1			2			3			4			5			6			7			8			9		
LMK ↓		G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I
Pair	Code																														
06 – 07	01	ZPK PEK ^{4a}			Auth Para																										
		U	A	U																											
14 – 15	02	PVK TPK KEYVAL TMK TKR PEK ^{4b} KT ¹ DbTAB KCA ¹ KMA ¹ KI ¹ PEK ¹ TK ¹ PVVK ²			TMK1 ¹			TMK2 ¹			IKEY ^φ			CVK CCK						KIA						PPASN					
		U	A	U	U	A	U	U	A	U	U	A	U	U	A	U				U	A	U									
16 – 17	03	TAK			TAKs			TAKr																							
		U	A	U	U	A	U	U	A	U																					
18 – 19	04				DTAB ¹			IPB																							
20 – 21	05				KML			KMX			KMP			KI _{S,6}			KM3L			KM3X			KMAC _{S4}			KMAC _{S5}			KMAC _{ACQ}		
					U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U
22 – 23	06	WWK			KMAC _{UPD}			KMAC _{MA}			KMAC _{CI} KMAC _{CISS}			KMSC _{ISS}			BKEM			BKAM											
		U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U									
24 – 25	07				KEK			KMC			SK-ENC			SK-MAC			SK-DEK KD- PERSO			ZKA MK						MK-KE			MK-AS		
					U	A	A	U	A	A										U	A	U				U	A	U	U	A	U
26 – 27	08	ZAK			ZAKs			ZAKr																							
		U	A	U	U	A	U	U	A	U																					
28 – 29	09	BDK-1			MK-AC			MK-SMI			MK-SMC			MK-DAC			MK-DN			BDK-2			MK-CVC3 MK-DCVV			BDK-3			BDK-4		
		U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U
30 – 31	0A	ZEK			ZEKs1			ZEK1r																							
		U	A	A _{U2}	U			U																							
32 – 33	0B	DEK TEK ¹			TEKs			TEK _r			TEK																				
		U			U			U			U	A																			
34 – 35	0C	RSA-SK			HMAC																										
		A	A ⁵	A ⁵	U	A	U																								
36 – 37	0D	RSA-PK									CK-ENC			CK-MAC			CK-DEK						Note 3			Note 3			Note 3		
		A		A							U	A	U	U	A	U	U	A	U				U	A	U	U	A	U	U	A	U
38 – 39	0E	Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved		

Key Type Table 1 - "Enforce key type 002 separation for PCI HSM compliance"

^φIKEY is also known as IPEK.

7.5.2 PCI Compliant

Variant ⇄		0			1			2			3			4			5			6			7			8			9		
LMK ↓		G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I	G	E	I			
Pair	Code																														
04 – 05	00	ZMK			ZMK (Comp)			KML			KEK _r			KEK _s																	
		A	A ⁶	A ⁷	U	A	U	U	A	U	U	A	U	U	A	U															
06 – 07	01	ZPK PEK			Auth Para																										
		U	A	U																											
14 – 15	02	PVK PVVK			TMK ₁			TMK ₂			IKEY ^Φ			CVK CSCK			KIA						PPASN								
		U	A	U	U	A	U	U	A	U	U	A	U	U	A	U				U	A	U									
16 – 17	03	TAK			TAK _s			TAK _r																							
		U	A	U	U	A	U	U	A	U																					
18 – 19	04				DTAB			IPB																							
20 – 21	05				KML KML _{ISS}			KM _X KM _X _{ISS}			KMP KMP _{ISS}			KI _{S,5}			KM3L KM3L _{ISS}			KM3X KM3X _{ISS}			KMAC _{S4}			KMAC _{S5}			KMAC _{ACQ} KMAC _{ACK}		
					U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U			
22 – 23	06	WWK			KMAC _{UPD}			KMAC _{MA}			KMAC _{CI} KMAC _{ISS}			KM _{SC} _{ISS}			BKEM			BKAM											
		U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U									
24 – 25	07				KEK			KMC			SK-ENC			SK-MAC			SK-DEK KD-PERSO			ZKA MK						MK-KE			MK-AS		
					U	A	A	U	A	A										U	A	U				U	A	U	U	A	U
26 – 27	08	ZAK			ZAK _s			ZAK _r																							
		U	A	U	U	A	U	U	A	U																					
28 – 29	09	BDK-1			MK-AC			MK-SMI			MK-SMC			MK-DAC			MK-DN			BDK-2			MK-CVC3 MK-DCVV			BDK-3			BDK-4		
		U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U	U	A	U			
30 – 31	0A	ZEK			ZEK _s			ZEK _r																							
		U	A	A ₂	U			U																							
32 – 33	0B	DEK TEK			TEK _s			TEK _r			TEK																				
		U			U			U			U	A																			
34 – 35	0C	RSA-SK			HMAC																										
		A	A ⁵	A ⁵	U	A	U																								
36 – 37	0D	RSA-PK									CK-ENC			CK-MAC			CK-DEK			DbTAB ¹			TPK KEYVAL PEK PEK			TMK KT KCA KMA KI T			TKR		
		A		A							U	A	U	U	A	U	U	A	U				U	A	U	U	A	U	U	A	U
38 – 39	0E	Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved			Reserved		

Key Type Table 2 - "Enforce key type 002 separation for PCI HSM compliance" set to 'Y' (i.e. key separation is PCI HSM compliant)

^ΦIKEY is also known as IPEK.

The tables above show the actions that can be applied to each specific LMK pair/variant. For each key type, the 3 boxes below the key type refer, from left to right, to:

G = Generate

E = Export

I = Import

Each of these 3 boxes contains one of the following entries to define permissions:

blank = Not allowed

A = allowed only in Authorized State

U = allowed Unconditionally, i.e. without Authorized State

Notes:

1. DTAB = Decimalization Table; DbTAB = Diebold Table
2. Authorization requirements for Import are dependent on the security setting "Enable ZEK encryption of ASCII data or Binary data or None".
3. Keys of this key type can be generated/imported/exported, but may only be used for other operations when the security setting "Enforce key type 002 separation for PCI HSM compliance" is set to 'Y'. This allows users to pre-generate their keys before migrating to a PCI-compliant environment.
4. The term "LMK pair" and the associated numbering scheme is retained for historical reasons, even though for triple-length LMKs each key is actually a triplet.
5. If the security setting "Enable import and export of RSA Private keys" is set to "YES", otherwise, this operation is not allowed.
6. If the security setting "Enable export of a ZMK" is set to "YES", otherwise, this operation is not allowed.
7. If the security setting "Enable import of a ZMK" is set to "YES", otherwise, this operation is not allowed.

Not all key type codes are available in all commands for security reasons.

The Key Type code used within commands is formed by using the Variant code as the first character then the LMK pair code as the second character. For example, the code for a ZPK is 001.

The payShield 10K HSM provides a set of commands for key generation, key export and key import. An export command is one that translates a key from LMK encryption to encryption under a ZMK or an RSA public key, for sending to another party. Import is the reverse, for receiving keys and translating to local storage. The Key Type Table controls 'permitted actions' for the console and host commands used to generate, import and export keys.

Errors are reported when an action breaks the rules imposed by the table.

For example:

29 : Key function not permitted

8 Key Block LMK Key Scheme

8.1 Introduction

A Key Block LMK is a more recent type of local master key, and is used to encrypt keys in a key block format. It is not compatible with a Variant LMK, and it can only be used to encrypt keys in key block form.

When using a Key Block LMK, working keys are encrypted as a Thales key Block.

Two types of Thales Key Block LMKs are supported in the payShield 10K:

- DES Key Block LMK - based on a triple-length 3-DES key. This key provides a security strength of 112-bits, and can be used to protect subordinate DES, TDES, RSA (up to 2048 bits), and HMAC keys.
- AES Key Block LMK - based on a 256-bit AES key. This key provides a security strength of 256-bits, and can be used to protect subordinate AES, DES, TDES, RSA, ECC & HMAC keys.

Notes:

- The term “Key Block LMK” refers to the 'key block' method of encrypting keys; a Key Block LMK is not itself stored in key block form.
- AES key block LMKs must be used to protect AES keys and RSA keys longer than 2048 bits and ECC keys.
- The tables in this chapter use blue shading to indicate that the information relates to keys encrypted using key blocks LMKs.

8.2 Relationship between Thales Key Blocks and TR-31 Key Blocks

Thales Key Blocks are similar to the Key Blocks as defined by the ANSI X9 committee in Technical Report 31 (TR-31), which are offered as a way of meeting the requirements of ANSI X9.24 when exporting keys between products from different vendors. However there are differences between Thales Key Blocks and TR-31 Key Blocks and the ways in which they are used.

The main difference between the Thales and TR-31 Key Blocks is that the TR-31 Key Block is less flexible in terms of Header values and supports only a small number of types of Optional Header blocks.

The TR-31 Key Block allows key exchange between HSMs from multiple vendors which have implemented the TR-31 specification.

The Thales Key Block is a superset of the TR-31 Key Block and can also be used for the secure exchange of keys between HSMs. However, because the Thales Key Block includes proprietary extensions it can be used to transfer keys only between Thales payment HSMs.

The Thales and TR-31 Key Blocks can be supported and utilized independently of one another. The combination of Thales and TR-31 Key Blocks defines a “trusted” HSM environment, in which key manipulation is practically infeasible.

8.3 Support for Thales Key Blocks

Not all console commands and host commands can be used with Key Block LMKs. Most commands that do not support the use of a Key Block LMK have a newer, alternative command that should be used in their place, or relate to functionality in optional licenses. You should check for Key Block support in the descriptions of all commands you intend to use in the appropriate documentation, e.g.,:

- *payShield 10K Console Guide*
- *payShield 10K Host Command Reference Manual*

8.4 Key Block Format

The Thales Key Block is denoted by key scheme "S" and has the following format:

Key Scheme Tag ("S") (1 byte)	Key Block Header (16 ASCII characters)	Optional Header (ASCII characters, variable length)	Encrypted Key Data (variable length, ASCII encoded)	Authenticator (8 or 16 ASCII characters)
---	--	---	---	--

	a 16-byte (clear) Key Block Header , which defines the key usage (e.g. Visa PVV) and mode of use (e.g. verification only), the algorithm with which the key is used (e.g. AES 256-bit) and the limitations on the exportability of the key (e.g. no export permitted); the Header also identifies the LMK used to encrypt the key in the Key Block;
	a (clear) Optional Header block, which can be used (for example) to define the period of validity of the key contained in the Key Block;
	the Encrypted Key Data , which includes the actual key itself, encrypted under the "encryption variant" of the identified LMK (or ZMK/TMK);
	a Key Block Authenticator , calculated using the "authentication variant" of the identified LMK (or ZMK/TMK); the use of the Authenticator prevents unauthorized modification to the Key Block.

The entire key block will be ASCII encoded.

8.4.1 Key Block Header

The Thales Key Block Header is 16 (ASCII) bytes in length and has the following format:

Byte(s)	Field	Comments
0	Version ID	value = "0" (X'30) when protected by a 3-DES key value = "1" (X'31) when protected by an AES key
1-4	Key Block Length	total length of key block
5-6	Key Usage	e.g. key encryption, data encryption
7	Algorithm	e.g. DES, 3-DES, AES
8	Mode of Use	e.g. encrypt only
9-10	Key Version Number	e.g. version of key in the key block or used to indicate that the key is a key component
11	Exportability	e.g. exportable under a trusted key
12-13	Number of optional blocks	number of Optional Header Blocks
14-15	LMK ID	LMK identifier (to support multiple LMKs); numeric values "00" - "19" (X'3030 - X'3139)

8.4.1.1 Key Block Length (Bytes 1-4)

Bytes 1-4 of the Header contain the length of the entire key block, namely Header, Optional Header Blocks, encrypted Key Data and the Authenticator. The length of the key block is calculated after encoding and is represented as 4 numeric (ASCII) digits.

For example, if the total key block length is 112 characters (bytes) then the value in byte 1 will be "0", the value in bytes 2 and 3 will be "1" and the value in byte 4 will be "2" (i.e. X'30313132).

8.4.1.2 Key Usage (Bytes 5-6)

Bytes 5-6 of the Header define the primary usage of the key contained in the key block. The following table defines the usage code. The table also indicates whether the usages applies to TR-31 Key Blocks:

- A blank entry means the usage is not appropriate to TR-31.
- An entry of 'Y' means that the usage is appropriate to TR-31.
- An entry of the form ("XX") means that the usage is not appropriate to TR-31 Key Blocks, but that when exporting from Thales Key Block format to TR-31 format the usage code is converted to the code in brackets.

Thales KB Key Usage	Equivalent TR-31 KB Key Usage	Algorithm	Description
'01'	-	DES/3DES	WatchWord Key (WWK)
'02'	-	RSA	RSA Public Key
'03'	-	RSA	RSA Private Key (for signing/key mgt)
'04'	-	RSA	RSA Private Key (for ICCs)
'05'	-	RSA	RSA Private Key (for PIN translation)
'06'	-	RSA	RSA Private Key (for TLS pre-master secret decryption)
'B0'	'B0'	3DES/AES	Base Derivation Key (BDK-1)
'41'	'B0'	3DES/AES	Base Derivation Key (BDK-2)
'42'	'B0'	3DES	Base Derivation Key (BDK-3)
'43'	'B0'	3DES/AES	Base Derivation Key (BDK-4)
'44'	'B0'	3DES	Base Derivation Key (BDK-5)
'B1'	'B1'	3DES/AES	DUKPT Initial Key (IKEY [®])
'C0'	'C0'	DES/3DES	Card Verification Key
'11'	'C0'	DES/3DES	Card Verification Key (American Express CSC)
'12'	'C0'	DES/3DES	Card Verification Key (Mastercard CVC)
'13'	'C0'	DES/3DES	Card Verification Key (Visa CVV)
'D0'	'D0'	DES/3DES/AES	Data Encryption Key (Generic)
'21'	'D0'	DES/3DES/AES	Data Encryption Key (DEK)
'22'	'D0'	DES/3DES/AES	Data Encryption Key (ZEK)
'23'	'D0'	DES/3DES/AES	Data Encryption Key (TEK)
'24'	-	AES	Key Encryption Key (Transport Key)
'25'	'D0'	AES	CTR Data Encryption Key (CTRDEK)

Thales KB Key Usage	Equivalent TR-31 KB Key Usage	Algorithm	Description
'E0'	'E0'	3DES/AES	EMV/Chip card Master Key: Application Cryptogram (MK-AC)
'E1'	'E1'	3DES	EMV/Chip card Master Key: Secure Messaging for Confidentiality (MK-SMC)
'E2'	'E2'	3DES/AES	EMV/Chip card Master Key: Secure Messaging for Integrity (MK-SMI)
'E3'	'E3'	3DES	EMV/Chip card Master Key: Data Authentication Code (MK-DAC)
'E4'	'E4'	3DES	EMV/Chip card Master Key: Dynamic Numbers (MK-DN)
'E5'	'E5'	3DES	EMV/Chip card Master Key: Card Personalization
'E6'	'E6'	3DES	EMV/chip card Master Key: Other
'E7'	-	3DES	EMV/Master Personalization Key
'31'	'E6'	3DES	Visa Cash Master Load Key (KML)
'32'	'E6'	3DES	Dynamic CVV Master Key (MK-CVC3)
'33'	-	AES	Mobile Remote Management Master key for message confidentiality (M_KEY_CONF)
'34'	-	AES	Mobile Remote Management Master key for message integrity (M_KEY_MAC)
'35'	-	AES	Mobile Remote Management Session key for message confidentiality (MS_KEY_CONF)
'36'	'M3'	AES	Mobile Remote Management Session key for message integrity (MS_KEY_MAC)
'37'	-	3DES	EMV Card Key for cryptograms
'38'	-	3DES	EMV Card Key for integrity
'39'	-	3DES	EMV Card Key for encryption
'40'	-	3DES	EMV Personalization System Key
'47'	-	3DES/AES	EMV Session Key for cryptograms
'48'	-	3DES/AES	EMV Session Key for integrity
'49'	-	3DES	EMV Session Key for encryption
'I0'	'I0'	-	Initialization Value
'K0'	'K0'	DES/3DES/AES	Key Encryption / Wrapping Key (Generic)
'51'	'K0' / 'K1'	DES/3DES/AES	Terminal Key Encryption (TMK)

Thales KB Key Usage	Equivalent TR-31 KB Key Usage	Algorithm	Description
'52'	'K0' / 'K1'	DES/3DES/AES	Zone Key Encryption (ZMK)
'53'	-	3DES	ZKA Master Key
'54'	-	3DES/AES	Key Encryption Key (KEK)
'55'	-	AES	Key Encryption Key (Transport Key)
'M0'	'M0'	3DES	ISO 16609 MAC algorithm 1 (using 3-DES)
'M1'	'M1'	DES/3DES	ISO 9797-1 MAC algorithm 1
'M2'	'M2'	DES/3DES	ISO 9797-1 MAC algorithm 2
'M3'	'M3'	3DES	ISO 9797-1 MAC algorithm 3
'M4'	'M4'	DES/3DES	ISO 9797-1 MAC algorithm 4
'M5'	-	AES	AES CBC MAC
'M6'	'M5'	AES	AES CMAC
'61'	-	HMAC	HMAC key (using SHA-1)
'62'	-	HMAC	HMAC key (using SHA-224)
'63'	-	HMAC	HMAC key (using SHA-256)
'64'	-	HMAC	HMAC key (using SHA-384)
'65'	-	HMAC	HMAC key (using SHA-512)
'P0'	'P0'	DES/3DES/AES	PIN Encryption Key (Generic)
'71'	'P0'	DES/3DES/AES	Terminal PIN Encryption Key (TPK)
'72'	'P0'	DES/3DES/AES	Zone PIN Encryption Key (ZPK)
'73'	P0'	DES/3DES	Transaction Key Scheme Terminal Key Register (TKR)
'V0'	'V0'	DES/3DES	PIN Verification Key (Generic)
'V1'	'V1'	DES/3DES	PIN Verification Key (IBM 3624 algorithm)
'V2'	'V2'	DES/3DES	PIN Verification Key (Visa PVV algorithm)

□ KEY is also known as IPEK.

8.4.1.3 Algorithm (Byte 7)

Byte 7 of the Header defines the cryptographic algorithm with which the key contained in the key block will be used. The following values are defined, although only four of these values are currently supported:

Value	Hex	Algorithm
'A'	X'41	AES

'D'	X'44	DES
'E'	X'45	Elliptic curve (included for future reference)
'H'	X'48	HMAC
'R'	X'52	RSA
'S'	X'53	DSA (included for future reference)
'T'	X'54	3-DES

8.4.1.4 Mode of Use (Byte 8)

Byte 8 of the Header defines the operation that the key contained in the key block can perform.

Value	Hex	Description
'B'	X'42	The key may be used to perform both encrypt and decrypt operations.
'C'	X'43	The key may be used to perform MAC calculation (both generate & verify) operations.
'D'	X'44	The key may only be used to perform decrypt operations.
'E'	X'45	The key may only be used to perform encrypt operations.
'G'	X'47	The key may only be used to perform MAC generate operations.
'N'	X'4E	No special restrictions apply.
'S'	X'53	The key may only be used to perform digital signature generation operations.
'V'	X'56	The key may only be used to perform digital signature verification operations.
'X'	X'58	The key may only be used to derive other keys.

The "C", "G" and "V" field values will apply to HMAC keys, as well as "ordinary" MAC keys.

The mode of use corresponding to the above values will be extended to include other key usage, as follows:

Value	Hex	Mode of Use
'B'	X'42	Both encryption and decryption
'C'	X'43	Both generate and verify
'D'	X'44	Decrypt only
'E'	X'45	Encrypt only
'G'	X'47	Generate only
'N'	X'4E	No special restrictions or not applicable
'V'	X'56	Verify only
'X'	X'58	Derivation only

For example, if, in a particular key block, bytes 5-8 have the values "11TV", this indicates that the key in the key block is a 3-DES key that can only be used for the verification of American Express CSCs. Similarly, if bytes 5-8 have the values "52TE" then the key in the key block is a 3-DES ZMK that can only be used for key export and if bytes 5-8 have the values "01DN" then the key is a single-length WatchWord key that can be used for both response generation and verification.

8.4.1.5 Key Version Number (Bytes 9-10)

Bytes 9-10 of the Header define the version number of the key contained in the key block or that the key is actually a key component. The following values will be supported:

Value (byte 9)	(byte 10)	Hex	Key Version Number
0	0	X'3030	Key versioning is not used for this key
c	Any	X'6300	A "c" in byte 9 indicates that the key carried in the key block is a key component.
Any other combination of printable characters			The version of the key or key component carried in the key block

Note: If byte 9 is set to "c" then byte 10 will be used to indicate the component number. For example, if bytes 9-10 = "c2" (X'6332) then this indicates that the key in the key block is the second component of a key. Note, however, that this gives no information regarding the total number of key components.

8.4.1.6 Exportability (Byte 11)

Byte 11 of the Header defines the conditions under which the key contained in the key block can be exported outside the cryptographic domain in which the key is found. A key is defined to be trusted if it is in either Thales Key Block or TR-31 Key Block format.

Any other format key is said to be untrusted. The following values will be supported.

Value	Hex	Exportability
'E'	X'45	May only be exported in a trusted key block, provided the wrapping key itself is in a trusted format.
'N'	X'4E	No export permitted.
'S'	X'53	Sensitive; all other export possibilities are permitted, provided such export has been enabled (existing Authorized State requirements remain).

"Sensitive" export includes, for example, when a Key Block LMK is used and the key is exported in ANSI X9.17 format. Export in X9.17 format is disabled by default in the payShield 10K security settings and must be specifically enabled (e.g. via the CS console command) if required.

Note: When a key block is exported in a Thales or TR-31 Key Block format, byte 11 in the exported key block dictates how further export must be handled by the recipient of the key block. Hence, if the received byte 11 has value "E" then further "trusted" export is permitted, but if byte 11 has value "N" then no further export will be permitted. Such considerations must be taken into account when the key is initially generated. It will be possible to change the value of byte 11 from "E" or "S" to "N" (so that no further export will be permitted), via an optional field at the end of the command message.

8.4.1.7 Number of Optional Blocks (Bytes 12-13)

The Thales Key Block format allows a key block to contain up to 99 Optional Header Blocks which can be used to include additional (optional) data within a Thales Key Block. Optional Header Blocks are described below.

Bytes 12-13 of the Header specify the number of Optional Header Blocks in the key block. A value of "00" (X'3030) indicates there are no optional blocks. A value of "12" (X'3132) indicates that there are 12 optional blocks.

8.4.1.8 LMK Identifier (Bytes 14-15)

Bytes 14-15 of the Header identify the LMK used to encrypt and authenticate the key block. Such identification is required to support multiple LMKs. The LMK Identifier field may take numeric values "00" to "19". The number of LMKs that can be loaded into the HSM is determined by the HSM's license file.

Note: A security setting exists to instruct the HSM to ignore the LMK Identifier stored in the header of Thales Key Blocks. This allows users to share Thales Key Blocks across multiple HSMs without requiring the LMK to be loaded into the same "slot" in all their HSMs.

Example

As an example, a Key Block Header of

00072**V**2**T****G**22**N**0033

indicates:

- a Key Block of 72 bytes,
- that the key contained in the Key Block may only be used with the Visa PVV algorithm (the "**V**2" field),
- that it is a 3-DES key (the "**T**" field),
- that it may only be used for PVV generation ("**G**"),
- The key version number is "22",
- the key may not be exported (the "**N**" field),
- there are no Optional Header blocks in the Key Block, and
- the Key Block has been encrypted/authenticated using the LMK with identifier "33".

With this example, the use of the Key Block is highly restricted. It cannot even be used to verify PVVs. Users must take considerable care when generating Key Blocks to ensure that the correct Header fields are created. The CS host command allows some modifications to Header fields - but only modifications that further restrict the use of the Key Block.

Carrying on with the above example, the Key Block may have been originally generated with the Mode of Use field (byte 8) set to "N", meaning that the key could be used for PVV generation or verification, and the Exportability field (byte 11) set to "E", meaning that "trusted" export of the key was permitted. Following export of the key, it would exist in two places, namely the PVV generation system and the PVV verification system. Once there, byte 8 could be changed to "G" or "V", respectively, and byte 11 set to "N". Thereafter the Header values cannot be changed any further.

8.4.2 Optional Header

An Optional Header Block has the following structure:

Byte(s)	Field	Comments
0-1	Identifier	Optional Header Block identifier.
2-3	Length	Optional Header Block length; this field contains the length in bytes of the complete Optional Header Block, represented as a hexadecimal value and ASCII encoded; for example, if the overall length is 24 (decimal), then this is represented as X'18 and encoded as X'3138; if an Optional Header Block contains no data then the length field contains the value X'04 (encoded as X'3034).
4-n (n ≤ 251)	Data	Optional Header Block data

Thus the maximum length of an Optional Header block is 255 bytes (and the minimum length is 4 bytes). The overall length of all the Optional Header Blocks must be a multiple of the encryption block length (8 bytes in the case of 3-DES), which is achieved (if necessary) by the inclusion of a "Padding" block that must be the last Optional Header Block.

Note: In theory, a Key Block may have a maximum of 99 optional blocks. In practice, however, a Key Block may not possess more than one optional block with the same identifier. If a key block (Thales or TR-31) contains more than one Optional Header Block with the same identifier then the HSM will return an error:

BC – Repeated optional block

This error will be returned if any attempt is made to use a host command to generate, import or export such a key block.

8.4.2.1 Optional Header Block Types

The following Optional Header Blocks will be supported in the Thales Key Block. The first three Optional Header Blocks are also supported by TR-31:

Optional Header Block ID	Hex	Field Name	Comments
KS	X'4B53	Key Set Identifier	See examples in ANSI X9.24 part 3.
KV	X'4B56	Key Block version	Used to define the version of the set of key block field values and that the key block contains provisional values not yet approved by ANSI. This field must contain 4 printable ASCII characters.
PB	X'5042	Padding block	Used to ensure that the overall length of the Optional Header Blocks is a multiple of the encryption block length; the data field is filled with readable (random) ASCII characters; if used, the Padding block must be the last Optional Header Block.

Optional Header Block ID	Hex	Field Name	Comments
00	X'3030	Key Status	Key status; permitted values: "E" (Expired) "L" (Live) "P" (Pending) "R" (Revoked) "T" (Test) See notes below on permitted Key Status transitions.
01	X'3031	Key Block Encryption	Algorithm and mode used to encrypt the Key Data in the key block; only permitted value: "00" (current mechanism).
02	X'3032	Key Block Authentication	Algorithm and mode used to authenticate the key block; only permitted value: "00" (current mechanism).
03	X'3033	Start Date/Time	Date and time from which the key block is valid; format: YYYY:MM:DD:HH
04	X'3034	End Date/Time	Date and time after which the key block is invalid; format: YYYY:MM:DD:HH
05	X'3035	Text	Any combination of printable characters; a zero length data field will not be permitted

The "KS", "KV" and "PB" optional blocks are inherited from the TR-31 standard, but the remaining optional block types are proprietary to the Thales key block. The "01" and "02" optional blocks are for future-proofing, in the event that the LMK is changed to another algorithm (as happened with AES).

For example, a key block that contains a "live" key could have an Optional Header Block structure as follows (the space is to aid readability and is not part of the Optional Header Block):

```
0005L PB0Brrrrrr
```

In this case, the "Number of Optional Blocks" field in the key block header (bytes 12-13) will contain the value "02" (X'3032).

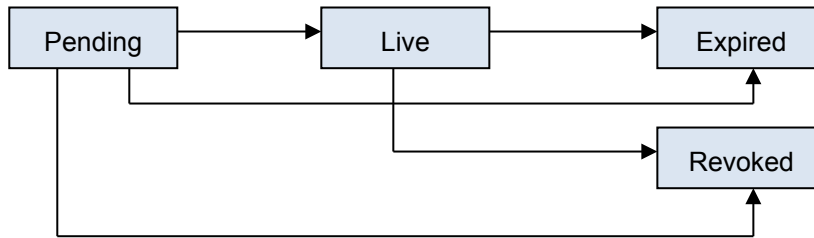
The Padding block (containing 7 random padding characters in this case) is required to ensure that the total length of the Optional Header Block is a multiple of 8 bytes. Note that any optional block must be at least 4 bytes in length.

Notes:

- The order in which optional blocks appear in a key block is immaterial, except that "PB" Padding block must be the last optional block.
- Optional Header Blocks with identifiers "01" and "02" will not normally be used, but are included in the event that multiple encryption and authentication modes may be required in the future. If they are used then they must have data value "00".

8.4.3 Permitted Key Status transitions

The following transitions (only) will be permitted for Key Status:



Notes:

- The host command CS may be used to modify the status of a key.
- The HSM will not permit the use of a key that is marked as "Pending", "Expired" or "Revoked".
- Keys with status "Test" or "Live" may be used as normal by an HSM.

8.5 Encrypted key data

The only part of the key block that is encrypted is the Key Data, which contains the actual key stored in the key block. The key types that may be protected in a Thales Key Block are DES and TDES keys, HMAC keys, AES keys, and RSA public and private keys. Note that an RSA public key is not encrypted, but the key block is still authenticated.

The encryption algorithm used to protect the Key Data depends on the specific Key Block scheme being used:

- When using a DES Key Block LMK, the Key Data portion of the key block is encrypted using 3-DES Cipher Block Chaining (CBC), using bytes 0-7 of the Header as the Initialization Vector (IV). The encryption key will be a variant of the LMK.
- When using an AES Key Block LMK, the Key Data portion of the key block is encrypted using AES Cipher Block Chaining (CBC), using bytes 0-15 of the Header as the Initialization Vector (IV). The encryption key will be cryptographically derived from the LMK.

Note: An Optional Header Block will be reserved so that other encryption algorithms and modes of encryption can be supported in the future.

The Key Data block has the following format:

Field	Length	Notes
Key Length	2 bytes	Contains the length in bits of the key that is to be encrypted (see next field); the length is written as a 16-bit binary number; for example, if the key is a 192-bit (triple-length) 3-DES key then this field contains the value X'00C0.
Key	variable, depending on key that is being encrypted	Contains the key data, in binary format; for example, a 192-bit (triple-length) 3-DES key would be represented as 24 bytes.
Padding	Variable	Contains random padding, used to ensure that the length of the entire Key Data block is a multiple of the block length of the encrypting key; for example, if a 3-DES key is used as the encryption key then the Key Data block must be a multiple of 8 bytes, so with the examples above the padding field could contain 6, 14, 22,... bytes; the padding field can be used to disguise the true length of the key in the key block, if required.

A security setting (e.g. in the CS console command) allows users to ensure that all double-length keys in Thales Key Blocks are padded with an additional 8 bytes of random values to disguise the length of the key. If this option is not selected then the minimum padding necessary will be applied in all cases.

Note: The minimum padding necessary for AES, RSA and HMAC keys will always be applied, regardless of this security setting.

8.6 Authenticator

The key block Authenticator ensures the integrity of the key block, and is calculated over the Header, Optional Header Blocks and the Key Data. The authentication algorithm used depends on the specific Key Block LMK being used:

- When using a DES Key Block LMK, the key block Authenticator is calculated using a 3-DES CBC-MAC, with a zero IV. (No padding is required, as the data to be authenticated is always a multiple of 8 bytes in length.) The leftmost 4 bytes of the result will be used as the Authenticator. The authentication key will be a variant of the LMK.
- When using an AES Key Block LMK, the key block Authenticator is calculated using an AES CMAC over the clear key block. The leftmost 8 bytes of the result will be used as the Authenticator. The authentication key will be cryptographically derived from the LMK.

Note: The Key Scheme Tag "S" is not included in the authenticated data.

8.7 Key Block Local Master Keys (LMKs)

A Key Block LMK is either a triple-length DES key, or a 256-bit AES key, and is used to encrypt keys in a key block format. It is not compatible with a Variant LMK, and it can only be used to encrypt keys in key block form.

Note: The term "Key Block LMK" refers to the 'key block' method of encrypting keys; a Key Block LMK is not itself stored in key block form.

8.7.1 3DES Key Block Test LMK

The value of the LMK contained in the 3DES Key Block Test LMK smartcard is shown in the table below. The two Passwords are also held in this device, and their values are also shown below.

The PIN for the 3DES Key Block Test LMK smartcard is: 1 2 3 4

Field	Length	Notes
Key Length	2 bytes	Contains the length in bits of the key that is to be encrypted (see next field); the length is written as a 16-bit binary number; for example, if the key is a 192-bit (triple-length) 3-DES key then this field contains the value X'00C0.
Key	variable, depending on key that is being encrypted	Contains the key data, in binary format; for example, a 192-bit (triple-length) 3-DES key would be represented as 24 bytes.
Padding	Variable	Contains random padding, used to ensure that the length of the entire Key Data block is a multiple of the block length of the encrypting key; for example, if a 3-DES key is used as the encryption key then the Key Data block must be a multiple of 8 bytes, so with the examples above the padding field could contain 6, 14, 22,.. bytes; the padding field can be used to disguise the true length of the key in the key block, if required.

The check value is 165126.

8.7.2 AES Key Block Test LMK

The value of the LMK contained in the AES Key Block Test LMK smartcards is shown in the table below. The AES Key Block Test LMK can only be used in smartcard authentication mode, so password authentication is not supported.

The PIN for each AES Key Block Test LMK smartcard is: 1 2 3 4

LMK	9B 71 33 3A13 F9 FA E7 2F 9D 0E 2D AB 4A D6 78 47 18 01 2F 92 44
-----	--

	03 3F 3F 26 A2 DE 0C 8A A1 1A
--	-------------------------------

The check value is 9D04A0.

8.8 Console Command examples

8.8.1 CK Console Command (Generate Check Value)

In this first example, the CK console command is used to generate a check value for a key contained in a Key Block. No additional fields are required.

```
Online> CK <Return>
Enter LMK id: 03 <Return>
Enter Key Block: S 00072V2TG22N0003xx.....xxxxx <Return>

Key check value: cccccc

Online>
```

8.8.2 KG Console Command (Generate Key)

In this example, the console KG command is used to generate a non-exportable, double length Base Derivation Key (BDK) used in the DUKPT scheme. The Key Block will contain start and end date/time optional blocks.

```
Online> KG <Return>
Enter LMK id: 07 <Return>
Enter key length [1,2,3]: 2 <Return>
Enter key scheme (LMK): S <Return>
Enter key scheme (ZMK): <Return>
Enter ZMK: <Return>
Enter key usage: B0 <Return>
Enter mode of use: N <Return>
Enter key version number: 00 <Return>
Enter exportability: N <Return>
Enter optional blocks? [Y/N]: Y <Return>
Enter optional block identifier: 03 <Return>
Enter optional block data: 2005:12:21:00 <Return>
Enter more optional blocks? [Y/N]: Y <Return>
Enter optional block identifier: 04 <Return>
Enter optional block data: 2007:12:21:00 <Return>
Enter more optional blocks? [Y/N]: N <Return>

Key under LMK:
S 00112B0TN00N030703112005:12:21:0004112007:12:21:00PB06rrxx.....xxxxx
Key check value: cccccc

Online>
```

In the above example, the Key Usage value "B0" indicates that the key is a BDK and the Mode of Use value "N" means that there are no specific restrictions on the use of the key.

8.9 Host Commands

Note: All Host commands are disabled by default.

8.9.1 Host Command Examples

Refer to the *payShield 10K Host Command Reference Manual* for examples of Host Commands and Responses when key block LMKs are being used.

8.9.2 Error Codes

Implementation of Key Blocks is a complex matter, so a large number of error codes are provided to aid development

and testing. For example:

- In an A6 command where a Key Usage of 72 has been specified, if a Mode of Use value other than "N" was input an error code would be generated (as "N" is the only permitted value when the Key Usage is "72").
- Similarly, if an Exportability value "E" was supplied then an error would be generated (since the imported key is untrusted).

Some of the more obvious (generic) key block error conditions include key block authentication failure, the use of incompatible key schemes or LMK identifiers, the attempted use of an expired or revoked key and the attempted use of an exportability mode that is not permitted. In addition, when used with specific commands only certain Header values are allowed; an invalid Header will cause an error code or error message to be returned.

9 Multiple LMKs

9.1 Introduction

The LMK (Local Master Key) on the payShield 10K is used to protect (by encryption) all of the operational keys plus some additional sensitive data that are processed by the HSM.

The payShield has the capability of Multiple LMKs, such that up to 20 LMKs of different types can be in use at any one time. Each LMK can be managed by a separate security team, allowing a single payShield 10K to be used for multiple purposes – such as different applications or different clients.

9.2 The Need for Multiple LMKs

9.2.1 About LMKs

The LMK is stored in the tamper-resistant memory of the HSM. All other cryptographic keys used in the system are encrypted under the LMK and are usually stored externally to the HSM (See Chapter titled “*User Storage*”), usually in a key database on the host system that is accessible by host applications. The LMK may be common to a number of HSMs, so that applications do not need to concern themselves with the particular HSM used for cryptographic processing. Since there is only a single key stored in the HSM, in the event of a problem with the device, recovery can be effected quickly and with minimal disruption to operations.

There are two types of LMK:

- Variant LMK
- Key Block LMK Multiple LMKs

9.2.2 Multiple LMKs

The LMK mechanism is simple, easy to understand and secure, but it does have some limitations. In particular, the use of a single LMK within an HSM (or cluster of HSMs) means that it is not possible to achieve cryptographic isolation of applications that are calling the HSM. This may breach the requirements of some PCI security standards and may be a cause for other concerns.

For example:

- In a bureau situation, there is generally a need to support keys from many different clients, yet maintain cryptographic isolation of such keys; the use of multiple LMKs is one way to achieve this;
- Some users may wish to use the HSM in both a live operational environment and a test or development environment (not recommended, but may be a necessity in some cases).

The availability of Multiple LMKs also makes it easier to migrate operational keys from an old LMK to a new one. Such LMK migration should be performed every few years for security purposes, but may also be necessary for operational reasons – for example when upgrading from double- to triple-length Variant LMKs or from Variant LMKs to Key Block LMKs.

Although the payShield 10K allows for changing the LMK, it means that all operational keys need to be translated from encryption under the old LMK to encryption under the new LMK before they can be used. A “big bang” approach typically requires very careful planning and coordination, with possible downtime or need for additional HSM capacity. As a consequence, some HSM users are reluctant to change the LMK.

The use of multiple LMKs allows users to adopt a phased approach to LMK change.

9.3 Multiple LMK Licensing

All payShield 10K HSMs are delivered with the capability to support 2 LMKs – one Variant LMK plus one Key Block

LMK.

Additional optional licenses can be added to the payShield 10K to enable use of up to 20 LMKs, of any mix of Variant and Key Block LMKs, on a single payShield 10K.

9.4 Managing Multiple LMKs

9.4.1 LMK component generation

LMK components are generated using the GK console command. This command also stores the component(s) to a smart card.

Note: This command may be used to generate components for the following types of LMKs:

- Double-length (2DES) Variant LMK
- Triple-length (3DES) Variant LMK
- Triple-length (3DES) Key Block LMK
- 256-bit AES Key Block LMK.

When creating a Variant LMK or a 3DES Key Block LMK, this command generates the data for a single LMK component card.

When creating an AES Key Block LMK, this command generates the data for all the required number of LMK component cards.

```
Secure> GK <Return>
Variant scheme or key block scheme? [V/K]: K <Return>
Key status? [L/T]: L <Return>
LMK component set [1-9]: 1 <Return>
Enter secret value A: <Return>
Enter secret value B: <Return>
Enter secret value C: <Return>
Insert blank card and enter PIN: ***** <Return>
    Writing key
    Checking key
Device write complete, check: 012345
Make another copy? [Y/N]: Y <Return>
...
Secure>
```

Equivalent functionality when using payShield Manager is found under Operational / Local Master Keys.

9.4.2 Migration from Old to New LMK

Once the LMK has been loaded, an "old" LMK can be loaded into key change storage (e.g. using the console LO command) and operational keys migrated from encryption under the old LMK to encryption under the new. Keys can be migrated from variant > variant LMK, variant > key block LMK or key block > key block LMK, but not from key block > variant LMK.


```
Secure-AUTH> LO <Return>
Enter LMK id: 02 <Return>
Enter comments: Old LMK for PQR Bank <Return>
Load old LMK from components
Insert card and enter PIN: ***** <Return>
Check: 678901
Load more components? [Y/N]: Y <Return>
...
Check: 085392
Load more components? [Y/N]: N <Return>
LMK id: 02
LMK key scheme: Key block
LMK algorithm: 3DES (3key)
LMK status: Live
Comments: Old LMK for PQR Bank
Confirm details? [Y/N]: Y <Return>
...
Secure-AUTH>
```

Equivalent functionality when using payShield Manager is found under Operational / Local Master Keys.

Once the old and new LMKs are loaded in the correct locations, migration of keys from old to new LMK (and from variant to key block, if required) can take place. The same mechanism can be used to provide a phased migration of keys from one LMK to another. In this case, the "old" key may continue to be used as an operational LMK at the same time as keys are migrated to encryption under a new LMK. This mechanism allows, for example, existing keys for different applications to be cryptographically isolated from each other.

9.5 Authorization

Authorization is required to enable certain console and host commands to be executed. Either the whole HSM can be put into Authorized State, which enables all commands requiring authorization to be executed. Alternatively, the HSM can be configured to support Multiple Authorized Activities: in this mode, authorization can be applied to individual activities or groups of activities.

In order to set authorization, two smartcard holders must authenticate themselves to the HSM by presenting the smartcards and entering the smartcards' PINs. These PINs are set up when the smartcards are formatted (e.g. by using the FC console command).

With multiple LMKs, each LMK will have its own authorizing smartcards and PINs. This means that commands can be authorized for a specific LMK. So, for example, in the previous section when using the LO command for loading an "old" LMK in key change storage, LMK with identifier=02 has been specified: authorization requires the use of the authorization smartcards and PINs used when setting up LMK 02 and so only permits the old LMK to be loaded in key change storage slot 02.

This technique provides for highly granular control over sensitive LMK operations. The A console command can be used to see which activities have been authorized:

```

Online> A <Return>
Enter LMK id: 03 <Return>
No activities are authorized for LMK id 03
List of authorizable activities:
...
...
The following activities are pending authorization for LMK id 03:
pin.mailer
First Officer:
Insert card for Security Officer and enter the PIN: ***** <Return>
Second Officer:
Insert card for Security Officer and enter the PIN: ***** <Return>
The following activities are authorized for LMK id 03:
pin.mailer

Online-AUTH>

```

In this example, only PIN mailer printing is authorized and then only for the LMK with identifier 03. Equivalent functionality when using payShield Manager is found under Operational / Local Master Keys.

Note: Different formats are used for LMK storage and saving HSM settings. payShield Manager cards do not need to be formatted.

9.6 LMK table

From a management perspective, it is important to know which LMKs have been loaded into the HSM (and in which locations). This information can be displayed using the VT console command:

```

Online> VT <Return>

LMK table:
ID Authorized Scheme Algorithm Status Check Comments
00 Yes(1H,1C) Variant 3DES(2key) Test 268604 For RST Bank
01 No Key block 3DES(3key) Test 999999 For XYZ Bank
02 Yes(4H,0C) Key block AES-256 Live 324365 For PQR Bank
03 Yes(0H,1C) Key block AES-256 Live 963272 Mngmnt LMK

Key change storage table:
ID Scheme Algorithm Status Check Comments
01 Variant 3DES(2key) Test 876543 For XYZ Bank
02 Key block 3DES(3key) Live 085392 For PQR Bank

Online>

```

As can be seen, details of operational LMKs and any LMKs that have been loaded into key change storage are displayed. The second column of the LMK table indicates whether any activities are authorized for that LMK and, if so, how many ("H" = Host commands, "C" = Console commands).

Equivalent functionality when using payShield Manager is found under Operational / Local Master Keys.

9.7 Deleting LMKs

With multiple LMKs in the HSM, it is necessary to be able to delete individual LMKs from the HSM's memory without deleting any other LMK. This is achieved using the DM console command:

```
Secure-AUTH> DM <Return>
Enter LMK id: 01 <Return>

LMK table entry:
ID Scheme  Algorithm  Status  Check  Comments          Auth
01 Key Block 3DES(3key) Test  999999  Test LMK for XYZ Bank  Yes (1)

Key change storage table entry:
ID Scheme  Algorithm  Status  Check  Comments
01 Variant  3DES(2key) Test  876543  Old test LMK for XYZ Bank

Confirm LMK deletion [Y/N]: Y <Return>
LMK deleted from main memory and key change storage

Secure>
```

As can be seen from the above example, if an LMK is deleted then any LMK in the corresponding "slot" in key change storage is also deleted.

Similarly, individual LMKs that have been loaded into key change storage may be deleted without deleting the live LMK by using the DO console command.

Equivalent functionality when using payShield Manager is found under Operational / Local Master Keys.

9.8 Identifying the Required LMK

Clearly, operational commands need to know which LMK should be used for processing.

9.8.1 Console Commands

Console commands are identified using the LMK identifier.

The Generate a Check Value (CK) console command is used to generate a key check value (KCV) for a key encrypted under a specified LMK.

```
Online> CK <Return>
Enter LMK id: 03 <Return>
Enter key block: S 0xxxxxxxxxxxx03xx.....xxxxx <Return>

Key check value: CCCCCC
Online>
```

In the above example, the LMK with identifier 03 is a key block LMK and the command returns the check value for the key contained within the key block. Note that the key block itself also includes the identifier of the LMK used to encrypt and authenticate the key block, in bytes 14-15. If this value does not agree with the value entered by the user then an error message will be displayed.

9.8.2 Host Commands

For host commands, the situation is somewhat more complicated and a number of mechanisms can be used to indicate which LMK to use in the processing of the command.

9.8.2.1 Key Block LMKs

One of the fields in a key block identifies the LMK used to encrypt and authenticate the key block (see bytes 14-15 of the key block in the earlier example).

The HSM can be set to ignore this field, using the security setting accessed via the CS console command ("Configure Security") or Configuration / Security Settings / Initial in payShield Manager.

Normal behavior is to not ignore this field, and so if Thales Key Blocks are presented in a host command, the LMK identified in the key block header(s) will be used, and if key blocks with different LMK identifiers are presented in the same command then the HSM will return an error.

However, if the HSM is set to ignore this field, then the method of identifying the LMK to use in with a command is identical to when a Variant LMK is used – described in the following sections.

9.8.2.2 Default LMK

One particular LMK is identified as the "default" LMK and, in the absence of any other indication in the command message, it is assumed that the default LMK is being used. The principal benefit of this mechanism is that it provides a backwards-compatibility for host applications written without consideration of the multiple LMK capability.

The identifier for the default LMK (which may be the same as the management LMK – see below) is defined using the CS console command ("Configure Security") or Configuration / Security Settings / General in payShield Manager.

9.8.2.3 Management LMK

One LMK is designated as the "management" LMK and is required for a small number of host commands, mainly associated with the HSM's audit functions.

The identifier for the management LMK (which may be the same as the default LMK discussed above) is defined using the CS console command ("Configure Security") or Configuration / Security Settings / General in payShield Manager.

9.8.2.4 Explicit LMK Identifier in Host Commands

An explicit mechanism that can be used is simply to include optional fields at the end of the command message to identify the correct LMK to use when processing the command. This technique can be used if there are no other fields in the command message that identify the LMK – for example, when using the A0 command to generate a key or when using a variant LMK, which is neither the default LMK nor the management LMK. The optional fields are a delimiter ("%") and the LMK identifier:

Field	Length & Type	Details
Delimiter	1 A	Optional; if present the following field must be present; value "%"
LMK Identifier	2 N	LMK identifier; permitted values "00" to "99"; must be present if the above Delimiter is present

If there is a "mis-match" between the LMK identifiers in a command then the HSM will return an error. This could happen, for example, if:

- no LMK is loaded in the identified location, or
- key blocks contain different identifiers, or
- the optional fields are present at the end of the command message and indicate an LMK that is different from the LMK identified elsewhere in the command message.

The explicit identification of the required LMK in the host command normally must match the LMK identification using the TCP Port number (see below). However, when the payShield 10K security setting ("Ensure LMK Identifier in command corresponds with host port") is set to "No", the HSM will ignore any mismatches, and just use the explicit identification of the LMK in the host command.

9.8.2.5 Ethernet TCP Port

An additional mechanism is available for Ethernet-attached host computers. The HSM can infer the LMK Identifier to use for a particular command from the TCP port on which the command is received.

When configuring the payShield 10K's host ports, a "Well-Known Port" is specified for host command traffic arriving from the host: by default this is 1500 (or 2500 if Secure Host Communications is being used). In the absence of any other indicator of the LMK to be used:

- host commands directed to the Well-Known Port will use the Default LMK;
- host commands directed to [Well-Known Port +1] will automatically use LMK Id 00;
- host commands directed to [Well-Known Port +2] will automatically use LMK Id 01;
- more generally, host commands directed to [Well-Known Port + n, where $n \leq 1$] will automatically use LMK Id [n-1].

The situation for an HSM using the default Well-Known Port value of 1500 is summarized in the table below:

Command received on TCP Port	LMK Used
1500	Default LMK Id
1501	LMK Id 00
1502	LMK Id 01
1503	LMK Id 02
...	...

The explicit identification of the required LMK in the host command (as described in the previous section) normally must match the LMK identification using the TCP Port number. However, when the payShield 10K security setting ("Ensure LMK Identifier in command corresponds with host port") is set to "No", the HSM will ignore any mismatches, and just use the explicit identification of the LMK in the host command.

10 Migrating LMKs

10.1 Introduction

Thales payment HSMs have always provided a facility to migrate between LMKs - i.e. to re-encrypt operational keys and other data from encryption under one (old) LMK to encryption under another (new) LMK. The need to do this is more important than in the past because:

- Card schemes are requesting that customers change their master keys every 2 years.
- Adoption of Key Block LMKs, with their added security, requires a migration from Variant LMKs.

This chapter outlines the migration process.

10.2 Multiple LMKs

By default, the payShield 10K is delivered with the ability to install 1 or 2 LMKs. If 2 LMKs are installed, one must be a Variant type and one must be a Key Block type (see below).

Each LMK can be managed by its own team of security officers.

The multiple LMK facility can be used to provide separation between multiple clients, applications, or purposes serviced on the same HSM - and they also make the process of migrating LMKs easier.

10.3 Overview of the LMK Migration Process

The LMK Migration process takes keys which are encrypted under an old LMK and re-encrypts them under a new LMK. Both the old and the new LMKs must be installed in the payShield 10K. There are two types of LMK storage:

- LMK Live storage. Transaction processing and other LMK functions can make use only of LMKs in Live storage.
- Key Change storage. LMKs in Key Change storage cannot be used for any purpose other than as part of the LMK migration process. Where multiple LMKs are deployed, there is one Key Change storage "slot" for each LMK in the Main storage.

There are 2 ways of allocating old/new keys to Main/Key Change storage:

- The new LMK (which has not yet been deployed for live operation) is loaded into Live storage, and the old LMK (which is still being used for live processing) is loaded into Key Change storage using the LO console command (as described later). This means that the payShield 10K being used for migration cannot be used to process transactions until the LMK migration process is completed and the new LMK comes into operational use, but it is then immediately ready to process transactions because the new LMK is already loaded in Live storage.
- The old LMK (still being used for live operation but about to be obsoleted) is left in Main Live, and the new LMK (which has not yet been deployed for live operation) is loaded into Key Change storage using the LN console command (as described later). This is a more recent option and means that the payShield 10K can continue processing transactions using the current LMK at the same time as it is used for migrating keys to the new LMK. On the other hand, when the new LMK is ready to go live, the new LMK must be loaded into Live storage before any transactions can be processed.

The steps needed to migrate from an old LMK to a new LMK are:

1. Create smartcards with components for the new LMK.
2. Load the new LMK (from components cards) into either LMK Live storage or LMK Key Change storage.
3. Either:
 - leave the old LMK in LMK Live storage and load the new LMK (from component cards) into LMK Key Change storage, or
 - load the new LMK (from component cards) into LMK Live storage and load the old LMK (from components cards) into LMK Key Change storage in the same HSM.
4. Re-encrypt the operational keys from the old LMK to the new LMK and hold these in a pending new key database.
5. Re-encrypt PINs from the old LMK to the new LMK and hold these in a pending new PIN database.

6. Re-encrypt decimalization tables from the old LMK to the new LMK and hold these in a pending new decimalization table database.
7. If the new LMKs have been loaded into Key Change storage, re-load them into Live storage.
8. Make the pending key/PIN/decimalization table databases the live databases.

The following sections describe each of these options.

Note: Subsequent sections look at the following associated considerations:

- Migrating from Variant to Key Block LMKs
- Key type changes for PCI HSM compliance.
- The Multiple LMK capability

10.3.1 Generating new LMK component smartcards

LMKs are set up in the payShield 10K by loading a number (typically 3) of components which are then combined within the HSM to form the LMK. (The formed LMK is never available outside of the HSM.) The LMK components are loaded from LMK smartcards.

The first stage, therefore, is to create smartcards which have the components for the new LMK. These components have completely random values, and are created on any payShield 10K.

Each component must be held by a different security officer, and access to the component cards must be securely controlled (e.g. by storing the card securely and requiring security officers to check the cards out and in).

All component cards are required to load (or form) an LMK, and so loss of any card or absence of a card holder prevents the LMK being loaded (or re-loaded at a later date if necessary). Therefore at least one backup should be made of each component card.

Note that the terms "LMK card" and "LMK component card" are interchangeable. Only LMK components are ever written to cards - the whole LMK is never written to a card.

10.3.2 Types of LMK component cards

There are two types of LMK component cards:

- HSM LMK cards - using the card reader built into the HSM. This type of card is created and used by operators using a console and the HSM card reader.
- payShield Manager RLMK cards - created by operators using payShield Manager and the card reader attached to the remote management PC.

The principles are the same for both types of card, although the detail of the processes is different. The two types of card are incompatible, although either type of card can be created from the other.

10.4 Formatting LMK smart cards

10.4.1 HSM LMK Cards

Before they can be written to, smart cards must be formatted.

Cards which have been used previously and are no longer required can be re-formatted to enable the new components to be written to them.

Do not re-format the component cards for the old LMK that you are about to migrate from - you will be needing them !

Each component holder should format their own card plus at least one backup per component.

HSM LMK smartcards are formatted using the FC console command (described in the *payShield 10K Console Guide*).

10.4.2 payShield Manager LMK Cards

With payShield Manager, the LMK components are written to RLMK cards which are provided by Thales. RLMK cards do not require formatting.

10.5 Generating LMK component cards

10.5.1 HSM LMK Cards

Each component holder should now generate a component and write it to their smart card and backup card(s). This is done using the GK console command (described in the *payShield 10K Console Guide*).

Various warnings and errors may be reported during this process. These are easy to understand, and appropriate responses should be made.

10.5.1.1 payShield Manager RLMK Cards

LMK components for use with payShield Manager are written to RLMK cards using the Generate button in payShield Manager's Operational / LMK Operations / Local Master Keys tab. These cards use a quorum (i.e. "m of n") approach to define how many of the cards must be used when loading an LMK. The operator provides the following information when generating the LMK:

- Number of LMK shares, i.e. "n" (Default: 2)
- Number of shares to rebuild, i.e. "m" (Default: 2)
- Key scheme (Variant or Key Block)
- Algorithm
- Status (Live or Test)

For more detailed information regarding payShield Manager, see the *payShield 10K Installation and User Guide*.

10.6 Creating Copies of LMK Component Cards

Because all component cards are needed when the LMK is loaded, copies of each LMK card should be made to allow for misplacement or for issuing to deputies. There are several ways of doing this.

10.6.1 HSM LMK cards

10.6.1.1 During LMK card generation

As mentioned above in the section on Generating HSM LMK cards, multiple copies may be made at the time of generating the LMK card.

10.6.1.2 Copying an existing HSM LMK card

It is possible at any time to copy an existing HSM LMK card using the DC console command.

10.6.2 payShield Manager RLMK card

10.6.2.1 Duplicating a payShield Manager RLMK card

A copy of an existing RLMK component card can be made using the Duplicate button in payShield Manager's Operational / LMK Operations / Local Master Keys tab.

10.7 Loading the new LMK

In the previous section we explained how to create a set of cards containing the components for the new LMK. Each component is "owned" by a different security officer, with no one security officer having access to more than one component. One holder of each of the required number of components must be present to allow the LMK to be loaded onto the payShield 10K using the component smart cards.

The new LMK now needs to be installed into either LMK Live storage or LMK Key Change storage depending on the approach being taken.

The new LMK can be loaded using a Console or payShield Manager.

10.7.1 Using the Console

10.7.1.1 Loading (or forming) the LMK

The LMK is loaded using either:

- the LK console command if the new LMK is to be loaded into LMK Live storage, or
- the LN console command if the new LMK is to be loaded into LMK Key Change storage.

The payShield 10K must be in Secure state. In addition, if the LN console command is being used then the HSM must be in Authorized state. If multiple authorized states is enabled, the activity category is admin (with no sub-category), and the console interface should be selected.

The smart cards used must be HSM cards - not cards created for payShield Manager.

10.7.1.2 Checking the LMK

It is recommended that a check is made that the new LMK has been properly loaded.

This can be done using the A console command, to put the HSM into authorized state (followed by the C command to cancel the authorized state). The A command can be run in any HSM state. The operation of this command depends on whether multiple authorized activities has been enabled in the security settings (e.g. by using the CS console command) – see the *payShield 10K Console Guide* for full details.

10.7.2 Load Using payShield Manager

10.7.2.1 Installing the LMK

The new LMK is loaded using the Install button in the appropriate payShield Manager tab:

- Operational / LMK Operations / Local Master Keys where the new LMK is to be loaded into LMK Live storage, or
- Operational / LMK Operations / Key Change Storage where the new LMK is to be loaded into LMK Key Change storage.

The LMK ID will need to be specified. See the *payShield Manager 10K Installation and User Guide* for more detailed information.

10.7.3 Checking the LMK

The installed LMK can be checked by viewing the LMK list displayed at Operational / LMK Operations / Local Master Keys or Operational / LMK Operations / Key Change Storage.

10.8 Loading the old LMK

So far we have created a set of cards containing the components for the new LMK, and used them to load into the HSM the "new" LMK that keys and data to be re-encrypted to.

To migrate keys from encryption under an old (current) LMK to encryption under the new LMK, we also need to have the old LMK into the HSM. The old LMK can be left in LMK Lives storage or loaded into LMK Key Change Storage, depending on the approach being taken.

If the old LMK is to be loaded into Key Change Storage, this can be done using a Console or payShield Manager.

10.8.1 Using the Console

The old LMK is loaded into Key Change Storage using the LO (where "O" is the alphabetic "O for Oscar") console command.

The payShield 10K must be in Secure state. In addition, the HSM must be in Authorized state. If multiple authorized states is enabled, the activity category is admin (with no sub-category), and the console interface should be selected.

The use of the LO console command is the same as for the LK console command mentioned previously, except that no existing LMK needs to be erased and so you will not be prompted to confirm an erasure.

After loading the old LMK, the HSM should be returned to Online state by turning the physical keys.

10.8.2 Load Using payShield Manager

The old LMK is loaded using the Install button in payShield Manager's Operational / LMK Operations / Key Change Storage tab. This can only be done if there is an LMK with the same ID in the LMK table. See the *payShield 10K Installation and User Guide* for more detailed information.

10.9 Migrating keys between Variant LMKs

We now have installed in the HSM both the old LMK that the operational keys are currently encrypted under and the new LMK that they need to be encrypted under for the future. We now need to take each existing operational key in the old key database (encrypted under the old LMK), re-encrypt it using the new LMK, and put it in a new key database.

In order to do this, an application needs to be set up at the host that:

- Takes each operational key (encrypted under the old LMK) from the old key database
- Sends the encrypted key to the HSM using the BW host command.
- Receives the BX response from the HSM containing the operational key encrypted under the new LMK.
- Puts the operational key encrypted under the new LMK into the new key database.

10.9.1 The BW Host command

Note: In order to migrate Single DES keys from 3DES LMKs to AES LMKs, ensure that the Single-DES security setting is enabled. Security settings are described in the *payShield 10K Security Manual*.

Here we will look at the BW host command as it is used to convert an operational key encrypted under an old LMK of the Variant type to encryption under a new LMK of the Variant type. Other ways of using the BW command are discussed later in this document.

The BW host command automatically adapts its processing depending on where the old and new LMKs are stored:

- If the old LMK was loaded into Key Change storage (e.g. the LO console command was used), then keys and data are re-encrypted from encryption under the (old) LMK in Key Change storage to encryption under the (new) LMK in Live storage.
- If the new LMK was loaded into Key Change storage (e.g. the LN console command was used), then keys and data are re-encrypted from encryption under the (old) LMK in Live storage to encryption under the (new) LMK in Key Change storage.

The table below indicates the structure of the BW host command when it is used in this way.

Field	Length & Type	Notes
Message Header	m A	This field contains whatever the user wants. The length of the field is defined using the <i>CH</i> console command or <i>Configuration / Host Settings</i> in payShield Manager. It is subsequently returned unchanged in the response to the host.
Command Code	2 A	Must have the value 'BW'.
Key Type code	2 H	Indicates the LMK under which the key is encrypted: '00' : LMK pair 04-05 (Key Type 000) '01' : LMK pair 06-07 (Key Type 001) '02' : LMK pair 14-15 (Key Type 002) '03' : LMK pair 16-17 (Key Type 003) '04' : LMK pair 18-19 (Key Type 004) '05' : LMK pair 20-21 (Key Type 005) '06' : LMK pair 22-23 (Key Type 006) '07' : LMK pair 24-25 (Key Type 007) '08' : LMK pair 26-27 (Key Type 008) '09' : LMK pair 28-29 (Key Type 009) '0A' : LMK pair 30-31 (Key Type 00A) '0B' : LMK pair 32-33 (Key Type 00B) '10' : Variant 1 of LMK pair 04-05 (Key Type 100) '42' : Variant 4 of LMK pair 14-15 (Key Type 402) 'FF' : Use this value where the key type is specified after the first ';' delimiter below. This allows key types other than those listed above to be specified.
Key length flag	1 N	'0' : for single-length key '1' : for double-length key '2' : for triple-length key.
Key	16/32 H or 1 A + 32/48 H	The operational key to be translated, encrypted under the old LMK.
Delimiter	1 A	Optional. Only present if 'FF' was supplied above for the Key Type code and the following field is present. Value ';'.

Field	Length & Type	Notes
Key Type	3 H	Where 'FF' was entered for Key Type Code, this is the 3-digit key type code of the key being translated. For a list of key type codes, see the appropriate table of Key Type Codes in the <i>payShield 10K Installation and User Guide</i> .
Delimiter	1 A	Optional. If present the following three fields must be present. Value ';':
Reserved	1 A	Optional. If present must be '0' (zero).
Key Scheme (LMK)	1 A	Optional. Key scheme for encrypting key under LMK (or '0' (zero)). For a list of key schemes, see the Key Scheme Table at Appendix A “Key Scheme Table”
Reserved	1 A	Optional. If present must be '0' (zero).
Delimiter	1 A	Value '%'. Optional; if present, the following field must be present.
LMK Identifier	2 N	Where the user is using multiple LMKs on the same HSM, this allows the ID of the LMK being migrated to be selected. Minimum value = '00'; maximum value is defined by license. This field must be present if the above Delimiter is present. If the field is not present, then the default LMK will be used.
End Message Delimiter	1 C	Must be present if a message trailer is present. Value X'19'.
Message Trailer	n A	Optional. The contents of the trailer is as required by the user, and is returned unchanged in the response. Maximum length 32 characters.

10.9.2 The BX Response to the Host

In response to the BW host command, the payShield 10K will return the following BX response to the host:

Field	Length & Type	Notes
Message Header	m A	This is a play-back of the header provided in the <i>BW</i> command.
Response Code	2 A	Has the value 'BX'.
Error code	2 N	<p>Indicating the general outcome of the <i>BW</i> command:</p> <p>'00' : No error</p> <p>'04' : Invalid key type code</p> <p>'05' : Invalid key length flag</p> <p>'10' : Key parity error</p> <p>'44' : migration not allowed: key migration requested when the security setting "Enforce key type 002 separation for PCI HSM compliance" has the value "Y".</p> <p>'45' : Invalid key migration destination key type.</p> <p>'68' : Command disabled</p> <p>or any standard error code (see Chapter 4 of the <i>payShield 10K Host Command Reference Manual</i>).</p>
Key	16/32 H or 1 A + 32/48 H	The resulting key, re-encrypted under the new LMK.
End Message Delimiter	1 C	Present only if a message trailer is present. Value X'19'.
Message Trailer	n A	This is simply a play-back of any trailer included in the <i>BW</i> command.

10.10 Migrating keys from Variant to Key Block LMKs

Key Block LMKs provide additional security compared to Variant LMKs.

The BW host command already described for Variant LMK > Variant LMK migration can also be used for Variant LMK > Key Block LMK migration. When used for this purpose, the BW command and BX response are modified as indicated below.

Note that it is **not** possible to migrate from:

- Key Block LMKs to Variant LMKs.
- AES Key Block LMKs to TDES Key Block LMKs.

10.10.1 The BW Host command

The table below indicates the structure of the BW host command when it is used to migrate from Variant-type LMKs to Key Block-type LMKs. Only the differences compared to Variant LMK > Variant LMK migration are described.

Full details on the use of the BW host command can be found in the *payShield 10K Host Command Reference Manual*.

Field	Length & Type	Notes
Message Header	m A	(As for Variant LMK ⇒ Variant LMK)
Command Code	2 A	Must have the value 'BW'.
Key Type code	2 H	(As for Variant LMK ⇒ Variant LMK)
Key length flag	1 N	(As for Variant LMK ⇒ Variant LMK)
Key	16/32 H or 1 A + 32/48 H	(As for Variant LMK ⇒ Variant LMK)
Delimiter	1 A	(As for Variant LMK ⇒ Variant LMK)
Key Type	3 H	(As for Variant LMK ⇒ Variant LMK)
Delimiter	1 A	(As for Variant LMK ⇒ Variant LMK)
Reserved	1 A	(As for Variant LMK ⇒ Variant LMK)
Key Scheme (LMK)	1 A	(As for Variant LMK ⇒ Variant LMK)
Reserved	1 A	(As for Variant LMK ⇒ Variant LMK)
Delimiter	1 A	(As for Variant LMK ⇒ Variant LMK)
LMK Identifier	2 N	(As for Variant LMK ⇒ Variant LMK)
Delimiter	1 A	Must have value '#'
Key Usage	2 A	The required key usage for the key encrypted under the Key Block LMK. This information is included in the Key Block header and should be determined using the Key Usage Table. Refer to the <i>payShield 10K Installation and User Guide</i> . This field determines type of the operational key (e.g. RSA private key, BDK, ZEK), and enforces key separation.
Mode of Use	1 A	The required mode of use for the key encrypted under the Key Block LMK. This information is included in the Key Block header, and should be determined using the Mode of Use Table. Refer to the <i>payShield 10K Installation and User Guide</i> . This field determines how the operational key can be used (e.g. encryption, decryption, MACing).
Key Version Number	2 N	A value from '00' to '99', for inclusion in the Key Block header. Determined by the user. '00' denotes that key versioning is not in use for this key.

Field	Length & Type	Notes
Exportability	1A	The required exportability for the key encrypted under the Key Block LMK. This information is included in the Key Block header, and should be determined using the Exportability Table. Refer to the <i>payShield 10K Installation and User Guide</i> . This field determines how the operational key can be exported (e.g. no export allowed, may only be exported as a Key Block).
Number of Optional Blocks	2 N	A value from '00' to '08', identifying how many optional data blocks the user wants to add into the Key Block Header. Optional data blocks are used to identify parameters (such as key validity dates, key status, algorithm). Optional header blocks are described in the <i>payShield 10K Installation and User Guide</i> . For a value greater than 0, the following three fields must be repeated for each optional block.
Optional Block Identifier	2 A	The available Optional Block Identifiers (or Types) are described in <i>payShield 10K Installation and User Guide</i> . Note that the value 'PB' may not be used.
Optional Block Length	2H	The length in bytes of the optional block (including the Identifier and Length). A value of X'04' indicates that the block contains only the identifier and length, and so the next field would not be present.
Optional Data Block	N A	The payload of the optional data block - see <i>payShield 10K Installation and User Guide</i> .
End Message Delimiter	1 C	(As for Variant LMK ⇒ Variant LMK)
Message Trailer	n A	(As for Variant LMK ⇒ Variant LMK)

10.10.2 The BX Response to the Host

In response to the BW host command, the payShield 10K will return the following BX response to the host:

Field	Length & Type	Notes
Message Header	m A	(As for Variant LMK ⇒ Variant LMK)
Response Code	2 A	Has the value 'BX'. (As for Variant LMK ⇒ Variant LMK)
Error code	2 N	(As for Variant LMK ⇒ Variant LMK)
Key	1 A + n A	The operational key, encrypted under the new Key Block LMK.

Field	Length & Type	Notes
End Message Delimiter	1 C	(As for Variant LMK ⇒ Variant LMK)
Message Trailer	n A	(As for Variant LMK ⇒ Variant LMK)

10.11 Migrating keys between Key Block LMKs

Migration of operational keys between Key Block LMKs is supported in addition to the Variant LMK > Variant LMK and Variant LMK > Key Block LMK migrations already described. This section describes the BW host command when used for this purpose.

Note that it is not possible to migrate from:

- Key Block LMKs to Variant LMKs.
- AES Key Block LMKs to TDES Key Block LMKs.

10.11.1 The BW Host command

The table below indicates the structure of the BW host command when it is used to migrate between Key Block-type LMKs. Only the differences compared to Variant LMK > Key Block LMK migration are described.

Field	Length & Type	Notes
Message Header	m A	(As for Variant LMK ⇒ Key Block LMK)
Command Code	2 A	Must have the value 'BW'.
Key Type code	2 H	Must be set to 'FF'.
Key length flag	1 H	Must be set to 'F'.
Key	1 A + n A	The operational key to be translated, encrypted under the old Key Block LMK.
Delimiter	1 A	Must have value ';'.
Key Type	3 H	Must be set to 'FFF'.
Delimiter	1 A	(As for Variant LMK ⇒ Key Block LMK)
Reserved	1 A	(As for Variant LMK ⇒ Key Block LMK)
Key Scheme (LMK)	1 A	(As for Variant LMK ⇒ Key Block LMK)
Reserved	1 A	(As for Variant LMK ⇒ Key Block LMK)
Delimiter	1 A	(As for Variant LMK ⇒ Key Block LMK)
LMK Identifier	2 N	(As for Variant LMK ⇒ Key Block LMK)
Delimiter	1 A	(As for Variant LMK ⇒ Key Block LMK)
Key Usage	2 A	(As for Variant LMK ⇒ Key Block LMK)
Mode of Use	1 A	(As for Variant LMK ⇒ Key Block LMK)
Key Version Number	2 N	(As for Variant LMK ⇒ Key Block LMK)
Exportability	1A	(As for Variant LMK ⇒ Key Block LMK)

Field	Length & Type	Notes
Number of Optional Blocks	2 N	(As for Variant LMK ⇒ Key Block LMK)
Optional Block Identifier	2 A	(As for Variant LMK ⇒ Key Block LMK)
Optional Block Length	2H	(As for Variant LMK ⇒ Key Block LMK)
Optional Data Block	N A	(As for Variant LMK ⇒ Key Block LMK)
End Message Delimiter	1 C	(As for Variant LMK ⇒ Key Block LMK)
Message Trailer	n A	(As for Variant LMK ⇒ Key Block LMK)

10.11.2 The BX Response to the Host

In response to the BW host command, the payShield 10K will return the following BX response to the host:

Field	Length & Type	Notes
Message Header	m A	(As for Variant LMK ⇒ Key Block LMK)
Response Code	2 A	Has the value 'BX'. (As for Variant LMK ⇒ Key Block LMK)
Error code	2 N	(As for Variant LMK ⇒ Key Block LMK)
Key	1 A + n A	(As for Variant LMK ⇒ Key Block LMK)
End Message Delimiter	1 C	(As for Variant LMK ⇒ Key Block LMK)
Message Trailer	n A	(As for Variant LMK ⇒ Key Block LMK)

10.12 Migrating keys from Key Block to Variant LMKs

This migration is not permitted because Variant LMKs are not as strong as key block LMKs.

10.13 Migrating keys for PCI HSM compliance

When it is required to make a payShield 10K compliant with the requirements of the PCI PTS HSM security standard, it may be necessary to move some keys from Variant key type 002 (LMK pair 14-15, Variant 0) to other key types.

Although this can be done as a separate operation, it can be achieved at the same time as migrating between LMKs using the BW host command by entering 'F2' as the Key Type Code, and the desired destination key type in the Key

Type field: see the *payShield 10K Host Command Reference Manual* for further details.

10.14 Re-encrypting PINs

Where PINs have been stored encrypted under the old LMK (in LMK Live storage or LMK Key Change storage) these will need to be re-encrypted using the new LMK (in LMK Key Change storage or LMK Live storage). This can be done by using the BG host command.

A host application will be required that takes each PIN from the old PIN database, re-encrypts it using the BG host command, and stores the re-encrypted PIN into the new PIN database.

10.14.1 The BG Host Command

The structure of the BG host command is given in the following table:

Field	Length & Type	Notes
Message Header	m A	This field contains whatever the user wants. The length of the field is defined using the <i>CH</i> console command or <i>Configuration / Host Settings</i> in payShield Manager. It is subsequently returned unchanged in the response to the host.
Command Code	2 A	Has the value 'BG'.
Account Number	12 N	The 12 right-most digits of the account number, excluding the check digit.
PIN	L ₁ N Or L ₁ H	The PIN encrypted under the old LMK, where L ₁ is the old encrypted PIN length. L ₁ N applies where PIN encryption algorithm A (Visa method) is specified in the security settings, and L ₁ H applies where PIN encryption algorithm B (Racal method) is specified.
Delimiter	1 A	Value '%'. Optional; if present, the following field must be present.
LMK Identifier	2 N	Where the user is using multiple LMKs on the same HSM, this allows the required LMK to be selected. Minimum value = '00'; maximum value is defined by license. This field must be present if the above Delimiter (%) is present. If the field is not present, then the default LMK will be used.
End Message Delimiter	1 C	Must be present if a message trailer is present. Value X'19'.
Message Trailer	n A	Optional. The contents of the trailer is as required by the user, and is returned unchanged in the response. Maximum length 32 characters.

10.14.2 The BH Response

The HSM will return the following BH response to the host's BG command:

Field	Length & Type	Notes
Message Header	m A	This is a play-back of the header provided in the <i>BG</i> command.
Response Code	2 A	Has the value 'BH'.
Error code	2 N	Indicating the general outcome of the <i>BG</i> command: '00' : No error '68' : Command disabled or any standard error code (see Chapter 4 of the <i>payShield 10K Host Command Reference Manual</i>).
PIN	L ₂ N Or L ₂ H	The PIN encrypted under the new LMK, where L ₂ is the new encrypted PIN length. L ₂ N applies where PIN encryption algorithm A (Visa method) is specified in the security settings, and L ₂ H applies where PIN encryption algorithm B (Racal method) is specified.
End Message Delimiter	1 C	Present only if a message trailer is present. Value X'19'.
Message Trailer	n A	This is simply a play-back of any trailer included in the BW command.

10.15 Re-encrypting decimalization tables

For security, it is recommended that decimalization tables are encrypted. They are encrypted under the LMK, and so will need to be re-encrypted when migrating to a new LMK.

This is achieved by having a host application which takes each decimalization table from the old decimalization table database and re-encrypting it under the new LMK using the LO (where "O" is the alphabetic "O for Oscar") host command (not to be confused with the LO console command discussed earlier!) and then storing it in a new decimalization table database. The new LMK can be in either Key Change storage or Live storage.

The structure of the LO host command is as follows:

Field	Length & Type	Notes
Message Header	m A	This field contains whatever the user wants. The length of the field is defined using the <i>CH</i> console command or <i>Configuration / Host Settings</i> in payShield Manager. It is subsequently returned unchanged in the response to the host.
Command Code	2 A	Most have the value 'LO'.
Decimalization Table (old LMK)	16 H	A decimalization table encrypted under the old LMK.
Delimiter	1 A	Value '%'. Optional; if present, the following field must be present.
LMK Identifier	2 N	Where the user is using multiple LMKs on the same HSM, this allows the required LMK to be selected. Minimum value = '00'; maximum value is defined by license. This field must be present if the above Delimiter (%) is present. If the field is not present, then the default LMK will be used.
End Message Delimiter	1 C	Must be present if a message trailer is present. Value X'19'.
Message Trailer	n A	Optional. The contents of the trailer is as required by the user, and is returned unchanged in the response. Maximum length 32 characters.

The payShield 10K will return the following LP response to the host:

Field	Length & Type	Notes
Message Header	m A	This is a play-back of the header provided in the <i>LO</i> command.
Response Code	2 A	Has the value 'LP'.
Error code	2 N	Indicating the general outcome of the <i>LO</i> command: '00' : No error '68' : Command disabled or any standard error code (see Chapter 4 of the <i>payShield 10K Host Command Reference Manual</i>).
Decimalisation Table (new LMK)	16 H	The decimalisation table encrypted under the new LMK.
End Message Delimiter	1 C	Present only if a message trailer is present. Value X'19'.
Message Trailer	n A	This is simply a play-back of any trailer included in the <i>BW</i> command.

10.16 Switching to the new LMK

Following the activities described above, the system is now in the following state:

- Old databases of operational keys, PINs, and decimalization tables, encrypted under the old LMK, are still being used for production.
- New databases of operational keys, PINs, and decimalization tables, encrypted under the new LMK are pending but not yet being used for production.
- One or more HSMs may have been taken out of service in order to re-encrypt the operational keys, PINs, and decimalization tables.
 - These would be HSMs that have the old (current) LMK (which is still being used on other HSMs for production) loaded in Key Change Storage (e.g. by using the LO console command), and the new LMK (not yet in use for production work) in their Live storage.
 - In this case there are other HSMs with the old LMK in their Live storage, which are doing production work using keys, PINs, and decimalization tables in the old versions of the databases.
- Production host applications are still using the old databases of operational keys, PINs, and decimalization tables.

In order to start using the new LMK, the following changes must be synchronized:

- Host production applications start using the new databases of operational keys, PINs, and decimalization tables.
- If the re-encryption of keys was done on an HSM with the new LMK in Live storage, then this HSM is immediately ready to start processing transactions using the new LMK. However, the new LMK needs to be loaded into LMK Live storage on those HSMs that were processing transactions using the old LMK.
- On the other hand, if the re-encryption of keys was done on an HSM with the new LMK in Key Change storage, then the new LMK needs to be loaded into LMK Live storage on all the HSMs in the system.

A total interruption of service can be avoided by a gradual switchover from the old LMK to the new - but in this case the host applications must know whether an HSM is using the old or new LMK and must retrieve the key or data from the appropriate database.

The use of the Multiple LMK feature of the payShield 10K offers additional options, and is described in the following section.

10.17 Taking advantage of Multiple LMKs

The payShield 10K supports multiple concurrent LMKs. The base product allows the user to implement one Variant-type LMK and one Key Block-type LMK, and optional licenses are available to provide up to 20 LMKs in any combination of types.

The multiple LMK feature offers a number of valuable benefits, and provides additional flexibility to simplify the process.

Here is an example of how the multiple LMK feature can be used where the old (still Live) LMK is in LMK Key Change storage and the new (future) LMK is in LMK Live storage:

Let us take as a starting point a production system which has the live LMK at LMK 00.

- LMK 00 is set up as the default LMK. This means that it is the LMK that is used by default in host commands where no LMK is identified: this provides backwards compatibility to applications developed before the multiple LMK facility was introduced.
- The future, new LMK is loaded as LMK 01 in LMK Live storage.
- The existing, "old" LMK, which is LMK 00 and is being used for production, is also loaded into LMK Key Change Storage for LMK 01.
- The BW, BG, and LO host commands can now be used to re-encrypt operational keys, PINs, and decimalization tables from the old LMK (which is in Key Change Storage, and also still in LMK 00 and therefore available for production) to the new LMK, which is loaded as LMK 01. This is achieved by setting the LMK Identifier in the host commands to a value of "01" : this must be preceded by a delimiter of "%".
- When all of the operational keys, PINs, and decimalization tables have been re-encrypted under the new LMK, the host application can start using the new key database when one of the following actions have been taken:
 - The new LMK is re-loaded on the payShield 10K as LMK 00. Or
 - Host commands sent to the payShield 10K are amended to use LMK 01 by either:

- Specifying the value "01" for the LMK identifier in host commands (see the structure of commands in the *payShield 10K Host Command Reference Manual*). Or
- Directing commands to the relevant TCP port (see the section on Multiple LMKs in Chapter 1 of the *payShield 10K Host Command Reference Manual*).

The benefit of this approach is that there is no need to take one or more HSMs out of productive use while the LMK migration is being performed, and therefore the key migration using the BW host command can be spread over as many HSMs as desired.

Multiple LMKs could also be used to avoid a "big bang" switchover from old to new LMKs: with the old LMK in one Live storage slot and the new LMK in a second Live storage slot, individual elements of the system can be moved to the new LMK one at a time.

10.18 Tidying up after migration to a new LMK


10.18.1 Deleting the Old LMK from Key Change Storage

The LMK in Key Change Storage should be deleted once it is no longer needed. There are multiple ways of doing this.

10.18.1.1 Using the console

The LMK can be deleted from Key Change Storage using the DO (where O is the alphabetic "O for Oscar") console command. The payShield 10K must be in Secure state.

10.18.1.2 Using payShield Manager

The LMK is deleted using the  button displayed against the LMK in payShield Manager's Operational / LMK Operations / Key Change Storage tab. This can only be done in Secure state. See the *payShield 10K Installation and User Guide* for more detailed information.

10.18.1.3 Using a Host Command

The BS host command allows the host to erase the LMK in Key Change Storage. The structure of the command is given in the following table:

Field	Length & Type	Notes
Message Header	m A	This field contains whatever the user wants. The length of the field is defined using the <i>CH</i> console command or <i>Configuration / Host Settings</i> in payShield Manager. It is subsequently returned unchanged in the response to the host.
Command Code	2 A	Must have the value 'BS'.
Delimiter	1 A	Value '%'. Optional; if present, the following field must be present.

Field	Length & Type	Notes
LMK Identifier	2 N	Where the user is using multiple LMKs on the same HSM, this allows the host to select which Old LMK is to be deleted. Minimum value = '00'; maximum value is defined by license. This field must be present if the above Delimiter (%) is present. If the field is not present, then the default LMK will be used.
End Message Delimiter	1 C	Must be present if a message trailer is present. Value X'19'.
Message Trailer	n A	Optional. The contents of the trailer is as required by the user, and is returned unchanged in the response. Maximum length 32 characters.

The BT response has the following structure:

Field	Length & Type	Notes
Message Header	m A	This is a play-back of the header provided in the <i>BS</i> command.
Response Code	2 A	Has the value 'BT'.
Error code	2 N	Indicating the general outcome of the <i>BS</i> command: '00' : No error '68' : Command disabled or any standard error code (see Chapter 4 of the <i>payShield 10K Host Command Reference Manual</i>).
End Message Delimiter	1 C	Present only if a message trailer is present. Value X'19'.
Message Trailer	n A	This is simply a play-back of any trailer included in the <i>BW</i> command.

10.18.2 Deleting the New LMK

In the section *Taking advantage of Multiple LMKs*, one suggested approach requires the new LMK to be unloaded from a temporary LMK Identifier or location (where it was loaded to enable the migration of keys and data to take place) and loaded to the location where it is required for production work.


The section *Loading the new LMK* explains how to load the LMK to its desired location, but in addition, the LMK should be deleted from its temporary location. This can be done by using one of the following methods:

10.18.2.1 Console

LMK deletion is achieved using the DM console command. This command requires Secure state and authorization - in a multiple authorize state environment, the activity to be authorized is admin.console.

Note that the DM console command also deletes the relevant old key in Key Change Storage, avoiding the need to do this separately.

10.18.2.2 Using payShield Manager

The LMK is deleted using the  button displayed against the LMK in payShield Manager's Operational / LMK Operations / Local Master Keys tab. This can only be done in Secure state. See the *payShield 10K Installation and User Guide* for more detailed information.

11 TR-31 Key Blocks

In the payments environment, keys often have to be shared between two or more parties, and these organizations may be using different Master File Keys (MFKs – the generic term for master keys performing the functions of the payShield 10K LMK) and/or types of HSM. There is therefore a need to securely export the operational keys from one system and import them into another.

The legacy way for implementing this is defined in the ANSI X9.17 standard. However, a number of weaknesses have been identified in this method, and the standard has been withdrawn. The ANSI standard X9.24 (Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques) of 2009 places requirements on the management of TDES keys which cannot be satisfied when using X9.17. The ANSI X9 Committee published Technical Report 31 (TR-31) on Interoperable Secure Key Exchange Key Block Specification for Symmetric Algorithms in 2010 to describe a method for secure key exchange which meets the needs of X9.24. This is based on the use of Key Blocks, which TR-31 defines.

The Thales payShield 10K supports the use of TR-31 Key Blocks. It also continues to support the use of X9.17 because this method, although no longer meeting the needs of standards, is still in widespread use.

This chapter describes TR-31 Key Blocks and their implementation in the Thales payShield 10K.

11.1 TR-31 Key Block Structure

Key Scheme Tag ("R") (1 byte)	Key Block Header (16 ASCII characters)	Optional Header (ASCII characters, variable length)	Encrypted Key Data (variable length, ASCII encoded)	Authenticator (8 or 16 ASCII characters)
----------------------------------	--	---	---	--

	a 16-byte (clear) Key Block Header , which defines the key usage (e.g. Visa PVV) and mode of use (e.g. verification only), the algorithm with which the key is used (e.g. TDES) and the limitations on the exportability of the key (e.g. no export permitted);
	a (clear) Optional Header block, which can be used (for example) to define the key set identifier;
	the Encrypted Key Data , which includes the actual key itself, encrypted under the "encryption variant" of the wrapping key;
	a Key Block Authenticator , calculated using the "authentication variant" of the wrapping key; the use of the Authenticator prevents unauthorized modification of the Key Block.

Currently, the TR-31 standard only permits key encryption data using the DES or TDES algorithms; the wrapping key must be a TDES key, but the key contained in the Key Block may be a single, double or triple length key using any one of the accepted algorithms. The standard has been designed to be sufficiently flexible so that other types of key can be wrapped in a Key Block in the future.

11.2 Key Block Header

The Key Block Header governs how the key contained in the Key Block may be used. Only specific field values for the Header are permitted for HSM commands. If a Key Block with the "wrong" Header is submitted in a command message then the HSM will return an error code. The structure of the Header is as follows:

Byte(s)	Field	Comments
0	Version ID	version of TR-31 key block format
1-4	Key Block Length	total length of Key Block (decimal)
5-6	Key Usage	e.g. key encryption, data encryption
7	Algorithm	e.g. DES, TDES
8	Mode of Use	e.g. encrypt only
9-10	Key Version Number	e.g. version of key in the Key Block or used to indicate that the key is a key component
11	Exportability	e.g. exportable under a trusted key
12-13	Number of optional blocks	number of Optional Header blocks (Decimal)
14-15	Reserved for future use	value = "00"

The sections that follow define valid values for the fields described in the Key Block Header.

Note: there are constant additions to the range of available values: the latest sets of values can be found in the TR-31 report.

11.2.1 Version ID (Byte 0)

Byte 0 of the Header contains a single character that indicates the format of the TR-31 key block:

Value	Definition
"A"	Originally defined in TR-31:2005 Key block protected using the Key Variant Binding Method
"B"	Originally defined in TR-31:2010 Key block protected using the Key Derivation Binding Method
"C"	Originally defined in TR-31:2010 Key block protected using the Key Variant Binding Method
"D"	Originally defined in TR-31:2018 Key block protected using the AES Key Derivation Binding Method.

11.2.2 Key Block Length (Bytes 1-4)

Bytes 1-4 of the Header contain the length of the entire key block, namely Header, Optional Header Blocks, encrypted Key Data and the Authenticator. The length of the key block is calculated after encoding and is represented as 4 numeric (ASCII) digits. For example, if the total key block length is 112 characters (bytes) then the value in byte 1 will be "0", the value in bytes 2 and 3 will be "1" and the value in byte 4 will be "2" (i.e. X'30313132).

11.2.3 Key Usage (Bytes 5-6)

Bytes 5-6 of the Header define the primary usage of the key contained in the key block. The following table defines the usage code, and how Thales key block usage codes are converted to TR-31 codes.

Value	Definition	Corresponding Thales KB Key Usage
B0	Base Derivation Key (BDK)	B0, 41, 42, 43
B1	DUKPT Initial Key (IKEY [®])	B1
C0	Card Verification Key	C0, 11, 12, 13
D0	Data Encryption Key (Generic)	D0, 21, 22, 23
E0	EMV/Chip card Master Key: Application Cryptogram (MKAC)	E0
E1	EMV/Chip card Master Key: Secure Messaging for Confidentiality (MKSMC)	E1
E2	EMV/Chip card Master Key: Secure Messaging for Integrity (MKSMI)	E2
E3	EMV/Chip card Master Key: Data Authentication Code (MKDAC)	E3
E4	EMV/Chip card Master Key: Dynamic Numbers (MKDN)	E4
E5	EMV/Chip card Master Key: Card Personalisation	E5
E6	EMV/chip card Master Key: Other	E6, 31, 32
I0	Initialization Value	I0
K0	Key Encryption / Wrapping Key (Generic)	K0, 51, 52
M0	ISO 16609 MAC algorithm 1 (using 3-DES)	M0
M1	ISO 9797-1 MAC algorithm 1	M1
M2	ISO 9797-1 MAC algorithm 2	M2
M3	ISO 9797-1 MAC algorithm 3	M3
M4	ISO 9797-1 MAC algorithm 4	M4
M5	ISO 9797-1 MAC algorithm 5	M6
P0	PIN Encryption Key (Generic)	P0, 71, 72, 73
V0	PIN Verification Key (Generic)	V0
V1	PIN Verification Key (IBM 3624 algorithm)	V1
V2	PIN Verification Key (Visa PVV algorithm)	V2
All-numeric values	Reserved for Proprietary Use	

Note: How Key Usage codes are converted when exporting from Thales Key Block format to TR-31 key format are also shown in Appendix C “*Thales Key Block / TR 31 Key Usage Conversion*”.

11.2.4 Algorithm (Byte 7)

Byte 7 of the Header defines the cryptographic algorithm with which the key contained in the key block will be used. The following values are defined, although only two of these values are currently supported:

Value	Hex	Algorithm
A	X'41	AES

D	X'44	DES (included for backwards compatibility)
E	X'45	Elliptic Curve (included for future reference)
H	X'48	HMAC –SHA1 (included for future reference)
R	X'52	RSA (included for future reference)
S	X'53	DSA (included for future reference)
T	X'54	Triple DES (TDES)
Numeric Values		Reserved for Proprietary Use

11.2.5 Mode of Use (Byte 8)

Byte 8 of the Header defines the operation that the key contained in the key block can perform.

Value	Hex	Definition
B	X'42	Both Encrypt and Decrypt
C	X'43	MAC Calculate (Generate or Verify)
D	X'44	Decrypt Only
E	X'45	Encrypt Only
G	X'47	MAC Generate Only
N	X'4E	No special restrictions or not applicable
S	X'53	Signature Only
T	X'55	Both Sign and Decrypt
V	X'56	MAC Verify Only
X	X'58	Key Derivation
Y	X'59	Key used to create key variants
Numeric values		Reserved for Proprietary Use

11.2.6 Key Version Number (Bytes 9-10)

Bytes 9-10 of the Header define the version number of the key contained in the key block or that the key is actually a key component. The following values will be supported:

First Character	Second Character	Key value field meaning
0	0	Key versioning is not used for this key.
c	Any character	The value carried in this Key Block is a component of a key. Local rules dictate the proper use of a component.
Any other combination of characters		The key version field indicates the version of the key carried in the Key Block.

11.2.7 Exportability (Byte 11)

Byte 11 of the TR-31 Key Block Header specifies the "exportability" of the key contained in the Key Block. This defines the conditions under which the key contained in the Key Block can be exported outside the cryptographic domain in which the key is found. A key is defined to be trusted if it is in either Thales Key Block or TR-31 Key Block format. Any other format key is said to be untrusted. The following values are supported:

Value	Exportability
E	May only be exported in a trusted Key Block, provided the wrapping key itself is trusted
N	No export permitted
S	Sensitive; all other export possibilities are permitted, provided such export has been enabled via the payShield 10K security settings
Numeric Values	Reserved for Proprietary Use

"Sensitive" export includes, for example, the case when a key is exported in ANSI X9.17 format. This type of export is disabled by default in the payShield 10K security settings and must be specifically enabled (e.g. using the CS console command) if required.

When a key is exported in a TR-31 Key Block format, byte 11 in the exported Key Block dictates how further export must be handled by the recipient of the Key Block. Hence, if the received byte 11 has value "E" then further "trusted" export is permitted, but if byte 11 has value "N" then no further export will be permitted.

Such considerations must be taken into account when the key is initially generated. It is possible to change the value of byte 11 from "E" or "S" to "N" (so that no further export will be permitted), via an optional field at the end of an HSM key export command message.

11.2.8 Number of Optional Blocks (Bytes 12-13)

The TR-31 Key Block format allows a key block to contain up to 99 Optional Header Blocks which can be used to include additional (optional) data within the Key Block. Optional Header Blocks are described below.

Bytes 12-13 of the Header specify the number of Optional Header Blocks in the key block. A value of "00" (X'3030) indicates there are no optional blocks. A value of "12" (X'3132) indicates that there are 12 optional blocks.

Example

As an example, a TR-31 Key Block Header of

A0072**V**2**T**G22**N**0000

indicates:

- a Key Block of 72 bytes,
- that the key contained in the Key Block may only be used with the Visa PVV algorithm (the "V2" field),
- that it is a TDES key (the "T" field),
- that it may only be used for PVV generation ("G"),
- the key version number is "22",
- the key may not be exported (the "N" field), and
- there are no Optional Header blocks in the Key Block.

11.3 Optional Header

The Optional Header comprises a number of Optional Header blocks, each with the structure:

Byte(s)	Field	Comments
---------	-------	----------

0-1	Identifier	Optional Header block identifier
2-3	Length	Optional Header block length (hexadecimal)
4-n	Data	Optional Header block data

Currently, only three types of optional block have been defined in TR-31:

Optional Header Block ID	Usage
KS	Key set identifier; as defined in ANSI X9.24-2004 Part 1, Annex E
KV	Key Block values; used to define the version of the set of Key Block field values
PB	Padding block; to ensure that the overall length of the Optional Header blocks is a multiple of the encryption block length
Numeric Values	Reserved for proprietary use. If proprietary identifiers are used, it is the responsibility of the application owner to ensure that all necessary devices support the proprietary Optional Block identifiers that they will use.

Other types of optional blocks will be supported by the payShield 10K as and when they are specified.

11.4 Using TR-31 Key Blocks

TR-31 Key Blocks are relevant to HSM commands that are used for key import or key export. Key Blocks themselves generally replace the existing X9.17 format encrypted keys in the HSM command or response messages.

In some cases, additional fields are required at the end of the command message to allow construction of the exported Key Block. Such fields are preceded by a special delimiter, the "&" character.

The key scheme identifier "R" is used to indicate to the HSM that the Key field is a TR-31 Key Block.

Most of the HSM legacy commands will not be upgraded to support TR-31 Key Blocks since there are more flexible versions of these commands available which will support TR-31.

11.4.1 Key Scheme Tags

The table below summarises the key scheme tags (or identifiers) relevant to key blocks supported in payShield 10K software:

Tag ID	Key Scheme
R	X9 TR-31 Key Block format
S	Thales Key Block format
V	VeriFone/GISKE Key Block format

11.5 GISKE Key Block

The GISKE Key Block is an early version of the TR-31 Key Block and is supported by some VeriFone terminals. The HSM's host A8 command has been modified to support export of terminal keys in the GISKE format. The command would be the same as for a command structured for standard TR-31 but with the "R" key scheme tag replaced with "V" to indicate that the exported key should be in GISKE Key Block format.

12 DES Key Support

12.1 DES Keys

The payShield 10K HSM host commands support single-, double and triple-length DES keys. The current command set is backward compatible with earlier versions. The commands support extensions to enable the specification of key length and key encryption scheme to use.

12.2 Key Usage

If the first character of the key is a hexadecimal character (0 – 9 or A - F) or "K" the commands will operate as previously specified. In most circumstances the key is single-length except for ZMKs when the ZMK length is configured for double-length or for specific keys that are double-length by definition. This is the 16H or 32H length and types referenced in the command structures in the *payShield 10K Host Command Reference Manual*.

To support double and triple-length keys throughout the command set key scheme tags have been defined these enable an HSM to determine the key length and encryption mechanism used for a key. The key scheme tag prefixes the key. This is the 1A+32H or 1A+48H length and types in the command structures.

12.3 Key Encryption Schemes

The HSM supports numerous encryption schemes:

12.3.1 ANSI X9.17 method

Each key of a double- or triple-length DES key is encrypted separately using the ECB mode of encryption. This scheme is available for import and export of keys and for keys encrypted under a Variant LMK. The use of this scheme must be enabled in the security settings (e.g. by using the CS (Configure Security) console command).

The tags for this scheme are:

- Z (optional) – Single-length DES keys
- X – Double-length DES keys
- Y – Triple-length DES keys

12.3.1.1 Variant method

The payShield 10K HSM supports the Thales Key Block method, for the encryption and authentication of keys.

12.3.1.2 Thales Key Block Method

The payShield 10K HSM supports the Thales Key Block method, for the encryption and authentication of keys.

12.3.1.3 X9 TR-31 Method

Information on the TR-31 key encryption scheme is provided in Chapter 11 “TR-31 Key Blocks”.

12.3.1.4 GISKE Method

For full details of the Verifone/GISKE key encryption scheme, refer to the GISKE specification [Reference 7].

12.4 Key Generate, Import and Export

All the key management commands have extensions to enable the specification of the key scheme to use when encrypting a key. This also defines the key length to generate within key generation commands. For import and export of keys the key schemes must be consistent as far as length is concerned i.e. if a double-length key is input the key scheme flag defining the output must also be for a double-length key.

The extension consists of a delimiter ";" and three single character option fields. If the extension is used all fields must be provided. If the command does not use an option, "0" or any valid value can be entered in that field. The option will be ignored during processing.

The option fields are:

- Key scheme for encrypting the output key under ZMK.
- Key scheme for encrypting the output key under LMK.
- Key check value type.

The valid values for these options are:

Key under ZMK	Z/blank	Thales Variant Method – for single-length DES keys
	U	Thales Variant Method – for double-length DES keys
	T	Thales Variant Method – for triple-length DES keys
	X	ANSI X9.17 Method – for double-length DES keys
	Y	ANSI X9.17 Method – for triple-length DES keys
	R	X9 TR-31 Key Block Method – for single/double/triple-length DES keys
	S	Thales Key Block Method – for all symmetric & asymmetric keys
Key under TMK	V	Verifone/GISKE Key Block Method – for double/triple-length DES keys
Key under LMK	Z/blank	Thales Variant Method – for single-length DES keys
	U	Thales Variant Method – for double-length DES keys
	T	Thales Variant Method – for triple-length DES keys
	X	ANSI X9.17 Method – for double-length DES keys
	Y	ANSI X9.17 Method – for triple-length DES keys
	S	Thales Key Block Method – for all symmetric & asymmetric keys
Key check value	0	KCV is 16-hex digits (backwards compatible mode) (except for DW & DY where an 8-hex KCV is returned)
	1	KCV is 6-hex digits
	2	KCV length is defined in the specific command

12.5 Rejection of Weak, Semi-Weak, & Possibly Weak Keys

All HSM commands that generate keys ensure that the standard DES weak, semi-weak, or possibly weak keys cannot be used. If the new key matches one of the listed weak, semi-weak, or possibly weak keys it is rejected and the key generation process is repeated.

12.5.1.1 DES Weak Keys

0101	0101	0101	0101
FEFE	FEFE	FEFE	FEFE
1F1F	1F1F	0E0E	0E0E
E0E0	E0E0	F1F1	F1F1

12.5.1.2 DES Semi-Weak Keys

01FE	01FE	01FE	01FE
FE01	FE01	FE01	FE01
1FE0	1FE0	0EF1	0EF1
E01F	E01F	F10E	F10E
01E0	01E0	01F1	01F1
E001	E001	F101	F101
1FFE	1FFE	0EFE	0EFE
FE1F	FE1F	FE0E	FE0E
011F	011F	010E	010E
1F01	1F01	0E01	0E01
E0FE	E0FE	F1FE	F1FE
FEE0	FEE0	FEF1	FEF1

12.5.1.3 DES Possibly Weak Keys

0101	1F1F	0101	0E0E
0101	E0E0	0101	F1F1
0101	FEFE	0101	FEFE
011F	1F01	010E	0E01
011F	E0FE	010E	F1FE
011F	FEE0	010E	FEF1
01E0	1FFE	01F1	0EFE
01E0	1FFE	01F1	F10E
01E0	E001	01F1	F101
01E0	FE1F	01F1	FE0E
01FE	1FE0	01FE	0EF1
01FE	E01F	01FE	F10E
01FE	FE01	01FE	FE01

1F01	011F	0E01	010E
1F01	E0FE	0E01	F1FE
1F01	FEE0	0E01	FEF1
1F1F	0101	0E0E	0101
1F1F	E0E0	0E0E	F1F1
1F1F	FEFE	0E0E	FEFE
1FE0	01FE	0EF1	01FE
1FE0	E01F	0EF1	F10E
1FE0	FE01	0EF1	FE01
1FFE	01E0	0EFE	01F1
1FFE	E001	0EFE	F001
1FFE	FE1F	0EFE	FE0E
E001	01E0	F101	01F1
E001	1FFE	F101	0EFE
E001	FE1F	F101	FE0E
E01F	01FE	F10E	01FE
E01F	1FE0	F10E	0EF1
E01F	FE01	F10E	FE01
E0E0	0101	F1F1	0101
E0E0	1F1F	F1F1	0E0E
E0E0	FEFE	F1F1	FEFE
E0FE	011F	F1FE	010E
E0FE	1F01	F1FE	0E01
E0FE	FEE0	F1FE	FEF1
FE01	01FE	FE01	01FE
FE01	1FE0	FE01	0EF1
FE1F	01E0	FE0E	01F1
FE1F	E001	FE0E	F101
FE1F	1FFE	FE0E	0EFE
FEE0	011F	FEF1	010E
FEE0	1F01	FEF1	0E01
FEE0	E0FE	FEF1	F1FE
FEFE	0101	FEFE	0101
FEFE	1F1F	FEFE	0E0E
FEFE	E0E0	FEFE	F1F1

13 AES Key Support

13.1 Overview

13.1.1 AES Keys

AES is (like DES and TDES) a symmetric algorithm used to encrypt data, but provides significantly higher encryption strength than DES/TDES. AES keys have lengths of 128, 192, or 256 bits. The comparative strengths of the algorithms are:

Algorithm	Key Length	Bits of Security
DES	56-bit	56
TDES	Double-length (112-bit)	80/112*
TDES	Triple-length (168-bit)	112
AES	128-bit	128
AES	192-bit	192
AES	256-bit	256

Note: Refer to NIST SP800-57 for details:

<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf>

When an AES Key Block LMK is installed, the payShield 10K supports the use of 128-bit, 192-bit and 256-bit AES keys for key management, data encryption and data authentication. AES keys are only supported when using the Thales Key Block scheme.

13.1.2 Key Management Keys

The payShield 10K supports the generation and use of two types of general purpose key encryption keys: a Zone Master Key (ZMK) and a Terminal Master Key (TMK).

The ZMK and TMK may be 128-bit, 192-bit or 256-bit AES keys, and the subordinate keys encrypted by the ZMK and TMK may be AES or DES keys.

The traditional DES-based ZMK and TMK are able to encrypt AES and DES subordinate keys.

13.1.3 Data Authentication Keys

The payShield 10K supports the generation and use of two types of data authentication keys: a Zone Authentication Key (ZAK) and a Terminal Authentication Key (TAK).

The ZAK and TAK may be 128-bit, 192-bit or 256-bit AES keys, and the supported authentication methods are CBC-MAC and CMAC.

Data Encryption Keys

The payShield 10K supports the generation and use of four types of data encryption keys: a Zone Encryption Key (ZEK), a Terminal Encryption Key (TEK), a Data Encryption Key (DEK), and an encryption key derived from a DUKPT BDK.

The ZEK, TEK and DEK may be 128-bit, 192-bit or 256-bit AES Keys, and the supported encryption modes are ECB, CBC, CFB8 and CFB64.

13.1.4 Key Encryption Schemes

There are a number of key encryption schemes supported by a payShield 10K, but only one scheme supports the use of AES keys:

- Thales Key Block Method

The payShield 10K HSM supports the Thales Key Block method, for the encryption and authentication of keys.

13.2 Support for AES

Support for the AES algorithm in the payShield 10K includes:

- 256-bit AES Key Block LMKs to protect AES working keys plus other keys (except single DES) and data protected in the HSM by LMKs.
- Use of AES for ZMKs and TMKs.
- Use of AES working keys for data encryption and MACing.
- Support for 128-, 192-, and 256-bit AES keys.
- Export/import of AES working keys using Thales Key Blocks. (TR-31:2018, Version 'D')
- Use of a Console or payShield Manager to manage AES keys and AES Key Block LMKs.

The next section in this chapter discusses implementing AES on existing payShield 10Ks. Later sections indicate how host commands, console commands, and Local payShield Manager menu options support AES.

The following table shows the keys used in the payShield 10K, indicating which keys can use AES.

Key Description	AES*	Key Block LMK Key Usage	Variant LMK Key Type ^ø	Key Type Code
Base Derivation Key (BDK)	Yes	B0	009	BDK
Base Derivation Key (BDK) Type 2	Yes	41	609	BDK2
Base Derivation Key (BDK) Type 3		42	809	BDK3
Base Derivation Key (BDK) Type 4	Yes	43	909	BDK4
Card Verification Key		C0	402	CVK
Card Verification Key (American Express CSC)		11	402	CSC K
Card Verification Key (Mastercard CVC)		12	402	CVK
Card Verification Key (Visa CVV)		13	402	CVK
CBC MAC (AES)	Yes	M5	n/a	n/a
CMAC (AES)	Yes	M6	n/a	n/a
Data Encryption Key	Yes	21	00B	DEK
Data Encryption Key (Generic)	Yes	D0	00B, 00A, 30B	DEK, ZEK, TEK
Dynamic CVV Master Key		32	709	MK-CVC3
EMV/Chip card Master Key: Application Cryptogram	Yes	E0	109	MK-AC
EMV/Chip card Master Key: Data Authentication Code		E3	409	MK-DAC

Key Description	AES*	Key Block LMK Key Usage	Variant LMK Key Typeø	Key Type Code
EMV/Chip card Master Key: Dynamic Numbers		E4	509	MK-DN
EMV/Chip card Master Key: Secure Messaging for Confidentiality		E1	309	MK-SMC
EMV/Chip card Master Key: Secure Messaging for Integrity	Yes	E2	209	MK-SMI
HMAC key (using SHA-1)		61	10C	HMA C
HMAC key (using SHA-224)		62	10C	HMA C
HMAC key (using SHA-256)		63	10C	HMA C
HMAC key (using SHA-384)		64	10C	HMA C
HMAC key (using SHA-512)		65	10C	HMA C
IPEK (DUKPT)	Yes	B1	302	IPEK
ISO 16609 MAC algorithm 1 (using 3-DES)		M0	003, 008	TAK, ZAK
ISO 9797-1 MAC algorithm 1		M1	003, 008	TAK, ZAK
ISO 9797-1 MAC algorithm 3		M3	003, 008	TAK, ZAK
Key Encryption / Wrapping Key (Generic)	Yes	K0	002/ 80D, 000	TMK, ZMK
PIN Encryption Key (Generic)	Yes	P0	002 / 70D	PEK
PIN Verification Key (Generic)		V0	002	PVK
PIN Verification Key (IBM 3624 algorithm)		V1	002	PVK
PIN Verification Key (Visa PVV algorithm)		V2	002	PVK
RSA Private Key		03	00C	RSA-SK
RSA Public Key		02	00D	RSA-PK
Terminal Key Encryption Key	Yes	51	002 / 80D	TMK
Terminal PIN Encryption Key	Yes	71	002 / 70D	TPK
Terminal/Data Encryption Key (TEK)	Yes	23	30B	TEK
Transaction Key Scheme Terminal Key Register		73	002 / 90D	TKR

Key Description	AES*	Key Block LMK Key Usage	Variant LMK Key Type ^Ø	Key Type Code
Visa Cash Master Load Key (KML)			200	KML
WatchWord Key		01	006	WWK
Zone Key Encryption Key	Yes	52	000	ZMK
Zone PIN Encryption Key	Yes	72	001	ZPK
Zone/Data Encryption Key	Yes	22	00A	ZEK

* Only available when using an AES Key Block LMK

Ø Where 2 key types are given (e.g. 002/70D), the key type depends on the security setting "Enforce key type 002 separation for PCI HSM compliance".

This table does not include keys used for AS2805, OBKM, and card issuing. The commands used in these licenses do not support key block LMKs and therefore cannot use AES keys.

As and when updates to the standards published by bodies such as ANSI X9, ISO, and EMVCo become available, the payShield 10K software will be extended to support additional AES key usages.

13.3 Implementing AES on an existing payShield 10K

13.3.1 Re-encrypting existing keys under an AES Key Block LMK

An AES Key Block LMK must be installed if it is required to use AES working keys: these keys cannot be protected using a Variant or Key Block TDES LMK.

Even if there is no requirement to use AES working keys, users may wish to use AES Key Block LMKs to protect TDES and other working keys because of the greater security strength that the 256-bit AES Key Block LMK provides compared to a TDES LMK.

Where a payShield 10K is equipped with an appropriate number of LMKs, it may not be necessary to re-encrypt existing working keys under a new AES Key Block LMK: the existing working keys could be left encrypted under the existing LMKs, with only new working keys (including AES keys) being encrypted under a new AES Key Block LMK.

The multiple LMK approach may not be satisfactory, for example if:

- The payShield 10K has an insufficient number of LMKs available
- Only one LMK can be used because the host applications have not been adapted to make use of multiple LMKs
- The AES Key Block LMK is being used to provide enhanced security for existing working keys and data.

In this case, it is necessary to re-encrypt existing working keys and data which are still required from the old LMK they are currently encrypted under to the new AES LMK.

The LMK migration process is described in Chapter 10, "*Migrating LMKs*". This process is based on:

- multiple uses of the BW host command, which re-encrypts a working key from an old LMK to a new LMK
- multiple uses of the BG host command to re-encrypt PINs
- multiple use of the LO host command to re-encrypt decimalization tables

The chapter on LMK migration also outlines how to install the AES Key Block LMK.

Note that:

Any type of working key (including key encryption keys, but excluding single-DES keys) and any data that can be encrypted under a payShield 10K LMK can be encrypted using an AES Key Block LMK. The use of AES Key Block LMKs is not limited to AES keys.

- AES keys cannot be encrypted under a TDES LMK (whether Variant or Key Block type)
- Keys can be re-encrypted from any type of LMK to an AES Key Block LMK
- Keys encrypted under an AES Key Block LMK can only be re-encrypted to another AES Key Block LMK - they

cannot be migrated to a TDES LMK of any type.

13.3.2 Creating AES working keys

Once an AES Key Block LMK has been installed, it is possible to create (i.e., generate or import) AES working keys, encrypted under the AES Key Block LMK, for use with the payShield 10K. It may also be necessary to export AES working keys.

This can be achieved with the key management facilities available using:

- Console commands
- Host commands

13.4 AES in Host commands

13.4.1 Processing data using AES keys

The commands in this table allow AES working keys to be used to process data. These commands also support the use of AES key block LMKs.

Host Command		Summary of Changes Made
M0	Encrypt Data Block	Support for AES encryption keys.
M2	Decrypt Data Block	Support for AES encryption keys.
M4	Translate Data Block	Support for Source/Destination AES encryption keys.
M6	Generate MAC	<p>Choice of MAC Algorithm extended to include: '5' : CBC-MAC (AES only) '6' : CMAC (AES only)</p> <p>Choice of Padding Method extended to include: '4' : AES CMAC Padding. Key may be a CMAC or CBC-MAC key.</p>
M8	Verify MAC	<p>Choice of MAC Algorithm extended to include: '5' : CBC-MAC (AES only) '6' : CMAC (AES only)</p> <p>Choice of Padding Method extended to include: '4' : AES CMAC Padding. Key may be a CMAC or CBC-MAC key.</p>
MY	Verify and Translate MAC	<p>Choice of Source/Destination MAC Algorithm extended to include: '5' : CBC-MAC (AES only) '6' : CMAC (AES only)</p> <p>Choice of Source/Destination Padding Method extended to include: '4' : AES CMAC Padding. Source/Destination Key may be a CMAC or CBC-MAC key.</p>

13.4.2 AES Key Management

The following key management Host commands can be used with AES working keys. These commands also support AES key block LMKs.

Host Command		Summary of Changes Made
A0	Generate a key	Choice of Algorithm when using AES key block LMKs extended to include: 'A1' – 128-bit AES key 'A2' – 192-bit AES key 'A3' – 256-bit AES key Generated key may be encrypted under an AES ZMK or TMK. AES keys can also be encrypted using a TDES ZMK/TMK.
A2	Generate and Print a Component	Choice of Algorithm when using AES key block LMKs extended to include: 'A1' – 128-bit AES key 'A2' – 192-bit AES key 'A3' – 256-bit AES key
A4	Form a Key from Encrypted Components	Allows an AES key to be formed from components and encrypted under an AES Key Block LMK.
A6	Import a Key	Allows an AES key to be imported. Support is provided for AES ZMKs, but the AES key can also be encrypted using a TDES ZMK.
A8	Export a Key	An AES key may be encrypted under a ZMK/TMK. Support is provided for AES ZMK/TMKs, but the AES key can also be encrypted using a TDES ZMK/TMK.
BY	Translate ZMK from ZMK to LMK encryption	The ZMK being translated may be an AES ZMK. And the encrypting ZMK may be an AES key. The LMK that the ZMK is to be translated to must, of course, be an AES Key Block LMK.
BW	Translate Keys from Old LMK to New LMK and Migrate to New Key Type	Supports AES keys & LMKs
NE	Generate and Print a Key as Split Components	Choice of Algorithm when using AES key block LMKs extended to include: 'A1' – 128-bit AES key 'A2' – 192-bit AES key 'A3' – 256-bit AES key

13.4.3 AES Key Block LMK support

The following host commands are able to accept keys and other data protected by AES key block LMKs.

Host Command		Summary of Changes Made
BA	Encrypt a Clear PIN	The output PIN may be encrypted using an AES Key Block LMK.

Host Command		Summary of Changes Made
BC	Verify a Terminal PIN Using the Comparison Method	The TPK may be encrypted using an AES Key Block LMK. If so, the TPK may be an AES key.
BE	Verify an Interchange PIN Using the Comparison Method	The ZPK may be encrypted using an AES Key Block LMK. If so, the ZPK may be an AES key.
BG	Translate a PIN and PIN Length	The input PIN may be encrypted using an AES Key Block LMK.
BK	Generate an IBM PIN Offset (of a customer selected PIN)	The Decimalisation Table, PIN Block Key and PVK may be encrypted using an AES Key Block LMK. If so, the PIN Block Key may be an AES key.
BQ	Translate PIN Algorithm	The input PIN may be encrypted using an AES Key Block LMK.
BS	Erase the Key Change Storage	Supports AES keys.
BU	Generate a Key Check Value	Supports AES keys.
CA	Translate a PIN from TPK to ZPK Encryption	The Source TPK and Destination Key may be encrypted using an AES Key Block LMK. If so, the Source TPK and Destination Key (only ZPK) may be an AES key.
CC	Translate a PIN from One ZPK to Another	The Source ZPK and Destination ZPK may be encrypted using an AES Key Block LMK. If so, either or both ZPKs may be an AES key.
CE	Generate a Diebold PIN Offset	The input PIN may be encrypted using an AES Key Block LMK.
CG	Verify a Terminal PIN Using the Diebold Method	The TPK may be encrypted using an AES Key Block LMK. If so, the TPK may be an AES key.
CI	Translate a PIN from BDK to ZPK Encryption (DUKPT)	Supports TDES BDK/ZPK encrypted using an AES Key Block LMK.
CK	Verify a PIN Using the IBM Method (DUKPT)	Support is provided for decimalization tables encrypted using an AES Key Block LMK.
CM	Verify a PIN Using the Visa PVV Method (DUKPT)	Supports TDES BDK/PVK encrypted using an AES Key Block LMK.
CO	Verify a PIN Using the Diebold Method (DUKPT)	Supports TDES BDK encrypted using an AES Key Block LMK.
CQ	Verify a PIN Using the Encrypted PIN Method (DUKPT)	The encrypted database PIN may be encrypted using an AES Key Block LMK.
CS	Modify Key Block Header	Supports keys encrypted using an AES Key Block LMK.
CU	Verify & Generate a Visa PVV (of a customer selected PIN)	The PIN Block Key and PVK may be encrypted using an AES Key Block LMK. If so, the PIN Block Key may be an AES key.
CW	Generate a Card Verification Code/Value	Supports TDES CVK encrypted using an AES Key Block LMK.
CY	Verify a Card Verification Code/Value	Supports TDES CVK encrypted using an AES Key Block LMK.

Host Command		Summary of Changes Made
DA	Verify a Terminal PIN Using the IBM Method	The TPK and PVK may be encrypted using an AES Key Block LMK. If so, the TPK may be an AES key.
DC	Verify a Terminal PIN Using the Visa Method	The TPK and PVK may be encrypted using an AES Key Block LMK. If so, the TPK may be an AES key.
DE	Generate an IBM PIN Offset (of an LMK encrypted PIN)	Support is provided for decimalization tables and PINs encrypted using an AES Key Block LMK.
DG	Generate a Visa PIN Verification Value (of an LMK encrypted PIN)	The input PIN may be encrypted using an AES Key Block LMK.
DU	Verify & Generate an IBM PIN Offset (of customer selected new PIN)	The Decimalization Table, PIN Block Key and PVK may be encrypted using an AES Key Block LMK. If so, the PIN Block Key may be an AES key.
EA	Verify an Interchange PIN Using the IBM Method	The ZPK and PVK may be encrypted using an AES Key Block LMK. If so, the ZPK may be an AES key.
EC	Verify an Interchange PIN Using the Visa Method	The ZPK and PVK may be encrypted using an AES Key Block LMK. If so, the ZPK may be an AES key.
EE	Derive a PIN Using the IBM Method	Support is provided for decimalization tables encrypted using an AES Key Block LMK. The generated PIN may be encrypted using an AES Key Block LMK.
EG	Verify an Interchange PIN Using the Diebold Method	The ZPK may be encrypted using an AES Key Block LMK. If so, the ZPK may be an AES key.
EI	Generate a Public/Private Key Pair	The output Private Key may be encrypted using an AES Key Block LMK.
EK	Load a Private Key	The input Private Key may be encrypted using an AES Key Block LMK.
EM	Translate a Private Key	The input and output Private Key may be encrypted using an AES Key Block LMK.
EO	Import a Public Key	The output Public Key may be protected using an AES Key Block LMK.
EQ	Validate a Public Key	The input Public Key may be protected using an AES Key Block LMK.
ES	Validate a Certificate and Import the Public Key	The input Private Key may be encrypted using an AES Key Block LMK.
EU	Translate a Public Key	The input and output Public Key may be protected using an AES Key Block LMK.
EW	Generate an RSA Signature	The input Private Key may be encrypted using an AES Key Block LMK.
EY	Validate an RSA Signature	The input Public Key may be protected using an AES Key Block LMK.
FU	Verify a Watchword Response	Supports TDES WWK may be encrypted using an AES Key Block LMK

Host Command		Summary of Changes Made
FW	Generate a Visa PIN Verification Value (of a customer selected PIN)	The PIN Block Key and PVK may be encrypted using an AES Key Block LMK. If so, the PIN Block Key may be an AES key.
G0	Translate a PIN from BDK to ZPK Encryption (3 DES DUKPT)	The input BDK and ZPK may be encrypted using an AES Key Block LMK.
GA	Derive a PIN using the Diebold method	The generated PIN may be encrypted using an AES Key Block LMK.
GI	Import Key under an RSA Public Key	The imported private key may be encrypted using an AES Key Block LMK. The output TDES key may be encrypted using an AES key block LMK.
GK	Export Key under an RSA Public Key	The input private, TDES, and HMAC keys may be encrypted using an AES Key Block LMK.
GO	Verify a PIN Using the IBM Method (3 DES DUKPT)	Support is provided for decimalization tables encrypted using an AES Key Block LMK.
GQ	Verify a PIN Using the Visa PVV Method (3 DES DUKPT)	Supports TDES BDK/PVK encrypted using an AES Key Block LMK.
GS	Verify a PIN Using the Diebold Method (3 DES DUKPT)	Supports TDES BDK encrypted using an AES Key Block LMK.
GU	Verify a PIN Using the Encrypted PIN Method (3 DES DUKPT)	The encrypted PIN may be encrypted using an AES Key Block LMK.
GW	Generate/Verify a MAC (3 DES DUKPT)	Supports TDES BDK encrypted using an AES Key Block LMK.
JA	Generate a Random PIN	The generated PIN may be encrypted using an AES Key Block LMK.
JC	Translate a PIN from TPK to LMK Encryption	The Source TPK may be encrypted using an AES Key Block LMK. If so, the Source TPK may be an AES key.
JE	Translate a PIN from ZPK to LMK Encryption	The Source Key may be encrypted using an AES Key Block LMK. If so, the Source Key may be an AES key.
JG	Translate a PIN from LMK to ZPK Encryption	The Destination ZPK may be encrypted using an AES Key Block LMK. If so, the Destination ZPK may be an AES key.
K0	Decrypt Encrypted Counters (EMV 4.x)	Input TDES MK-AC key may be encrypted using an AES Key Block LMK.
K2	Verify Truncated Application Cryptogram (Mastercard CAP)	Input TDES MK-AC key may be encrypted using an AES Key Block LMK.
KQ	ARQC Verification and/or ARPC Generation (EMV 3.1.1)	Input TDES MK-AC key may be encrypted using an AES Key Block LMK.
KS	Data Authentication Code and Dynamic Number Verification (EMV 3.1.1)	Input TDES MK-DAC and MK-DN keys may be encrypted using an AES Key Block LMK.
KU	Generate Secure Message (EMV 3.1.1)	Input TDES MK-SMI, MK-SMC, TK, source PEK, and MK-AC keys may be encrypted using an AES Key Block LMK.

Host Command		Summary of Changes Made
KY	Generate Secure Message (EMV 4.x)	Input TDES MK-SMI, MK-SMC, TK, source PEK, and MK-AC keys may be encrypted using an AES Key Block LMK.
KW	ARQC Verification and/or ARPC Generation (EMV 4.x)	Input TDES MK-AC key may be encrypted using an AES Key Block LMK.
L0	Generate an HMAC Secret Key	Output HMAC key may be encrypted using an AES Key Block LMK.
LC	Verify the Diebold Table in User Storage	The Diebold table may be encrypted using an AES Key Block LMK.
LK	Generate a Decimal MAC	Supports TDES TAK encrypted using an AES Key Block LMK.
LM	Verify a Decimal MAC	Supports TDES TAK encrypted using an AES Key Block LMK.
LO	Translate Decimalization Table from Old to New LMK	Support is provided for input/output decimalization tables encrypted using an AES Key Block LMK.
LQ	Generate an HMAC on a Block of Data	Input HMAC key may be encrypted using an AES Key Block LMK.
LS	Verify an HMAC on a Block of Data	Input HMAC key may be encrypted using an AES Key Block LMK.
LU	Import an HMAC key under a ZMK	Supports TDES ZMK encrypted under an AES Key Block LMK. The output HMAC key may be encrypted using an AES Key Block LMK.
LW	Export an HMAC key under a ZMK	Supports TDES ZMK encrypted under an AES Key Block LMK. The input HMAC key may be encrypted using an AES Key Block LMK.
LY	Translate a HMAC Key from Old LMK to New LMK	The old and new LMKs may be AES Key Block LMKs.
NG	Decrypt an Encrypted PIN	The input PIN may be encrypted using an AES Key Block LMK.
OA	Print a PIN Solicitation Mailer	Reference number check value can use an AES Key Block LMK.
PE	Print PIN/PIN and Solicitation Data	The input PIN may be encrypted using an AES Key Block LMK.
PG	Verify PIN/PIN and Solicitation Mailer Cryptography	The input PIN may be encrypted using an AES Key Block LMK.
PM	Verify a Dynamic CVV/CVC	Supports TDES MK-DCVV encrypted using an AES Key Block LMK.
Q0	Translate Audit Record MAC key	The old and new LMKs may be AES Key Block LMKs.
QA	Load Solicitation Data to User Storage	Reference number may be protected using an AES Key Block LMK.
QC	Final Load of Solicitation Data to User Storage	Reference number may be protected using an AES Key Block LMK.
RA	Cancel Authorized Activities	Authorized activities for an AES Key Block LMK may be cancelled.

Host Command		Summary of Changes Made
RC	Verify Solicitation Mailer Cryptography	Reference number check value can use an AES Key Block LMK.
RI	Transaction Request With a PIN (T/AQ Key)	The input Terminal Key Register may be encrypted using an AES Key Block LMK. The output Response MAC Residue and TPK may be encrypted using an AES Key Block LMK.
RK	Transaction Request Without a PIN	The input Terminal Key Register may be encrypted using an AES Key Block LMK. The output Response MAC Residue may be encrypted using an AES Key Block LMK.
RM	Administration Request Message	The input Terminal Key Register may be encrypted using an AES Key Block LMK. The output Response MAC Residue may be encrypted using an AES Key Block LMK.
RO	Transaction Response with Auth Para from Card Issuer	The input Terminal Key Register, ZPK, and Request MAC Residue may be encrypted using an AES Key Block LMK. The output Response MAC Residue and Terminal Key Register may be encrypted using an AES Key Block LMK.
RQ	Generate Auth Para and Transaction Response	The input Terminal Key Register and Request MAC Residue may be encrypted using an AES Key Block LMK. The output Response MAC Residue and Terminal Key Register may be encrypted using an AES Key Block LMK.
RS	Confirmation	The input Terminal Key Register and Request MAC Residue may be encrypted using an AES Key Block LMK.
RU	Transaction Request With a PIN (T/CI Key)	Support for output KEYVAL encrypted under an AES Key Block LMK.
RW	Translate KEYVAL	Support for input KEYVAL encrypted under an AES Key Block LMK.
RY	(Mode 3) Calculate Card Security Codes	Support for input CSCK encrypted under an AES Key Block LMK.
RY	(Mode 4) Verify Card Security Codes	Support for input CSCK encrypted under an AES Key Block LMK.

13.5 Support for AES in console commands

13.5.1 Managing the HSM

These console commands, used to manage the payShield 10K, take account of the AES algorithm.

Console Command		Summary of Changes Made
DT	Diagnostic Test	Tests the AES algorithm
VR	View Software Revision Number	Reports on availability of AES algorithm and relevant NIST certifications.

13.5.2 AES Key Management

The following key management console commands may be used with AES working keys. These commands also support AES key block LMKs.

Console Command		Summary of Changes Made
CK	Generate a Check Value	Can generate a value for keys, including AES keys, encrypted under an AES key block LMK.
EC	Encrypt Clear Component	Can encrypt an AES component. TDES components may be, and AES components must be, encrypted under an AES key block LMK.
FK	Form Key from Components	Can form an AES key from components. TDES keys may be, and AES keys must be, encrypted under an AES key block LMK.
GC	Generate Key Component	Can generate an AES key component. TDES components may be, and AES components must be, encrypted under an AES key block LMK.
GS	Generate Key and Write Components to Smartcard	Can generate AES keys/components. TDES keys may be, and AES keys must be, encrypted under an AES key block LMK.
IK	Import Key	Can import an AES key. The ZMK used to encrypt TDES or AES keys to be imported may be an AES key and/or may be encrypted under an AES key block LMK. Imported TDES keys may be, and imported AES keys must be, encrypted under an AES key block LMK.
KE	Export Key	Can export an AES key. For the key to be exported, TDES keys may be, and AES keys must be, encrypted under an AES key block LMK. The ZMK used to encrypt the exported keys may be an AES key and/or may be encrypted under an AES key block LMK.
KG	Generate Key	Can generate an AES key. TDES keys may be, and AES keys must be, encrypted under an AES key block LMK.

13.5.3 AES Key Block LMK support

The following console commands are able to accept keys and other data protected by AES key block LMKs.

Console Command		Summary of Changes Made
A	Enter the Authorized State / Authorize Activity	Can authorize for AES key block LMKs.
CO	Create an Authorizing Officer Smartcard	Can copy cards containing AES key block LMK components.
CV	Generate a Card Verification Value	Can use a CVK encrypted under an AES key block LMK.
DC	Duplicate LMK Component Sets	Can copy cards holding AES key block LMK components.
DM	Delete LMK	Can delete AES key block LMKs.
DO	Delete 'Old' LMK from Key Change Storage	Can delete AES key block LMKs.
ED	Encrypt Decimalization Table	Decimalisation tables can now be encrypted under an AES key block LMK.
GK	Generate LMK Component	Can generate AES key block LMK components. A single use of the command generates all the required components.
LK	Load LMK	Can load AES key block LMKs.
LO	Move 'Old' LMKs into Key Change Storage	Can work with AES key block LMKs. AES Key Block LMKs cannot be loaded as the old LMK where the new LMK is TDES (either variant or key block).
MI	Generate a MAC on an IPB	Can use an AES key block LMK.
PV	Generate a Visa PVV	Can use an AES key block LMK.
R	Load the Diebold Table	The table may be encrypted using an AES key block LMK.
TD	Translate Decimalization Table	Decimalisation tables may be translated to encryption under an AES key block LMK.
V	Verify LMK Store	Now works with AES key block LMKs
VC	Verify the Contents of a Smartcard	Can verify cards holding AES key block LMK components.
VT	View LMK Table	Reports on AES key block LMKs

13.6 Support for AES in payShield Manager

13.6.1 Managing the HSM



These payShield Manager options, used to manage the payShield 10K, have been modified to take account of the AES algorithm.

Menu Option	Summary of Changes Made
Operational / Local master keys / Duplicate	Can copy cards holding AES key block LMK components.

Menu Option	Summary of Changes Made
Status / Health Statistics/Diagnostics / Diagnostics	Tests the AES algorithm
Status / Software Info / FIPS/Licensing	Reports on availability of AES and relevant NIST certifications.

13.6.2 AES Key Block LMK support

The following payShield Manager options are able to manage LMKs protected by AES key block LMKs.

Menu Option	Summary of Changes Made
Operational / LMK Operations / Local Master Keys / Verify	Can verify status of an AES RLMK card
Operational / LMK Operations / Local Master Keys / Generate	Can generate AES key block RLMK cards.
Operational / LMK Operations / Local Master Keys / Duplicate	Can copy AES key block RLMK cards.
Operational / LMK Operations / Local Master Keys / Install	Can load AES key block LMKs.
Operational / LMK Operations / Local master Keys / 	Can uninstall or replace AES key block LMKs.
Operational / LMK Operations / Key Block Storage / Install	Can install AES key block LMKs into Key Change Storage. AES Key Block LMKs cannot be loaded as the old LMK where the new LMK is TDES (either variant or key block).
Operational / LMK Operations / Key Block Storage / 	Can uninstall or replace AES key block LMKs in Key Change Storage

14 User Storage

14.1 Introduction

The most common way of storing and managing keys and other sensitive data is on the host system, outside of the payShield 10K. In this scenario, the data is protected by being encrypted using the HSM's Local Master Key (LMK), the volume of stored data and its retrieval mechanism is limited only by the host system's capacity and functional capabilities (such as database systems), and the keys do not need to be replicated to multiple locations. The data can be backed up and replicated as desired - e.g., to create Disaster Recovery data centers.

In addition, keys and other data can also be stored in the User Storage area inside the payShield 10K itself. This method does not provide the capacity or flexibility that is experienced when storing at the host, but ensures that the data is protected by the HSM's security.

14.2 User Storage Area

The User Storage can be used to store any data, not just LMK-encrypted keys. This can be any data that the user wants to protect, and the following data types that are used by the payShield 10K:

- PIN Solicitation data
- PIN Blocks
- The Diebold table
- Decimalization tables

The user storage area has a capacity of 98,304 Bytes. It can hold up to 4,096 blocks of data:

- Each block is identified by an index, with values ranging from "000" to "FFF" in hexadecimal characters (equivalent to 0 to 4,095 in decimal).
- Blocks have either:
 - A fixed size (of 16, 32, or 48 hexadecimal characters - corresponding to the size of single-, double-, and triple-length TDES keys), or
 - A variable sizedepending on the block size setting.

- Blocks can contain any information that the user desires

The user storage area is in tamper-protected volatile memory, and so the stored data will be lost on events such as a tamper, a power cycle, use of the RESET console command

There are two different modes of User Storage:

- Fixed Block Size User Storage
- Variable Block Size User Storage

For each mode, the following facilities for storing data are supported:

- General facility to store any data
- Specific method for storing and referencing RSA and ECC keys

Using the general facility to store and retrieve any data:

- Data is stored in User Storage using Host Command 'LA' (Load Data to User Storage).
- Data can be retrieved using Host Command 'LE' (Read from User Storage).
- Alternatively, keys in the appropriate format in User Storage can be referenced using a "Key Reference" (e.g. SK123). This is passed into host commands instead of the key. Restrictions on this facility are provided later below.

Using the specific Key Index facility for RSA and ECC keys:

- Host Command 'EK' (Load Private Key) is used to load a private RSA or ECC key into the Key Index storage

- For use with RSA keys, a Key Index can be used with Host Commands EW, GI, GK
- For use with ECC keys, currently a Key Index can be used with Host Command EW only.
- Key Index storage for up to 21 RSA or ECC private keys is supported with both Fixed and Variable Block Size User Storage

Please note, the following restrictions apply when using Fixed Block Size User Storage:

- Key References are not supported for RSA and ECC keys for any LMK type.
- Key References are not supported for any key type when using a Key Block LMK.

14.3 Defining Block Size

The block size is set using the CS console command or using payShield Manager under Configuration / Security Settings / General.

Using the CS console command for an example, the relevant prompt is:

User storage key length [S/D/T/V](SINGLE):

Valid entries are S, D, T, or V (or, in fact, any string beginning with these characters) representing the following block sizes:

S = Fixed Block Size for Single-length keys, i.e., block size of 16 hexadecimal characters (8 Bytes)

D = Fixed Block Size for Double-length keys, i.e., block size of 32 hexadecimal characters (16 Bytes)

T = Fixed Block Size for Triple-length keys, i.e., block size of 48 hexadecimal characters (24 Bytes)

V = Variable Block Size

The Variable block size capability provides more flexibility than the S/D/T options, and operates differently. In the sections that follow, the operation of User Storage when the block size is selected to be Single/Double/Triple-length is described first. The operation of User Storage when Variable Block Size is selected is described later.

14.4 User Storage when using Fixed Block Sizes

14.4.1 Loading Data into User Storage with fixed block sizes

Data is loaded into User Storage using the LA host command. This command includes the following fields:

- Index Address (3H - Valid range "000" to "FFF") - the index pointing to the location where the first data block is to be written.
- Block Count (2H - Valid range "00" to "20") - the number of data blocks which are included in the command. Up to 32 data blocks can be included with the command. The sum of [(Index Address) + (Block Count-1)] must not exceed FFF (4,095 decimal).
- Data Blocks - a number of blocks of data to be stored. Data must be entered as ASCII-encoded hexadecimal characters (e.g. the byte with a binary value 00111101, hexadecimal 3D, would be entered as the two characters "3" and "D" and stored as 2 bytes.) The number of blocks must correspond to the value of the Block Count field. The size of the data block must correspond to the length set using CS or payShield Manager Configuration / Security Settings / General; if the data has a shorter length than this, the block should be padded with hexadecimal F characters.

Keys shorter than the specified block size can be stored by padding the key to the specified block size with hexadecimal F characters.

14.4.2 Reading Data from User Storage with fixed block sizes

Data can be read from user storage using the LE host command. The command specifies:

- Index Address (3H) - the index pointing to the location where the first data block is to be read from. (Valid range is "000" to "FFF".)
- Block Count (2H) - the number of data blocks which are to be retrieved. Up to 32 data blocks can be requested.

(Valid range is "00" to "20".) The sum of [(Index Address) + (Block Count-1)] must not exceed FFF (4,095 decimal).

14.4.2.1 Storing and accessing Symmetric Keys

The host system will receive Variant LMK-encrypted TDES keys from the payShield 10K when, for example, the HSM is used to generate keys (A0 host command) or import a key (A6 host command). Typically the LMK-encrypted keys are stored in the host system's key database, but optionally the host system, can store the LMK-encrypted key inside the payShield 10K user storage, by using the method outlined above. The stored key in this case does not include the alphabetic key scheme indicator (e.g., "U" for double-length Triple DES, "T" for Triple-length Triple DES).

There are two ways that LMK-encrypted keys can be retrieved from user storage by the host system. First, the data can be read by the host system using the LE host command, as outlined above, and then inserting the returned key into a host command which is then sent to the payShield 10K.

Alternatively, host commands can point to an LMK-encrypted key held in user storage, avoiding the need to first retrieve the encrypted key from the payShield 10K. In order to do this, the field in the host command which would normally provide the LMK-encrypted key would instead include a key reference consisting of:

- The key scheme - e.g., [null] for single-length DES, "U" for double-length Triple DES, "T" for Triple-length Triple DES
- "K" to indicate that what follows is an index to user storage
- The 3-hex character index to the LMK-encrypted key in user storage

For example:

LMK-encrypted key	Type	Index in User Storage	Key in host command if key provided by host	Key reference in host command if key stored in user storage
9A73BC83A90012BD	SingleDES	3A8	9A73BC83A90012BD	K3A8
23A7BB616A198EF F1298FAE25D3A4 BF5	Double TDES	B17	U23A7BB616A198EF F1298FAE25D3A4BF 5	UKB17
18A7DE43129C2C D6BBA965823F5A7 8A99987AA4F5890 B0C3	Triple TDES	179	T18A7DE43129C2C D6BBA965823F5A78 A99987AA4F5890B0 C3	TK179

Note that it is possible to store and recover keys of a length inconsistent with the Block Size. This is useful, for example, if it is necessary to store keys of different lengths.

- If the Block Size is set to Single (i.e. 16 hex characters), user storage can still be used to store double and triple-length keys. In this case, 2 blocks must be allocated for a double-length key, and three blocks for a triple-length key. When inserting a key reference in a command (e.g. "UKB17"), the appropriate number of blocks will be retrieved. Working in this way reduces the number of keys that can be stored because more than one index address is required for a key - for example, if only double-length keys were being used, then only 2,048 keys could be stored.
 - As an example, if the double-length encrypted key 92385025801691228682054947200919 was to be stored at index 0FE, then the example of the LA host command entitled "LA Command - Load Data to User Storage (double-length TDES key encrypted under a variant LMK)" could be used.
- If the Block Size is set to Triple (i.e. 48 hex characters), for example, single- or double-length keys can still be stored. However, the shorter keys must be padded with hexadecimal F characters to bring them up to the Triple Block size.

14.4.3 Key Block LMKs

When using Key Block LMKs, the LMK-encrypted key can be written to user storage using the LA host command. The host system can later read the key back from user storage using the LE host command and insert the it (with the preceding key scheme indicator of "S") into another host command sent to the payShield 10K.

Note that the key block must be expanded to hex encoding. For example, the following key block:

```
0007271TN00S0002B7CC796C2A4909FF12174898C4286E5AE2E8230E89284ED715D1D7F0
```

should be written to user storage as:

```
30303037323731544E303053303030324237434337393643324134393039464631323137343839384334323836453541453245383233304538393238344544373135443144374630
```

plus any required "F" padding.

If the User Storage Block Size has been defined as Double (32 hexadecimal characters) then this key would be written as the following 5 blocks:

```
30303037323731544E30305330303032
42374343373936433241343930394646
31323137343839384334323836453541
45324538323330453839323834454437
3135443144374630FFFFFFFFFFFFFFFFFFFF
```

The facility to use key references in commands (e.g., "SK123") is not available when using key block LMKs.

14.4.4 Storing general data

As has been mentioned, any hexadecimal character string can be written to User Storage and subsequently retrieved.

For example, let's say we wanted to store the following information into User Storage:

```
Mary had a little lamb,
Its fleece was white as snow.
```

This would have to be converted to a hexadecimal string:

```
4D617279206861642061206C6974746C65206C616D622C0D0A49747320666C656563652077617
320776869746520617320736E6F772E
```

If the User Storage Block Size has been set to Double (i.e., 32 Hexadecimal characters) this input would have to be written to User Storage as four 32-character blocks, padded with Hexadecimal F:

```
4D617279206861642061206C6974746C
65206C616D622C0D0A49747320666C65
65636520776173207768697465206173
20736E6F772EFFFFFFFFFFFFFFFFFFFF
```

14.4.5 PIN Solicitation Data

When a card issuer has solicited a desired PIN from the cardholder, the cleartext PIN provided by the cardholder must be encrypted and passed, together with the PAN, to the card issuer system. In order to ensure that the cleartext PAN and cleartext PIN do not exist together on the host system, a reference number is substituted for the PAN. A process is required to convert the reference number + cleartext PIN into a PAN + encrypted PIN. To do this, the reference number and PIN are loaded into user storage using the QA and QC commands.

Up to 1,260 records can be loaded using the QC command, and multiple preceding QA commands (each containing up to 25 records) can be used to load a total of 2,520 records. The user cannot define where this data is to be written in the user storage area.

The PIN solicitation data overwrites existing data in user storage. Therefore any user storage data must be reloaded after PIN Solicitation has been performed.

The PIN solicitation process is described in Chapter 3, "PIN Printing and Solicitation".

14.4.6 PIN Blocks

If the user storage area has been used to store PIN Blocks, host commands requiring input of a PIN Block can directly access the PIN Block held in user storage. This is achieved by replacing the input PIN Block in the command with the ["K" + index] structure described earlier for referencing keys held in user storage.

14.4.7 Diebold Table

If a Diebold Table is loaded in the payShield 10K, it is loaded into user storage using the R console command. The table consists of 32 contiguous blocks of 16 hexadecimal characters each. Each LMK has its own Diebold table.

When using the R console command, the user must enter the Index Address for the start of the table. This can have any value from "000" to "FE0" (to ensure that the 32nd block is within the index limit of FFF).

The Diebold Table shares the same storage blocks as other data in user storage, and so the user must ensure that the 32 blocks allocated to the Diebold table are not being used to store any other data.

The Diebold Table stored in the HSM is encrypted using the LMK. For variant LMKs, this will use LMK pair 14-15 variant 0 or LMK pair 36-37 variant 6, depending on the security settings relating to PCI HSM compliance.

The Diebold Table in user storage can be verified using the LC host command.

14.4.8 Decimalization Tables

Plaintext and Variant LMK-encrypted decimalization tables can be loaded into storage as blocks of 16 hexadecimal characters in the same way as TDES keys or other data.

Where the decimalization table is required in host commands (such as BK, DE, EE, GO), it can be referenced in the command using the same ["K" + index] structure described earlier for symmetric keys.

14.4.9 Storing a mix of data types

As an example, let's say we have set User Storage Block Size to Double (i.e., 32 Hexadecimal characters) and we want to store:

- The key block example in the section on User Storage and Key Block LMKs, at index decimal 123 (hexadecimal 07B)
- The single-length variant-LMK encoded key 9A73BC83A90012BD, at the next available index - which would be 128 (hexadecimal 080)
- The example data in the section on Storing general data, at the next available index - which would be 129 (hexadecimal 081)

This could be done using a single LA host command, where:

- Index address = "07B"

- Block Count = "0A" (i.e., decimal 10)
- The following 10 data blocks are provided:

30303037323731544E30305330303032	} Key Block (5 blocks)
42374343373936433241343930394646	
31323137343839384334323836453541	
45324538323330453839323834454437	
3135443144374630FFFFFFFFFFFFFFFFFFFF	
9A73BC83A90012BDFFFFFFFFFFFFFFFFFFFF	- Single-length key (1 block)
4D617279206861642061206C6974746C	} General data (4 blocks)
65206C616D622C0D0A49747320666C65	
65636520776173207768697465206173	
20736E6F772EFFFFFFFFFFFFFFFFFFFF	

Alternatively, this could be done using 3 LA host commands, where:

- Command 1 – Key Block
 - Index address = "07B"
 - Block Count = "05"
 - The following 5 data blocks are provided:


```
30303037323731544E30305330303032
42374343373936433241343930394646
31323137343839384334323836453541
45324538323330453839323834454437
3135443144374630FFFFFFFFFFFFFFFFFFFF
```
- Command 2 – Single-length Key
 - Index address = "080"
 - Block Count = "01"
 - The following 1 data block is provided:


```
9A73BC83A90012BDFFFFFFFFFFFFFFFFFFFF
```
- Command 3 – General Data
 - Index address = "081"
 - Block Count = "04"
 - The following 4 data blocks are provided:


```
4D617279206861642061206C6974746C
65206C616D622C0D0A49747320666C65
65636520776173207768697465206173
20736E6F772EFFFFFFFFFFFFFFFFFFFF
```


14.5 User Storage when using Variable Block Sizes

When V is selected for the block size, RSA and ECC keys and Key Block LMK-encrypted keys can be held in User Storage, and to be referenced by host commands.

Note: Stored keys and data do not have to be of a fixed length. This eliminates any need to specify a block size or to pad data blocks to the appropriate size.

The sections that follows assume that the reader is familiar with the information in the earlier section on the operation of User Storage when using fixed block sizes, and therefore identifies only the differences from that information.

14.5.1 Enabling the Enhanced Functionality

The option V(ARIABLE) must be selected for the User storage key length parameter in the payShield 10K Security Settings:

User storage key length [S/D/T/V](VARIABLE):

If the S(ingle), D(ouble), or T(riple) options are selected, user storage operates as described earlier in this chapter, and the enhanced functionality for variable-length user storage is not available.

If the V(ariable) option is selected, the enhanced functionality described here is activated. In this case, the user storage data structure is incompatible with that for the other options, and any existing data in user storage must be re-loaded.

The facility described earlier to store up to a total of 21 RSA or ECC private keys in the HSM continues to be available even if the V(ariable) option is selected.

14.5.2 New Data Structure

The enhanced user storage allows for 4,096 entries, as previously, but the total storage space has been increased by more than a factor of 5. These entries can now be of variable length, subject to the following limits:

- For Index Addresses 000-07F (0-127 decimal), the maximum length is 1,000 bytes. This enables the storage of longer data items such as RSA keys or the Diebold Table, but can be used to store any data such as AES or TDES keys.
- For Index Addresses 080-FFF (128-4095 decimal, i.e., 3,968 entries), the maximum length is 100 bytes. This is designed for use with shorter data items such as AES or TDES keys.

Data to be stored can now be defined as being ASCII, ASCII-encoded hexadecimal, or binary. It is no longer necessary to pad data to the block size.

The stored data can, as previously, be keys, PIN solicitation data, a Diebold table, decimalization tables, or any other data that the user desires.

14.5.3 Loading Data into user storage

As previously, data is entered into user storage using the LA host command. However, the command is modified in the following way:

- A single data item can be loaded at a time, and as a result there is no need to specify the number of blocks being loaded
- The data being loaded can be ASCII, ASCII-encoded hexadecimal, or binary. The command defines which type-of data is to be loaded
- To allow for variable-length data, the command now includes a field to specify the data length

Where the loaded data represents keys, the keys can be encrypted using any type of LMK (i.e., including AES and TDES Key Block types).

Note that ECC keys can only be encrypted using an AES key block LMK.

14.5.4 Storing a Diebold table

The Diebold Table must be stored as a single block of 512 hexadecimal characters in one of the longer user storage blocks with an index in the range 000-07F (000-127 in decimal).

14.5.5 Reading Data from User Storage

Data can still be read from user storage using the LE host command. As only a single record is returned, a block count is no longer required by the command.

14.5.6 Referencing keys stored in user storage

Once a key is held in user storage it can be referenced by host commands which normally provide the key as part of the command. This now applies to any key type encrypted under any LMK type. The reference is constructed from the Index Flag "K" plus the Index Address of the key in user storage: the value of "K" for the Index Flag is used irrespective of the Index Flag used in the LA host command when the key was loaded into User Storage.

Here are some examples. <...> indicates binary data represented here as hexadecimal characters.

LMK-encrypted key	LMK Type	Key Type	Index in User Storage	Key in host command if key provided by host	Key reference in host command if key stored in user storage
9A73BC83A90012BD	Var TDES	Sngle DES	3A8	9A73BC83A90012BD	K3A8
23A7BB616A198EFF1298FAE25D3A4BF5	Var TDES	Dble TDES	B17	U23A7BB616A198EFF1298FAE25D3A4BF5	UKB17
18A7DE43129C2CD6BBA965823F5A78A99987AA4F5890B0C3	Var TDES	Triple TDES	179	T18A7DE43129C2CD6BBA965823F5A78A99987AA4F5890B0C3	TK179
00072B1TN00S00013FC2C2BD71EFA6A1F65A01CC3EA930CED51E07D8A818AC06742D1651	Kbl TDES	Dble TDES	A12	S00072B1TN00S00013FC2C2BD71EFA6A1F65A01CC3EA930CED51E07D8A818AC06742D1651	SKA12
<AC5DC5A50CA3D9293629994FF8452E767 ... (344 bytes of binary data) ... 402E0399AC47527F7E881BA68F1> (344 bytes of binary data)	Var TDES	RSA Prv 1024	01C	<AC5DC5A50CA3D9293629994FF8452E767 ... (344 bytes of binary data) ... 402E0399AC47527F7E881BA68F1> (344 bytes of binary data)	K01C
0037603RN00N02020005TPB0B4vzoOWb<F343F5C367CB557BC217195EBFD4821 ... (336 bytes of binary data) ... 09E3AD2CFCE4B045CEC4D2C23061455>99653B1A	Kbl TDES	RSA Prv 1024	06E	S0037603RN00N02020005TPB0B4vzoOWb<F343F5C367CB557BC217195EBFD4821 ... (336 bytes of binary data) ... 09E3AD2CFCE4B045CEC4D2C23061455>99653B1A	K06E

14.6 Storing RSA and ECC Keys Using Key Indexes

The User Storage mechanism, when using fixed block size as described above, is not ideal for storing RSA keys in the HSM because of the length and variability on length of such keys. Therefore if fixed block sizes for user storage has been specified, the payShield 10K provides a separate mechanism for storing up to 21 LMK- encrypted RSA and ECC private keys inside the HSM and referencing them using an alternative method in the selected host commands.

Note: ECC keys can only be used with an AES Key Block LMK and in this case the facility to use key references in commands (e.g., "SK123") is not available when using key block LMKs.

14.6.1 Storing the RSA and ECC Private Key

The EK host command is used to store LMK-encrypted RSA and ECC private keys inside the payShield 10K. The RSA keys can be encrypted using Variant or Key Block LMKs. The ECC keys can only be encrypted using an AES Key Block LMK.

The command requires a Key index in the range 00-20 (decimal) to identify which location the key is to be loaded to.

14.6.2 Using RSA and ECC Private Keys held in the payShield 10K

Once an RSA or an ECC private key has been stored inside the payShield 10K using the EK command, it can be directly referenced by relevant host commands which need to use it - EW, GI, GK for RSA and in the current release just EW for ECC.

These commands include a Private Key Flag field to identify the location (i.e., 00-20) where the required private key is stored. (Entering 99 into this field indicates that the private key is included in the command rather than being stored in the HSM.)

14.7 Host Command Summary

The following Host commands are typically used with User Storage:

- CA Command - Translate a PIN from TPK to ZPK Encryption (using keys and PIN Block held in user storage)
- DE Command - Generate an IBM PIN Offset
- EW Command - Generate a signature (with variant LMK-encrypted RSA key held in user storage)
- EW Command - Generate a signature (with key block LMK-encrypted RSA key held in user storage)
- JE Command - Translate a PIN from ZPK to LMK encryption
- JG Command - Translate a PIN from LMK to ZPK encryption
- LA Command - Load Data to User Storage (double-length TDES key encrypted under a variant LMK)
- LA Command - Load Data to User Storage (RSA private key encrypted under a TDES variant LMK)
- LA Command - Load Data to User Storage (RSA private key encrypted under a TDES key block LMK)
- LE Command - Read Data from User Storage (single/double/triple block size setting)
- LE Command - Read Data from User Storage (variable block size setting)
- M0 Command - encrypt a block of data (using a key block LMK-encrypted key held in user storage)

15 SNMP

15.1 Introduction

The payShield 10K supports industry-standard SNMP (Simple Network Management Protocol) to provide information about the payShield 10K's state to external management devices. Information can be obtained from the payShield 10K in two operational modes:

- By the management device polling the payShield 10K to request information;
- By the payShield 10K automatically providing an SNMP "Trap" when a significant event has occurred. The use of traps reduces network traffic compared with polling, but some polling will generally still be required, for example to detect if the HSM has been taken offline or has failed.

The information provided by the payShield 10K is defined in its SNMP MIB (Management Information Base). The MIB is provided with the payShield 10K software, and is included in this manual at Appendix F "*SNMP MIB*".

15.2 Network Connectivity

SNMP traffic to and from the payShield 10K can use any one of the following 3 Ethernet ports:

- Host Port 1
- Host port 2
- Management Port

SNMP is enabled/disabled and the desired port selected by using the SNMP console command. Users of payShield Manager should use the Configuration / SNMP Settings dialogue box:

ary Status Operational Domain **Configuration** Virtual Console Quick Links Terminate Session

SNMP Settings

SNMP State

Enabled ☐

Enabled on Port: Management

Undo Apply

Version 3 (V3) Users

Name	Authentication Algorithm	Privacy Algorithm
	None	None
	Password	Password

Clear

The ports themselves are configured using the CH, CM, and CA console commands. payShield Manager users can configure host ports using Configuration / Host Settings and the management port using Configuration / Management Settings.

SNMP traffic to the payShield 10K is received on UDP port 161, and SNMP traffic is sent to a specified UDP port (usually 162) at the management device.

15.3 SNMP Version

payShield 10K supports SNMP version 3.

Users are set up under a username with details of authentication/privacy algorithms and passwords. This configuration is done using the SNMPADD (or SNMPEL to delete a User) console command or payShield Manager Configuration / SNMP Settings.

15.4 Configuring Traps

For Traps to be sent by the payShield 10K, both SNMP (e.g., using the SNMP console command) and traps (e.g., using the TRAP console command) must be enabled.

Multiple traps can be configured:

- Current Traps can be viewed using the TRAP console command.
- The UDP port at the management device and the User who is to receive the trap are configured using the TRAPADD console command.
- Traps are deleted using the TRAPEL console command.

15.5 Information provided by the payShield 10K through SNMP

See the MIB at Appendix F “*SNMP MIB*”, for the detailed definition of information available.

15.5.1 Traps issued by the payShield 10K

A Trap message is issued whenever a test in the daily diagnostic self-tests fails.

The trap is issued using the ps10KDiagnosticTestFailureAlarm MIB object, and the following supporting information is provided:

- The test which has failed is identified in ps10KDiagnosticID: see Appendix F “*SNMP MIB*” for the list of test IDs which may be reported, and
- The failed test is also described as a string in ps10KDiagnosticString

15.5.2 Tamper

If the payShield 10K detects a tamper, a Trap is sent when the unit restarts.

The Trap is issued as MIB object ps10KTamperAlarm, and the following supporting information is provided:

- Cause of the tamper is reported in ps10KStateTamperCause
- Date and time of the tamper is reported in ps10KStateTamperDate
- Current state of the unit is reported in ps10KStateTamperState

15.5.3 Powering up

When the payShield 10K is powered up, a Trap is sent using MIB object ps10KPowerOnAlarm.

15.5.4 Use of the Erase button

When the Erase button on the payShield 10K is used, a Trap is sent using MIB object ps10KEraseAlarm and the following information is provided:

- Date and time of the use of the Erase button is provided in ps10KAlarmEraseTimeandDate

15.5.5 Fraud Detection

If a fraud attempt is detected, a Trap is issued in the ps10KFraudAlarm MIB object. The following supporting information is available:

- Cause of the fraud detection in ps10KFraudType

15.5.6 Installation of a new license

A Trap is issued when an attempt is made to install a new license on the payShield 10K, using MIB object ps10KNewLicenseAlarm.

15.5.7 Installation of new software

A Trap is issued when an attempt is made to install new software on the payShield 10K, using MIB object ps10KNewSoftwareAlarm.

15.5.8 Power Supply Unit (PSU) failure

When a PSU failure is detected, a Trap is issued using MIB object ps10KPSUFailureAlarm. Supporting information is:

- Identity of the failed PSU in ps10KPsuNumber

15.5.9 Abnormal fan speed

When a fan running at an abnormal rotational speed is detected, a Trap is issued using MIB object ps10KFanAlarm, with the following supporting information:

- Identity (1 or 2) of the abnormal fan in ps10KFanID
- Nature of the abnormality in ps10KFanState

15.5.10 New Error Log entry

If a new entry is put into the Error Log a trap is issued using MIB object ps10KErrorlogAlarm. The following supporting information is provided:

- New error log entry in ps10KErrorLogData

15.5.11 Invalid or unexpected data received at a host port

A Trap is issued if there is a protocol violation in data received at a host port. This is notified in MIB object ps10KHostPortBadDataAlarm. The following supporting information is also provided:

- Physical port involved, in ps10KBadDataPhysicalPort
- Protocol involved in ps10KBadDataProtocol

15.5.12 Actual or impending battery problem

The payShield 10K cannot function if the battery maintaining the volatile memory fails. If the battery fails or is expected to do so shortly a Trap is issued using MIB object ps10KBatteryAlarm. The following supporting information is provided:

- battery state in ps10KBatteryState

16 PIN Block Formats

16.1 General

For PIN verification and PIN translation, the HSM requires that the PIN to be input as an encrypted 16-character PIN block. The plaintext data within the PIN block comprises the PIN and some form of padding to ensure that this data is 16 characters. The padding mechanism is known as the PIN block format. The HSM supports a number of PIN block formats, each identified by a 2-digit PIN block format code. Formats 34, 35, 41 and 42 are used for EMV PIN change operations and are only available to the KU and KY commands.

16.1.1 PCI HSM Compliance

In order to comply with the requirements of the PCI HSM Certification, there are restrictions on the PIN Block format translations and usage which are allowed. These are enforced by the payShield 10K security setting "Restrict PIN block usage for PCI compliance".

16.2 Thales PIN Block Formats

The following table lists the PIN Block formats supported by the standard base software, together with their default state: each format is described below. Note that PIN Blocks implemented as part of a custom software development will always be enabled.

Thales Format	ISO Format	Description	Algorithm supported	Default
01	0	ISO 9564-1 & ANSI X9.8 format 0	DES/3DES	Enabled
02	-	Docutel ATM format	DES/3DES	Disabled
03	-	Diebold & IBM ATM format	DES/3DES	Disabled
04	-	PLUS Network format	DES/3DES	Disabled
05	1	ISO 9564-1 format 1	DES/3DES	Enabled
34	2	Standard EMV 1996 format	DES/3DES	Disabled
35	-	Mastercard Pay Now & Pay Later format	DES/3DES	Enabled
41	-	Visa/Amex new PIN only format	DES/3DES	Enabled
42	-	Visa/Amex new & old PIN format	DES/3DES	Enabled
47	3	ISO 9564-1 & ANSI X9.8 format 3	DES/3DES	Enabled
48	4	ISO 9564-1 format 4	AES	Enabled

16.2.1 Format 1

Format 01 is the ISO 9564-1 Format 0, equivalent to the ANSI X9.8 Format 0 PIN Block, and can only be encrypted using a DES/3DES key.

The format combines the customer PIN and account number as follows:

- A 16-digit block is made from the digit 0, the length of the PIN, the PIN, and a pad character (hexadecimal F). For example, for the 5-digit PIN 92389, the block is:
0592 389F FFFF FFFF
- Another 16-digit block is made from four zeros and the 12 right-most digits of the account number, excluding the check digit. For example, for the 13-digit account number 4000 0012 3456 2, where the check digit is 2, the block is:

0000 4000 0012 3456

- The two blocks are exclusive-OR added:

	05	92	38	9F	FF	FF	FF	FF
	00	00	40	00	00	12	34	56
PIN block:	05	92	78	9F	FF	ED	CB	A9

16.2.2 Format 02

Format 02 supports Docutel ATMs, and can only be encrypted using a DES/3DES key. A PIN block is created from the PIN length, a 6-digit PIN, and a user-defined numeric padding string.

If the PIN has fewer than 6 digits, it is left-justified and zero filled.

For example, for the 5-digit PIN 92389, the PIN digits are 923890.

With pad characters added, the PIN block could be, for example:

5923 8909 8765 4321

Where 98765 4321 is the padding string.

16.2.3 Format 3

Format 03 supports Diebold and IBM ATMs, and can only be encrypted using a DES/3DES key. It also applies to the Docutel format that does not include a PIN length. The PIN block is created from the customer PIN and the hexadecimal F padding character. For example, for the 5-digit PIN 92389, the PIN block is:

9238 9FFF FFFF FFFF

16.2.4 Format 4

Format 04 is the PIN block format adopted by the PLUS network, and can only be encrypted using a DES/3DES key. The format combines the customer PIN and the related account number as follows:

- A 16-digit block is made from the digit 0, the length of the PIN, the PIN, and a pad character (hexadecimal F). For example, for the 5-digit PIN 92389, the block is:

0592 389F FFFF FFFF

- Another 16-digit block is made from four zeros and the left-most 12 digits of the account number. For example, for the 16-digit account number 2283 4000 0012 3456, where the check digit is 6, the block is:

0000 2283 4000 0012

The two blocks are exclusive-OR added:

	0592	389F	FFFF	FFFF
	0000	2283	4000	0012
PIN block:	0592	1A1C	BFFF	FFED

Notes: Any transaction that requires a PIN block as a parameter accepts Format 04. The major impact of this format is on the account number field length: when a PIN block is formatted according to Format 04, the account number field becomes 18 digits in length.

For the PIN translation CA and CC commands, there are two format fields; if either is 04, the account number field must be 18 digits. If the account number is less than 18 digits, it must be right-justified and padded with X'F' on the left.

The following commands can use this format:

BC, BE, CA, CC, CG, DA, DC, EA, EC, EG, G0, JC, JE, BK, FW, CU, DU, JG, KU, KY, GO, GQ, GS, GU and CI.

When reviewing the details for these commands, consider the change to the account field that this format requires.

16.2.5 Format 5

Format 05 is the ISO 9564-1 Format 1 PIN Block, and can only be encrypted using a DES/3DES key.

The PIN block is represented by the following 16 hexadecimal values:

1NPP..P R . . R

Where:

N is the PIN length (4 - C),

PP..P is the N-digit PIN,

R . . R is random padding.

The following validity checks are carried out on incoming Format 05 PIN blocks:

- The first character of the PIN block has value 1
Error Code 20 is returned if this check fails
- The PIN digits (in positions 3 - (N+2)) are in the range 0 to 9
Error Code 20 is returned if this check fails.
- The second character (N) is in the hexadecimal range 4 - C
Error Code 24 is returned if this check fails.

16.2.6 Format 34

Format 34 is the ISO 9564-1 Format 2 PIN Block (the standard EMV PIN block format), and can only be encrypted using a DES/3DES key. It is only available as an Output from EMV PIN change commands (KU and KY).

The PIN block is created from a fixed Control Field, the length of the PIN, the customer PIN itself and the hexadecimal F padding character. PINs from 4 to 12 digits in length are accommodated.

The 16-digit (8 byte) block is constructed as follows:

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F
---	---	---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----	---	---

Where:

C is a fixed control field of binary value 0010 (X'2).

N is the length of the PIN and can be any binary value from 0100 to 1100 (X'4 to X'C).

P is a digit of the PIN and can be any binary value from 0000 to 1001 (X'0 to X'9).

P/F is either a PIN digit or the binary value 1111 (X'F) filler depending on the length of the PIN.

F is a filler of binary value 1111 (X'F).

Thus for a 5 digit PIN of 34567, the PIN block would hold the 16 hexadecimal values as shown below:

2	5	3	4	5	6	7	F	F	F	F	F	F	F	F	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

16.2.7 Format 35

PIN Block format 35 is the PIN Block required by Europay/Mastercard for their Pay Now & Pay Later products, and can only be encrypted using a DES/3DES key.

The PIN block is created from a fixed Control Field, the length of the PIN, the customer PIN itself and the customer account number. PINs from 4 to 12 digits in length are accommodated.

A 16-digit (8 byte) block is constructed as follows:

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F
---	---	---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----	---	---

Where:

- C is a fixed control field of binary value 0010 (X'2).
- N is the length of the PIN and can be any binary value from 0100 to 1100 (X'4 to X'C).
- P is a digit of the PIN and can be any binary value from 0000 to 1001 (X'0 to X'9).
- P/F is either a PIN digit or the binary value 1111 (X'F) filler depending on the length of the PIN.
- F is a filler of binary value 1111 (X'F).

Thus for a 5 digit PIN of 34567, the block would hold the 16 hexadecimal values as shown below:

2	5	3	4	5	6	7	F	F	F	F	F	F	F	F	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Another 16-digit block is made from four zeros and the 12 right-most digits of the account number, excluding the check digit.

For account number 1234 0000 0123 4562 where 2 is the check digit, the block is:

0	0	0	0	4	0	0	0	0	0	1	2	3	4	5	6
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The two blocks are exclusive-ORed on a bit by bit basis:

2	5	3	4	5	6	7	F	F	F	F	F	F	F	F	F
0	0	0	0	4	0	0	0	0	0	1	2	3	4	5	6
2	5	3	4	1	6	7	F	F	F	E	D	C	B	A	9

Format 41

PIN Block Format 41 is the Visa format for PIN change without using the current PIN. The method for constructing the PIN block is defined in section C.11.2 of reference 4, and can only be encrypted using a DES/3DES key.

The PIN Block is created using the new PIN and part of the card's unique DEA Key as follows:

- Construct a 16 hexadecimal digit block of data, by extracting the eight rightmost digits of the card application's Unique DEA Key A (UDK-A) and zero filling it on the left with eight hexadecimal zeros:

0	0	0	0	0	0	0	0								
---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--

8 Rightmost digits of card app's unique DEA key A

- Create a second 16 hexadecimal digit block of data as follows:

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F
---	---	---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----	---	---

Where:

- C is a fixed control field of binary value 0000 (X'0).
- N is the length of the new PIN and can be any binary value from 0100 to 1100 (X'4 to X'C).
- P is a digit of the new PIN and can be any binary value from 0000 to 1001 (X'0 to X'9).
- P/F is either a PIN digit or the binary value 1111 (X'F) filler depending on the length of the PIN.

The Thales HSM terminology for the UDK is DK-AC. This is the card-unique key that is derived from MK-AC, the Master Key for Application Cryptograms.

- F is a filler of binary value 1111 (X'F).

- Perform an exclusive-OR operation on the blocks of data created in steps 1 and 2 to create the final PIN block.

16.2.8 Format 42

PIN Block Format 42 is the Visa format for PIN change using the current (old) PIN. The method for constructing the PIN block is defined in section C.11.1 of reference 4, and can only be encrypted using a DES/3DES key.

The PIN Block is created using the old PIN, the new PIN and part of the card's unique DEA Key as follows:

- Construct a 16 hexadecimal digit block of data, by extracting the eight rightmost digits of the card application's Unique DEA Key A (UDK-A) and zero filling it on the left with eight hexadecimal zeros:

0	0	0	0	0	0	0	0								
---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--

8 Rightmost digits of card app's unique DEA key A

- Create a second 16 hexadecimal digit block of data as follows:

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F
---	---	---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----	---	---

Where:

C is a fixed control field of binary value 0000 (X'0).

N is the length of the new PIN and can be any binary value from 0100 to 1100 (X'4 to X'C).

P is a digit of the new PIN and can be any binary value from 0000 to 1001 (X'0 to X'9).

P/F is either a PIN digit or the binary value 1111 (X'F) filler depending on the length of the PIN.

F is a filler of binary value 1111 (X'F).

- Create a third 16 decimal digit block of data using the old PIN as follows:

P	P	P	P	P	P/0	P/0	P/0	P/0	P/0	P/0	P/0	P/0	0	0	0	0
---	---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----	---	---	---	---

Where:

P is a digit of the old PIN and can be any binary value from 0000 to 1001 (X'0 to X'9).

P/0 is either a PIN digit or the binary value 0000 (X'0) filler depending on the length of the PIN.

0 is a filler of binary value 0000 (X'0).

² The Thales HSM terminology for the UDK is *DK-AC. This is the card-unique key that is derived from *MK-AC, the Master Key for Application Cryptograms.

16.2.9 Format 47

PIN Block Format 47 is the ISO 9564-1 Format 3 PIN Block, and can only be encrypted using a DES/3DES key.

This PIN block is constructed by modulo 2 addition of two 64 bit fields: the plain text PIN field and the account number field.

The plain text PIN field is formatted as follows:

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F
---	---	---	---	---	---	-----	-----	-----	-----	-----	-----	-----	-----	---	---

Where:

C = Control field - Binary 0011 (X'3).

N = PIN length - 4 bit binary number with permissible values of 0100 (X'4) to 1100 (X'C).

P = PIN digit - 4 bit field with permissible values of 0000 (X'0) to 1001 (X'9).

P/F = PIN/Fill digit - Designation of these fields is determined by the PIN length field.

F = Fill digit - 4-bit field, with values from 1010 (X'A) to 1111 (X'F), where the fill-digit values are randomly or sequentially selected from this set of six possible values, such that it is highly unlikely that the identical configuration of

4. Decrypt the result of step 3 using the key, K.
5. Extract the PIN from the result of step 4.

17 Key Component Printing

17.1 Introduction

Payment processing is a multi-organizational activity, with the various organizations involved needing to exchange encryption keys. The working keys that the parties need to exchange are protected by encrypting them using a KEK; this term covers any Key Encrypting Key, including specialized KEKs such as Zone Master Keys (ZMKs) or Terminal Master Keys (TMKs). The problem is how the KEKs themselves are to be exchanged.

Further information on the TMD can be found in the payShield 10K Installation and User Guide and in the TMD User Guide.

Because all of these parties may be using different, incompatible IT infrastructures, KEKs are typically exchanged by splitting them into multiple components which are printed out, with each component being given to a different officer in the recipient organization; the KEK is re-formed at the recipient organization by the component holders coming together and entering the components into a key-forming application. No one individual acting alone has complete knowledge of the KEK or the ability to form it in the recipient's system.

This chapter describes how the payShield 10K could be used to generate and print KEK components at the issuing organization, and to re-form the KEK from components at the recipient organization.

Note: You will need to review the suggested methods described in this chapter with your organization's security team to ensure that these methods meet the your organizations' security and operational requirements.

17.2 Process Description

17.2.1 Overview

Key Encrypting Keys (KEKs), including specialized keys such as TMKs, ZMKs, and ZCMKs, are shared between organizations to allow them to securely exchange working keys by encrypting them using the KEKs.

A KEK that is to be shared between two organizations is generated by one of the parties and conveyed to the other in a secure manner.

Each user organization designs its own component distribution process to meet its operational and security requirements. One possible sequence of events, using a payShield 10K to provide a high level of security, is as follows:

- At the KEK issuing organization:
 - Set up Key Component Print Form
 - Generate and print the desired number of KEK components, and retain a copy of the components encrypted using the payShield 10K's Local Master Key (LMK)

Deliver each printed component to its designated officer in the recipient organization

Use the encrypted components to form the KEK using a payShield 10K at the issuing organization

Store the KEK in the issuing organization's key database, encrypted using the payShield 10K's LMK.

- At the recipient organization:
 - The component officers come together at a payShield 10K and individually enter their components
 - The recipient organization's payShield 10K forms the key from the entered components
 - The payShield 10K displays the KEK formed from the components, encrypted with the payShield 10K's LMK.
 - The LMK-encrypted KEK is entered into the recipient organization's key database. It can then be included in commands used to exchange working keys with the issuing organization.

Note: An alternative approach is to use the Thales Trusted Management Device (TMD). The TMD replaces the Thales

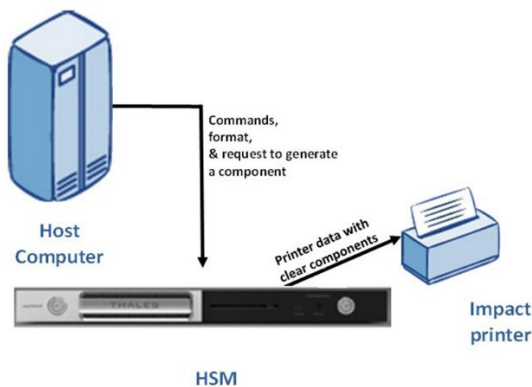
Key Management Device (KMD).

The possible process model outlined here can be modified to the needs of individual organizations. As an example of this, an approach taken by some users of Thales payment HSMs for setting up TMKs is to pre-print batches of components in secure envelopes, with an associated reference number; the component check value has been used for this by at least one organization, but in this case a process must be put in place to allow for the (unlikely) occurrence of a non-unique value. A supply of components is provided to each component holder. All the components are also held at the host system. The component holders visit the ATMs, and each takes any one of their stock of components and enters it into the ATM to form the ATM's copy of the TMK. Each component holder then calls the operations center and provides the reference number of the component they used. The Host computer can then retrieve the same components and form its copy of the TMK.

As mentioned above, each organization needs to design its own procedures to provide the level of security that the organization feels is appropriate. It is suggested that these procedures include the disposal of printed components after they have been used.

17.2.2 Directly Attached Printers

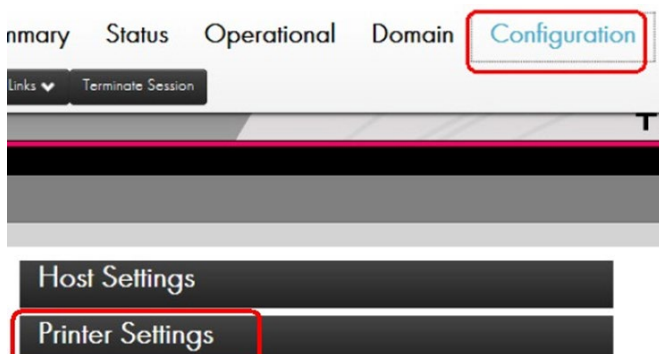
Multiple components need to be printed, but they must only be disclosed to the single user that they are to be addressed to. Using the payShield 10K, the components are created one at a time. It is suggested that these components are printed at an impact printer directly attached to the HSM, and onto specialized multi-part, tamper-evident mailers that prevent the component being accessed until the mailer is opened. Non-sensitive data, such as date and addressee, can be printed on the outside of the mailer.



The payShield 10K can print to:

- A parallel printer attached to a USB port on the payShield 10K via a USB-to-parallel cable available from Thales
- A serial printer attached to a USB port on the payShield 10K via a USB-to-serial cable available from Thales
- A USB printer attached to a USB port on the payShield 10K via a standard USB cable.

To choose which type of printer to use and to configure the printer, use the CP (Configure Printer) console command or payShield Manager Configuration / Printer Settings.



17.2.2.1 Non-impact Printers

It is unlikely that laser or ink-jet printers can be used in this way, because use of this type of printer to print secret information (such as PINs) requires specialized stationery with plasticized windows; it is unlikely that stationery with large enough windows for component printing will be readily available.

Any type of printer could be used to print components onto plain stationery. However, if plain stationery is to be used, security measures should be in place to ensure that the component can be seen by no more than one person and that any one person cannot see more than one component.

17.2.3 Operation without a directly attached Printer

If no directly attached printer is available, components can be displayed to the users. In this case, security processes must be in place to ensure that each component can be seen by no more than one person, that any one person cannot see more than one component, and that procedures are in place to protect the components (e.g., if they are transcribed onto paper).

17.2.4 Setting up Key Component Print Forms

A form definition needs to be loaded into the payShield 10K so that it can format the output sent to the printer.

The sections that follow describes how this can be used for key component printing.

The payShield 10K can hold a single form definition at a time. Formatting data is loaded onto the HSM as text strings. This data consists of:

- Formatting symbols to format the data
- Constants (text strings)
- Variable print field markers, which will be populated with data when the PIN mailers are to be printed. A key component print formatting string might look as follows:
`>L>003^0>033^1>L>003KEY COMPONENT PART 1: ^P>L>003 KEY COMPONENT PART 2: ^Q>L>003 KEY COMPONENT PART 3: ^R>L>L>003 KEY CHECK VALUE: ^T>L>L>003 DO NOT DISCLOSE THIS COMPONENT TO ANYONE ELSE>L>F`

In this example, the following print formatting symbols are used:

- >L carriage return + line feed (CR/LF)
- >nnn skip to output column nnn
- ^n - insert variable print field n: the actual value will be provided at print time
- >F Form Feed (FF)
- ^T - the key component check value.
- ^P, ^Q, and ^R have the following meanings:

Key length	^P	^Q	^R
Single	1 st 8 hex digits of component	2 nd 8 hex digits of component	N/A
Double	1 st 16 hex digits of component	2 nd 16 hex digits of component	N/A
Triple	1 st 16 hex digits of component	2 nd 16 hex digits of component	3 rd 16 hex digits of component

The full set of print formatting symbols is defined in Appendix D “Print Formatting Symbols”.

The above example would print the following key component form:

```
..(Field 0).....(Field 1)
```

```

..KEY.COMPONENT.PART.1: .xxxxxxxxxxxxxxxxxxxx
..KEY.COMPONENT.PART.2: .yyyyyyyyyyyyyyyyyy
..KEY.COMPONENT.PART.3: .zzzzzzzzzzzzzzzzzz

```

```

..KEY.CHECK.VALUE: .vvvvvv

```

Notes:

- "." is used in this example to indicate where a space would be printed
- "(Field 0)" etc. represents variable data (such as date and addressee) that the host computer application will provide at print time
- xxx...x, yyy...y, zzz...z represent parts of the key component, each representing 16 hexadecimal digits
- vvvvvv represents the component check value

The maximum length of a form definition is 299 symbols and characters of constant data

This form definition is loaded onto the HSM using Host commands. The required host commands are as follows - see the *payShield 10K Host Command Reference Manual* for full details:

payShield 10K Host Command	
ID	Command Description
PA	Load Formatting Data to HSM - to load the first part of the set of symbols and constant characters.
PC	Load Additional Formatting Data to HSM - to load any continuation of the symbols and constant characters. (PC is a continuation of PA, and must be preceded by a PA command.)

Generating and printing components

If using a directly-attached printer

The following single payShield 10K Host command performs these actions:

- Generates a random key component.
- Prints the key component at the printer directly attached to the HSM, using the print form currently installed on the HSM (using the PA and PC host commands).
- Encrypts the component using the HSM's LMK and returns it to the originating organization's host computer system.

payShield 10K Host Command	
ID	Command Description
A2	Generate and print a component

Typically 3 components will be generated and printed in this way, by running the A2 command once for each component.

Each component will be printed in its own mailer, which should be designed such that the component cannot be read from the outside of the mailer, and any attempt to access the component inside the mailer will be evident.

The HSM will provide 2 responses to the A2 command sent by the host:

payShield 10K Response to Host	
ID	Command Description
A3	Initial response, before printing. This returns the component encrypted under the HSM's LMK, and the component check value.
AZ	Second response, providing the outcome of the printing operation.

The issuer's host will receive and hold the encrypted component.

If no HSM-attached printer is available

The A2 Host command depends on the payShield 10K having a directly attached printer. Where such a printer is not

available, key components can be generated using the following Console command:

payShield 10K Console Command	
ID	Command Description
GC	Generate key component

This command generates both the clear component and the component encrypted under the payShield 10K's LMK and displays them to the user.

The output from this command must be manually written or printed onto the mailer using a separate system.

17.2.5 Security considerations

Whenever components are output in any way other than inside a secure, tamper-evident mailer, the originating organization should satisfy themselves that procedures are in place to ensure that not more than one person can see any component, and that any one person cannot see more than one component.

17.2.5.1 Avoiding use of Legacy Commands

The payShield 10K also provides the NE Host command. This might appear to be a more convenient method of creating components because a single command generates the key and prints multiple components.

However, the NE command is provided purely for backwards compatibility with older systems that were designed to make use of it. It is recommended that use of this command is avoided, and that it is disabled using the CONFIGCMDS Console command or in the payShield Manager Configuration / Configure Commands / Host dialogue box, because it generates components as split keys rather than full-length components which are XORed together.

17.2.5.2 Delivery of printed components

These mailers are delivered to the appropriate key management officers at the recipient organization, using secure delivery methods.

17.2.6 Forming the KEK at the issuer system

17.2.6.1 If components were printed at a directly attached printer

If a payShield 10K-attached printer has been used to print components, typically 3 components will have been generated using the HSM's A2 Host command and encrypted under the HSM's LMK. These encrypted components are held on the issuer's host system.

The issuer's host now forms the KEK from the encrypted components it is holding by sending the components to a payShield 10K in the following host command:

payShield 10K Host Command	
ID	Command Description
A4	Form a key from encrypted components

The payShield 10K will form the KEK, encrypt it using its LMK, and return the LMK-encrypted KEK to the host system. The host can now add this encrypted KEK to its key database. Whenever a cryptographic operation requires the KEK, the encrypted KEK will be included in the host command sent to the HSM and the HSM will decrypt the KEK and use it within its secure boundary.

17.2.6.2 If a directly attached printer was not used

Where no HSM-attached printer is available and components have been generated using the GC Console command,

then the KEK should be formed in the same way.

17.2.7 Forming the KEK at the recipient system

The key management officers at the recipient organization will each have received their own KEK component. Each one of them has access to only one component, and all of the component holders need to come together to allow the KEK to be formed from the components.

There are 3 ways that this can be done using the payShield 10K. In each case:

- the component holders come together
- each enters the component they hold
- the KEK is displayed, encrypted under the HSM's LMK
- the encrypted KEK is noted down and entered into the recipient host system's key database, using the appropriate key entry tool at the host.

The three payShield 10K mechanisms available for doing this are as follows.

17.2.7.1 Console

The following Console command allows the component holders to enter their components sequentially and then displays the encrypted key and its check value:

payShield 10K Console Command	
ID	Command Description
FK	Form key from components

This Console command can also form a key from encrypted components (e.g. if the encrypted components generated by the GC Console command option had been provided to the recipient).

17.2.7.2 payShield Manager

The ability to Install an LMK from RLMK Card Set can be found in payShield Manager at Operational / LMK Operations / Local master Keys.

17.2.7.3 payShield Trusted Management Device (TMD)

The standalone Thales Trusted Management Device (TMD) can be used to form an LMK-encrypted key from components.

Note: The TMD replaces the Thales Key Management Device (KMD).

18 Moving to PCI HSM Compliance

18.1 Introduction

The PCI SSC (Payment Card Industry Security Standards Council) publishes security standards relating to the issuing of credit/debit cards and the processing of their transactions. When these standards become mandated by the card schemes, organizations are audited against those standards that relate to their operations and equipment manufacturers must ensure that their products comply with the standards which are relevant to them.

The PCI HSM standard relates specifically to HSMs, such as the payShield 10K. Compliance with the PCI HSM standard is sought by many users and is required for compliance with certain other PCI standards (e.g., Point-to-Point Encryption and mPOS; Card Production).

The payShield 10K hardware has been certified as being compliant with this standard. In order to achieve this certification, a number of changes were made to the payShield 10K software, some of which apply to user operation of the device and some of which are relevant to application developers. This chapter focusses on the changes which may impact on developers:

- PIN Blocks
- Updating Key Type 002 in host commands
- Migrating keys from Key Type 002
- Diebold Table re-encryption

Note that where changes have been made to the payShield 10K to enable it to comply with the requirements of the PCI HSM standard, the user can decide whether to use these changes through the payShield 10K's security settings. If the changes to the software are not used then the payShield 10K can run with full compatibility with legacy units, but it is then not compliant with the requirements of PCI HSM.

18.2 PIN Blocks

18.2.1 Using PIN Block formats in a Compliant Manner

PCI HSM compliance requires adherence to the recommendations of ISO 9564 / ANSI X9.8 in terms of allowed PIN Block Format translations and usage.

The allowed translations and usage are a sub-set of what was previously available on the payShield 10K. Therefore users could already be compliant with ISO 9564 / ANSI X9.8 with older HSMs, simply by using only the allowed functionality. However, for PCI HSM compliance the payShield 10K must enforce the limitations.

For users not already conforming to ISO 9564/ANSI X9.8, the payShield 10K allows the user to continue operating in this fashion until such time as the user wishes to become PCI HSM compliant. This is done by manipulating the security setting Restrict PIN block usage for PCI compliance:

- if this is not set the user can continue to using PIN Block Format translations or usage outside of those allowed by ISO 9564/ANSI X9.8 and PCI HSM
- if it is set then only the permitted functionality is available.

18.2.2 Permitted PIN Block Format Translations

When the security setting Restrict PIN block usage for PCI compliance is set, the permissible PIN Block format translations are:

Translation to:												
Thales PIN	01	02	03	04	05	34	35	41	42	46	47	48

Block Format		ISO Format	0	-	-	-	1	2	-	-	-	-	3	4
Translation from:	01	0	<input checked="" type="checkbox"/>										<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	02	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	03	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	04	-	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	05	1	<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	34	2	<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	35	-	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	41	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	42	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	46	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	47	3	<input checked="" type="checkbox"/>										<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	48	4	<input checked="" type="checkbox"/>										<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Note: The rows shaded light blue indicate that these translations are not currently used in any payShield 10K Host commands.

With this security setting, users must ensure that their applications are not requesting non-permissible PIN Block Format translations when sending any of the following Host commands to the payShield 10K:

Command	
CA	Translate PIN from TPK to ZPK
CC	Translate PIN from one ZPK to another
G0	Translate a PIN from BDK to BDK or ZPK Encryption (3DES DUKPT)
G6	Translate PIN from one ZPK to another with SEED (LIC020)
G8	Translate PIN from TPK to ZPK with SEED (LIC020)

18.2.3 PIN Block Format for Values Derived from PIN & PAN

When the security setting Restrict PIN block usage for PCI compliance is set, payShield 10K enforces the ISO 9564 / ANSI X9.8 and PCI HSM requirement that only Thales PIN Block formats 01 (ISO format 0), 47 (ISO format 3), and 48 (ISO format 4) may be used when calculating values (e.g. offsets, PVV) from the PIN and PAN.

The Host commands affected by this setting are:

Command	
BK	Generate IBM PIN Offset (customer-selected PIN)
CE	Generate Diebold PIN Offset
CU	Verify & Generate a Visa PVV
DE	Generate IBM PIN Offset
DG	Generate Visa PVV
DU	Verify & Generate IBM PIN Offset
FW	Generate Visa PVV (customer-selected PIN)

Where the security setting is not set, this restriction is not enforced and users can continue to use other PIN Block formats to calculate offsets, PVVs, etc., until such time as the user wants to implement an ISO 9564/ANSI X9.8 and PCI HSM compliant environment.

18.2.4 PIN Translations to/from LMK Encryption

When the security setting Restrict PIN block usage for PCI compliance is set, payShield 10K will not permit PIN translations to/from any 3DES LMK (since all types of 3DES LMK use proprietary PIN block formats when encrypting PINs). However, an AES Key Block LMK uses ISO 9564-1 format 4 when encrypting PINs, and so PIN block translations to/from encryption under an AES Key Block LMK are always permitted.

18.2.5 Summary of required Actions

- If not already the case, change applications to use only permitted PIN Block format translations.
- If not already the case, change applications to use only ISO format 0, 3, and 4 PIN Blocks to calculate values from PIN and PAN.

18.3 Updating Key Type 002 in Host Commands

Note: This change does not affect users who have implemented Key Block LMKs. Moving to Key Block LMKs is the better approach from a strategic viewpoint: users who wish to continue working with Variant LMKs should implement the guidelines provided below.

PCI HSM certification requires additional key separation to be applied to the HSM. This means that, for users of Variant LMKs, various data and keys in key type 002 (i.e. encrypted with LMK Pair 14-15, variant 0) must be moved to a different key type.

Users can plan when they want to move the keys from the legacy key type (002) to the new PCI HSM compliant key types by making use of the security setting Enforce key type 002 separation for PCI HSM compliance:

If this is not set, then the affected keys remain in key type 002: this provides interoperability with legacy systems but means that the payShield 10K is not PCI HSM compliant.

If the setting is set, the affected keys use the new key types as required for PCI HSM compliance.

In order to operate with the new key types, Host applications issuing commands to the HSM which specify the key type must be modified to specify the new key type.

The Host commands affected by this are:

Host Command	
A0	Generate a Key
A2	Generate and Print a Component
A4	Form a Key from Encrypted Components
A6	Import a Key
A8	Export a Key
B0	Translate Key Scheme
BK	Gen IBM PIN Offset (customer selected PIN)
BU	Generate a Key Check Value
BW	Translate Keys from Old to New LMK
CU	Verify & Generate a Visa PVV (of a customer selected PIN)
DU	Verify & Generate an IBM PIN Offset (of customer selected new PIN)
FW	Generate a Visa PIN Verification Value (of a customer selected PIN)
GI	Import Key under an RSA Public Key
GK	Export Key under an RSA Public Key

Host Command**NE Gen and Print Key as Split Components**

18.3.1 Example

If it is required to use the A0 Host command to generate a TPK where:

- the key is to be encrypted under the default Variant LMK
- the key does not need to be encrypted under a ZMK or TMK
- the key does not need to be exported in TR-31 format
- no command message trailer is required.

A legacy application might send the following Host command:

Field	Value
Message Header	1234
Command Code	A0
Mode	0
Key Type	002
Key Scheme	U

If the security setting Enforce key type 002 separation for PCI HSM compliance is set, the Host command would be modified to the following:

Field	Value
Message Header	1234
Command Code	A0
Mode	0
Key Type	70D
Key Scheme	U

18.3.2 Interoperability with older HSMs

Where users have a mixed estate of PCI HSM compliant and non-compliant payShield 10K and HSM 8000 HSMs, all of their HSMs must be able to use the same key types.

HSM 8000 users should upgrade to v3.3a or later. These versions of HSM 8000 software include the same capability of changing key types as described above. Some Host commands will also be disabled. (Note that the HSM 8000 is not PCI HSM compliant, even with version 3.3a of software installed.)

Users of payShield 10K with software versions earlier than v1.2a should upgrade their HSMs to v1.2a or later. This will provide interoperability but will not necessarily make the payShield 10K PCI HSM compliant.

It is not possible for a PCI HSM compliant payShield 10K HSM to inter-operate with older Thales RG6000 and RG7000 payment HSMs.

18.3.3 Summary of Required Actions

- Amend key type values specified in Host commands to take account of the new key types.

18.4 Migrating keys from Key Type 002

Note: This change does not affect users who have implemented Key Block LMKs. Moving to Key Block LMKs is the better approach from a strategic viewpoint: users who wish to continue working with Variant LMKs can migrate their keys following the guidelines provided below.

As described in the section Updating Key Types 002 in Host commands above, a number of keys will have different key types if the security setting Enforce key type 002 separation for PCI HSM compliance has the PCI HSM compliant value. This means that the users' relevant operational keys will need to be migrated from key type 002 to their new key type.

It is suggested that this can be done as part of the regular LMK refreshes that users should be performing as part of their security procedures. LMK refresh is performed by a Host application making use of the payShield 10K's BW Host command. The BW Host command includes an option for the migration of keys from key type 002 at the same time as refreshing the LMK. (The BW command can continue to be used just for LMK refresh, and can also be used to perform the key migration without LMK refresh.)

The processes below indicate approaches that might be taken to achieve migration, but users will need to design processes that suit their own environment. The starting point in these processes is that transaction processing is using the existing key database, and the security setting Enforce key type 002 separation for PCI HSM compliance has not been set on any HSM.

18.4.1 Migrating keys WITHOUT a change of LMK

18.4.1.1 Creating a new key database for the new Key Types

- Create a Host process which will be able to take each affected key from a key database, Use the BW Host command (example below) to migrate the key from key type 002 to the new key type, and replace the original key in the key database with the re-encrypted key returned by the BX response.
- Clone each live key database to a new key database. At this stage the new key database contains the un-migrated keys and is not being used for processing - which continues using the live key database.
- For each set of LMKs in use:
 - Set up one (or more) payShield 10K which has the active LMKs loaded with the security setting Enforce key type 002 separation for PCI HSM compliance not set.
 - Use this HSM with the Host process to migrate all the affected keys in the new key database. The HSM can continue to be used for transaction processing using the old unmigrated keys at the same time, if required.

18.4.1.2 Switching the HSMs to the new Key Types

- The HSMs need to be switched over to use the new key types at a convenient point. During this switch-over, the HSMs will be unavailable for processing for a period of up to a few minutes. Depending on the nature of the applications and the systems design, it may be appropriate to do this switchover for:
 - All HSMs using a particular key database. (This might be all HSMs, or all HSMs for an application, or all HSMs for an LMK - dependent on the organization of the key databases). This makes it easier for the Host application to switch between old and new key databases, but could involve an interruption of service if a Disaster Recovery site is not available to pick up processing using the old unmigrated keys during this switchover.
 - A single HSM or a sub-set of the HSMs using a key database, leaving other HSMs available to continue providing a service using the old unmigrated keys. This avoids a complete interruption of service, but requires the Host application to determine which key database to use for each HSM - e.g. by using the NO command.

- For the HSMs to be switched over:
 - Suspend processing on the HSMs, or use a failover mechanism (such as an SRM) to re-direct commands to an alternative HSM.
 - Put the HSMs into secure state.
 - Set the security setting Enforce key type 002 separation for PCI HSM compliance.
 - Resume processing on the HSMs, but using the new key database and the Host application modified to use the new key types.
- When the process is complete
 - archive the old key database

18.4.2 Migrating keys WITH a change of LMK

- Create a Host process which will take each key from a key database, use the BW Host command (example below) to change the LMK for each key and migrate relevant keys from key type 002 to the new key type, and write the re-encrypted key returned by the BX response back to the key database.
- Clone each live key database to a new key database. At this stage the new key database holds the old un-migrated keys and is not being used for processing - processing continues using the live key database.
- For each set of LMKs in use:
 - Set up one (or more) payShield 10K which has the old LMK (i.e., the one currently being used to process transactions) loaded as its live LMK (e.g. by using the LK Console command) and the new LMK stored in its Key Change Storage (e.g. by using the LN Console command). The security setting Enforce key type 002 separation for PCI HSM compliance must not be set. It can still be used for transaction processing if required.
 - Use this HSM with the Host process to migrate all the keys in the new key database by overwriting the old keys with keys newly encrypted using the new LMK and new key types.

18.4.2.1 Switching the HSMs to the new Key Types

- The HSMs need to be switched over to use the new key types at a convenient point. During this switch-over, the HSMs will be unavailable for processing for a period of up to a few minutes. Depending on the nature of the applications and the systems design, it may be appropriate to do this switchover for:
 - All HSMs using a key database. (This might be all HSMs, or all HSMs for an application, or all HSMs for an LMK - dependent on the organization of the key databases). This makes it easier for the Host application to switch between old and new key databases, but could involve an interruption of service if a Disaster Recovery is not available.
 - A single HSM or a sub-set of the HSMs using a key database, leaving other HSMs available to continue providing a service. This avoids a complete interruption of service, but requires the Host application to determine which key database to use for each HSM - e.g., by using the NO command.
- For the HSMs to be switched over:
 - Suspend processing on the HSMs, or use a failover mechanism (such as an SRM) to re-direct commands to an alternative HSM
 - Put the HSMs into secure state
 - On each HSM set the security setting Enforce key type 002 separation for PCI HSM compliance
 - On each HSM delete the old LMK (e.g., using the DM Console command) and load the new LMK (e.g. using the LK Console command)
 - Resume processing on the group of HSMs, but using the new key database and the modified Host application using the new key types

Note: The BW command does not allow keys to be migrated back to key type 002. Therefore if there is any possibility that the security setting Enforce key type 002 separation for PCI HSM compliance is to be unset then the old key

database must be retained.

18.4.3 Using Disaster Recovery (DR) sites

Where a DR site is available that replicates the capabilities of the live site, it may be appropriate to implement the changes on the DR site where they can be trialed and tested. This will increase the level of confidence prior to going live.

18.4.4 Examples of using the BW Host Command

If a key is being migrated without change of LMK, where the key:

- Is a TPK
- Is a Triple-length DES key
- Is using the default LMK
- Has no message trailer

Field	Value
Message Header	1234
Command Code	BW
Key Type code	E2
Key length	2
Key	U12345678...ABCDEF
Delimiter	;
Key Type	70D

Note that when the Key Type Code is E2, no LMK must be installed in the Key Change Storage.

If the same key is being migrated but this time with a change of LMK:

Field	Value
Message Header	1234
Command Code	BW
Key Type code	F2
Key length	2
Key	U12345678...ABCDEF
Delimiter	;
Key Type	70D

Note that when Key Type Code is F2, both the old LMK (that is being moved away from) and the new LMK (that is being moved to) must be installed on the payShield 10K.

18.4.4.1 Thales Professional Services

Thales professional services can work with users in planning and implementing their key migration.

18.4.4.2 Summary of Actions required

- Create a Host process to migrate keys from key type 002 (and, if appropriate, change LMK) using the BW Host command, and generate a new key database.

- At the appropriate time, set the HSM security setting Enforce key type 002 separation for PCI HSM compliance and start using the new key database.

18.5 Diebold Table re-encryption

This change will only affect users who are employing the Diebold Table.

The Diebold Table in legacy systems is encrypted using LMK pair 14-15 variant 0 (key type 002); because of the PCI HSM key separation requirements this will be encrypted under LMK pair 36-37 variant 6 (key type 60D) when the HSM is required to be PCI HSM compliant. The encryption key type used is controlled by the security setting Enforce key type 002 separation for PCI HSM compliance.

Immediately after setting the security setting Enforce key type 002 separation for PCI HSM compliance on a payShield 10K, the following actions must be taken:

- Erase the existing encrypted version of the Diebold table using the LA Host command to overwrite it with hexadecimal F digits. The Index Address must be the same as that used when the encrypted Diebold table was set up; 32 blocks of 16 hexadecimal F digits should be written. (This step is unnecessary if the re-encrypted Diebold table is to be written to the same location in user storage as the existing encrypted table.)
- Use the R Console command to load the new Diebold Table. This requires access to the plaintext version of the Diebold Table
- Use the LC Host command to verify the Diebold Table
- Repeat the above steps for each LMK which has an encrypted Diebold Table

Note that if for any reason the security setting Enforce key type 002 separation for PCI HSM compliance is unset the Diebold table will have to be re-entered again.

18.5.1 Summary of Actions required

- Re-enter the Diebold table whenever the value of the security setting Enforce key type 002 separation for PCI HSM compliance is changed.

Appendix A - Key Scheme Table

Whenever a key is entered into the HSM, it must be prefixed by a Key Scheme Tag which allows the HSM to interpret the key correctly. The following values are supported:

Key Scheme Tag	Notes
None/Z	Encryption of a single-length DES key using the ANSI X9.17 method. Used for encrypting keys for local use (under a Variant LMK) or for importing/exporting keys (e.g. under a ZMK). The use of this scheme requires some security settings to be changed.
U	Encryption of a double-length DES key using the Thales Variant method. Used for encrypting keys for local use (under a Variant LMK) or for importing/exporting keys (e.g. under a ZMK). The use of this scheme for import/export requires some security settings to be changed.
T	Encryption of a triple-length DES key using the Thales Variant method. Used for encrypting keys for local use (under a Variant LMK) or for importing/exporting keys (e.g. under a ZMK). The use of this scheme for import/export requires some security settings to be changed.
X	Encryption of a double-length DES key using the ANSI X9.17 method. Used for encrypting keys for local use (under a Variant LMK) or for importing/exporting keys (e.g. under a ZMK). The use of this scheme requires some security settings to be changed.
Y	Encryption of a triple-length DES key using the ANSI X9.17 method. Used for encrypting keys for local use (under a Variant LMK) or for importing/exporting keys (e.g. under a ZMK). The use of this scheme requires some security settings to be changed.
V	Encryption of double/triple-length DES keys using Verifone/GISKE methods. Only used for exporting keys (e.g. under a ZMK).
R	Encryption of single/double/triple-length DES keys using the X9 TR-31 Key Block methods. Only used for importing/exporting keys (e.g. under a ZMK).
S	Encryption of all DES, AES, RSA, ECC & HMAC keys using Thales Key Block methods. Used for encrypting keys for local use (under a Key Block LMK) or for importing/ exporting keys (e.g. under a ZMK).

Appendix B - Reduced Character Sets

The HSM optionally (through the security setting “Enable ZEK/TEK encryption”) restricts the character set that can be processed by the data encryption/ decryption commands. The three options are described in the table below:

"ZEK/TEK Encryption" Setting	Byte Value (hex)
ASCII data	0x20 – 0x7E
Binary data	0x00 – 0xFF
None	-

Appendix C - Thales Key Block / TR 31 Key Usage Conversion

When exporting from Thales Key Block format to TR-31 format, the following table is used to convert the key blocks' Key Usage field:

Internal (Thales) Key Block	Key Usage	Exported (TR-31) Key Block
'41'	Base Derivation Key Type 2 (BDK-2)	'B0'
'42'	Base Derivation Key Type 3 (BDK-3)	'B0'
'43'	Base Derivation Key Type 4 (BDK-4)	'B0'
'11'	Card verification (American Express CSC)	'C0'
'12'	Card verification (Mastercard CVC)	'C0'
'13'	Card verification (Visa CVV)	'C0'
'21'	Data encryption using a DEK (DEK)	'D0'
'22'	Data encryption using a ZEK (ZEK)	'D0'
'31'	Visa Cash Master Load Key (KML)	'E6'
'32'	Dynamic CVV Master Key (MK-CVC3)	'E6'
'51'	Terminal key encryption (TMK)	'K0'
'52'	Zone key encryption (ZMK)	'K0'
'71'	Terminal PIN encryption (TPK)	'P0'
'72'	Zone PIN encryption (ZPK)	'P0'
'73'	Terminal Key Register (TKR)	'P0'

Appendix E - Print Formatting Symbols

Printer Formatting

The table shows the EBCDIC and ASCII codes for the printer formatting when printing from the payShield 10K:

Symbol	EBCDIC	ASCII	Meaning
>L	6E D3	3E 4C	Line feed, carriage return.
>V	6E E5	3E 56	Vertical tab.
>H	6E C8	3E 48	Horizontal tab.
>F	6E C6	3E 46	Form feed.
>nnn	6E Fn Fn Fn	3E 3n 3n 3n	Skip column nnn in relation to left margin, where nnn is a 3-digit decimal number.
^M	5F D6	5E 49	For a key document, print third clear component.
^P	5F D7	5E 50	For a PIN mailer, print clear PIN for mailer 1. For a key document, print clear component.
^Q	5F D8	5E 51	For a PIN mailer, print clear PIN for mailer 2. For a key document, print clear component or encrypted TMK (only one-up printing allowed for key documents).
^R	5F D9	5E 52	Print reference number for PIN mailer 1.
^S	5F E2	5E 53	Print reference number for PIN mailer 2.
^T	5F E3	5E 54	Print last 6 account number digits on PIN mailer 1.
^U	5F E4	5E 55	Print last 6 account number digits on PIN mailer 2.
<L><hh hh hh ..>	6A <L> <hh hh hh ..>	7C <L> <hh hh hh ..>	Send binary data to printer for example printer control string. character followed by the length of the string in bytes <L> 0 - F then the expanded hex string <hh hh hh ..>. NOTE: in the columns to the left, the "<" and ">" characters represent field boundaries, and are NOT part of the data to be sent to the printer. For example (using ASCII), a string of 4 bytes which would be represented in the 3 rd column as 7C<4><1B283358> would actually be sent as 41B283358 .
^0	5F F0	5E 30	Insert Print Field 0. Insert Print Field 1.
^1	5F F1	5E 31	Insert Print Field 2.
^2	5F F2	5E 32	.

Symbol	EBCDIC	ASCII	Meaning
.	.	.	.
.	.	.	.
.	.	.	Insert Print Field 15. Insert Print Field 16. Insert Print Field 17. Insert Print Field 18.
^F	5F C6	5E 46	.
^^10	5F 5F F1 F0	5E 5E 31 30	.
^^11	5F 5F F1 F1	5E 5E 31 31	Insert Print Field 31.
^^12	5F 5F F1 F2	5E 5E 31 32	
.	.	.	
.	.	.	
^^1F	5F 5F F1 C6	5E 5E 31 46	

Printing PINs in Word Format

Two print formatting symbols are provided for printing PINs in word format. For example:

ONE TWO THREE FOUR.

English is used as the default setting. The symbols can be used in addition to the symbols for printing PINs in numeric format (e.g., 1234).

Symbol	EBCDIC	ASCII	Meaning
^V	5F E3	5E 56	Print the clear PIN in word format for mailer 1. Can be used for either a one-up or a two-up PIN mailer. e.g., ONE TWO THREE FOUR
^W	5F E6	5E 57	Print the clear PIN in word format for mailer 2. Can be used only for a two-up PIN mailer. e.g., ONE TWO THREE FOUR

Printing PINs in Columns

Four print formatting symbols are provided for printing PINs (both words and numerics) in columns. For example:

```
1    ONE
2    TWO
3    THREE
4    FOUR
```

For the following definition of print symbols an n is used to indicate which digit of a PIN is to be printed. The relationship between PIN digits and n is as follows:

PIN Digit	1	2	3	4	5	6	7	8	9	10	11	12
'n'	1	2	3	4	5	6	7	8	9	A	B	C

Symbol	EBCDIC	ASCII	Meaning
^Pn	5F D7 F1-F9 or C1-C3	5E 50 31-39 or 41-43	Print the clear PIN digit n in number format for mailer 1. Can be used for either a one-up or a two-up PIN mailer. e.g. 1
^Qn	5F D8 F1-F9 or C1-C3	5E 51 31-39 or 41-43	Print the clear PIN digit n in number format for mailer 2. Can be used only for two-up PIN mailer. e.g. 1
^Vn	5F E3 F1-F9 or C1-C3	5E 56 31-39 or 41-43	Print the clear PIN digit n in word format for mailer 1. Can be used for either a one-up or a two-up PIN mailer. e.g. ONE
^Wn	5F E6 F1-F9 or C1-C3	5E 57 31-39 or 41-43	Print the clear PIN digit n in word format for mailer 2. Can be used only for two-up PIN mailer. e.g. ONE

Appendix F - Example laser printer formatting control codes

The following is an example of how an application can pass format control commands for a laser printer to a payShield 10K as print formatting symbols in PA and PC Host commands, as discussed in Chapter 3 “*PIN Printing and Solicitation*”. This information, based on certain types of HP printer, should be viewed simply as an example to help developers generate an appropriate symbol sequence for their own printer.

The example symbol sequence is as follows, and is explained in the subsequent table.

```
>L
>L|91B2661393030763048|91B2666313030793358|51B28313055|B1B28733170333676323554|D1
B266136373930763133343048
>005^P
>L|A1B287330703130763354|91B2661393030763048
>005^^00
>L
>005^^01
>L
>005^^02
>L
>005^^03
>L
>005^^04
>L
>005^^05
>L
>005^^06
>L
>F
```

Symbol	Interpretation	Notes
>L	Line feed, carriage return	
>L	Line feed, carriage return	
91B2661393030763048	Send the following 9 bytes of binary data to the printer: "&a900v0H"	Position the print head 900 decipoints vertically, 0 decipoints horizontally.
91B2666313030793358	Send the following 9 bytes of binary data to the printer: "&f100y3X"	Select overlay
51B28313055	Send the following 5 bytes of binary data to the printer: ".(10U"	Select font ID=10U (PC-8)
B1B28733170333676323554	Send the following 11 bytes of binary data to the printer: ".(s1p36v25T"	Select font Type Face
D1B266136373930763133343048	Send the following 13 bytes of binary data to the printer: "&a6790v1340H"	Position the print head 6790 decipoints vertically, 1340 decipoints horizontally.

Symbol	Interpretation	Notes
>005^P	Skip to col. 5 and print the PIN	
>L	Line feed, carriage return	
A1B287330703130763354	Send the following 10 bytes of binary data to the printer: ".(s0p10v3T"	Select font Type Face
91B2661393030763048	Send the following 9 bytes of binary data to the printer: ".&a900v0H"	Position the print head 900 decipoints vertically, 0 decipoints horizontally.
>005^^00	Skip to col. 5 and print field 0	
>L	Line feed, carriage return	
>005^^01	Skip to col. 5 and print field 1	
>L	Line feed, carriage return	
>005^^02	Skip to col. 5 and print field 2	
>L	Line feed, carriage return	
>005^^03	Skip to col. 5 and print field 3	
>L	Line feed, carriage return	
>005^^04	Skip to col. 5 and print field 4	
>L	Line feed, carriage return	
>005^^05	Skip to col. 5 and print field 5	
>L	Line feed, carriage return	
>005^^06	Skip to col. 5 and print field 6	
>L	Line feed, carriage return	
>F	Form feed	

Appendix G - SNMP MIB

This Appendix provides the MIB (Management Information Base) for the data that the payShield 10K can provide over SNMP. The MIB is also included on the payShield 10K software media.

```
-- Thales eSecurity, Inc.
-- Version: 1.2
-- Date:    February 7, 2020
-- Changes:
```

```
THALES-ESECURITY-PAYSHIELD-MIB DEFINITIONS ::= BEGIN
```

```
IMPORTS
```

```
    NOTIFICATION-TYPE, OBJECT-TYPE, MODULE-IDENTITY,
    enterprises, IpAddress, Integer32, Gauge32
    FROM SNMPv2-SMI
    MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
    FROM SNMPv2-CONF
    MacAddress, DisplayString, DateAndTime, TruthValue
    FROM SNMPv2-TC;
```

```
payShieldMIB MODULE-IDENTITY
```

```
    LAST-UPDATED "202002070810Z"
```

```
    ORGANIZATION
```

```
        "Thales eSecurity"
```

```
    CONTACT-INFO
```

```
        "954 888 6200"
```

```
    DESCRIPTION
```

```
        "Added new objects payShieldVersionSoftwareFirmwareVersion and
payShieldVersionSoftwareDeploymentVersion.
```

```
        Deprecated payShieldVersionSoftwareBuildNumber."
```

```
    REVISION "201902260810Z"
```

```
    DESCRIPTION
```

```
        "Provides information for following components for Thales
eSecurity's payShield devices (10K or later):
```

```
        Enabled host commands, Fraud, Health, Host connection, Licensing,
LMK, Logs, Management,
```

```
        PCI Security Settings, Printer, State, Utilization, Version,
Auxiliary Ethernet."
```

```
    REVISION "201902260810Z"
```

```
    DESCRIPTION
```

```
        "Deprecated payShieldVersionSoftwareHSMCoreAPIVersion and
payShieldRestartAlarm.
```

```
        Updated description of payShieldUtilHostCmdVolume."
```

```
    REVISION "201811120810Z"
```

```
    DESCRIPTION
```

```
        "Initial revision"
```

```
::= { raProductMibs 10000 }
```

```
-- This MIB describes the structure of Thales eSecurity's Management
```

```
-- Information Base. The variables are structured based on the RFC
-- for the Structure of Management Information (SMI), RFC1155, and
-- the format of the definitions is as per RFC1212, the Concise MIB
-- Definitions RFC
-- The object identifiers below have been assigned by the Thales eSecurity
-- Management Group. All new Thales eSecurity MIB's should be inserted into
this
-- MIB.

thalesEsecurity      OBJECT IDENTIFIER ::= { enterprises 4096 }
thalesEsecurityDevs  OBJECT IDENTIFIER ::= { thalesEsecurity 1 }
payshield            OBJECT IDENTIFIER ::= { thalesEsecurityDevs 10000 }
thalesEsecurityMibs  OBJECT IDENTIFIER ::= { thalesEsecurity 2 }
raProductMibs        OBJECT IDENTIFIER ::= { thalesEsecurityMibs 2 }

-- Utilization statistics provide information on the device's current host
command
-- processing load.

payShieldUtil        OBJECT IDENTIFIER ::= { payShieldMIB 1 }

-- State information for the payShield device.

payShieldState        OBJECT IDENTIFIER ::= { payShieldMIB 2 }
payShieldStateTamper   OBJECT IDENTIFIER ::= { payShieldState 2 }
payShieldStateFan      OBJECT IDENTIFIER ::= { payShieldState 3 }
payShieldStatePSU      OBJECT IDENTIFIER ::= { payShieldState 4 }

-- LMK information for the payShield device.

payShieldLmk          OBJECT IDENTIFIER ::= { payShieldMIB 3 }

-- Communication ports information for the payShield device.

payShieldComms         OBJECT IDENTIFIER ::= { payShieldMIB 4 }
payShieldCommsMgmt     OBJECT IDENTIFIER ::= { payShieldComms 1 }
payShieldCommsHost     OBJECT IDENTIFIER ::= { payShieldComms 2 }
payShieldCommsHostPort OBJECT IDENTIFIER ::= { payShieldCommsHost 4 }

-- System information for the payShield device.

payShieldSystem        OBJECT IDENTIFIER ::= { payShieldMIB 5 }

-- Provides information on whether or not the Fraud Detection limits have been
exceeded.

payShieldFraud         OBJECT IDENTIFIER ::= { payShieldMIB 6 }

-- Contains information to identify the software running on the payShield.

payShieldVersion       OBJECT IDENTIFIER ::= { payShieldMIB 7 }

-- Provides information pertaining to the licensing of the payShield.

payShieldLicensing     OBJECT IDENTIFIER ::= { payShieldMIB 8 }

-- Provides the number of and list of host commands that are enabled on this
payShield.
```

```
payShieldEnabledHostCommands OBJECT IDENTIFIER ::= { payShieldMIB 9 }

-- Info on the system logs.

payShieldLogs OBJECT IDENTIFIER ::= { payShieldMIB 10 }

-- Info on the Error log

payShieldLogsErrorlog OBJECT IDENTIFIER ::= { payShieldLogs 1 }

-- Info about the system audit log.

payShieldLogsAuditlog OBJECT IDENTIFIER ::= { payShieldLogs 2 }

-- Data pertaining to the diagnostic self tests and the HealthCheckCounts.

payShieldHealth OBJECT IDENTIFIER ::= { payShieldMIB 11 }

-- Provides the number of diagnostic tests last performed and the results of
each test.

payShieldHealthDiagSelfTest OBJECT IDENTIFIER ::= { payShieldHealth 1 }

-- Health check counts for the payShield. These cover reboots, tampers and
suspicious
-- pin verification failure patterns

payShieldHealthCheckCounts OBJECT IDENTIFIER ::= { payShieldHealth 2 }

-- Contains information about the configuration of Host interfaces. For enabled
interfaces
-- details about the physical and protocol configuration are provided.

payShieldHostConnection OBJECT IDENTIFIER ::= { payShieldMIB 12 }

-- Settings and states of the Host ethernet interfaces.
-- Only available when payShieldHostConnectionEthernetEnabled is TRUE.

payShieldHostConnectionEthernet OBJECT IDENTIFIER ::= { payShieldHostConnection
3 }

payShieldHostConnectionFICON OBJECT IDENTIFIER ::= { payShieldHostConnection
4 }

-- The detailed configuration info of the management ethernet interface.

payShieldManagement OBJECT IDENTIFIER ::= { payShieldMIB 13 }

-- Configuration information about the management ethernet interface.

payShieldManagementEthernet OBJECT IDENTIFIER ::= { payShieldManagement 1 }

-- Data concerning the printer configuration of the unit. This includes the
universally
-- used settings for all printers and the printer report on the actual printer
hooked up.
-- This is the identical report the console QP command and the payShield
Manager give.
```



```
payShieldPrinter          OBJECT IDENTIFIER ::= { payShieldMIB 14 }

-- payShield settings.

payShieldSettings         OBJECT IDENTIFIER ::= { payShieldMIB 15 }

-- Configuration information about the auxiliary ethernet interface.

payShieldAuxiliaryEthernet OBJECT IDENTIFIER ::= { payShieldMIB 16 }

-- Notifications

notifications            OBJECT IDENTIFIER ::= { thalesEsecurity 999 }
payShieldNotifications    OBJECT IDENTIFIER ::= { notifications 2 }

payShieldAlarmObjects     OBJECT IDENTIFIER ::= { payShieldNotifications 1 }
payShieldTraps            OBJECT IDENTIFIER ::= { payShieldNotifications 2 }

-- Information pertaining to the operation of the payShield fans.

payShieldAlarmFans        OBJECT IDENTIFIER ::= { payShieldAlarmObjects 1 }

-- Information on the status of the payShield power supply units

payShieldAlarmPSU         OBJECT IDENTIFIER ::= { payShieldAlarmObjects 2 }

-- Information on the packets with bad port received on the payShield host
ports

payShieldAlarmBadPortData OBJECT IDENTIFIER ::= { payShieldAlarmObjects 3 }

-- Information on error log

payShieldAlarmErrorLog    OBJECT IDENTIFIER ::= { payShieldAlarmObjects 4 }

-- Information on the status of the payShield battery

payShieldAlarmBattery     OBJECT IDENTIFIER ::= { payShieldAlarmObjects 5 }

-- Information on diagnostic failures

payShieldAlarmDiagnostic  OBJECT IDENTIFIER ::= { payShieldAlarmObjects 6 }

-- Information on fraud attempts

payShieldAlarmFraud       OBJECT IDENTIFIER ::= { payShieldAlarmObjects 7 }

-- Information on SW/Settings erase

payShieldAlarmErase       OBJECT IDENTIFIER ::= { payShieldAlarmObjects 8 }

-- Information on modified settings

payShieldAlarmSettingsModified OBJECT IDENTIFIER ::= { payShieldAlarmObjects 9
}

-- Information on the new payShield state

payShieldAlarmStateChange OBJECT IDENTIFIER ::= { payShieldAlarmObjects 10 }
```

-- Information on tamper

payShieldAlarmTamper OBJECT IDENTIFIER ::= { payShieldAlarmObjects 11 }

payShieldUtilLoad OBJECT-TYPE

 SYNTAX Gauge32 (0..100)

 MAX-ACCESS read-only

 STATUS current

 DESCRIPTION

 "Displays the instantaneous load for all host commands processed by the payShield in the previous time period. The load is represented as a percentage of the total [licensed] processing capacity of the device."

 ::= { payShieldUtil 1 }

-- The octet string is formatted as follows (where CC is command code, e.g. A0):

--
-- <measurement period in seconds>,<num commands in report><LINE FEED>
-- "CC",<total number commands>;"CC",<total number commands>; ... etc...

payShieldUtilHostCmdVolume OBJECT-TYPE

 SYNTAX OCTET STRING (SIZE(0..484))

 MAX-ACCESS read-only

 STATUS current

 DESCRIPTION

 "Displays the measurement period in seconds and the number of commands in the report followed by a list of all host commands and the number of times each of those host commands that have been processed within the previous measurement period. Note that if a large variety of commands are processed, then this list may be incomplete and will contain the commands that consumed the highest proportion of the device's capacity."

 ::= { payShieldUtil 2 }

payShieldUtilEnabled OBJECT-TYPE

 SYNTAX TruthValue

 MAX-ACCESS read-only

 STATUS current

 DESCRIPTION

 "True if Utilization turned on"

 ::= { payShieldUtil 3 }

payShieldStateDevice OBJECT-TYPE

 SYNTAX INTEGER {
 stateUnavailable (1),
 stateOnline (2),
 stateOffline (3),
 stateSecure (4)
 }

 MAX-ACCESS read-only

 STATUS current

 DESCRIPTION

 "Indicates the current state of the payShield."

 ::= { payShieldState 1 }

```
payShieldStateTamperState OBJECT-TYPE
    SYNTAX      INTEGER {
        stateUnknown (1),
        stateOK (2),
        stateTampered (3)
    }
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
        "Indicates whether the device is currently in a tamper state."
    ::= { payShieldStateTamper 1 }

payShieldStateTamperDate OBJECT-TYPE
    SYNTAX      DateAndTime
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
        "The Date and Time that the last tamper event occurred. A NULL value
        indicates that the unit has not registered any tamper events."
    ::= { payShieldStateTamper 2 }

payShieldStateTamperCause OBJECT-TYPE
    SYNTAX      INTEGER {
        causeUnavailable (1),
        causeTemperatureOutOfRange (2),
        causeBatteryLow (3),
        causeEraseButtonPressed (4),
        causeSensorProcessorWatchdog (5),
        causeSensorProcessorRestart (6),
        causeVoltageOutOfRange (7),
        causeMotionDetected (8),
        causeCaseTampered (9),
        causePowerLoss (10)
    }
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
        "The cause of the last tamper event"
    ::= { payShieldStateTamper 3 }

payShieldStateFanTable OBJECT-TYPE
    SYNTAX SEQUENCE OF PayShieldStateFanEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
        "A table that returns the status of fans."
    ::= { payShieldStateFan 1 }

payShieldStateFanEntry OBJECT-TYPE
    SYNTAX      PayShieldStateFanEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
        "Row in the fan status table"
    INDEX { payShieldStateFanIndex }
    ::= { payShieldStateFanTable 1 }

PayShieldStateFanEntry ::= SEQUENCE {
```

```

    payShieldStateFanIndex
        Integer32,
    payShieldStateFanSerialNum
        DisplayString,
    payShieldStateFanState
        INTEGER
}

payShieldStateFanIndex OBJECT-TYPE
    SYNTAX      Integer32 (1..2)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The index into the fan status table"
    ::= { payShieldStateFanEntry 1 }

payShieldStateFanSerialNum OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Displays the serial number of this fan unit."
    ::= { payShieldStateFanEntry 2 }

payShieldStateFanState OBJECT-TYPE
    SYNTAX      INTEGER {
        stateOK (1),
        stateFailure (2),
        stateNotDetected (3)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the current state of this fan."
    ::= { payShieldStateFanEntry 3 }

payShieldStatePSUTable OBJECT-TYPE
    SYNTAX SEQUENCE OF PayShieldStatePSUEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
        "A table that returns the status of power supply units."
    ::= { payShieldStatePSU 1 }

payShieldStatePSUEntry OBJECT-TYPE
    SYNTAX      PayShieldStatePSUEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Row in the PSU status table"
    INDEX { payShieldStatePSUIndex }
    ::= { payShieldStatePSUTable 1 }

PayShieldStatePSUEntry ::= SEQUENCE {
    payShieldStatePSUIndex
        Integer32,
    payShieldStatePSUSerialNum
        DisplayString,
    payShieldStatePSUState

```

```
        INTEGER
    }

payShieldStatePSUIndex OBJECT-TYPE
    SYNTAX      Integer32 (1..2)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The index into the PSU status table"
    ::= { payShieldStatePSUEntry 1 }

payShieldStatePSUSerialNum OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Displays the serial number of this power supply unit."
    ::= { payShieldStatePSUEntry 2 }

payShieldStatePSUState OBJECT-TYPE
    SYNTAX      INTEGER {
        stateOK (1),
        stateFailure (2),
        stateNotDetected (3)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the current state of this power supply unit."
    ::= { payShieldStatePSUEntry 3 }

payShieldStateBattery OBJECT-TYPE
    SYNTAX      INTEGER {
        stateOK (1),
        stateWarning (2),
        stateFailure (3)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the current state of the battery."
    ::= { payShieldState 5 }

payShieldLmkNumLoaded OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of LMKs currently loaded into the device."
    ::= { payShieldLmk 1 }

payShieldLmkNumTestLmksLoaded OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the number of 'test' LMKs loaded (as opposed to
production).
        When the payShield is operating in a live environment, no test
```

LMKs should be loaded and this field should be returned as 0."
 ::= { payShieldLmk 2 }

payShieldLmkNumOldLmksLoaded OBJECT-TYPE

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Indicates the number of old LMKs currently loaded in the payShield.
 Old LMKs should be stored in the device only as long as they are

needed for

translation purposes. Once all LMK-protected data has been placed

under the

control of the new LMK set, then the old LMK set should be deleted."

::= { payShieldLmk 3 }

payShieldLmkStatusTable OBJECT-TYPE

SYNTAX SEQUENCE OF PayShieldLmkStatusEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A table that returns the status of all possible LMK sets."

::= { payShieldLmk 4 }

payShieldLmkStatusEntry OBJECT-TYPE

SYNTAX PayShieldLmkStatusEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"Row in the LMK Status table"

INDEX { payShieldLmkStatusIndex }

::= { payShieldLmkStatusTable 1 }

PayShieldLmkStatusEntry ::= SEQUENCE {

payShieldLmkStatusIndex

Integer32,

payShieldLmkStatusLoaded

TruthValue,

payShieldLmkStatusAuth

TruthValue,

payShieldLmkStatusNumAuthActivities

Integer32,

payShieldLmkStatusScheme

INTEGER,

payShieldLmkStatusAlgorithm

INTEGER,

payShieldLmkStatusLiveTest

INTEGER,

payShieldLmkStatusComments

OCTET STRING,

payShieldLmkStatusCheckDigits

DisplayString

}

payShieldLmkStatusIndex OBJECT-TYPE

SYNTAX Integer32 (1..20)

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

```

        "The index into the LMK Status table"
    ::= { payShieldLmkStatusEntry 1 }

payShieldLmkStatusLoaded OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates whether or not an LMK set is currently loaded at this
index
        in the table. Returns TRUE if loaded, else FALSE."
    ::= { payShieldLmkStatusEntry 2 }

payShieldLmkStatusAuth OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Authorized state of this LMK set. A value of TRUE indicates that
the
        LMK is in an authorized state; FALSE indicates that it is not
authorized."
    ::= { payShieldLmkStatusEntry 3 }

payShieldLmkStatusNumAuthActivities OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the number of authorized activities if the payShield
is set to Multi-Auth mode. If not in multi-auth mode, returns 0."
    ::= { payShieldLmkStatusEntry 4 }

payShieldLmkStatusScheme OBJECT-TYPE
    SYNTAX      INTEGER {
        lmkSchemeUnknown (1),
        lmkSchemeVariant (2),
        lmkSchemeKeyblock (3),
        lmkSchemeAES (4)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the scheme of the loaded LMK set (variant, keyblock, or
AES)."
    ::= { payShieldLmkStatusEntry 5 }

payShieldLmkStatusAlgorithm OBJECT-TYPE
    SYNTAX      INTEGER {
        lmkAlgorithmUnknown (1),
        lmkAlgorithm3DES2Key (2),
        lmkAlgorithm3DES3Key (3),
        lmkAlgorithmAES256 (4)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the algorithm used by the LMK set for encryption."
    ::= { payShieldLmkStatusEntry 6 }

```

```
payShieldLmkStatusLiveTest  OBJECT-TYPE
    SYNTAX      INTEGER {
        lmkStatusUnknown (1),
        lmkStatusLive (2),
        lmkStatusTest (3)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the status of the currently loaded LMK set.
        This will be either 'Live' or 'Test'. Note that test LMKs should not
        be loaded into a payShield operating in a live customer
environment."
    ::= { payShieldLmkStatusEntry 7 }

payShieldLmkStatusComments  OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE(0..41))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Textual description of the LMK set loaded. A string of length 0
        indicates that no description is stored."
    ::= { payShieldLmkStatusEntry 8 }

payShieldLmkStatusCheckDigits  OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates the check digits for an LMK."
    ::= { payShieldLmkStatusEntry 9 }

payShieldCommsMgmtConsoleState  OBJECT-TYPE
    SYNTAX      INTEGER {
        consoleUp (1),
        consoleDown (2),
        consoleDisabledByGui (3),
        consoleUnavailable (4)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates whether the process to service Console Management
requests
        is currently running. payShieldCommsMgmtConsoleStateUp (1) indicates
that
        it is; all the other states indicate that the console is currently
inactive."
    ::= { payShieldCommsMgmt 1 }

payShieldCommsMgmtGuiState  OBJECT-TYPE
    SYNTAX      INTEGER {
        guiUp (1),
        guiDown (2),
        guiUnavailable (3)
    }
    MAX-ACCESS  read-only
    STATUS      current
```


DESCRIPTION

"Indicates whether the process to service GUI requests is currently running.
payShieldCommsMgmtGuiStateUp (1) indicates that it is; all the other states

indicate that the GUI is currently inactive."
::= { payShieldCommsMgmt 2 }

payShieldCommsHostTCPServer OBJECT-TYPE

SYNTAX INTEGER {
serverUp (1),
serverDown (2),
serverNotEnabled (3)
}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Indicates whether the process to service host commands is currently running over the Ethernet host port using the TCP protocol.

payShieldCommsHostTCPServerUp (1) indicates that it is;
all the other states indicate that TCP is currently inactive."

::= { payShieldCommsHost 1 }

payShieldCommsHostUDPServer OBJECT-TYPE

SYNTAX INTEGER {
serverUp (1),
serverDown (2),
serverNotEnabled (3)
}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Indicates whether the process to service host commands is currently running over the Ethernet host port using the UDP protocol.

payShieldCommsHostUDPServerUp (1) indicates that it is; all the

other

states indicate that UDP is currently inactive."

::= { payShieldCommsHost 2 }

payShieldCommsHostFICONServer OBJECT-TYPE

SYNTAX INTEGER {
serverUp (1),
serverDown (2),
serverNotEnabled (3)
}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Indicates whether the process to service host commands is currently running over the FICON host port. payShieldCommsHostFICONServerUp

(1)

indicates that it is; all the other states indicate that FICON is currently inactive. "

::= { payShieldCommsHost 3 }

payShieldCommsHostPortEthernet1 OBJECT-TYPE

SYNTAX INTEGER {
portUp (1),
portDown (2),

```
        portUnavailable (3),
        portNotConfigured (4)
    }
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Indicates if the first host ethernet port is up and running."
    ::= { payShieldCommsHostPort 1 }

payShieldCommsHostPortEthernet2 OBJECT-TYPE
    SYNTAX INTEGER {
        portUp (1),
        portDown (2),
        portUnavailable (3),
        portNotConfigured (4)
    }
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Indicates if the second host ethernet port is up and running."
    ::= { payShieldCommsHostPort 2 }

payShieldCommsHostPortFICON OBJECT-TYPE
    SYNTAX INTEGER {
        portUp (1),
        portDown (2),
        portUnavailable (3),
        portNotConfigured (4)
    }
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Indicates if the FICON port (if present) is up and running."
    ::= { payShieldCommsHostPort 3 }

payShieldSystemDateAndTime OBJECT-TYPE
    SYNTAX DateAndTime
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The date and time as reported by the system's Real Time Clock."
    ::= { payShieldSystem 1 }

payShieldSystemSerialNum OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(0..32))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Displays the serial number of the payShield system."
    ::= { payShieldSystem 2 }

payShieldSystemModel OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Displays the model/series information of the payShield system."
    ::= { payShieldSystem 3 }
```

```

payShieldFraudPinVerifyLimitsExceeded  OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Returns TRUE if fraud detection is turned on, and either the
        allowable PIN verifications/minute, or PIN verifications/hour, have
        been exceeded."
    ::= { payShieldFraud 1 }

payShieldFraudPinAttackLimitsExceeded  OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Returns TRUE if fraud detection is turned on, AND the total number
        of PIN attacks have exceeded the allowed count."
    ::= { payShieldFraud 2 }

payShieldVersionSoftwareBaseRelease  OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Base release of the software running on the payShield."
    ::= { payShieldVersion 1 }

payShieldVersionSoftwareRevision  OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Revision of the software running on the payShield."
    ::= { payShieldVersion 2 }

payShieldVersionSoftwareBuildNumber  OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      deprecated
    DESCRIPTION
        "***** THIS OBJECT IS DEPRECATED *****

        Build number of the software running on the payShield."
    ::= { payShieldVersion 3 }

payShieldVersionSoftwareHSMCoreAPIVersion  OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      deprecated
    DESCRIPTION
        "***** THIS OBJECT IS DEPRECATED *****

        HSM core API version of the software running on the payShield."
    ::= { payShieldVersion 4 }

payShieldVersionSoftwareCPLDVersion  OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current

```

DESCRIPTION

"Version of CPLD running on the payShield."

::= { payShieldVersion 5 }

payShieldVersionSoftwareSPVersion OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Version of Sensor Processor software running on the payShield."

::= { payShieldVersion 6 }

payShieldVersionSoftwareSPBootVersion OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Version of Sensor Processor boot on the payShield."

::= { payShieldVersion 7 }

payShieldVersionSoftwareBootstrapVersion OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Version of bootstrap on the payShield."

::= { payShieldVersion 8 }

payShieldVersionSoftwareFirmwareVersion OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Semantic version of firmware on the payShield."

::= { payShieldVersion 9 }

payShieldVersionSoftwareDeploymentVersion OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Semantic deployment version of firmware on the payShield."

::= { payShieldVersion 10 }

payShieldLicensingPerformanceModel OBJECT-TYPE

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The maximum calls per second this payShield unit is licensed for."

::= { payShieldLicensing 1 }

payShieldLicensingPackage OBJECT-TYPE

SYNTAX OCTET STRING (SIZE(0..2048))

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The license package loaded on this payShield unit."

::= { payShieldLicensing 2 }

```
payShieldLicensingOptionalLicenseCount  OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of optional licenses this payShield unit has installed."
    ::= { payShieldLicensing 3 }

payShieldLicensingOptionalLicensesList  OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE(0..2048))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The optional licenses this payShield unit is licensed for separated
by ';'."
    ::= { payShieldLicensing 4 }

payShieldLicensingCryptoAlgorithmCount  OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of Cryptographic Algorithms enabled by the payShield's
licensing."
    ::= { payShieldLicensing 5 }

payShieldLicensingCryptoAlgorithmList  OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "List of licensed Cryptographic Algorithms. These will be
descriptive strings
        for each algorithm terminated by a semi-colon ';'.
        The number of licensed algorithms in the list will be
        equal to payShieldLicensingCryptoAlgorithmCount."
    ::= { payShieldLicensing 6 }

payShieldEnabledHostCommandsCount  OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of host commands enabled on this payShield."
    ::= { payShieldEnabledHostCommands 1 }

payShieldEnabledHostCommandsList  OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE(0..2048))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Entire list of enabled host commands separated by a semi-colon
';'."
    ::= { payShieldEnabledHostCommands 2 }

payShieldLogsErrorlogTotalCount  OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
```

```
STATUS      current
DESCRIPTION
    "Total number of entries in the error log."
::= { payShieldLogsErrorlog 1 }
```

```
payShieldLogsAuditlogTotalCount  OBJECT-TYPE
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Total number of entries in the audit log."
::= { payShieldLogsAuditlog 1 }
```

```
payShieldHealthDiagSelfTestTimeOfDay  OBJECT-TYPE
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of minutes after midnight the diagnostic tests begins
at."
::= { payShieldHealthDiagSelfTest 1 }
```

```
payShieldHealthDiagSelfTestOK  OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "True unless one or more of the tests in the last self test failed."
::= { payShieldHealthDiagSelfTest 2 }
```

```
payShieldHealthDiagSelfTestCount  OBJECT-TYPE
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of self tests run last test cycle on this payShield."
::= { payShieldHealthDiagSelfTest 3 }
```

```
payShieldHealthDiagSelfTestList  OBJECT-TYPE
SYNTAX      OCTET STRING (SIZE(0..2048))
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The complete results of the last run self test.
    Syntax of string Testname:testresult;
    Test/results pairs are separated by a colon ':'
    They are delimited by a semi-colon ; .
    Results are the string 'passed' or 'failed'.
    The number of self tests in the list will be
    equal to payShieldHealthDiagSelfTestCount."
::= { payShieldHealthDiagSelfTest 4 }
```

```
payShieldHealthHealthCheckEnabled  OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Whether the payShield is presently collecting health check data."
::= { payShieldHealthCheckCounts 1 }
```

payShieldHealthCheckCountsStartTime OBJECT-TYPE
SYNTAX DateAndTime
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Date/Time of last health stats reset."
 ::= { payShieldHealthCheckCounts 2 }

payShieldHealthCheckCountsEndTime OBJECT-TYPE
SYNTAX DateAndTime
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Date/time of when stats collecting was last disabled.
If health stats are not disabled, the Date/Time when this field was
read."
 ::= { payShieldHealthCheckCounts 3 }

payShieldHealthCheckCountsRebootCount OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Number of times the payShield rebooted since the last reset of
health counters."
 ::= { payShieldHealthCheckCounts 4 }

payShieldHealthCheckCountsTamperCount OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of tampers detected since the last reset of health
counters."
 ::= { payShieldHealthCheckCounts 5 }

payShieldHealthCheckCountsPinFailuresMinuteLimit OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of minutes in which the maximum pin verify failures was
exceeded
since the last reset of health counters."
 ::= { payShieldHealthCheckCounts 6 }

payShieldHealthCheckCountsPinFailuresHourLimit OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of hours in which the maximum allowable pin failures per
hour was
exceeded since the last reset of health counters."
 ::= { payShieldHealthCheckCounts 7 }

payShieldHealthCheckCountsPinAttackLimitExceeded OBJECT-TYPE
SYNTAX Integer32

```

MAX-ACCESS read-only
STATUS      current
DESCRIPTION
    "The number of times the pin attack limit was exceeded since the
last reset of
    health counters."
 ::= { payShieldHealthCheckCounts 8 }

payShieldHostConnectionEthernetEnabled OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Whether the payShield is configured to provide ethernet host
interfaces."
 ::= { payShieldHostConnection 1 }

payShieldHostConnectionFICONEnabled OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Whether or not the payShield is configured to provide a FICON host
interface."
 ::= { payShieldHostConnection 2 }

payShieldHostConnectionEthernetIfCount OBJECT-TYPE
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of physical ethernet interfaces enabled to process host
commands.
    This will be 1 or 2."
 ::= { payShieldHostConnectionEthernet 1 }

payShieldHostConnectionEthernetSSLEnabled OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "If true TLS/SSL has been enabled."
 ::= { payShieldHostConnectionEthernet 2 }

payShieldHostConnectionEthernetACLEnabled OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Whether Access Control Lists are enabled."
 ::= { payShieldHostConnectionEthernet 3 }

payShieldHostConnectionEthernetUDPEnabled OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Whether UDP is enabled."
 ::= { payShieldHostConnectionEthernet 4 }

```


payShieldHostConnectionEthernetTCPEnabled OBJECT-TYPE

```
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Whether TCP is enabled."
 ::= { payShieldHostConnectionEthernet 5 }
```

payShieldHostConnectionEthernetMaxTCPConnections OBJECT-TYPE

```
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The maximum number of TCP sessions allowed per interface.
    (0-64)"
 ::= { payShieldHostConnectionEthernet 6 }
```

payShieldHostConnectionEthernetTCPKeepalive OBJECT-TYPE

```
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "TCP keep alive timeout in minutes.
    (0-120)"
 ::= { payShieldHostConnectionEthernet 7 }
```

payShieldHostConnectionEthernetWellKnownPortTCP OBJECT-TYPE

```
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The well known TCP port.
    (0-65535)"
 ::= { payShieldHostConnectionEthernet 8 }
```

payShieldHostConnectionEthernetWellKnownPortTLS OBJECT-TYPE

```
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The well known TLS port.
    (0-65535)"
 ::= { payShieldHostConnectionEthernet 9 }
```

payShieldHostConnectionEthernetTable OBJECT-TYPE

```
SYNTAX SEQUENCE OF PayShieldHostConnectionEthernetEntry
MAX-ACCESS not-accessible
STATUS      current
DESCRIPTION
    "The interface specific ethernet configuration/status information.
    For interfaces that are disabled or fields that are not relevant,
    all values will be set to a NULL value."
 ::= { payShieldHostConnectionEthernet 10 }
```

payShieldHostConnectionEthernetEntry OBJECT-TYPE

```
SYNTAX      PayShieldHostConnectionEthernetEntry
MAX-ACCESS  not-accessible
STATUS      current
```

DESCRIPTION

"An entry containing information applicable to the host ethernet connection"

```
INDEX { payShieldHostConnectionEthernetIndex }
::= { payShieldHostConnectionEthernetTable 1 }
```

```
PayShieldHostConnectionEthernetEntry ::= SEQUENCE {
    payShieldHostConnectionEthernetIndex
        Integer32,
    payShieldHostConnectionEthernetConfigured
        TruthValue,
    payShieldHostConnectionEthernetInterfaceNumber
        Integer32,
    payShieldHostConnectionEthernetConfigMethod
        INTEGER,
    payShieldHostConnectionEthernetHostName
        DisplayString,
    payShieldHostConnectionEthernetIpAddress
        IpAddress,
    payShieldHostConnectionEthernetSubnetMask
        IpAddress,
    payShieldHostConnectionEthernetGateway
        IpAddress,
    payShieldHostConnectionEthernetMacAddress
        MacAddress,
    payShieldHostConnectionEthernetPortSpeed
        DisplayString,
    payShieldHostConnectionNumberOfConnectionsUsed
        Integer32
}
```

payShieldHostConnectionEthernetIndex OBJECT-TYPE

SYNTAX Integer32 (1..2)

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A unique value (> 0) for each host ethernet connection interface"

```
::= { payShieldHostConnectionEthernetEntry 1 }
```

payShieldHostConnectionEthernetConfigured OBJECT-TYPE

SYNTAX TruthValue

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Tells whether a row of the table is configured. For a 1 interface system it is possible that the configured interface could be either the first or second one."

```
::= { payShieldHostConnectionEthernetEntry 2 }
```

payShieldHostConnectionEthernetInterfaceNumber OBJECT-TYPE

SYNTAX Integer32 (1..2)

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This binds a row to a physical host ethernet interface.

1 corresponds to Ethernet interface HostPortEthernet1

2 corresponds to Ethernet interface HostPortEthernet2"

```
::= { payShieldHostConnectionEthernetEntry 3 }
```

payShieldHostConnectionEthernetConfigMethod OBJECT-TYPE

```
SYNTAX      INTEGER {  
    ipMethodUnknown (0),  
    ipMethodStatic (1),  
    ipMethodDHCP (2)  
}
```

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"DHCP and static IP configuration are the possibilities. When statically configured, the addresses reported are from the payShield's configuration for this interface.

When DHCP, the addresses obtained via DHCP are reported."

```
::= { payShieldHostConnectionEthernetEntry 4 }
```

payShieldHostConnectionEthernetHostName OBJECT-TYPE

```
SYNTAX      DisplayString
```

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Network Name is present for a DHCP configured port only.

If the ethernet port is configured statically set to 'NotApplicable'.

If the ethernet port is not configured set to 'Interface not enabled'."

```
::= { payShieldHostConnectionEthernetEntry 5 }
```

payShieldHostConnectionEthernetIpAddress OBJECT-TYPE

```
SYNTAX      IpAddress
```

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"IP address for this interface.

If this interface is not enabled returns 0.0.0.0"

```
::= { payShieldHostConnectionEthernetEntry 6 }
```

payShieldHostConnectionEthernetSubnetMask OBJECT-TYPE

```
SYNTAX      IpAddress
```

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Subnet mask for this interface.

If this interface is not enabled returns 0.0.0.0"

```
::= { payShieldHostConnectionEthernetEntry 7 }
```

payShieldHostConnectionEthernetGateway OBJECT-TYPE

```
SYNTAX      IpAddress
```

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Gateway IP address used by this interface.

If this interface is not enabled returns 0.0.0.0"

```
::= { payShieldHostConnectionEthernetEntry 8 }
```

payShieldHostConnectionEthernetMacAddress OBJECT-TYPE

```
SYNTAX      MacAddress
```

MAX-ACCESS read-only

```

STATUS      current
DESCRIPTION
    "MAC address for this ethernet interface.
    If this interface is not enabled returns 00:00:00:00:00:00"
::= { payShieldHostConnectionEthernetEntry 9 }

```

```

payShieldHostConnectionEthernetPortSpeed  OBJECT-TYPE
SYNTAX      DisplayString
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Describes the speed/duplex this port is configured at.
    If the port is configured for 'ethernet autoselect' returns:
    'ethernet autoselect(speed/duplex )'
    If the ethernet interface is not configured returns:
    'Interface not enabled'"
::= { payShieldHostConnectionEthernetEntry 10 }

```

```

payShieldHostConnectionNumberOfConnectionsUsed  OBJECT-TYPE
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The Number of TCP connections in use for this physical interface.
    range (0-payShieldHostConnectionMaxTCPConnections)"
::= { payShieldHostConnectionEthernetEntry 11 }

```

```

payShieldHostConnectionFICONControlUnitImage  OBJECT-TYPE
SYNTAX      DisplayString
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "This is the control unit address defined in the mainframe I/O
definition
    (CUADD on CNTLUNIT statement).
    (0-255)"
::= { payShieldHostConnectionFICON 1 }

```

```

payShieldHostConnectionFICONControlUnitAddress  OBJECT-TYPE
SYNTAX      DisplayString
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The starting unit address for this control unit.
    16 devices are enumerated from this point (UNITADD on CNTLUNIT
statement).
    (0-255)"
::= { payShieldHostConnectionFICON 2 }

```

```

payShieldHostConnectionFICONMIH  OBJECT-TYPE
SYNTAX      Integer32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "This specifies the missing interrupt handler value to be used in
the read device
    characteristics CCW for the mainframe. It is in minutes.
    If set to 0, the mainframe setting is used
    (0-60)"

```

```

::= { payShieldHostConnectionFICON 3 }

payShieldManagementEthernetEnabled OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Whether management ethernet interface is enabled."
    ::= { payShieldManagementEthernet 1 }

payShieldManagementEthernetConfigMethod OBJECT-TYPE
    SYNTAX      INTEGER {
        ipMethodUnknown (0),
        ipMethodStatic (1),
        ipMethodDHCP (2)
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The method the management ethernet interface uses to obtain its
        ethernet address.
        Static - uses local stored information
        DHCP - obtains settings from the network"
    ::= { payShieldManagementEthernet 2 }

payShieldManagementEthernetNetworkName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The name the payShield assigns to this interface if in DHCP mode.
        If the ethernet port is configured statically, set to
        'DoesNotApply'."
    ::= { payShieldManagementEthernet 3 }

payShieldManagementEthernetIpAddress OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The management ethernet interface's IP address.
        If payShieldManagementEthernetConfigMethod is set to DHCP, it will
        be set by the network.
        If payShieldManagementEthernetConfigMethod is set to static, it will
        be set to the payShield's
        configured value.
        A value of 0.0.0.0 means it is not successfully fetched."
    ::= { payShieldManagementEthernet 4 }

payShieldManagementEthernetSubnetMask OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The management ethernet interface's IP subnet mask.
        If payShieldManagementEthernetConfigMethod is set to DHCP, it will
        be set by the network.
        If payShieldManagementEthernetConfigMethod is set to static, it will
        be set to the payShield's

```

configured value.

A value of 0.0.0.0 means it was not successfully fetched."

```
::= { payShieldManagementEthernet 5 }
```

payShieldManagementEthernetGateway OBJECT-TYPE

SYNTAX IPAddress

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The management ethernet interface's IP gateway address.

If payShieldManagementEthernetConfigMethod is set to DHCP, it will be set by the network.

If payShieldManagementEthernetConfigMethod is set to Static it will be set to the payShield's configured value.

A value of 0.0.0.0 means it was not successfully fetched."

```
::= { payShieldManagementEthernet 6 }
```

payShieldManagementEthernetMacAddress OBJECT-TYPE

SYNTAX MacAddress

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The management ethernet interface's MAC address.

Will be set to 00:00:00:00:00:00 if not successfully fetched."

```
::= { payShieldManagementEthernet 7 }
```

payShieldManagementEthernetPortSpeed OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"String that describes the speed configured and the actual connection properties.

Form is config (actual)

eg : ethernet autoselect (100baseTX full-duplex)"

```
::= { payShieldManagementEthernet 8 }
```

payShieldPrinterLFCRReversed OBJECT-TYPE

SYNTAX TruthValue

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Whether LF and CR are reversed."

```
::= { payShieldPrinter 1 }
```

payShieldPrinterTimeout OBJECT-TYPE

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Printer timeout value in milliseconds.

(1000, 86400000)"

```
::= { payShieldPrinter 2 }
```

payShieldPrinterDelay OBJECT-TYPE

SYNTAX Integer32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Printer delay in milliseconds
(0-7200000)"

::= { payShieldPrinter 3 }

payShieldPrinterReport OBJECT-TYPE

SYNTAX OCTET STRING (SIZE(0..2000))

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The report of the dynamic state of printers in the system as produced for QP and the payShield Manager. It will consist of printable characters."

::= { payShieldPrinter 4 }

payShieldSettingsPCISCompliant OBJECT-TYPE

SYNTAX TruthValue

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"True if all security settings are PCI compliant."

::= { payShieldSettings 1 }

payShieldAuxiliaryEthernetEnabled OBJECT-TYPE

SYNTAX TruthValue

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Whether auxiliary ethernet interface is enabled."

::= { payShieldAuxiliaryEthernet 1 }

payShieldAuxiliaryEthernetConfigMethod OBJECT-TYPE

SYNTAX INTEGER {
ipMethodUnknown (0),
ipMethodStatic (1),
ipMethodDHCP (2)
}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The method the auxiliary ethernet interface uses to obtain its ethernet address.

Static - uses local stored information

DHCP - obtains settings from the network"

::= { payShieldAuxiliaryEthernet 2 }

payShieldAuxiliaryEthernetNetworkName OBJECT-TYPE

SYNTAX DisplayString

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The name the payShield assigns to this interface if in DHCP mode.

If the ethernet port is configured statically, set to

'DoesNotApply'."

::= { payShieldAuxiliaryEthernet 3 }

payShieldAuxiliaryEthernetIpAddress OBJECT-TYPE

SYNTAX IpAddress

MAX-ACCESS read-only

```

STATUS      current
DESCRIPTION
    "The auxiliary ethernet interface's IP address.
    If payShieldAuxiliaryEthernetConfigMethod is set to DHCP, it will be
set by the network.
    If payShieldAuxiliaryEthernetConfigMethod is set to static, it will
be set to the payShield's
    configured value.
    A value of 0.0.0.0 means it is not successfully fetched."
    ::= { payShieldAuxiliaryEthernet 4 }

payShieldAuxiliaryEthernetSubnetMask  OBJECT-TYPE
SYNTAX      IPAddress
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The auxiliary ethernet interface's IP subnet mask.
    If payShieldAuxiliaryEthernetConfigMethod is set to DHCP, it will be
set by the network.
    If payShieldAuxiliaryEthernetConfigMethod is set to static, it will
be set to the payShield's
    configured value.
    A value of 0.0.0.0 means it was not successfully fetched."
    ::= { payShieldAuxiliaryEthernet 5 }

payShieldAuxiliaryEthernetGateway  OBJECT-TYPE
SYNTAX      IPAddress
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The auxiliary ethernet interface's IP gateway address.
    If payShieldManagementEthernetConfigMethod is set to DHCP, it will
be set by the network.
    If payShieldManagementEthernetConfigMethod is set to static, it will
be set to the payShield's
    configured value.
    A value of 0.0.0.0 means it was not successfully fetched."
    ::= { payShieldAuxiliaryEthernet 6 }

payShieldAuxiliaryEthernetMacAddress  OBJECT-TYPE
SYNTAX      MacAddress
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The auxiliary ethernet interface's MAC address.
    Will be set to 00:00:00:00:00:00 if not successfully fetched."
    ::= { payShieldAuxiliaryEthernet 7 }

payShieldAuxiliaryEthernetPortSpeed  OBJECT-TYPE
SYNTAX      DisplayString
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "String that describes the speed configured and the actual
connection properties.
    Form is config (actual)
        eg: ethernet autoselect (100baseTX full-duplex)"
    ::= { payShieldAuxiliaryEthernet 8 }

```



```
payShieldFanNumber OBJECT-TYPE
    SYNTAX      INTEGER {
        fan1 (1),
        fan2 (2)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Identifies fan1 or fan2."
    ::= { payShieldAlarmFans 1 }

payShieldFanState OBJECT-TYPE
    SYNTAX      INTEGER {
        stateOK (1),
        stateFailure (2),
        stateNotDetected (3)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Provides the state of the fan."
    ::= { payShieldAlarmFans 2 }

payShieldPSUNumber OBJECT-TYPE
    SYNTAX      INTEGER {
        psu1 (1),
        psu2 (2)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Identifies the power supply unit 1 or 2."
    ::= { payShieldAlarmPSU 1 }

payShieldPSUState OBJECT-TYPE
    SYNTAX      INTEGER {
        stateOK (1),
        stateFailure (2),
        stateNotDetected (3)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Provides the state of the PSU."
    ::= { payShieldAlarmPSU 2 }

payShieldBadDataPhysicalPort OBJECT-TYPE
    SYNTAX      INTEGER {
        portHost1 (1),
        portHost2 (2),
        portFICON (3)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "The physical port the bad data was detected on."
    ::= { payShieldAlarmBadPortData 1 }

payShieldBadDataProtocol OBJECT-TYPE
```

```
SYNTAX      INTEGER {
    protocolTCP (1),
    protocolUDP (2),
    protocolFICON (3)
}
MAX-ACCESS accessible-for-notify
STATUS      current
DESCRIPTION
    "The method that bad data detected was transmitted."
 ::= { payShieldAlarmBadPortData 2 }
```

-- Text of last error sent to Error Log

```
payShieldErrorLogData OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS accessible-for-notify
    STATUS      current
    DESCRIPTION
        "A textual error log entry as stored in the system."
    ::= { payShieldAlarmErrorLog 1 }
```

```
payShieldBatteryState OBJECT-TYPE
    SYNTAX      INTEGER {
        stateOK (1),
        stateWarning (2),
        stateFailure (3)
    }
    MAX-ACCESS accessible-for-notify
    STATUS      current
    DESCRIPTION
        "State of the battery."
    ::= { payShieldAlarmBattery 1 }
```

```
payShieldDiagnosticID OBJECT-TYPE
    SYNTAX      INTEGER {
        diagDES (1),
        diagAES (2),
        diagECDSA (3),
        diagHMAC (4),
        diagMD5 (5),
        diagSHA (6),
        diagRSA (7),
        diagRNG (8),
        diagRTC (9),
        diagMemory (10),
        diagPower (11),
        diagBattery (12),
        diagFans (13),
        diagTemperature (14),
        diagVoltage (15),
        diagUDP (16),
        diagTCP (17),
        diagFICON (18)
    }
    MAX-ACCESS accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Diagnostic identifier as a code."
    ::= { payShieldAlarmDiagnostic 1 }
```

```
payShieldDiagnosticString OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Diagnostic identifier as a string."
    ::= { payShieldAlarmDiagnostic 2 }

payShieldFraudType OBJECT-TYPE
    SYNTAX      INTEGER {
        fraudExceededFailuresPerMinute (1),
        fraudExceededFailuresPerHour (2),
        fraudExceededAttackLimit (3)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Indicates what sort of fraud event took place."
    ::= { payShieldAlarmFraud 1 }

payShieldAlarmEraseTimeandDate OBJECT-TYPE
    SYNTAX      DateAndTime (SIZE(8))
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Indicates the time and date when the erase button was pressed."
    ::= { payShieldAlarmErase 1 }

payShieldModifiedSetting OBJECT-TYPE
    SYNTAX      INTEGER {
        settingSecurity (1),
        settingPCI (2),
        settingAuditedItems (3),
        settingEnabledPinblocks (4)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Collection of settings whose modifications will trigger a
notification."
    ::= { payShieldAlarmSettingsModified 1 }

payShieldDeviceState OBJECT-TYPE
    SYNTAX      INTEGER {
        stateUnavailable (1),
        stateOnline (2),
        stateOffline (3),
        stateSecure (4)
    }
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "Provides the new state of the payShield."
    ::= { payShieldAlarmStateChange 1 }

payShieldTamperCause OBJECT-TYPE
    SYNTAX      INTEGER {
        causeUnavailable (1),
```

```

        causeTemperatureOutOfRange (2),
        causeBatteryLow (3),
        causeEraseButtonPressed (4),
        causeSensorProcessorWatchdog (5),
        causeSensorProcessorRestart (6),
        causeVoltageOutOfRange (7),
        causeMotionDetected (8),
        causeCaseTampered (9),
        causePowerLoss (10)
    }
MAX-ACCESS accessible-for-notify
STATUS      current
DESCRIPTION
    "The cause of the tamper event."
 ::= { payShieldAlarmTamper 1 }

payShieldTamperDate OBJECT-TYPE
SYNTAX      DateAndTime
MAX-ACCESS  accessible-for-notify
STATUS      current
DESCRIPTION
    "The Date and Time that the tamper event occurred."
 ::= { payShieldAlarmTamper 2 }

payShieldPowerOnAlarm NOTIFICATION-TYPE
STATUS      current
DESCRIPTION
    "SNMP Trap indicating the unit has been powered up."
 ::= { payShieldTraps 1 }

payShieldRestartAlarm NOTIFICATION-TYPE
STATUS      deprecated
DESCRIPTION
    "***** THIS NOTIFICATION IS DEPRECATED *****

    SNMP trap generated when the reset button has been pressed."
 ::= { payShieldTraps 2 }

payShieldEraseAlarm NOTIFICATION-TYPE
OBJECTS { payShieldAlarmEraseTimeandDate }
STATUS      current
DESCRIPTION
    "SNMP trap indicating that the Erase button was pressed."
 ::= { payShieldTraps 3 }

payShieldNewLicenseAlarm NOTIFICATION-TYPE
STATUS      current
DESCRIPTION
    "SNMP trap indicating the installation of a new license."
 ::= { payShieldTraps 4 }

payShieldNewSoftwareAlarm NOTIFICATION-TYPE
STATUS      current
DESCRIPTION
    "SNMP trap indicating that new software has been installed."
 ::= { payShieldTraps 5 }

payShieldPSUAlarm NOTIFICATION-TYPE
OBJECTS { payShieldPSUNumber,

```

```
    payShieldPSUState }
STATUS      current
DESCRIPTION
    "SNMP trap indicating a change in the state of a Power supply unit."
::= { payShieldTraps 6 }

payShieldBatteryAlarm NOTIFICATION-TYPE
OBJECTS { payShieldBatteryState }
STATUS      current
DESCRIPTION
    "SNMP trap indicating a change in the state of the Battery."
::= { payShieldTraps 7 }

payShieldFanAlarm NOTIFICATION-TYPE
OBJECTS { payShieldFanNumber,
          payShieldFanState }
STATUS      current
DESCRIPTION
    "SNMP trap notifying a change in the state of a Fan unit."
::= { payShieldTraps 8 }

payShieldTamperAlarm NOTIFICATION-TYPE
OBJECTS { payShieldTamperCause,
          payShieldTamperDate }
STATUS      current
DESCRIPTION
    "SNMP trap indicating a tamper has been detected. Detected tamper is
reported."
::= { payShieldTraps 9 }

payShieldHostPortBadDataAlarm NOTIFICATION-TYPE
OBJECTS { payShieldBadDataPhysicalPort,
          payShieldBadDataProtocol }
STATUS      current
DESCRIPTION
    "SNMP trap indicating that protocol breaking data has arrived on a
port.
    Indicates which physical interface it arrived on and what protocol
was being used."
::= { payShieldTraps 10 }

payShieldFraudAlarm NOTIFICATION-TYPE
OBJECTS { payShieldFraudType }
STATUS      current
DESCRIPTION
    "SNMP trap that indicates a fraud limit was exceeded."
::= { payShieldTraps 11 }

payShieldDiagnosticTestFailureAlarm NOTIFICATION-TYPE
OBJECTS { payShieldDiagnosticID,
          payShieldDiagnosticString }
STATUS      current
DESCRIPTION
    "SNMP trap indicating that a diagnostic test has failed. Indicates
which test failed."
::= { payShieldTraps 12 }

payShieldErrorlogAlarm NOTIFICATION-TYPE
OBJECTS { payShieldErrorLogData }
```

```
STATUS      current
DESCRIPTION
    "SNMP trap that indicates an error log entry has been filled.
Contains the entry."
 ::= { payShieldTraps 13 }

payShieldSettingsModifiedAlarm NOTIFICATION-TYPE
OBJECTS { payShieldModifiedSetting }
STATUS      current
DESCRIPTION
    "SNMP trap that indicates a setting of the payShield has been
modified.
    It includes a payShieldModifiedSetting object which identifies the
setting that was changed."
 ::= { payShieldTraps 14 }

payShieldDeviceStateAlarm NOTIFICATION-TYPE
OBJECTS { payShieldDeviceState }
STATUS      current
DESCRIPTION
    "SNMP trap that indicates a change in payShield device state.
    It includes a payShieldDeviceState object which provides the new
state of the payShield."
 ::= { payShieldTraps 15 }

-- Conformance information

payShieldMIBConformance      OBJECT IDENTIFIER ::= { payShieldMIB 100 }
payShieldMIBCompliances      OBJECT IDENTIFIER ::= { payShieldMIBConformance 1 }
payShieldMIBGroups           OBJECT IDENTIFIER ::= { payShieldMIBConformance 2 }

-- Compliance statements

payShieldMIBCompliance MODULE-COMPLIANCE
STATUS deprecated
DESCRIPTION
    "***** THIS COMPLIANCE IS DEPRECATED *****

    The compliance statement for SNMP entities which
    implement the payShield MIB.

    This compliance is deprecated and replaced by
    payShieldMIBCompliance2."
MODULE -- this module
    MANDATORY-GROUPS { payShieldDeprecatedObjectGroup,
                        payShieldDeprecatedNotificationGroup }

 ::= { payShieldMIBCompliances 1 }

payShieldMIBCompliance2 MODULE-COMPLIANCE
STATUS current
DESCRIPTION
    "The compliance statement for SNMP entities which
    implement the payShield MIB."
MODULE -- this module
    MANDATORY-GROUPS { payShieldObjectGroup,
                        payShieldAlarmObjectGroup,
```

```
payShieldNotificationGroup }

 ::= { payShieldMIBCompliances 2 }

payShieldObjectGroup OBJECT-GROUP
  OBJECTS {
    payShieldUtilLoad,
    payShieldUtilHostCmdVolume,
    payShieldUtilEnabled,
    payShieldStateDevice,
    payShieldStateTamperState,
    payShieldStateTamperDate,
    payShieldStateTamperCause,
    payShieldStateFanSerialNum,
    payShieldStateFanState,
    payShieldStatePSUSerialNum,
    payShieldStatePSUState,
    payShieldStateBattery,
    payShieldLmkNumLoaded,
    payShieldLmkNumTestLmksLoaded,
    payShieldLmkNumOldLmksLoaded,
    payShieldLmkStatusLoaded,
    payShieldLmkStatusAuth,
    payShieldLmkStatusNumAuthActivities,
    payShieldLmkStatusScheme,
    payShieldLmkStatusAlgorithm,
    payShieldLmkStatusLiveTest,
    payShieldLmkStatusComments,
    payShieldLmkStatusCheckDigits,
    payShieldCommsMgmtConsoleState,
    payShieldCommsMgmtGuiState,
    payShieldCommsHostTCPServer,
    payShieldCommsHostUDPServer,
    payShieldCommsHostFICONServer,
    payShieldCommsHostPortEthernet1,
    payShieldCommsHostPortEthernet2,
    payShieldCommsHostPortFICON,
    payShieldSystemDateAndTime,
    payShieldSystemSerialNum,
    payShieldSystemModel,
    payShieldFraudPinVerifyLimitsExceeded,
    payShieldFraudPinAttackLimitsExceeded,
    payShieldVersionSoftwareBaseRelease,
    payShieldVersionSoftwareRevision,
    payShieldVersionSoftwareBuildNumber,
    payShieldVersionSoftwareCPLDVersion,
    payShieldVersionSoftwareSPVersion,
    payShieldVersionSoftwareSPBootVersion,
    payShieldVersionSoftwareBootstrapVersion,
    payShieldLicensingPerformanceModel,
    payShieldLicensingPackage,
    payShieldLicensingOptionalLicenseCount,
    payShieldLicensingOptionalLicensesList,
    payShieldLicensingCryptoAlgorithmCount,
    payShieldLicensingCryptoAlgorithmList,
    payShieldEnabledHostCommandsCount,
    payShieldEnabledHostCommandsList,
    payShieldLogsErrorlogTotalCount,
    payShieldLogsAuditlogTotalCount,
```

```

payShieldHealthDiagSelfTestTimeOfDay,
payShieldHealthDiagSelfTestOK,
payShieldHealthDiagSelfTestCount,
payShieldHealthDiagSelfTestList,
payShieldHealthHealthCheckEnabled,
payShieldHealthCheckCountsStartTime,
payShieldHealthCheckCountsEndTime,
payShieldHealthCheckCountsRebootCount,
payShieldHealthCheckCountsTamperCount,
payShieldHealthCheckCountsPinFailuresMinuteLimit,
payShieldHealthCheckCountsPinFailuresHourLimit,
payShieldHealthCheckCountsPinAttackLimitExceeded,
payShieldHostConnectionEthernetEnabled,
payShieldHostConnectionFICONEnabled,
payShieldHostConnectionEthernetIfCount,
payShieldHostConnectionEthernetSSLEnabled,
payShieldHostConnectionEthernetACLEnabled,
payShieldHostConnectionEthernetUDPEnabled,
payShieldHostConnectionEthernetTCPEnabled,
payShieldHostConnectionEthernetMaxTCPConnections,
payShieldHostConnectionEthernetTCPKeepalive,
payShieldHostConnectionEthernetWellKnownPortTCP,
payShieldHostConnectionEthernetWellKnownPortTLS,
payShieldHostConnectionEthernetConfigured,
payShieldHostConnectionEthernetInterfaceNumber,
payShieldHostConnectionEthernetConfigMethod,
payShieldHostConnectionEthernetHostName,
payShieldHostConnectionEthernetIpAddress,
payShieldHostConnectionEthernetSubnetMask,
payShieldHostConnectionEthernetGateway,
payShieldHostConnectionEthernetMacAddress,
payShieldHostConnectionEthernetPortSpeed,
payShieldHostConnectionNumberOfConnectionsUsed,
payShieldHostConnectionFICONControlUnitImage,
payShieldHostConnectionFICONControlUnitAddress,
payShieldHostConnectionFICONMIH,
payShieldManagementEthernetEnabled,
payShieldManagementEthernetConfigMethod,
payShieldManagementEthernetNetworkName,
payShieldManagementEthernetIpAddress,
payShieldManagementEthernetSubnetMask,
payShieldManagementEthernetGateway,
payShieldManagementEthernetMacAddress,
payShieldManagementEthernetPortSpeed,
payShieldPrinterLFCRReversed,
payShieldPrinterTimeout,
payShieldPrinterDelay,
payShieldPrinterReport,
payShieldSettingsPCICompliant,
payShieldAuxiliaryEthernetEnabled,
payShieldAuxiliaryEthernetConfigMethod,
payShieldAuxiliaryEthernetNetworkName,
payShieldAuxiliaryEthernetIpAddress,
payShieldAuxiliaryEthernetSubnetMask,
payShieldAuxiliaryEthernetGateway,
payShieldAuxiliaryEthernetMacAddress,
payShieldAuxiliaryEthernetPortSpeed

```

```

}

```

```

STATUS    current

```


DESCRIPTION

"Group of all payShield MIB objects."

::= { payShieldMIBGroups 1 }

payShieldDeprecatedObjectGroup OBJECT-GROUP

```
OBJECTS {  
    payShieldVersionSoftwareHSMCoreAPIVersion  
}
```

STATUS deprecated

DESCRIPTION

"Group of deprecated payShield MIB objects."

::= { payShieldMIBGroups 2 }

payShieldAlarmObjectGroup OBJECT-GROUP

```
OBJECTS {  
    payShieldFanNumber,  
    payShieldFanState,  
    payShieldPSUNumber,  
    payShieldPSUState,  
    payShieldBadDataPhysicalPort,  
    payShieldBadDataProtocol,  
    payShieldErrorLogData,  
    payShieldBatteryState,  
    payShieldDiagnosticID,  
    payShieldDiagnosticString,  
    payShieldFraudType,  
    payShieldAlarmEraseTimeandDate,  
    payShieldModifiedSetting,  
    payShieldDeviceState,  
    payShieldTamperCause,  
    payShieldTamperDate  
}
```

STATUS current

DESCRIPTION

"Group of objects used in payShield notifications."

::= { payShieldMIBGroups 3 }

payShieldNotificationGroup NOTIFICATION-GROUP

```
NOTIFICATIONS {  
    payShieldPowerOnAlarm,  
    payShieldEraseAlarm,  
    payShieldNewLicenseAlarm,  
    payShieldNewSoftwareAlarm,  
    payShieldPSUAlarm,  
    payShieldBatteryAlarm,  
    payShieldFanAlarm,  
    payShieldTamperAlarm,  
    payShieldHostPortBadDataAlarm,  
    payShieldFraudAlarm,  
    payShieldDiagnosticTestFailureAlarm,  
    payShieldErrorlogAlarm,  
    payShieldSettingsModifiedAlarm,  
    payShieldDeviceStateAlarm  
}
```

STATUS current

DESCRIPTION

"Group of payShield notifications."

::= { payShieldMIBGroups 4 }

```
payShieldDeprecatedNotificationGroup  NOTIFICATION-GROUP
    NOTIFICATIONS {
        payShieldRestartAlarm
    }
    STATUS deprecated
    DESCRIPTION
        "Group of deprecated payShield notifications."
    ::= { payShieldMIBGroups 5 }

END
```

Technical Support Contacts

Our team of knowledgeable and friendly support staff are available to help. If your product is under warranty or you hold a support contract with Thales, do not hesitate to contact us using the link below. For more information, consult our standard Terms and Conditions for Warranty and Support.

<https://supportportal.thalesgroup.com/csm>



Contact us

For all office locations and contact
information, please visit
cpl.thalesgroup.com/contact-us

> cpl.thalesgroup.com <

