THALES

cpl.thalesgroup.com

payShield 9000 v3.5

Applications Using the payShield 9000

1270A647-038

26 July 2021

Contents

End User License Agreement

Use of this product is subject to the Thales Cloud Protection & Licensing *End User License Agreement* found at:

https://cpl.thalesgroup.com/legal

Revision Status

Document No.	Manual Set	Software Version	Release Date
1270A647-038	Issue 38	payShield 9000 v3.5	July 2120

Introduction

The payShield 9000 documentation set includes manuals which describe the operation and features of the payShield 9000 HSM – for example in the *payShield 9000 Console Reference Manual* and the *payShield 9000 Host Command Reference Manual*. The payShield 9000 Host Programmer's Manual provides information relevant to developers of applications on the user organization's systems which will make use of the payShield 9000, and includes descriptions of some specific applications which are likely to be part of any payment system, such as PIN mailer printing and key component printing.

This document on the other hand looks at some application areas which users *may* wish to implement and make use of the payShield 9000 in. The information presented is not intended to be prescriptive or definitive in terms of stating how an application must be developed, but rather aims to provide background information and possible approaches to help application developers in designing the systems that meet the needs of their own organizations.

Chapter 1 - General data protection

Introduction

Whilst the functionality of the payShield 9000 is aimed specifically at cards and payments, it provides a set of commands that have been designed to protect general data being used in this environment. (For general data protection in other environments, the Thales Luna HSM product line may be more appropriate.) This is for use in two different scenarios:

- encryption of data at rest, and
- encryption of transient data.

"Data at rest" refers to data that is stored and retrieved by the same entity – for example, when encrypting account number records in a database. In this case, the encryption and decryption operations are performed by the same cryptographic module. In the payShield 9000, the encryption and decryption of data at rest is performed using a Data Encryption Key (DEK).

"Transient data" refers to data that is to be communicated between different entities – for example, when encrypting transaction data sent from an ATM to an acquiring host. In this case, the encryption and decryption functions are performed by different cryptographic modules. In the payShield 9000, the encryption and decryption of transient data is performed using a Zone Encryption Key (TEK, ZEK, or BDK).

The relevant host commands provided in the payShield 9000 software are:

- M0 – Encrypt a block of data: the HSM simply encrypts the supplied plaintext data using the given encryption key (DEK or ZEK). The resulting ciphertext is returned to the host.
- M2 – Decrypt a block of data: the HSM decrypts the supplied ciphertext using the given decryption key (DEK or ZEK). The resulting plaintext is returned to the host.
- M4 – Translate a block of data: the HSM converts ciphertext encrypted under the 'source' key (DEK/ZEK) to encryption under a new 'destination' key (DEK/ZEK). The resulting ciphertext is returned to the host.
- M6 – Generate a MAC on a block of data: this uses a symmetric key shared by the sender and recipient and allows the integrity and authenticity of the data to be verified by the recipient.
- M8 – Verify a MAC on a block of data: this allows the recipient of a block of data to verify its integrity.
- MY – Verify and Translate a MAC for a block of data: this verifies the MAC on a received block of data and generates a MAC for onward transmission under a different encryption key.
- GM – Hash a block of data using a hashing algorithm: this allows the user to verify the integrity of the data by repeating the same hashing action on the data and comparing results.
- EW – Generate an RSA signature on a block of data: this enables the sender to prove their identity to the recipient.
- EY – Validate an RSA signature on a block of data: this enables the recipient to validate the signature provided by the sender.

Use of these commands on the payShield 9000 requires optional license HSM9-LIC008. If commands requiring RSA are to be used, then optional license HSM9-LIC002 will also be required, and if the AES algorithm is going to be used then optional license HSM9-LIC007 will be required. All of these licences are available in HSM9-PAC302, HSM9-PAC303, and HSM9-PAC908

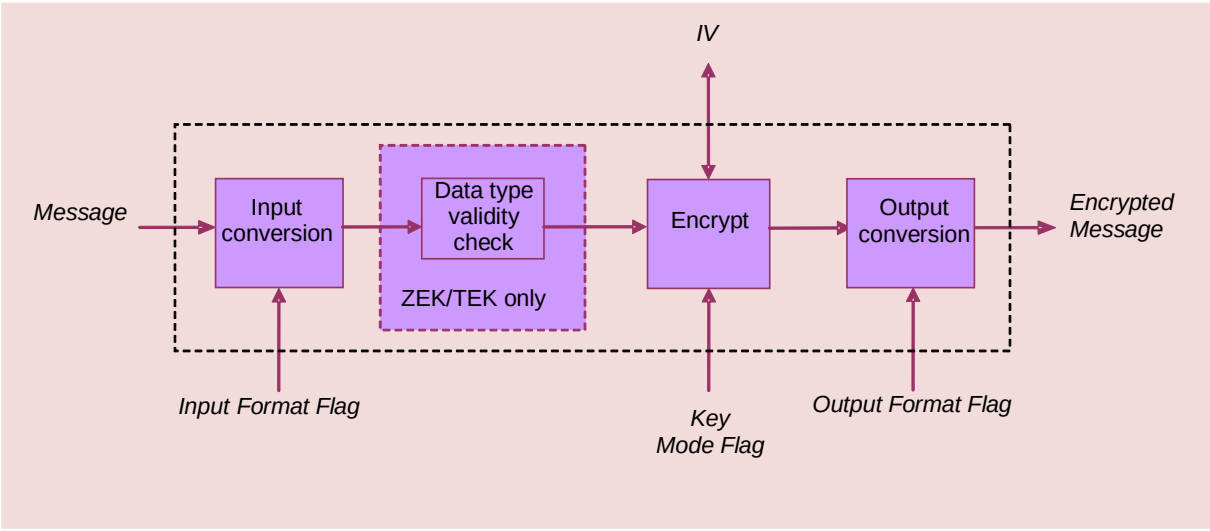
Message Encryption and Decryption

The HSM provides three generic commands to support general purpose data encryption:

- **M0 (Encrypt Data Block)** – to encrypt the supplied data using the supplied key.
- **M2 (Decrypt Data Block)** – to decrypt the supplied data using the supplied key.
- **M4 (Translate Data Block)** – to decrypt the supplied data using the first supplied key, and then re-encrypt the decrypted data using the second supplied key.

Encrypt Data Block

The *M0 Encrypt Data Block* host command encrypts a supplied message with a supplied key, using various flags to control the encryption process. The diagram below shows the overall process of the command:



Encrypt Data Block host command process

Notes:

- *The following key types may be used to decrypt data:*
  - *Using Variant LMK:*
    - *BDK Type 1, 2, 3, or 4 (Key Types 009, 609, 908, 909)*
    - *ZEK (Key type 00A)*
    - *TEK (Key type 30B)*
    - *DEK (Key type 00B)*
  - *Using Key Block LMK:*
    - *BDK Type 1, 2, 3, or 4 (Key Usage B0, 41, 42, 43)*
    - *ZEK (Key Usage 22)*
    - *TEK (Key Usage 23)*

- *DEK (Key Usage D0, 21)*

- *The data type validity check applies only when a ZEK or TEK key type is used to perform the encrypt operation, and applies the user security setting "Enable ZEK/TEK encryption of ASCII data or Binary data or None".*
- *Where a message consists of multiple data blocks, the IV (Initialisation Vector) output from one block of data is used as input to the next block of data.*

The tables below describe the *M0 Encrypt Data Block* command's input and output fields respectively. Optional fields such as headers and trailers are not shown for the sake of brevity and clarity.

The document *payShield 9000 Host Command Examples* gives some examples of the *M0* command with actual data.

M0 (Encrypt a block of data) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"M0"
Mode Flag	2 N	Specifies the desired encryption mode. The four possible values are: "00": ECB (Electronic Code Book) "01": CBC (Cipher Block Chaining) "02": CFB8 (8-bit Cipher Feedback) "03": CFB64 (64-bit Cipher Feedback) When using ECB mode, no padding is applied, so the input data must be a multiple of 8 bytes. For all other modes, an Initialisation Vector (IV) must be supplied, and a new IV is returned.

Input Field Name	Format	Description
Input Format Flag	1 N	Specifies the format of the supplied input *Message* field. The three possible values are: "0": Binary "1": Hex-Encoded Binary "2": Text For Binary formatted input messages, each byte in the *Message* field can take the hex values 00..FF. The message is encrypted with no conversion process taking place. Note: The length of the supplied *Message* must be a multiple of 8 bytes. For Hex-Encoded Binary formatted input messages, each byte in the *Message* field can take the ASCII values '0'..'F'. Prior to encryption, the HSM converts the input message to binary (resulting in a message half the size of the input *Message* field). Note: The length of the supplied *Encrypted Message* must be a multiple of 16 bytes. For Text formatted input messages, each byte in the *Message* field contains only printable characters. If the HSM is in ASCII mode, the input *Message* field must contain only valid ASCII characters. If the HSM is in EBCDIC mode, the input *Message* field must contain only valid EBCDIC characters – and these are converted to ASCII prior to encryption. Note: The length of the supplied *Encrypted Message* must be a multiple of 8 bytes.
Output Format Flag	1 N	Specifies the format of the output *Encrypted Message* field. The two possible values are: "0": Binary "1": Hex-Encoded Binary For Binary formatted output messages, the *Encrypted Message* field contains the raw encrypted message – no conversion process takes place. Each byte in the *Encrypted Message* field can take the hex values 00..FF. For Hex-Encoded Binary formatted output messages, the HSM converts the raw encrypted message to hex-encoded binary (resulting in a message twice the size of the raw encrypted message). Each byte in the *Encrypted Message* field can take the ASCII values '0'..'F'.

Input Field Name	Format	Description
Key Type	3 H	Specifies the type of the encryption key. The possible values are: "009": Type-1 BDK (DUKPT Base derivation Key) "609": Type-2 BDK (DUKPT Base derivation Key) "809": Type-3 BDK (DUKPT Base derivation Key) "00A" : ZEK (Zone Encryption Key) "00B": DEK (Data Encryption Key) "30B": TEK (Terminal Encryption Key) "FFF": Key Block LMK being used
Key	16 H or 1 A + 16/32/48 H or 1 A + n A	Specifies the encryption key. This is used in conjunction with the *Mode Flag* and *IV* input fields as appropriate to encrypt the supplied *Message*.
KSN Descriptor	3 H	Required only for BDK encryption keys.
KSN	12-20 H	Key serial Number - required only for BDK encryption keys.
IV	16 H or 32 H	Specifies the input Initialisation Vector – only present when *Mode Flag* indicates an encryption mode which uses an IV (CBC, CFB8 or CFB64).

<table>
<tr><th>Input Field Name</th><th>Format</th><th>Description</th></tr>
<tr><td>Message Length</td><td>4 H</td><td>The length of the Message field. If encryption of a larger message is required, the original message should be divided into smaller blocks, each of which is less than 32000 bytes.

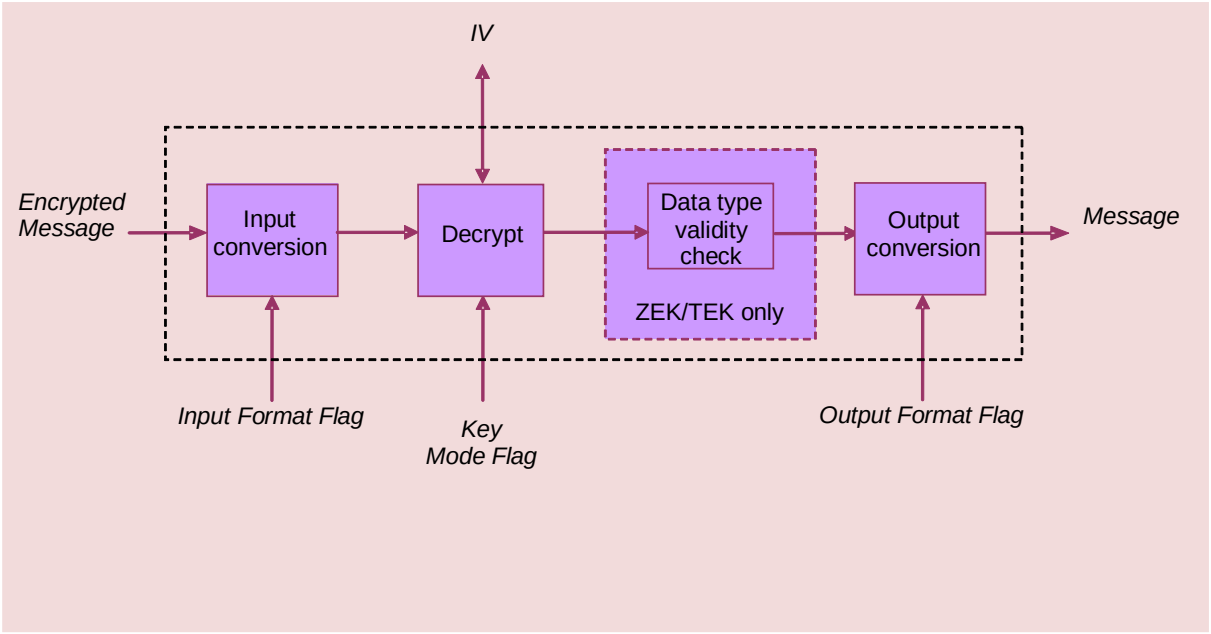
Although the HSM will accept up to 32000 characters in the Message field, this figure may be affected by the value of the Input Format Flag and Output Format Flag fields. The table below indicates the actual maximum values for the given conditions:
<table>
<tr><th>Input Format Flag</th><th>Output Format Flag</th><th>Maximum Message Length (Bytes)</th><th>Maximum Encrypted Message Length (Bytes)</th><th>Actual number of bytes encrypted</th></tr>
<tr><td>Binary/ Text</td><td>Binary</td><td>32000</td><td>32000</td><td>32000</td></tr>
<tr><td>Hex-Encoded Binary</td><td>Binary</td><td>32000</td><td>16000</td><td>16000</td></tr>
<tr><td>Binary/ Text</td><td>Hex-Encoded Binary</td><td>16000</td><td>32000</td><td>16000</td></tr>
<tr><td>Hex-Encoded Binary</td><td>Hex-Encoded Binary</td><td>32000</td><td>32000</td><td>16000</td></tr>
</table>
</td></tr>
<tr><td>Message</td><td>n B
or
n H
or
n A</td><td>Specifies the input message to be encrypted. The format of the data within this field is defined by the Input Format Flag field.</td></tr>
</table>

M1 Response (to M0 Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"M1"
Error Code	2A	"00" indicates No Error
IV	16 H or 32 H	Contains the output Initialisation Vector – only present when *Mode Flag* indicates an encryption mode which uses an IV (CBC, CFB8 or CFB64).
Encrypted Message Length	4 H	Indicates the length of the following field.
Encrypted Message	n B or n H	Contains the encrypted message data. The format of the data within this field is defined by the *Output Format Flag* field.

Decrypt Data Block

The *M2 Decrypt Data Block* host command decrypts a supplied encrypted message with a supplied key, using various flags to control the decryption process. The diagram below shows the overall process of the command:



Decrypt Data Block host command

Notes:

- *The following key types may be used to decrypt data:*
  - *Using Variant LMK:*
    - *BDK Type 1, 2, or 3 (Key Types 009, 609, 908)*
    - *ZEK (Key type 00A)*
    - *TEK (Key type 30B)*
    - *DEK (Key type 00B)*
  - *Using Key Block LMK:*

- *BDK Type 1, 2, or 3 (Key Usage B0, 41, 42)*
- *ZEK (Key Usage 22)*
- *TEK (Key Usage 23)*
- *DEK (Key Usage D0, 21)*

➢ *The data type validity check applies only when a ZEK or TEK key type is used to perform the encrypt operation, and applies the user security setting "Enable ZEK/TEK encryption of ASCII data or Binary data or None".*

➢ *Where a message consists of multiple data blocks, the IV (Initialisation Vector) output from one block of data is used as input to the next block of data.*

The tables below describe the Encrypt Data Block command's input and output fields respectively. The document *payShield 9000 Host Command Examples* gives an example of the *M2* command with actual data.

M2 (Decrypt a block of data) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"M2"
Mode Flag	2 N	Specifies the desired decryption mode. The four possible values are: "00": ECB (Electronic Code Book) "01": CBC (Cipher Block Chaining) "02": CFB8 (8-bit Cipher Feedback) "03": CFB64 (64-bit Cipher Feedback)
Input Format Flag	1 N	Specifies the format of the supplied input *Encrypted Message* field. The two possible values are: "0": Binary "1": Hex-Encoded Binary For Binary formatted input messages, each byte in the *Encrypted Message* field can take the hex values 00..FF. The encrypted message is decrypted as is – no conversion process takes place. Note: The length of the supplied *Encrypted Message* must be a multiple of 8 bytes. For Hex-Encoded Binary formatted input messages, each byte in the *Encrypted Message* field can take the ASCII values '0'..'F'. Prior to decryption, the HSM converts the input message to binary (resulting in a message half the size of the input *Encrypted Message* field). Note: The length of the supplied *Encrypted Message* must be a multiple of 16 bytes.

Input Field Name	Format	Description
Output Format Flag	1 N	Specifies the format of the output Decrypted Message field. The three possible values are: "0": Binary "1": Hex-Encoded Binary "2": Text For Binary formatted output messages, the output Message field contains the raw decrypted message – no conversion process takes place. Each byte in the Message field can take the hex values 00..FF. For Hex-Encoded Binary formatted output messages, the HSM converts the raw decrypted message to hex-encoded binary (resulting in a message twice the size of the raw decrypted message). Each byte in the output Message field may take the ASCII values '0'..'F'. For Text formatted output messages, each byte in the output Message field contains only printable characters. If the HSM is in EBCDIC mode, the raw decrypted message will be converted to EBCDIC prior to being output.
Key Type	3 H	Specifies the type of the decryption key. The possible values are: "009": Type 1 BDK (DUKPT Base Derivation Key) "609": Type 2 BDK (DUKPT Base Derivation Key) "809": Type 3 BDK (DUKPT Base Derivation Key) "00A": ZEK (Zone Encryption Key) "00B": DEK (Data Encryption Key) "30B": TEK (Terminal Encryption Key "FFF": Key Block LMK being used Refer to sections 2 and 3 for more details.
Key	16 H or 1 A + 16/32/48 H or 1 A + n A	Specifies the decryption key. This is used in conjunction with the Mode Flag and IV input fields as appropriate to decrypt the supplied Encrypted Message.
KSN Descriptor	3 H	Required only for BDK encryption keys.
KSN	12-20 H	Key serial Number - required only for BDK encryption keys.
IV	16 H or 32 H	Specifies the input Initialisation Vector – only present when Mode Flag indicates a decryption mode which uses an IV (CBC, CFB8 or CFB64).

<table>
<tr><th>Input Field Name</th><th>Format</th><th>Description</th></tr>
<tr><td>Encrypted Message Length</td><td>4 H</td><td>The length of the following Encrypted Message field. If decryption of a larger message is required, the original message should be divided into smaller blocks, each of which is less than 32000 bytes.

Although the HSM will accept up to 32000 characters in the Encrypted Message field, this figure may be affected by the value of the Input Format Flag and Output Format Flag fields. The table below indicates the actual maximum values for the given conditions:

<table>
<tr><th>Input Format Flag</th><th>Output Format Flag</th><th>Maximum Encrypted Message Length (Bytes)</th><th>Maximum Message Length (Bytes)</th><th>Actual number of bytes decrypted</th></tr>
<tr><td>Binary</td><td>Binary/ Text</td><td>32000</td><td>32000</td><td>32000</td></tr>
<tr><td>Hex-Encoded Binary</td><td>Binary/ Text</td><td>32000</td><td>16000</td><td>16000</td></tr>
<tr><td>Binary</td><td>Hex-Encoded Binary</td><td>16000</td><td>32000</td><td>16000</td></tr>
<tr><td>Hex-Encoded Binary</td><td>Hex-Encoded Binary</td><td>32000</td><td>32000</td><td>16000</td></tr>
</table>
</td></tr>
<tr><td>Encrypted Message</td><td>n B
or
n H</td><td>Specifies the input message to be decrypted. The format of the data within this field is defined by the Input Format Flag field.</td></tr>
</table>

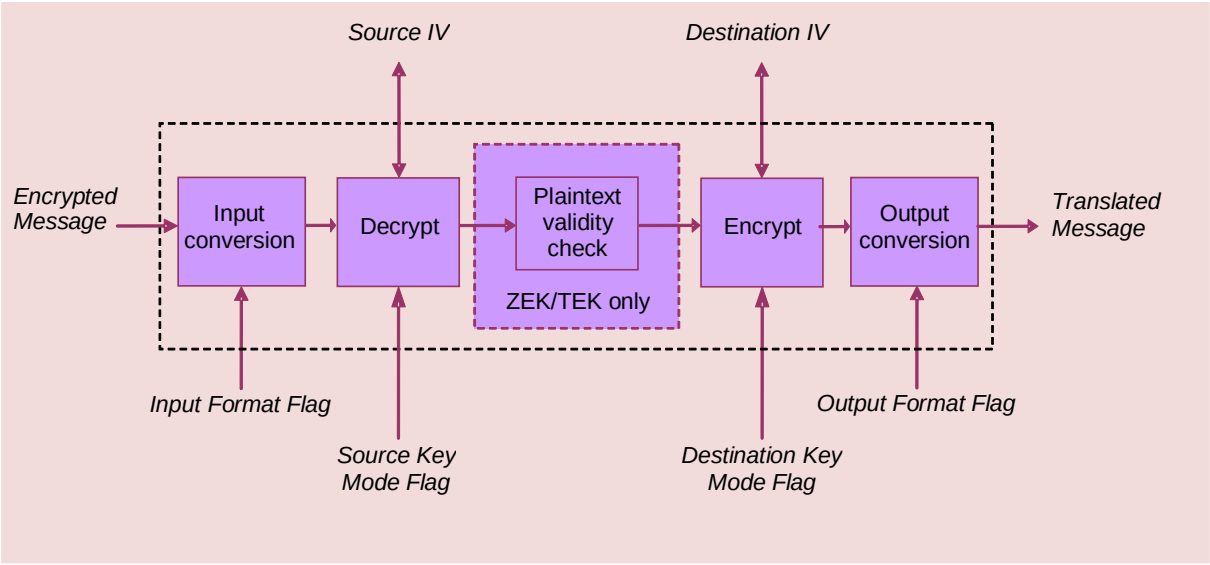
M3 Response (to M2 Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"M3"
Error Code	2A	"00" indicates No Error
IV	16 H or 32 H	Contains the output Initialisation Vector – only present when *Mode Flag* indicates an encryption mode which uses an IV (CBC, CFB8 or CFB64).
Decrypted Message Length	4 H	Indicates the length of the following field.

Output Field Name	Format	Description
Decrypted Message	n B or n H or n A	Contains the decrypted message data. The format of the data within this field is defined by the *Output Format Flag* field.

Translate Data Block

The *M4 Translate Data Block* host command decrypts a supplied encrypted message with a supplied key, and then re-encrypts the decrypted message using a second supplied key, using various flags to control the encryption/decryption processes. The diagram below shows the overall process of the command:



Translate Data Block host command

Notes:

- *The following key types may be used to translate data:*
  - *Using Variant LMK:*
    - *BDK Type 1, 2, or 3 (Key Types 009, 609, 908)*
    - *ZEK (Key type 00A)*
    - *TEK (Key type 30B)*
    - *DEK (Key type 00B)*
  - *Using Key Block LMK:*
    - *BDK Type 1, 2, or 3 (Key Usage B0, 41, 42)*
    - *ZEK (Key Usage 22)*
    - *TEK (Key Usage 23)*
    - *DEK (Key Usage D0, 21)*
- *The data type validity check applies only when a ZEK or TEK key type is used to perform the encrypt operation, and applies the user security setting "Enable ZEK/TEK encryption of ASCII data or Binary data or None".*
- *Where a message consists of multiple data blocks, the IV (Initialisation Vector) output from one block of data is used as input to the next block of data.*

The tables below describe the Translate Data Block command's input and output fields respectively. The document *payShield 9000 Host Command Examples* provides ab example of an *M4* command,

M4 (Translate a data block) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"M4"
Source Mode Flag	2 N	Specifies the desired decryption mode. The four possible values are: "00": ECB (Electronic Code Book) "01": CBC (Cipher Block Chaining) "02": CFB8 (8-bit Cipher Feedback) "03": CFB64 (64-bit Cipher Feedback)
Destination Mode Flag	2 N	Specifies the desired encryption mode. The four possible values are: "00": ECB (Electronic Code Book) "01": CBC (Cipher Block Chaining) "02": CFB8 (8-bit Cipher Feedback) "03": CFB64 (64-bit Cipher Feedback)
Input Format Flag	1 N	Specifies the format of the input *Encrypted Message* field. The two possible values are: "0": Binary "1": Hex-Encoded Binary For Binary formatted input messages, each byte in the *Encrypted Message* field can take the hex values 00..FF. The encrypted message is translated as is – no conversion process takes place. Note: The length of the supplied *Encrypted Message* must be a multiple of 8 bytes. For Hex-Encoded Binary formatted input messages, each byte in the *Encrypted Message* field can take the ASCII values '0'..'F'. Prior to translation, the HSM converts the input message to binary (resulting in a message half the size of the input *Encrypted Message* field). Note: The length of the supplied *Encrypted Message* must be a multiple of 16 bytes.
Output Format Flag	1 N	Specifies the format of the output *Translated Message* field. The two possible values are: "0": Binary "1": Hex-Encoded Binary For Binary formatted output messages, the output *Message* field contains the raw translated message – no conversion process takes place. Each byte in the *Message* field can take the hex values 00..FF. For Hex-Encoded Binary formatted output messages, the HSM converts the raw translated message to hex-encoded binary (resulting in a message twice the size of the raw translated

		message). Each byte in the output *Message* field may take the ASCII values '0'..'F'.
Source Key Type	3 H	Specifies the type of the decryption key. The possible values are: "009": Type-1 BDK (DUKPT Base Derivation Key) "609": Type-2 BDK (DUKPT Base Derivation Key) "809": Type-3 BDK (DUKPT Base Derivation Key) "00A": ZEK (Zone Encryption Key) "00B": DEK (Data Encryption Key) "30B": TEK (Terminal Encryption Key) "FFF": Key Block LMK being used. Refer to sections 2 and 3 for more details.
Source Key	16 H or 1 A + 16/32/48 H or 1 A + n A	Specifies the decryption key. This is used in conjunction with the *Source Mode Flag* and *Source IV* input fields as appropriate to decrypt the supplied *Encrypted Message*.
Source KSN Descriptor	3 H	Required only for BDK encryption keys.
Source KSN	12-20 H	Key serial Number - required only for BDK encryption keys.
Destination Key Type	3 H	Description as for Source Key Type
Destination Key	16 H or 1 A + 16/32/48 H or 1 A + n A	Specifies the encryption key. This is used in conjunction with the *Destination Mode Flag* and *Destination IV* input fields as appropriate to re-encrypt the decrypted *Encrypted Message*.
Destination KSN Descriptor	3 H	Required only for BDK encryption keys.
Destination KSN	12-20 H	Key serial Number - required only for BDK encryption keys.
Source IV	16 H or 32 H	Specifies the input Initialisation Vector used in the decryption process – only present when *Source Mode Flag* indicates a decryption mode which uses an IV (CBC, CFB8 or CFB64).
Destination IV	16 H or 32 H	Specifies the input Initialisation Vector used in the re-encryption process – only present when *Destination Mode Flag* indicates an encryption mode which uses an IV (CBC, CFB8 or CFB64).
Encrypted Message Length	4 H	The length of the following *Encrypted Message* field. If translation of a larger message is required, the original message should be divided into smaller blocks, each of which is less than 32000 bytes. Although the HSM will accept up to 32000 characters in the *Encrypted Message* field, this figure may be affected by the value of the *Input*

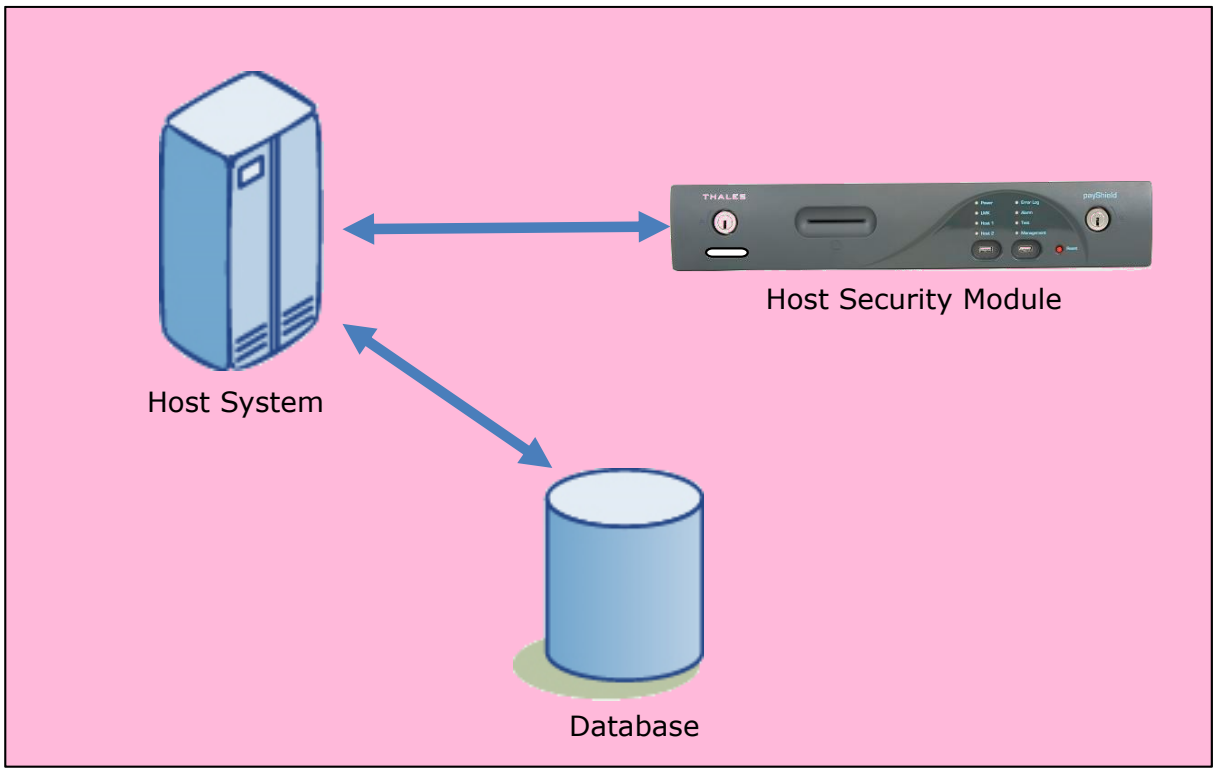
<table>
<tr><td></td><td></td><td>Format Flag and Output Format Flag fields. The table below indicates the actual maximum values for the given conditions:
<table>
<tr><th>Input Format Flag</th><th>Output Format Flag</th><th>Maximum Encrypted Message Length (Bytes)</th><th>Maximum Message Length (Bytes)</th><th>Actual number of bytes decrypted</th></tr>
<tr><td>Binary</td><td>Binary</td><td>32000</td><td>32000</td><td>32000</td></tr>
<tr><td>Hex-Encoded Binary</td><td>Binary</td><td>32000</td><td>16000</td><td>16000</td></tr>
<tr><td>Binary</td><td>Hex-Encoded Binary</td><td>16000</td><td>32000</td><td>16000</td></tr>
<tr><td>Hex-Encoded Binary</td><td>Hex-Encoded Binary</td><td>32000</td><td>32000</td><td>16000</td></tr>
</table></td></tr>
<tr><td>Encrypted Message</td><td>n B
or
n H</td><td>Specifies the input message to be translated. The format of the data within this field is defined by the Input Format Flag field.</td></tr>
</table>

M5 Response (to M4 Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"M5"
Source IV	16 H or 32 H	Contains the output Initialisation Vector used in the decryption process – only present when *Source Mode Flag* indicates a decryption mode which uses an IV (CBC, CFB8 or CFB64).
Destination IV	16 H or 32 H	Contains the output Initialisation Vector used in the re-encryption process – only present when *Destination Mode Flag* indicates an encryption mode which uses an IV (CBC, CFB8 or CFB64).
Translated Message Length	4 H	Indicates the length of the following field.
Translated Message	n B or n H	Contains the translated message data. The format of the data within this field is defined by the *Output Format Flag* field.

Encryption of Data At Rest

The term "data at rest" refers to data stored and maintained by a single host system. How or where the data is stored is immaterial – what is important is that there is no requirement for any other system to decrypt the data. Typical applications that fall into this category include database record encryption, file encryption, secure backup/archiving, etc.



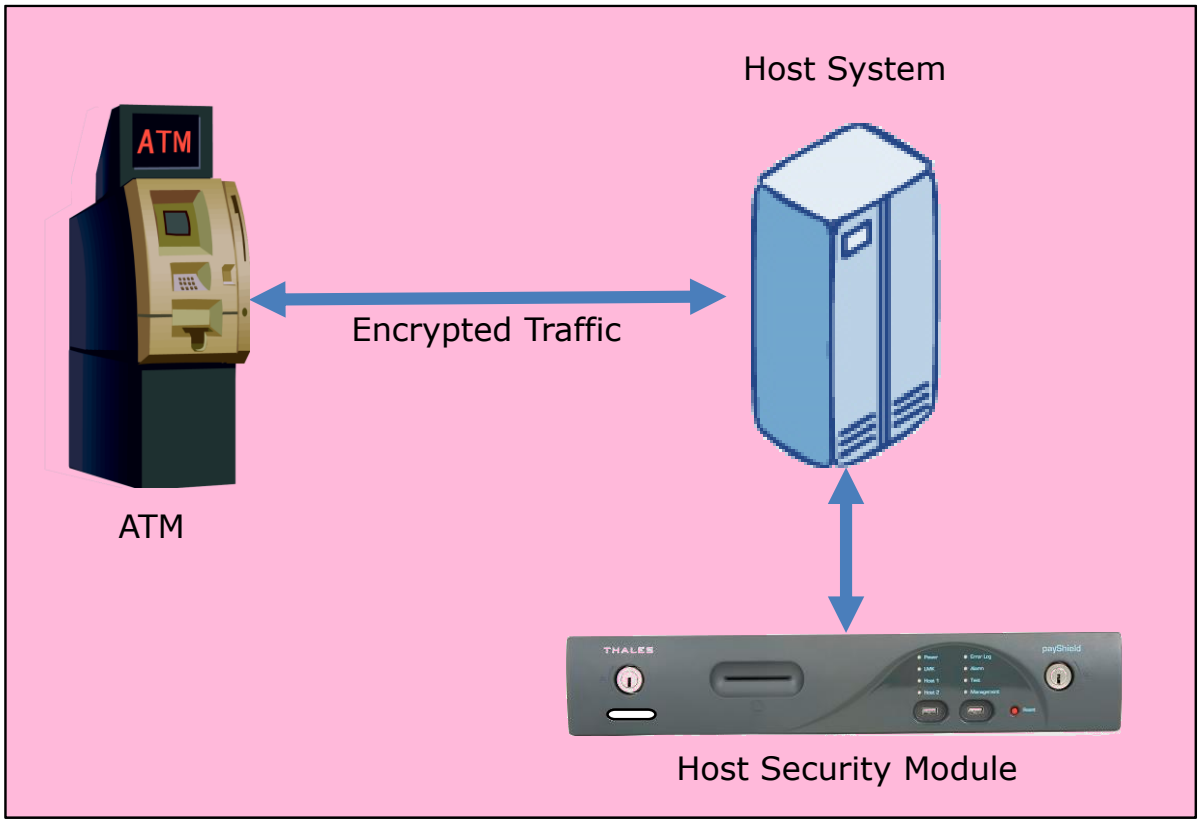
Typical application for encryption of Data at Rest

To perform the encryption of data at rest, a DEK (Data Encryption Key) is used. What restricts a DEK to encrypting data at rest is that a DEK cannot be exported out of, or imported into, the HSM. A DEK can only be used by the HSM that generated it (or another HSM with the same Local Master Key).

A DEK can be used with the host commands to Encrypt/Decrypt/Translate Data Blocks, and there is no restriction as to the type of data that can be encrypted.

Encryption of Transient Data

The term "transient data" refers to data exchanged between two independent parties. The most important feature of encryption of transient data is that the encryption key needs to be shared between the two parties. Typical applications that fall into this category include encryption of traffic between an ATM or PoS terminal and host system.



Typical Application for encryption of Transient Data

Encryption of transient data is performed using a TEK (Terminal Encryption Key) or ZEK (Zone Encryption Key). A TEK/ZEK may be generated, imported or exported by the HSM using existing commands.

However, the type of data that may be encrypted, decrypted or translated by a TEK/ZEK can be restricted using the *CS* console command or payShield Manager *Configuration / Security Settings / Initial* dialogue box: this can be set to allow the processing of any (binary) data, or of only ASCII data, or the encryption/decryption/translation of data using a TEK/ZEK can be blocked altogether.

```
Secure> CS <Return>
..
..
..
Enable ZEK/TEK encryption of ASCII data or Binary data or None? [A/B/N] (NONE):
..
..
Secure>
```

Configure Security console command

The three Message Encryption host commands implement a Data Type Validity Check at the following locations:

- Encrypt Data Block – immediately prior to encryption (see the diagram at *Encrypt Data Block*).
- Decrypt Data Block – immediately after decryption (see the diagram at *Decrypt Data Block*).
- Translate Data Block – in between decryption & encryption (see the diagram at *Translate Data Block*).

Protecting Data

Encryption of data is only one aspect of protecting data. The sender and recipient may wish to ensure that:

- the integrity of the message is preserved – i.e. the received message has not been corrupted (maliciously or accidentally), and that
- the recipient and sender are authenticated – i.e. they are the legitimate parties in the data exchange.

These goals can be achieved using a number of approaches:

- **Hashing** the data. This uses a standard algorithm to create a short digest (or hash) of the data. There is an insignificantly low probability that any change to the data could result in the algorithm generating the same hash, and so the hash effectively guarantees the integrity of the data. No key is used in hashing, and hashing is often used when signing data using RSA (see below) because the size of the hash is smaller than the length of the RSA key (whereas the original data may be longer than the length of the RSA key).
- **MACing** the data. This also generates a short digest of the data, but the MACing algorithm uses a symmetric key (a TAK (Terminal Authentication Key) or ZAK (Zone Authentication Key)) known only to the sender and recipient. The MAC (Message Authentication Code) therefore both assures integrity of the data and authenticates the sender and recipient.
- **Digital Signatures**. Signatures make use of asymmetric algorithms such as RSA. The sender creates a key pair, consisting of a Private key and a Public Key. The Public Key can be given out freely. The sender uses his Private Key to generate a digital signature on his data. (Because the data length is restricted to the modulus size of the RSA key – e.g. 1024 or 2048 bits – the signature is usually applied to a hash of the data.) The signature accompanies the data and the recipient can use the Public Key to verify the signature. This assures both the integrity of the message and that the message came from the expected sender (as the Public Key would work only with signatures created using the sender's Private key.)

The following sections describe the host commands available on the payShield 9000 to implement these protection methods.

Hashing

Hashing is accomplished by using the *GM* host command. This will generate a hash over a block of data of up to 32,000 bytes.

GM (Hash a block of data) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"GM"
Hash Identifier	2 N	Identifies the hash algorithm to be used. Available options are: "01": SHA-1 "02": MD5 "03": ISO 10118-2 "05": SHA-224 "06": SHA-256 "07": SHA-384 "08": SHA-512
Data Length	5 N	Length of message to be hashed.
Message data	n B	Data to be hashed

GN Response (to GM Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"GN"
Error Code	2 A	"00" indicates No Error
Hash value	n B	Hash result. The length of the hash depends on the algorithm that was used.

MACing

MACing is achieved by using the *M6 Generate MAC*, *M8 Verify MAC*, and *MY Verify/Translate MAC* host commands.

M6 (Generate MAC) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"M6"
Mode Flag	1 N	This data block is: are: "0": the only block in the message. "1": the first block of a multi-block message. "2": a middle block of a multi-block message "3": the final block of a multi-block message.

Input Field Name	Format	Description
Input Format Flag	1 N	Specifies the format of the supplied input *Message* field. The three possible values are: "0": Binary "1": Hex-Encoded Binary "2": Text For Binary formatted input messages, each byte in the *Message* field can take the hex values 00..FF. The message is encrypted with no conversion process taking place. Note: The length of the supplied *Message* must be a multiple of 8 bytes. For Hex-Encoded Binary formatted input messages, each byte in the *Message* field can take the ASCII values '0'..'F'. Prior to encryption, the HSM converts the input message to binary (resulting in a message half the size of the input *Message* field). Note: The length of the supplied *Encrypted Message* must be a multiple of 16 bytes. For Text formatted input messages, each byte in the *Message* field contains only printable characters. If the HSM is in ASCII mode, the input *Message* field must contain only valid ASCII characters. If the HSM is in EBCDIC mode, the input *Message* field must contain only valid EBCDIC characters – and these are converted to ASCII prior to encryption. Note: The length of the supplied *Encrypted Message* must be a multiple of 8 bytes.
MAC size	1 N	The possible values are: "0": 8 Hex digits "1": 16 Hex digits
MAC Algorithm	1 N	The possible values are: "1": ISO 9797 MAC Algorithm 1. DES only. (=X9.9 when used with single-length key). "3": ISO 9797 MAC Algorithm 1. DES only. (=X9.19 when used with double-length key). "5": CBC-MAC. AES only. "6": CMAC. AES only.
Padding Method	1 N	The possible values are: For MAC Algorithm 1, 3 & 5: "0": No padding. '1' : ISO 9797 Padding method 1 (i.e. pad with 0x00). "2": ISO 9797 Padding method 2. "3": ISO 9797 Padding method 3. For MAC Algorithm 6: "4": AES CMAC Padding.

Input Field Name	Format	Description
Key Type	3 H	The possible values are: "003": TAK "008": ZAK "FFF" : Key Block LMK being used
Key	16 H or 1 A + 16/32/48 H or 1 A + n A	The MACing key, used in conjunction with the IV, if appropriate, to generate the MAC.
IV	16 H or 32 H	The intermediate IV. When MACing the middle or final blocks of a series of blocks, this should be the IV returned from MACing the previous block. If using a DES/3 DES key, the IV will be a 16 H field. If using an AES key, the IV will be a 32 H field. Only present if Mode Flag = '2' or '3'.
Message Length	4 H	The length of the message to be MACed. The maximum message length is 32,000 bytes (7D00 in Hex)
Message	n B or n H or n A	The message to be MACed.

M7 Response (to M6 Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"M7"
Error Code	2 A	"00" indicates No Error
IV	16 H or 32 H	The intermediate IV, used as input for the next block.
MAC	8 H or 16 H	The calculated MAC.

M8 (Verify MAC) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"M8"
Mode Flag	1 N	Same as for M6 command.
Input Format Flag	1 N	Same as for M6 command.
MAC size	1 N	Same as for M6 command.
MAC Algorithm	1 N	Same as for M6 command.
Padding Method	1 N	Same as for M6 command.
Key Type	3 H	Same as for M6 command.
Key	16 H or 1 A + 16/32/48 H or 1 A + n A	Same as for M6 command.
IV	16 H or 32 H	Same as for M6 command.
Message Length	4 H	Same as for M6 command.
Message	n B or n H or n A	Same as for M6 command.
MAC	8 H or 16 H	The MAC that is to be verified for the message.

M9 Response (to M8 Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"M9"
Error Code	2 A	"00" indicates No Error. Other codes represent various MAC Verification failure modes.
IV	16 H or 32 H	The intermediate IV, used as input for the next block.

MY (Translate MAC) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"MY"
Mode Flag	1 N	Same as for M6 command.
Input Format Flag	1 N	Same as for M6 command.
Source MAC size	1 N	Same as for M6 command.
Source MAC Algorithm	1 N	Same as for M6 command.
Source Padding Method	1 N	Same as for M6 command.
Source Key Type	3 H	Same as for M6 command.
Source Key	16 H or 1 A + 16/32/48 H or 1 A + n A	Same as for M6 command.
Destination MAC size	1 N	Same as for M6 command.
Destination MAC Algorithm	1 N	Same as for M6 command.
Destination Padding Method	1 N	Same as for M6 command.
Destination Key Type	3 H	Same as for M6 command.
Destination Key	16 H or 1 A + 16/32/48 H or 1 A + n A	Same as for M6 command.
Source IV	16 H or 32 H	Same as for M6 command.
Destination IV	16 H or 32 H	Same as for M6 command.
Message Length	4 H	Same as for M6 command.
Message	n B or n H or n A	Same as for M6 command.

Input Field Name	Format	Description
Source MAC	8 H or 16 H	The MAC that is to be verified for the message using the Source Key.

MZ Response (to MY Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"MZ"
Error Code	2 A	"00" indicates No Error. Other codes represent various MAC Verification failure modes.
Source IV	16 H or 32 H	The intermediate IV, used as input for the next block using the Source Key.
Destination IV	16 H or 32 H	The intermediate IV, used as input for the next block using the Destination Key.
Destination MAC	8 H or 16 H	The MAC generated using the Destination Key.

Digital Signatures

The payShield 9000 provides the *EW Generate an RSA Signature* and *EY Validate an RSA Signature* host commands.

EW (Generate RSA Signature) Command Structure

Input Field Name	Format	Description
Command Code	2 A	"EW"
Hash Identifier	2 N	Identifies the hash algorithm to be used. Available options are: "01": SHA-1 "02": MD5 "03": ISO 10118-2 "05": SHA-224 "06": SHA-256 "07": SHA-384 "08": SHA-512
Signature Identifier	2 N	Always "01" (for RSA).
Pad Mode Identifier	2 N	Always "01" (PKCS#1 v1.5)
Data Length	4 N	Length of data (in Bytes) to be signed.
Message Data	n B	Data to be signed.

Input Field Name	Format	Description
Delimiter	1 A	Always ";". Used to define the end of the message data.
Private Key Flag	2 N	The payShield 9000 can store 21 RSA Private Keys internally. These can be referenced by indexes "00"-"20" entered here. If the RSA Private Key is to be included in the command, then "99" is entered here.
Private Key Length	4 N or 4 H	The length of the Private Key, in Bytes. (For Key Block MKs, this field is set to "FFFF").
Private Key	n B or 1 A + n B	The RSA Private Key to be used to generate the signature.

EX Response (to EW Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"EX"
Error Code	2 A	"00" indicates No Error.
Signature Length	4 N	Length of the signature, in bytes
Signature	n B	The calculated signature.

EY (Verify RSA Signature) Command Structure

The structure below applies when using a Variant LMK. See the *payShield 9000 Host Command Reference Manual* for the differences when using a Key Block LMK.

Input Field Name	Format	Description
Command Code	2 A	"EY"
Hash Identifier	2 N	Identifies the hash algorithm to be used. Available options are: "01": SHA-1 "02": MD5 "03": ISO 10118-2 "05": SHA-224 "06": SHA-256 "07": SHA-384 "08": SHA-512
Signature Identifier	2 N	Always "01" (for RSA).
Pad Mode Identifier	2 N	Always "01" (PKCS#1 v1.5)
Signature Length	4 N	Length of signature (in Bytes) to be verified.
Signature	n B	Signature to be verified.

Input Field Name	Format	Description
Delimiter	1 A	Always ";". Used to define the end of the signature.
Data Length	4 N	Length of data (in Bytes) for which the signature is to be verified.
Message Data	n B	The data for which the signature is to be verified.
Delimiter	1 A	Always ";". Used to define the end of the message data.
MAC	4 B	MAC on the Public Key and authentication data.
Public Key	n B	The Public Key used to validate the signature.
Authentication Data	n B	Optional additional data to be included in the MAC calculation.

EZ Response (to EY Command) Structure

Output Field Name	Format	Description
Response Code	2 A	"EZ"
Error Code	2 A	"00" indicates No Error. Other codes identify the cause of a signature verification failure.

Host Command Examples

Examples of the host commands used for general data protection are available in the document *payShield 9000 Host Command Examples*.

Chapter 2 – Remote Key Loading

Introduction

Encryption is used universally to protect sensitive data being transferred between a host payment application and a remote terminal device, such as an ATM (Automated Teller Machine) or POS (Point of Sale) terminal. In order for the encryption to work it is necessary to securely install the encryption keys on the terminal device (with the corresponding keys also being available to the host application), and to have a secure method of updating the keys both at the terminal and at the host application.

The traditional way of updating keys at an ATM was to create a key component for each of two members of staff who would visit the ATM and enter their components to allow the ATM to form the key. This was very expensive in terms of staff time, and required an infrastructure to securely generate and manage the key components. Often the keys loaded were not unique to each ATM, and while this made key management easier it did not meet the requirements of the card schemes.

The problems with this approach make it completely impracticable for managing POS terminals because of the large population of such devices and their relatively low cost.

To address these problems, operators of terminals or the host systems they connect to (or independent service providers) have developed systems to allow updated keys to be loaded from a central key loading facility (KLF) down to remote terminals. The designs of these systems vary, and may be proprietary - but they generally have some common characteristics.

This chapter considers a typical approach to securely implement remote key loading (RKL) and how the Thales payShield 9000 HSM can be used to provide security.

Background to terminal key management

Terminal devices generally use 3 types of working key to protect data flowing between the terminal and the host payment application that it connects to:

- TPK - Terminal PIN Key to encrypt PINs (in the format of a PIN Block)
- TAK - Terminal Authentication Key, for authentication data.
- TEK - Terminal Encryption Key, to encrypt other data.

These keys are currently TDES (Triple DES) symmetric keys, with a likely move to AES in the future.

These keys would have to be loaded when the ATM was first deployed, and subsequently re-loaded if the existing keys were compromised or as part of refresh based on the age or volume of use of the existing keys.

In some early ATM systems, all of these keys were loaded manually onto the ATM. In order to reduce the amount of effort, Master Session Key Management was introduced using a TMK - Terminal Master Key. Here the idea is that only the TMK has to be loaded manually on the ATM, and that the working keys can be loaded by sending them encrypted using the TMK from a central Key Loading Facility (KLF).

The TMK will need to be re-loaded in the event of a compromise of the key or as part of a planned refresh. Some implementations have allowed a new TMK to be sent to the ATM by encrypting it with the old TMK that it is about to replace. However, this approach has some security vulnerabilities, cannot be used where the old TMK has been compromised, and does not meet the requirements of ISO and ANSI standards.

To avoid the danger that the compromise of a TMK at one ATM could lead to attacks on other ATMs using the same TMK, the use of unique TMKs became mandated.

The key management problem for ATMs can therefore be condensed down to a consideration of how a unique TMK can be initially loaded on the ATM and subsequently refreshed. The initial loading may be achieved by injecting the keys before delivery of the terminal to its operational site or by remote loading to the terminal after its delivery (but before it is put into operation).

For POS terminals, the key management issue boils down to the same consideration. Many POS terminals use DUKPT (Derived Unique Key Per Transaction) key management, whereby the terminal and host systems independently derive the same unique key for each transaction. In this case, working keys (TPK, TAK, TEK) do not need to be loaded on to the terminal, and there is therefore no TMK. However, the terminal must be provided with an Initial Key (originally referred to as an IPEK - Initial PIN Encryption Key) from which the unique transaction keys are derived, with the IPEK being updated as required.

So in a DUKPT environment, the core issue is the same as in a Master Session Key environment - except that it is an IPEK that has to be loaded instead of a TMK.

See chapter 4 for of the *payShield 9000 Host Programmer's Manual* more information about DUKPT.

The traditional way - and its problems

The way that updated keys have traditionally been loaded onto ATMs has been to have at least two people visit each ATM whenever a new key has to be loaded. In outline, this process requires the following steps:

- Generation of at least 2 key components for each ATM. No person may have access to more than one component. There must be a secure way to store the components until they are presented to their custodian.
- Each component for an ATM must be provided to a different custodian, with security procedures in place to ensure that no one else can see the component.
- All custodians visit each HSM, and enter their component for that ATM. The ATM will generate the key from the components. The custodians must then securely dispose of their components.
- The key that was generated by the ATM from the components must also be created in encrypted form for loading into the host system's key database.
- The host application will verify the new key at the HSM and activate it.

This is clearly a labour-intensive, expensive, and slow process, costing up to $500 per ATM. This approach becomes even more unsustainable for POS terminals because of their greater population and lower cost.

There is also the danger that the long, random strings of digits in the key components will be incorrectly entered into the terminal.

Today's method - using PKI

Modern Remote Key Loading schemes vary in their design, but most of them use RSA-based PKI technology to load a TMK or DUKPT Initial Key to the terminal. (PKI could also be used to load working keys such as a TPK or TEK, but this is generally not done because of the performance impact of using RSA keys.)

This document describes a generic process for remote key loading using RSA-based PKI, and how the payShield 9000 can be used to accomplish this securely. Although the actual process design will vary with different manufacturers and different KLFs (Key Loading Facilities), the payShield 9000 capabilities described here will generally satisfy all the cryptographic needs of the process.

The *Remote Key Transport White Paper* is a document available from various web sites which describes how Diebold, Inc. (a manufacturer of ATMs) uses this approach for loading TMKs (or Master Terminal Keys in Diebold terminology) to ATMs.

The principle PKI is not described here, as it is adequately described in a wide range of publicly available material.

Outline of loading keys using PKI

Typically the KLF (the organisation that loads the working keys onto the terminal and processes transactions from the terminal) is different to the original manufacturer of the terminal. There are therefore three stages in establishing a secure key loading process:

1. Initializing the trusted environment between the KLF and terminal, using the terminal manufacturer as a trusted CA (Certificate Authority).
2. Secure loading of TMKs or DUKPT Initial Keys (IK) to the terminals.
3. Secure loading of working keys to the terminals (not required for DUKPT).

The initialization process works by the terminal manufacturer:

- loading a unique RSA key pair and the manufacturer's public key into each terminal and providing the terminal public keys to the KLF.
- providing the manufacturer's public key to the KLF.
- acting as a CA to certify all the public keys used in the process.

The TMK or IK loading process will be used infrequently, and consists of:

- the KLF and terminal exchanging their public keys certified by the manufacturer, and each verifying the other's public key.
- the KLF generating a TMK/IK, encrypting it with the terminal's public key, and sending it to the terminal. The terminal recovers the TMK/IK by decrypting it using its secret key.
- The KLF may also sign the message including the TMK/IK with its private key so that the terminal can use the KLF's public key to verify the authenticity and integrity of the message.

Loading of working keys (TPK, TEK, TAK) will be a more frequent activity, and does not in itself make use of PKI. This is only required where a TMK was loaded to the terminal (as working keys are generated within the terminal when DUKPT is being used), and the process is:

- the KLF generates unique working keys for each terminal, encrypts them using the terminal's TMK, and sends them to the terminal.
- the terminal recovers the working keys by decrypting them using the TMK.

Where DUKPT key management is being used, the method to load the IK can be modified. Instead of loading an IK as indicated above, a TMK could be loaded to the terminal (using the method described above) and the IK could subsequently be loaded in the way described for the loading of working keys - i.e. by encrypting it using the TMK.

Detailed Description

Conventions used below

In the detailed process description below, keys and other data items relevant to RSA are identified using the following convention:

$$X_{Owner}^{Validator}$$

Where:

X is the data type:

- SK = Secret (a.k.a. Private) Key
- PK = Public Key
- UID = Unique Identifier

Owner is the owner of the key:

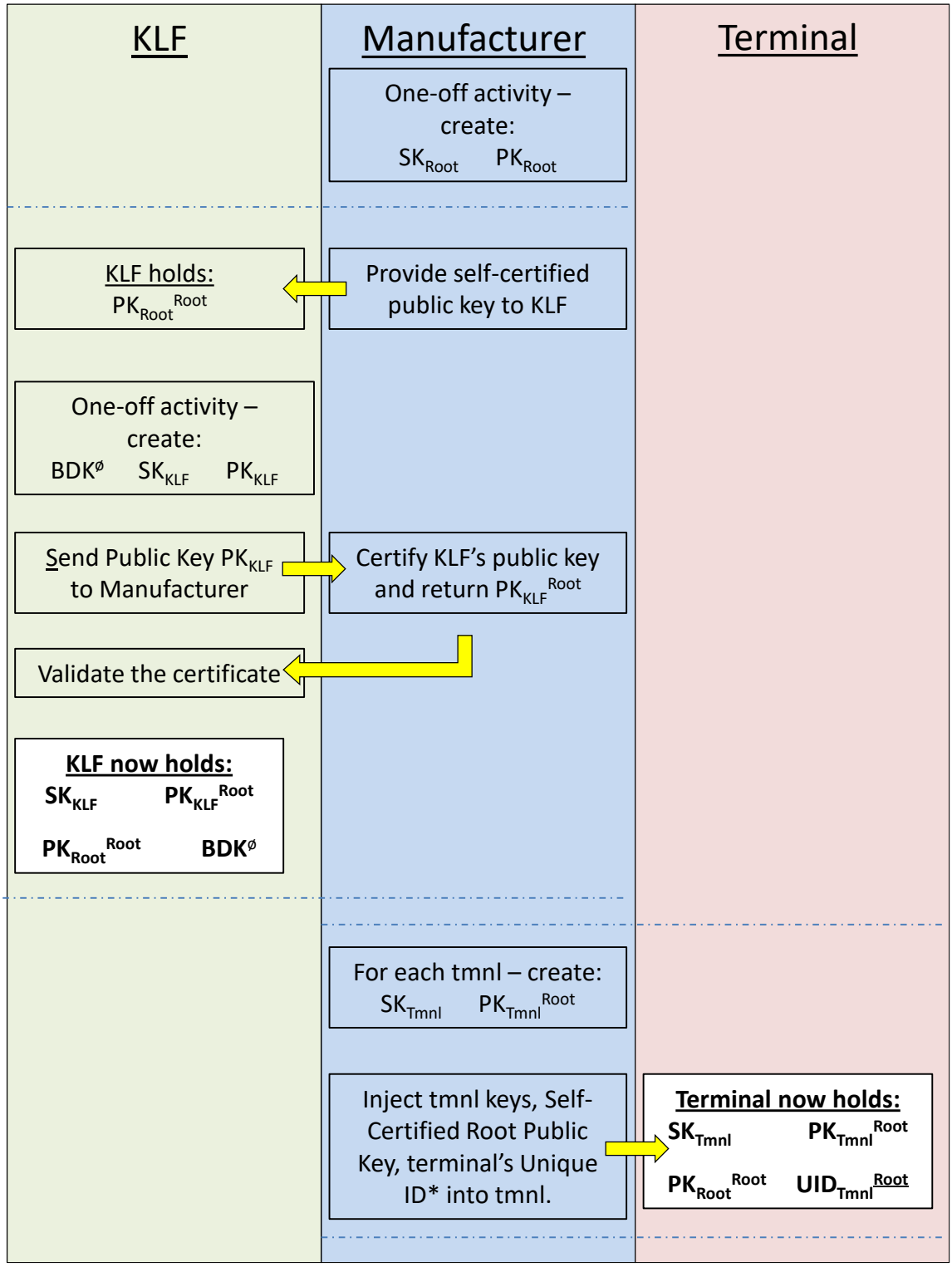
- Root = terminal manufacturer
- Tmnl = Terminal device
- KLF = Key loading facility

Validator indicates data is certified or signed:

- Root = certified using the manufacturer's secret key.
- Root = signed using the manufacturer's secret key.
- Tmnl = certified using the terminal's secret key

Initialization

The Process



* Use of a Unique Terminal ID (UID) is not universal, but is used by some terminal manufacturers. In the case of DUKPT, a UID in the form of the KSN (Key Sequence Number) is required to calculate the IK.

ø A BDK is needed only if DUKPT is being used.

Role of the payShield 9000

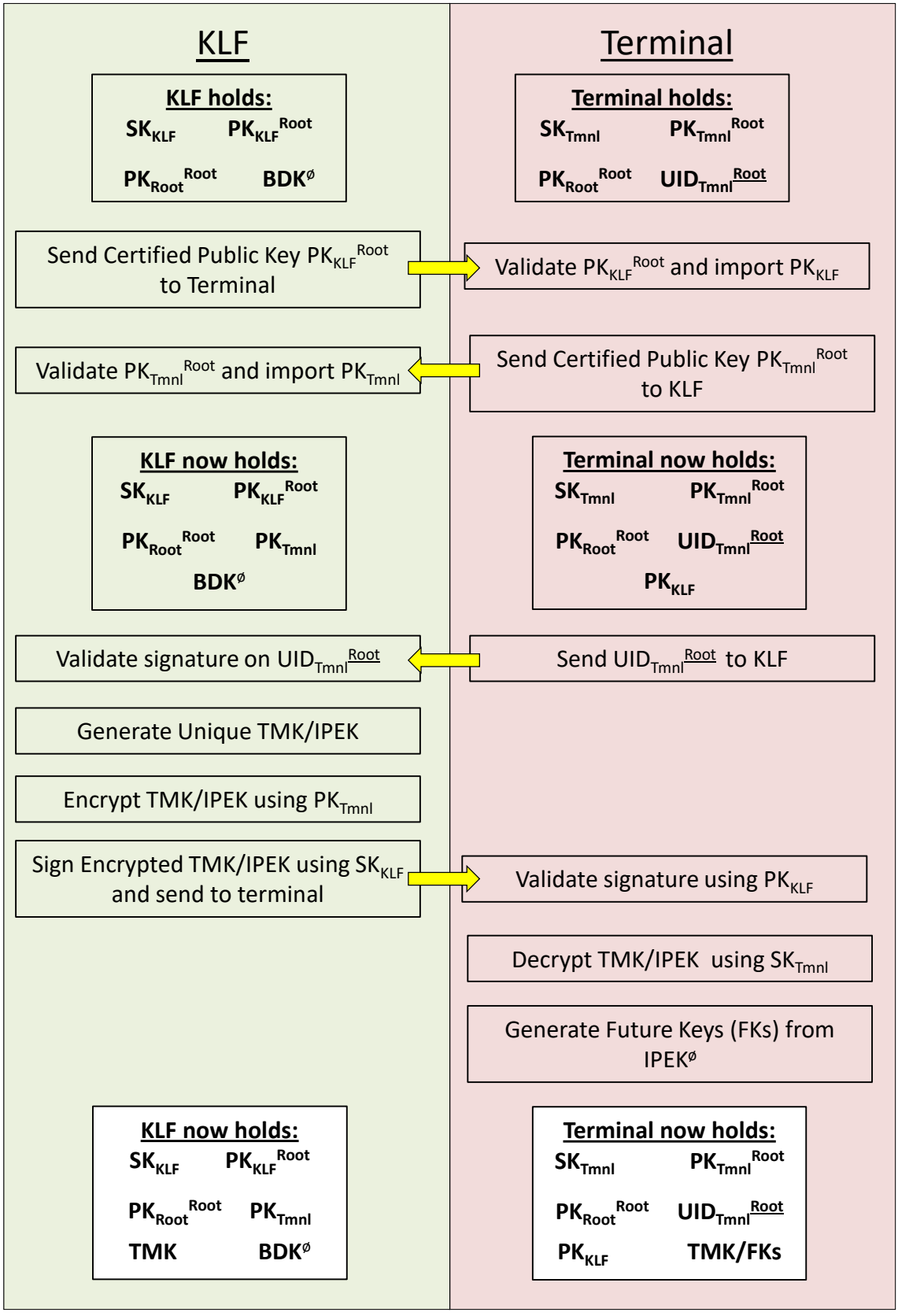
The payShield 9000 can be used by both the KLF and the Manufacturer to provide the cryptographic functions for this purpose. In reality, the Manufacturer is less likely to use a payShield 9000 for this because they do not have the need for the other specialised payments functionality offered by this type of HSM - instead they are more likely to use a "general purpose" HSM such as the Thales Luna HSM range.

The payShield 9000 host commands that would be used here are:

payShield 9000 Host Command	
ID	**Command Use**
EI	Generate an RSA key pair (i.e. SK_{KLF} & PK_{KLF}; SK_{Root} & PK_{Root} ; SK_{Tmnl} & PK_{Tmnl})
ES	Validate the certificate in PK_{KLF}^{Root}, using PK_{Root}
A0	Generate a BDK.

Loading the TMK/IK to the terminal

The Process



ø Only if DUKPT is being used.

Role of the payShield 9000

The payShield 9000 provides the required cryptographic functions to the KLF through the following host commands:

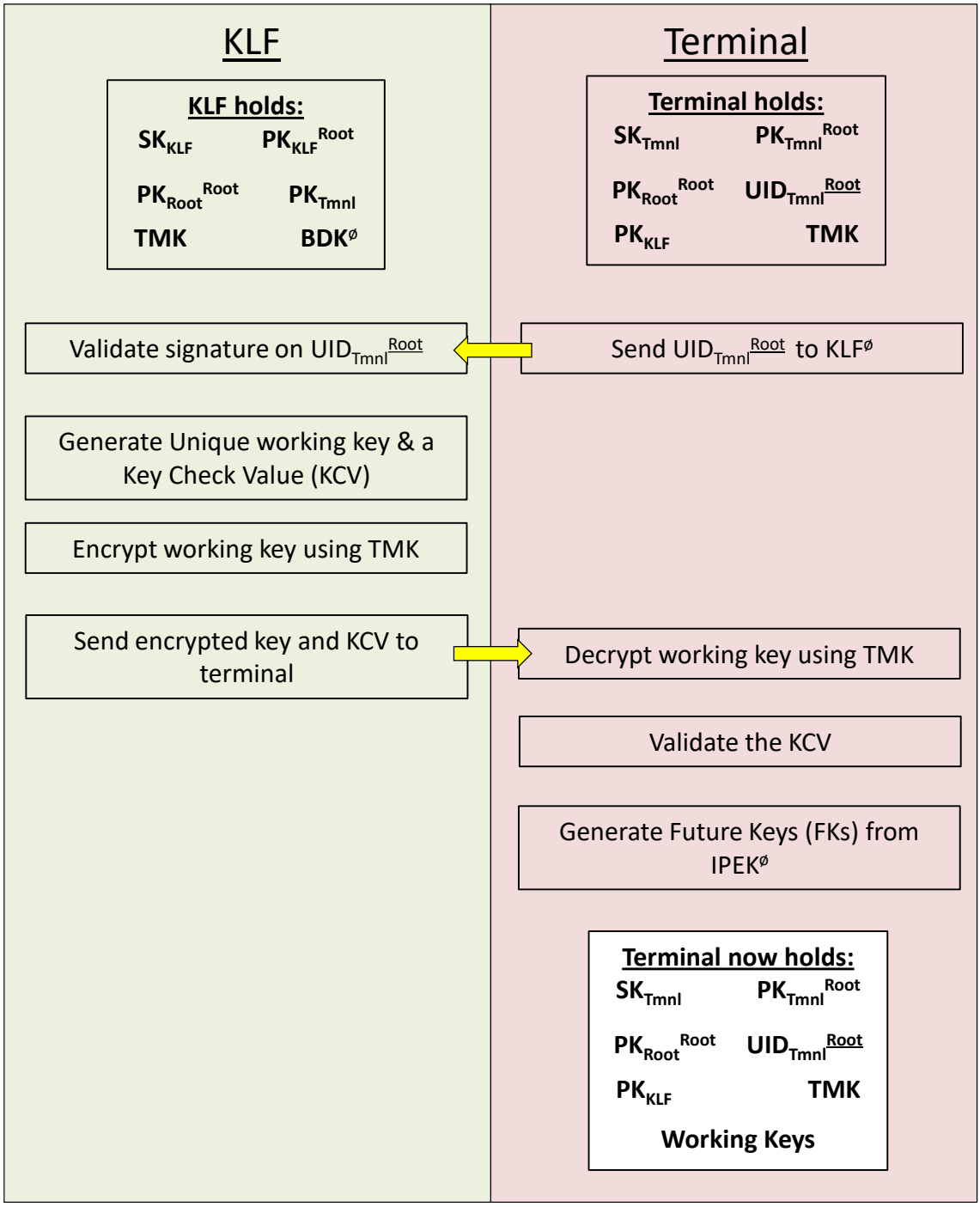
payShield 9000 Host Command	
ID	**Command Use**
ES	Validate the certificate in PK_{Tmnl}^{Root}, using PK_{Root}, and import PK_{Tmnl}.
EW	Generate signature on encrypted TMK/IPEK
EY	Validate the signature on UID_{Tmnl}^{Root}, using PK_{Root}
A0	Generate a TMK/IPEK, encrypted under the payShield 9000 LMK
GK	Take the TMK/IPEK encrypted under the LMK and re-encrypt it under PK_{Tmnl}, and sign it using SK_{KLF}.

Loading working keys to the terminal

Where Master/Session key management is being used, this process is used to load TPKs, TEKs, and TAKs - either when the terminal is first put into use, or subsequently to refresh the keys.

This method can also be used to load the IK where DUKPT key management is implemented.

The Process



ø Only if this method is being used to load an IK

Role of the payShield 9000

The payShield 9000 provides the required cryptographic functions to the KLF through the following host commands:

payShield 9000 Host Command	
ID	**Command Use**
EY	Validate the signature on UID_{Tmnl}^{Root}, using PK_{Root}
A0	Generate a TPK, TEK, TAK, or IPEK, encrypted under the payShield 9000 LMK and under the terminal's TMK. This command also generates the Key Check Value.

Chapter 3 – Online PIN delivery

Introduction

One of the problems facing card issuers is how to securely deliver PINs to their customers when a new card is issued. Traditionally this has been done by printing tamper-evident PIN mailers, which prevent the PIN from being read unless the envelope is opened, and sending them to the customer by the postal service. The payShield 9000 includes host commands to facilitate the printing of PIN mailers, and the process is described in Chapter 3 of the *payShield 9000 Host Programmer's manual*.

More recently, some card issuers have developed online PIN delivery systems, which push PINs to the customer. The PIN is generated by the card issuer, and a secure method is provided for the customer to access the PIN online. Thales payment HSMs are in use by card issuers who have implemented online PIN delivery systems.

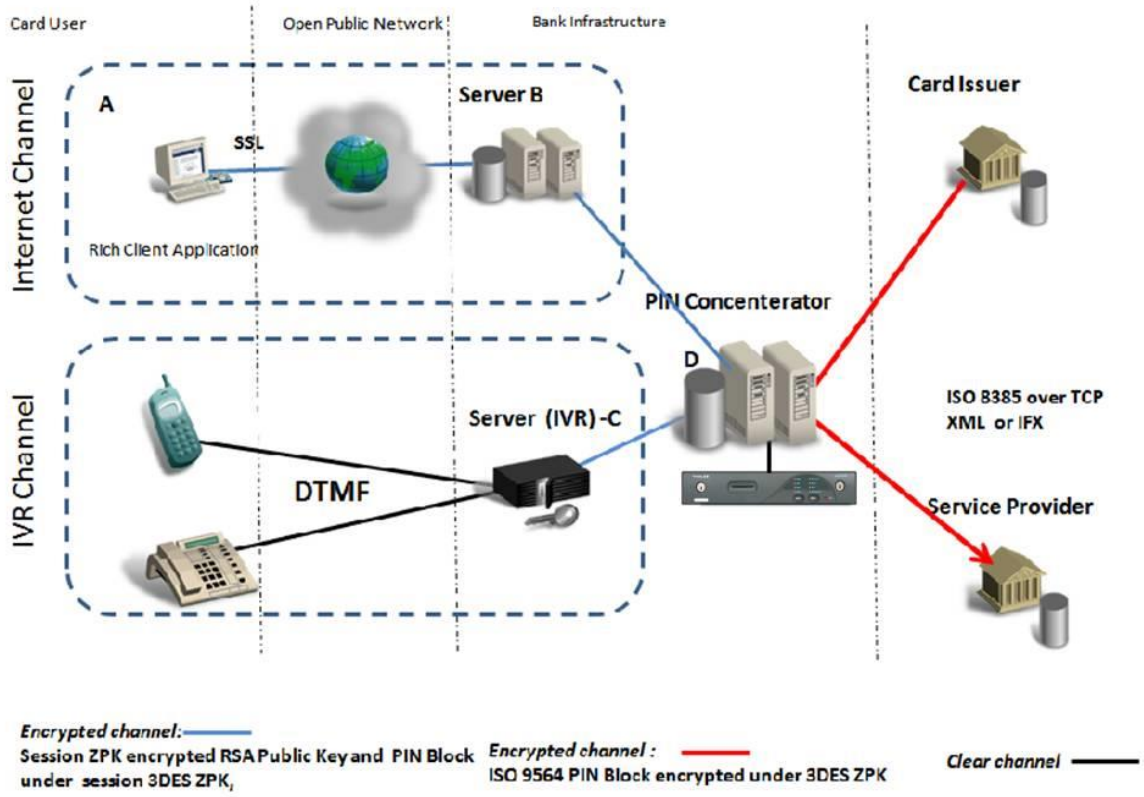
A third approach is to effectively by-pass the need to deliver PINs by providing a secure mechanism for customers to set their PINs when a new card is issued, or subsequently to change their PIN. This approach also mitigates the need to have a process to retain PINs when a new PAN is allocated (e.g. when a customer's lost card is replaced) because the customer can just set the same PIN as for their old card when the new card is issued.

This chapter outlines one possible system architecture for this approach and shows how the payShield 9000 can be used to provide the secure cryptography to support it. It also outlines how customers without Internet access can be supported by being offered access using IVR (Interactive Voice Response).

Overview of the process

1. The customer uses their browser to set up an SSL session to their account on the bank's web site.
2. The customer downloads a rich client application (RCA).
3. The card issuer provides the RCA with a RSA public session key.
4. The RCA invites the client to enter the PIN.
5. The RCA generates a symmetric session ZPK.
6. The RCA generates a PIN Block encrypted under the session ZPK. Thales PIN Block Format 05 (ISO format 1) is used. This format does not make use of the PAN, which is not used at all in this process: a randomized reference number is provided by the bank which the bank can use to identify the PAN that the new PIN refers to.
7. The RCA sends the PIN Block to the bank.
8. The RCA sends the session ZPK encrypted under the RSA public session key to the bank.
9. The bank system decrypts the session ZPK using the RSA private session key.
10. The bank system translates the PIN Block to Thales PIN Block format 01 (ISO format 0). This format includes the PAN.
11. The new PIN Block is included in the data provided to the card issuer back-office function and the service provider who manufactures and delivers the card.

Architecture



In this section we will consider just the Internet Channel in the above diagram. The IVR channel is discussed later in this document.

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.

The next example is the *CW* host command, used to generate a Visa Card Verification Value (CVV). In this case, no additional fields are required.

Functions of the Rich Client Application (RCA) – A

1. Provision of user interface and user-facing functionality.
2. Communication with Server B.
3. Generation of session ZPK.
4. Downloading the RSA public session key from the PIN concentrator (D) RSA. This key would typically be encoded using ASN.I DER with 2's complement representation.
5. Downloading other data required by this process. Note that the PAN is not provided.
6. Receiving PIN entry from the customer.
7. Generation from the entered PIN a PIN Block using format "05" and encrypting it using the session ZPK.
8. Encryption of the session ZPK via the RSA Public Session Key.
9. Transmission of these cryptograms and other data to Server B.

Functions of Server B

1. Communication with Rich Client Application (A).
2. Downloading of the RSA public session key from the PIN concentrator (D).
3. Receipt of the cryptograms from the Rich Client Application (A):
   a. PIN Block encrypted under the ZPK session key.
   b. Session ZPK encrypted under the RSA public session key.
4. Communication with the PIN Concentrator (D)

Functions of the PIN Concentrator (D)

1. Communication with the payShield 9000 HSM.
2. Generation of the RSA session key set and storage in the key database, with relevant key length in ASN.I DER PKCS#1 format. The RSA session key set is generated encoded as ASN.I DER with 2's complement representation. The RSA public session key is key type 00D (i.e. encrypted using LMK pair 36-37) and the RSA private session key is key type 00C (i.e. encrypted using LMK pair 34-35): these RSA keys are never available in the clear.
3. Providing the encrypted RSA private session key to the payShield 9000.
4. Importing the session ZPK (under which the received PIN Block is encrypted), which is encrypted with the RSA public session key, as key type 001 (i.e. encrypted by the payShield 9000 using LMK key pair 06-07).
5. Translation of the received PIN Block (which is in Thales PIN Block format 05 and is encrypted under the session ZPK key received from server B) to a PIN Block which can be used by the card issuer back-office and card production service provider (using Thales PIN Block format 01 and encrypted under the agreed ZPK).

payShield 9000 functions used

This section considers which payShield 9000 Host Commands are used to provide the secure cryptographic functions used in the above process.

RSA session key set generation

Example:

EI Command:

```
HSM1EI1102402
      HSM1              Header
      EI                Command Code
      1                 Key Type indicator (Key management only)
      1024              Modulus length in bits
      02                DER encoding for ASN.1 Public Key (INTEGER
                        uses 2's complement representation).
```

EJ Response:

```
HSM1EJ0030818902818100aaa…aaa020301000130333434bbb…bbb
      HSM1              Header
      EJ                Response Code
      00                Error code (00 = No error)
      308189028181*     DER encoding of public key (length129 bytes,
                        first byte always zero)
```

*00aaa…aaa *	*Public key129 bytes (258 Hex digits), beginning with '00'*
*0203 *	*DER encoding of exponent*
*010001 *	*Exponent (65537)*
0344	*Length of private key (344 bytes)*
bbb…bbb	*The Private Key (344 bytes, 688 Hex digits), encrypted under the LMK.*

* Sent as binary

Load RSA private session key (to decrypt the received ZPK)

This loads the RSA private session key from the key database into the payShield 9000 so that the session ZPK received from the RCA can be decrypted. This is done using the *EK* Host Command. The key passed to the payShield 9000 is encrypted under LMK key pair 34-35.

Example:

EK Command:

HSM3EK000344ccc…ccc

HSM3	*Header*
EK	*Command Code*
00	*Index number for the Private Key to be stored*
0344	*Private Key Length (344 bytes)*
ccc…ccc	*Private key encrypted under LMK pair 34-35 (344 bytes, 688 Hex digits)*

EL Response:

HSM3EL00

HSM3	*Header*
EL	*Response Code*
00	*Error code (00 = No error)*

Translate session ZPK encrypted under RSA public session key to encryption under LMK

The *GI* command is used to recover the session ZPK sent from the RCA encrypted under the RSA public session key and re-encrypt it using the appropriate LMK variant encrypted under LMK key pair 34-35.

Example:

GI Command:

HSM9GI010106000128ddd…ddd;00;0U0

HSM9	*Header*
GI	*Command Code*
01	*Identifier of algorithm used to decrypt the key. (01 = RSA)*
01	*Identifier of the Pad Mode used in the encryption process. (01 = PKCS#1 v1.5 method (EME-PKCS1-v1_5)).*
06	*LMK key pair that key is to be encrypted under (06 = LMK pair 06-07)*
00	*LMK key pair variant that key is to be encrypted under.*
0128	*Length (in bytes) of Data Block field.*

*ddd…ddd **	*ZPK encrypted under the RSA public session key (128 bytes, 256 Hex digits).*
;	*Delimiter*
00	*Index of stored RSA private key. (See example above of loading RSA private key.)*
;	*Delimiter*
0	*Ignored.*
U	*Key Scheme*
0	*KCV calculation method (0 = 16-digit KCV).*

GJ Response:

```
HSM9GJ0003FF053D8BBF3F55Ueee...eeefff…fff
```

HSM9	*Header*
GJ	*Response Code*
00	*Error code (00 = No error)*
*03FF…3F55 **	*Initialization Value*
U	*Key Scheme*
*eee...eee **	*ZPK encrypted under LMK pair 06-07.*
*fff...fff **	*KCV*

* Sent as binary

Translate PIN Block for transmission to card issuer or service provider.

The *CC* command is used to translate the PIN Block received from the RCA (in Thales PIN Block format 05, encrypted using the session ZPK) to a PIN Block that is sent to the card issuance service provider (in Thales PIN Block format 01, encrypted using the ZPK agreed with the service provider). The output PIN Block includes the PAN.

Example:

CC Command:

```
HSM5CCUggg...gggUhhh…hhh12829FE22EE6BA56EB0501201006062222
```

HSM5	*Header*
CC	*Command Code*
U	*Key scheme for the ZPK received from the RCA.*
ggg...ggg	*ZPK received from the RCA, encrypted under LMK pair 06-07..*
U	*Key scheme for the ZPK agreed with the service provider.*
hhh…hhh	*ZPK agreed with service provider, encrypted under LMK pair 06-07.*
12	*Maximum PIN length*
829F…56EB	*PIN Block received from the RCA, encrypted under session ZPK*
05	*Format of PIN Block received from RCA.*
01	*Format of PIN Block to be sent to service provider.*
201006062222	*Account Number – part of the PAN.*

CD Response:

```
HSM5CD000415F9A4DB0FAA118501
```

HSM5	*Header*
CD	*Response Code*
00	*Error code (00 = No error)*

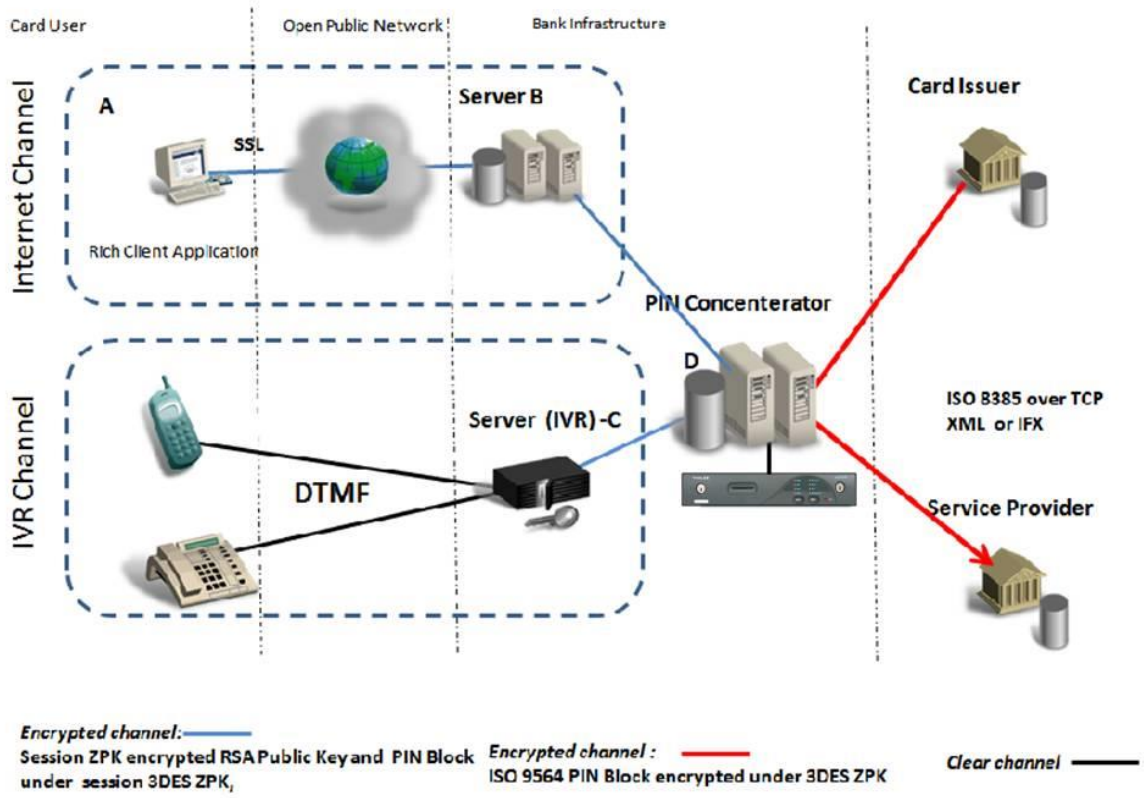
04	PIN Length
15F9…1185	PIN Block to be sent to service provider, encrypted under ZPK agreed with service provider.
01	Format of PIN Block to be sent to service provider.

Operating without internet access

The preceding sections apply where the customer has Internet access to the bank's web site. But how can customers without Internet access be accommodated?

An obvious fallback position is to send them PIN mailers in the traditional way. But the architecture we have described above can be extended to enable any customer with access to a 'phone to set their PIN. This is done by making use of IVR (Interactive Voice Response) technology.

Here is the architecture diagram that we discussed previously:



In the IVR channel, we see that the customer makes a phone connection to the bank's IVR Server (C).

Functions of the IVR system

In essence, the IVR system fulfills the functions of the Rich Client and Server B in the Electronic channel:

1. Providing a user interface to customer.
2. Accepting and translating voice or keypad data input by the customer.

3. Passing customer input data to Server C.
4. Communication with the PIN Concentrator (D),
5. Generation of session ZPK,
6. Downloading the RSA public session key from the PIN concentrator (D). This key is encoded with ASN I DER with 2's complement representation.
7. Downloading the entered PIN from the IVR system.
8. Generating a PIN Block using Thales Format 05, encrypted under the session ZPK.
9. Encrypting the session ZPK under the RSA public session key.
10. Sending the cryptograms to the PIN Concentrator (D):
    a. Session ZPK encrypted under the RSA public session key,
    b. The PIN Block in format 05 encrypted under the session ZPK.

Security of the process

Use of the payShield 9000

All of the sensitive cryptographic operations are protected within the payShield 9000, connected to the PIN Concentrator (D).

Server B does not perform any cryptographic functions other than managing the SSL session with the RCA – it is essentially a communications switch that does not need to process any of the data passing through it. As a result, it does not present any security vulnerabilities and does not need to make use of an HSM.

Server C fulfills similar tasks to Server B. In addition, it performs the same cryptographic functions as the RCA – see next section. Again, there is no need for Server C to make use of an HSM.

Cryptographic functions of the RCA & Server C

The RCA and Server C perform the following cryptographic functions in software:

- Generation of the session ZPK.
- Encryption of the session ZPK under the RSA public session key provided by the PIN Concentrator (D).

The session ZPK is used for transmission of a single PIN Block and then not used again. Therefore any compromise of the ZPK does not present a threat to any subsequent transactions. There is therefore no need to protect the key by using technology like an HSM. However, it is important that the key generation process incorporates an effective random number generator to ensure that the ZPKs are not predictable and so a payShield 9000 could be to good use in generating the ZPK.

The RSA public key is also a session key, and has no value in attempting to mount any subsequent attacks (e.g. by changing the PIN to a value known by the attacker.)

Use of Thales PIN Block Format 05

Thales Format 05 (ISO Format 1) is used for the PIN Block created at the RCA because it does not require any knowledge of the PAN.

Format 05 is generally viewed as a weaker format, precisely because it does not incorporate the PAN. It therefore enables codebook attacks where the attacker builds a table of known PINs against their encrypted values, enabling the encrypted PIN of another user to be decrypted. This attack works in an environment where the same PIN key is used for a large number of transactions.

In the process described here, the ZPK used to encrypt the PIN is used only once, and therefore the codebook attack cannot be used and the perceived weakness of PIN Block Format 05 does not apply.

Avoiding use of the PAN

As has been mentioned, the PAN is not involved in the customer-facing portion of the process – i.e. it is not disclosed anywhere to the left of the PIN Concentrator in the diagram used previously in this chapter.

This is an important aspect of security, as even if the PIN is compromised there is no way of associating it with a card.

Because no PAN is involved in the process to the left of the PIN Concentrator, this part of the process is out-of-scope as far as PCI DSS is concerned. (*From the PIN Concentrator to the right, access to PANs is required, and so this part of the system will need to meet the needs of PCI DSS audits*.)

Chapter 4 – Point-to-Point Encryption & mPOS

Introduction

Point-to-Point Encryption (P2PE) encrypts cardholder data sent from the merchant to the acquirer/processor/gateway: The document *Point-to-Point Encryption - Solution Requirements and Testing Procedures: Encryption, Decryption, and Key Management within Secure Cryptographic Devices (Hardware/Hardware)* published by PCI SSC indicates that a P2PE solution "*provides a method for … merchants to reduce the scope of their PCI DSS assessments when using a validated P2PE solution for account data acceptance and processing*".

A particular application of P2PE is to enable the implementation of Mobile Point-of-Sale (mPOS) - also referred to as Mobile Acceptance. This allows merchants to use standard mobile communications equipment (such as a smartphone or tablet PC) with an approved card reader and PIN entry device to accept card payments.

This chapter describes how the payShield 9000 can be used as part of P2PE and mPOS systems to provide security and help the systems meet the required security standards. It considers the management of keys and processing of transactions up to the point where a transaction enters the established payments network, e.g. at an acquirer. From that point on, the payShield 9000 fulfils its traditional role of providing security for payment transactions.

Point-to-Point Encryption (P2PE)

During card payment processing, traditionally only some of the transaction data, such as the PIN, has been protected cryptographically. P2PE is a technology area designed to protect other cardholder data (such as PANs, CVVs, and expiry dates) in the merchant space by encrypting the data when entered at the Point of Interaction (where the customer interacts with the POS (Point of Sale) terminal) on the merchant premises and subsequently transmitted to a P2PE Solutions Provider.

One of the drivers behind the adoption of P2PE is that encryption of the PAN takes it out of scope in terms of PCI DSS assessment and thereby reduces the cost of compliance. The PCI DSS standard imposes requirements on securing cardholder data that is being held: P2PE encrypts data at the point of its acceptance by the terminal, and so avoids the merchant being involved in protecting the data because the merchant has no access to the clear text PAN or the cryptographic keys needed to decrypt the data.

In addition, the PCI DSS standard includes the following requirement:

> *4.1 Use strong cryptography and security protocols (for example, SSL/TLS, IPSEC, SSH, etc.) to safeguard sensitive cardholder data during transmission over open, public networks.*
>
> *Examples of open, public networks that are in scope of the PCI DSS include but are not limited to:*
>
> - *The Internet*

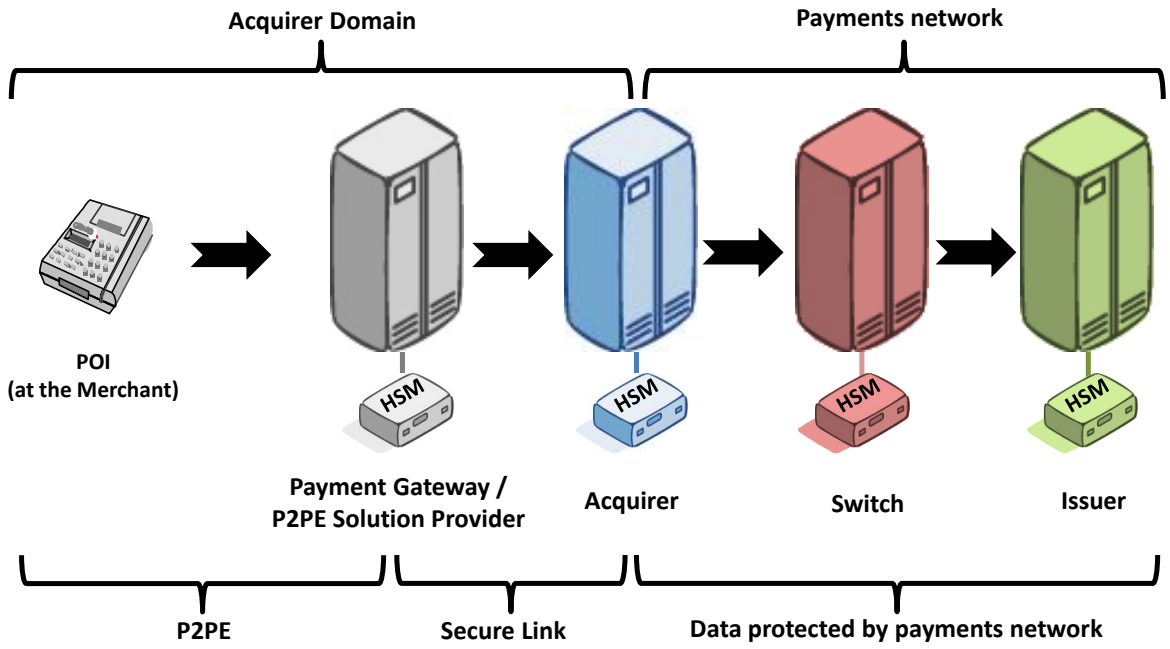
- *Wireless technologies,*
- *Global System for Mobile communications (GSM)*
- *General Packet Radio Service (GPRS).*

Sensitive cardholder data includes any data held on the card, including the PAN.

P2PE satisfies the PCI DSS requirement 4.1.

Local Impact of P2PE

P2PE covers only the merchant and the P2PE Solution Provider. The P2PE Solution Provider may be the merchant's Acquirer, the merchant's payment processing system, or a third party service provider (sometimes referred to as a P2PE Solution Provider or a Payment Gateway) operating between the merchant and acquirer. The Acquirer reformats the transaction and submits it to the standard payments network for processing.



POI (Point of Interaction) refers to the components of the Point of Sale system where the payment card is read and the PIN is entered.

The P2PE Solution Provider will operate the agreed P2PE technology with the merchant: this will format data and transactions as required by the P2PE solution, but when the transaction is passed into the standard payments network it will conform to the standard message formats and protocols required by the payments system. (This chapter does not discuss the handling of the transaction in the payments network.)

Because the use of P2PE is localised to the merchant-P2PE Solution Provider relationship, it does not need to interact with other parts of the payments system. There is therefore no defined way in which P2PE has to operate - although the deployed system will need to meet the security requirements as defined by PCI SSC.

As a result, there is a variety of P2PE solutions based on different products, technologies, and architectures. This chapter therefore outlines the facilities in the payShield 9000

which can be used generically in P2PE solutions, and does not attempt to describe the definitive use of the HSM for any particular P2PE solution.

What Does a P2PE Solution Involve?

Both the merchant and the Payment Gateway or P2PE Solution Provider must use the same P2PE solution, with the merchant being a client of the service and using the terminal devices and procedures required by the solution.

The P2PE solution encompasses:

1. Injection of P2PE keys into the POI device.
2. Loading of keys into the POI device while it is in place at the merchant site.
3. Encryption of the cardholder data at the POI device and subsequent decryption at the P2PE Solution Provider.
4. Transfer of data to the POI device while it is in situ (e.g. software updates).

These tasks are described later in this chapter, together with explanations of how the payShield 9000 can be used to build a secure solution that meets the required security standards.

Security Standards for P2PE

PCI has published two sets of standards for security requirements for P2PE Solutions

1. Hardware/Hardware solutions - *Point-to-Point Encryption - Solution Requirements and Testing Procedures: Encryption, Decryption, and Key Management within Secure Cryptographic Devices (Hardware/Hardware),* and
2. Hardware/Hybrid solutions - *PCI Point-to-Point Encryption - Solution Requirements and Testing Procedures: Encryption and Key Management within Secure Cryptographic Devices, and Decryption of Account Data in Software (Hardware/Hybrid)*.

These standards do not remove the need for compliance with other standards from PCI (including PCI DSS and PCI PA-DSS), although use of solutions complying with the above standards can reduce the scope of PCI DSS assessment.

The requirements in these documents are defined in 6 domains:

- Domain 1 - Encryption Device Management: use secure encryption devices and protect devices from tampering.
- Domain 2 - Application Security: secure applications in the P2PE environment.
- Domain 3 - Encryption Environment: secure environments where POI devices are present.
- Domain 4 - Segmentation between Encryption and Decryption Environments: segregate duties and functions between encryption and decryption environments.
- Domain 5 - Decryption Environment and Device Management: secure decryption environments and decryption devices.

- Domain 6 - P2PE Cryptographic Key Operations: use strong cryptographic keys and secure key-management functions.

All of these domains need to be considered by P2PE solution providers, but in the context of the payShield 9000 it is Domains 5 and 6 that particularly concern us. These cover the payment gateway or Acquirer where the terminals' encrypted transactions are decrypted, and where the payShield 9000 is deployed. Both types of solution require the use of PCI HSM-approved HSMs.

Hardware/Hardware Solutions

PCI P2PE-compliant solutions must meet the security requirements defined in the PCI standard for hardware/hardware solutions.

This standard defines requirements for all aspects of a P2PE solution where both encryption (at the merchant) and decryption (at the P2PE solution provider) are performed in secure cryptographic devices such as an HSM. The diagram on page 16 (of version 1.1 of the requirements) identifies the security requirements that apply to the various elements of the solution. Of particular relevance to this Application Note is Domain 5 covering the Solution Provider's Secure Decryption Environment, which requires the use of an HSM for key management and cryptography which is FIPS 140-2 or PCI HSM approved.

Domain 6 covering Cryptographic Key Operations requires the use of secure methods for encryption, generation, distribution, usage, and administration of keys. Use of the payShield 9000 greatly eases the effort required to meet these requirements.

The payShield 9000 offers PCI HSM approved versions which can therefore be used to build and secure P2PE solutions.

Hardware/Hybrid Solutions

In December 2012 PCI published the standard for hardware/hybrid solutions. In this environment, the encryption at the merchant site is still performed in hardware, but decryption at the P2PE solution provider uses a hybrid of software and secure cryptographic devices (i.e. HSMs).

This architecture is described in the following terms:

> *The solution provider's decryption environment may consist of multiple Host Systems in one or more locations. ... Where a Host System is comprised of multiple hardware components … the connectivity between this hardware must be made through physical connections rather than using a network connection. Alternatively, the Host System may comprise of a partition on a mainframe computing system.*
>
> *The Host System is connected to one or more HSMs to securely protect the data-decryption keys (DDKs) when not in use. The HSM(s) is a fundamental component of a hardware/hybrid P2PE solution; however, unlike hardware/hardware solutions, the decryption of account data is performed outside of the HSM on the Host System. When the Host System is required to decrypt encrypted account data received from POI, the DDK is retrieved from a key store protected by the HSM, and then passed to the Host System. The Host System then uses the DDK to decrypt the account in the software of the Host System. The Host System will*

temporarily retain DDKs in volatile memory for the purpose of decrypting account data. When the DDK reaches the end of its cryptoperiod, it will be erased from memory

The Host System and HSM(s) must reside on a network that is dedicated to decryption operations and transaction processing, which may also include services required to support these functions. The decryption network must be segmented from any other network or system that is not performing or supporting decryption operations or transaction processing.

Because sensitive functions are now performed on the host, there are additional security requirements the host system must meet, such as:

- The host system must be dedicated to the P2PE solution
- It must be on a dedicated network.
- The host must ensure that decryption keys are not used beyond their service life (or cryptoperiod).
- Many of the security requirements in Domains 5 and 6 applying to the HSM in a Hardware/Hardware solution become the responsibility of the host system in a Hardware/Hybrid solution.

Identifying approved P2PE solutions.

As P2PE Solutions and Applications become approved by PCI, they are listed on the PCI web site at:

https://www.pcisecuritystandards.org/assessors_and_solutions/point_to_point_encryption_solutions

Mobile Point of Sale, or Mobile Acceptance

Mobile Point of Sale (mPOS), or Mobile Acceptance, is an emerging and high-profile area of technology and products which generally employs P2PE to provide data protection and meet the required security standards, such as PCI DSS relating to protection of cardholder data. It allows commercial off-the-shelf communication devices such as smartphones and tablets in conjunction with approved Point of Interaction (POI) peripherals and applications to be used by merchants as Point of Sale (POS) terminals.

The mPOS terminal typically uses a wireless Internet or mobile 'phone connection to a PCI-approved P2PE solution provider. The P2PE solution provides the security for mPOS transactions, and passes transactions on to Acquirers or, for "sub-merchants" with lower sales volumes, to Payment Service Providers.

What are the attractions of mPOS?

mPOS avoids the need to deploy "traditional" POI terminals which are designed for fixed, wired connection. Compared with these, mPOS offers a lower cost for the terminal and avoids the need for a communications infrastructure at the merchant location other than the Internet or mobile 'phone network.

This is attractive to small merchants whose level of business cannot support the cost of a traditional terminal, or whose nature of business (e.g. a stall holder, or a sole trader who works on their customers' premises) means that they cannot operate with a fixed communications infrastructure. MasterCard has found that of the 1.2 million mPOS solutions deployed in 2010 and 2011, 75% were used by merchants who did not previously accept card payments (see *MasterCard Best Practices for Mobile Point of Sale Acceptance*).

mPOS is also being adopted by merchants who already have a history of using "traditional" POS terminals for card payments. Here the attractions include the ability to add terminals anywhere in the premises (providing more convenient POS locations for their consumers and the ability to reduce queues at tills) and using apps to enhance the consumer experience at the point of sale.

The mPOS Terminal

The core of an mPOS terminal is an off-the-shelf intelligent device with internet or mobile communications capability, such as a smartphone or tablet computer, which is supported by the P2PE Solution Provider. The required application is downloaded onto this device to enable it to function as an mPOS terminal.

A card-reading peripheral must be attached to the device to enable the mobile communications device to interface to the payment card, turning it into an mPOS terminal. This peripheral will be either a Secure Card Reader (SCR) to swipe magnetic stripe cards or a PIN Entry Device (PED) with card reader for EMV cards. Many of these devices utilize PCI SRED (secure reading and exchange of data) technology which encrypts the data inside the device and therefore releases an encrypted data stream to the smart phone or tablet. The advantage of this approach is that the mobile device only ever handles encrypted data and therefore simplifies PCI DSS compliance for the merchant.

All components of the mPOS terminal must be both supported by the P2PE solution provider as part of their approved solution and be approved by PCI. The requirements for PCI approval can be found in PCI's *PIN Transaction Security (PTS) Point of Interaction (POI) Modular Security Requirements*. PCI-approved PEDs and SCRs (not just those for use in mPOS) can be identified from the PCI web site at:

https://www.pcisecuritystandards.org/approved_companies_providers/approved_pin_transaction_security.php#

A printer may also be attached to the mPOS terminal (e.g. by Bluetooth) for receipt printing. Where a printer is not attached, the consumer must agree to receive receipts electronically, such as by e-mail.

mPOS Solutions

Solutions for mPOS (i.e. encompassing the mPOS terminal device, the mPOS application, and the payment gateway) will generally be developed and marketed by a third party solutions provider.

The solution provider will identify which mobile devices can be used with their solution, and will identify (or sell) the card readers that can be used: these card readers must be approved by PCI.

The mPOS solution is built around P2PE technology. For PCI compliance, the P2PE solution must comply with the PCI P2PE standard; the solution provider will issue a P2PE Instruction Manual to tell the user how to safeguard their system.

Security Standards for mPOS

No formal security standards have been published for mPOS yet, but various publications provide security guidance. These publications include:

- PCI: Mobile Payment Acceptance Security - Accepting Mobile Payments with a Smartphone or Tablet ("At a Glance" document series)
- PCI: Mobile Payment Acceptance Security Guidelines for Developers
- Mastercard: Best Practices for Mobile Point of Sale Acceptance
- Visa: Security Best Practices for Mobile Payment Acceptance Solutions

Host Commands for transaction processing (Hardware/Hardware Solutions)

The format and content of data sent between the terminal and P2PE Solution Provider will be determined by the design of the solution. However, a common element of all solutions is that the data will be encrypted. The keys and processes used will depend on the solution and the type of key management scheme employed.

This section shows how the payShield 9000 provides the building blocks to build a secure solution.

Master/Session Key Schemes.

All the data in the transaction should be encrypted as it will often be passing over a public network, and best security practise requires key separation to ensure that different types of keys are used for different types of data.

When this data is received by the P2PE Solution provider, it will need to be decrypted and processed. For example, MACs (Message Authentication Codes) may need to be validated, and data may need to be re-formatted and re-encrypted when passed to the payments network. Similarly, data from the payments network will need to be passed to the POI terminal. Again, key separation is required.

In Thales terminology, the relevant key types are:

Data type	Key type	
	Data to/from POI terminal	Data to/from payments network
PIN (in the form of a PIN Block)	Terminal PIN Key (TPK)	Zone PIN Key (ZPK)
Other cardholder data	Terminal Encryption Key (TEK)	Zone Encryption Key (ZEK)
Message Authentication data	Terminal Authentication Key (TAK)	Zone Authentication Key (ZAK)

The payShield 9000 provides a wide range of host commands to support all of these processes. The commands that are actually used will depend on the design of the solution, and the solution designer can choose which commands are most appropriate to their solution. Commands which are of use to process transaction data from the terminal are:

payShield 9000 Host Command	
ID	**Command Description**
JC	Translate a PIN Block from encryption under the TPK to encryption under the HSM's LMK. This would be relevant where processing (other than PIN Block translation) of the PIN is required at the host. (*NOTE: the PIN is never available in plaintext to the host.*)
CA	Translate a PIN Block from encryption under the TPK to encryption under a ZPK (Zone PIN Key). This would be relevant where the PIN Block is to be passed into the payments network.
JG	Translate a PIN from encryption under the HSM's LMK to encryption under a ZPK. This might be used where a PIN was previously translated from encryption under the TPK to encryption under the LMK, and now needs to be passed into the payments network.
M2	Decrypt a block of data encrypted under a TEK.
M4	Translate a block of data from encryption under a TEK to encryption under a ZEK - to pass the data into the payments network.
M0	Encrypt a block of data under a ZEK, to pass the data to the payments network
M8	Verify a MAC using a TAK.
MY	Verify a MAC using a TAK, and create a new MAC using a ZAK - e.g. when passing the transaction to the payments network.
EY	Validate a signature, using the POI terminal's public key.

Where transaction information is being returned to the POI terminal:

payShield 9000 Host Command	
ID	**Command Description**
M2	Decrypt a block of data encrypted under a ZEK.
M4	Translate a block of data from encryption under a ZEK to encryption under the POI terminal's TEK.
M0	Encrypt a block of data under the POI terminal's TEK, to pass the data to the payments terminal
M6	Generate a MAC using the POI terminal's TAK.

payShield 9000 Host Command	
ID	**Command Description**
MY	Verify a MAC using a ZAK, and create a new MAC using the POI terminal's TAK.
EW	Generate a signature on a message, using the P2PE Solution Provider's private key.
GM	Hash a block of data

DUKPT

Further information about DUKPT and the role that the payShield 9000 can play is provided in Chapter 4 of the *payShield 9000 Host Programmer's Manual*.

In the DUKPT scheme, the terminal will create a unique key for the transaction, plus derivatives of this key for different purposes such as PINs, authentication, and general data.

The payShield 9000 used by the P2PE Solution Provider can re-create the key that the terminal uses from knowledge of the BDK and the KSN (Key Serial Number - including the terminal's unique identifier), and therefore decrypt and encrypt data exchanged with the terminal.

payShield 9000 host commands that support DUKPT transaction processing when receiving transaction data at the P2PE Solution Provider from the POI terminal are:

payShield 9000 Host Command	
ID	**Command Description**
G0	Translate a PIN block from encryption using DUKPT to encryption under a ZPK (to pass the PIN as a PIN Block into the payments network.)
M2	Decrypt a block of data encrypted using DUKPT.
M4	Translate a block of data from encryption using DUKPT to encryption under a ZEK.
M0	Encrypt a block of data under a ZEK
GO	Verify a PIN Using the IBM Method, with optional MAC verification.
GQ	Verify a PIN Using the VISA PVV Method, with optional MAC verification.
GS	Verify a PIN Using the Diebold Method, with optional MAC verification.
GU	Verify a PIN Using the Encrypted PIN Method, with optional MAC verification.
GW	Verify a MAC with a MAC key derived using DUKPT.

payShield 9000 Host Command	
ID	**Command Description**
M6	Generate a MAC using a ZAK - when passing the transaction to the payments network.

The following commands can be used when preparing transaction data for transmission from the P2PE Solution Provider to the POI terminal:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a block of data using DUKPT
M2	Decrypt a block of data encrypted using a ZEK.
M4	Translate a block of data from encryption under a ZEK to encryption using DUKPT.
GW	Generate a MAC with a MAC key derived using DUKPT.
EW	Generate a signature on a message, using the P2PE Solution Provider's private key.
GM	Hash a block of data

Host Commands for key management (Hardware/Hybrid Solutions)

Introduction

In a Hardware/Hybrid solution, the cryptography at the P2PE solution provider is performed in the solution provider's host computer software rather than in an HSM. This means that cleartext data encryption keys will be held in the host's volatile memory, as will transaction and cardholder data while a transaction is being processed. As a consequence, the host computer system has to take on the responsibility for protecting keys and data that would be protected by the HSM in a Hardware/Hardware solution and for meeting the P2PE security requirements appropriate to Domains 5 and 6. In a Hardware/Hybrid environment, the PCI SSC standard places more requirements on the host system compared with the Hardware/Hardware environment in order to assure that security is maintained.

The role of the HSM in a Hardware/Hybrid environment is to:

- generate cryptographically strong keys under the control of authorised personnel,
- protect keys when they are being stored, and
- provide the data encryption keys to the host system when required.

Note that PINs must still be processed inside an HSM and cannot exist in the clear in the P2PE solution host - the P2PE requirements relate to non-PIN data such as account numbers.

Security impact on the host system

In the Hardware/Hybrid environment, the host system performs many of the cryptographic functions that would be performed by the HSM in a Hardware/Hardware solution. As a result, many of the P2PE security requirements in domains 5 and 6 that in a Hardware/hardware environment would relate only to the HSM now relate also to the host system. The considerations that now apply to the host include the following - it is assumed that key generation and loading of keys onto the host is performed entirely using a payShield 9000, and that therefore the relevant security considerations (in Domain 6) do not apply to the host system:

- Inventory control and monitoring (Domain 5, Requirement B):
  - recording and tracking all hardware and its serial numbers (with annual reviews)
  - provision of procedural, physical and logical security (covering operation, storage, and transport), including use of dual control
  - access controls for staff
  - procedures for loss of devices
  - detection and management of any deviations from expectations
  - validation of integrity of hardware
  - ensuring currency of software
- Implement security management processes (Domain 5, Requirement C):
  - Documented operational security procedures, including for maintenance/repair and withdrawal from service
  - Dual control
  - Control of privileges of staff
  - Tamper-detection and response mechanisms
- Maintenance of a secure environment (Domain 5, Requirement D):
  - Logging of activity
  - Detection and management of suspicious events, inc. decryption failure, unexpected transactions
  - PCI DSS compliance
- Implementing of secure processes (Domain 5, Requirement E):
  - Full configuration data
  - Isolation/dedication of the host system
  - Control of software used on the host

- Self-testing and response to unexpected conditions
- Protection of decryption keys
- User access controls and password management
- Security of user interface methods
- Physically secure environment, with CCTV

- Secure encryption methodologies (Domain 6, Requirement A):
  - Use of approved algorithms and key lengths
- Secure usage of keys (Domain 6, Requirement E):
  - Prevention of use or substitution by unauthorised keys
  - Prevention of keys being used for unintended purposes
  - No sharing of keys between production and non-production systems
- Secure key management (Domain 6, Requirement G):
  - Key lifecycle (cryptoperiod) management.
  - Secure erasure of keys
  - Use of secure key transport between host and HSM

Using the payShield 9000 in a Hardware/Hybrid environment

The payShield 9000 protects working keys such as Data Encryption Keys by encrypting them with its Local Master Key (LMK). The LMK-encrypted keys are returned by the payShield 9000 to the host system, which stores them in its key database.

In traditional operations, when the host issues a command to the payShield 9000 it provides any necessary working keys in their LMK-encrypted form, which the host retrieves from its key database. The payShield 9000 decrypts the working key using the LMK in order to process the command, such that the cleartext key is resident only in the payShield 9000's secure cryptographic module and only while the command is being processed.

For Hardware/Hybrid P2PE, the host would issue a command which includes the LMK-encrypted working key to the payShield 9000, and the payShield 9000 would return the working key encrypted under a P2PE symmetric Zone Master Key (ZMK) which is known by the host. The command sent to the payShield 9000 would also include the LMK-encrypted P2PE ZMK to be used.

The P2PE ZMK needs to be available to both the payShield 9000 and the host, and could be generated by either. When the P2PE ZMK is passed between the host and payShield 9000 it must be protected by:

- Transferring it as components
- Encrypting it under another already-shared ZMK
- Encrypting is using asymmetric (RSA) keys.

The payShield 9000 offers a rich smorgasbord of key management functions. The ones particularly relevant to the Hardware/Hybrid environment include the following:

payShield 9000 Console Command	
ID	**Command Description**
KG	To generate a P2PE ZMK and display it in both LMK- and ZMK-encrypted forms. The encrypted P2PE ZMK would be manually entered into the host system.
GS	To generate a P2PE ZMK as components on smartcards, which can be read by both the payShield 9000 and the host.
GC	To generate components which can be used by both the host and the payShield 9000 to derive the P2PE ZMK)
IK	Import a P2PE ZMK generated by the host and encrypted by it under another ZMK, and display the LMK-encrypted P2PE ZMK (which would then be manually entered into the host's key database).
FK	To form a P2PE ZMK from components on smartcards (e.g. generated by the GS console command) or printed on paper (e.g. by the host computer system or GC console command) and return the ZMK in LMK-encrypted form.

payShield 9000 Host Command	
ID	**Command Description**
A0	To generate a P2PE ZMK. This function will return the LMK-encrypted P2PE ZMK for storage in the key database and the P2PE ZMK encrypted under another ZMK which is known to the host computer.
BY	To import a P2PE ZMK generated by the host encrypted under another ZMK and return it in LMK-encrypted form. This command could be used where a ZMK has already been established between the host and payShield 9000 and is being used to provide (or update) the P2PE ZMK.
EI	Generate an RSA key pair to allow the host to encrypt a P2PE ZMK that it has generated.
GI	To import a P2PE ZMK generated by the host and encrypted by the host using an RSA public key, and return the P2PE ZMK in LMK-encrypted format.
A8	To take the LMK-encrypted working key sent by the host to the payShield 9000 and to return it to the host as a ZMK-encrypted working key, using a ZMK that the host also knows.

Host Commands for transferring data to terminals

There may be a need to transfer data (other than keys) to the POS terminals after they have been deployed to merchant sites: one example of this is the updating of firmware.

Some of this data may be non-sensitive, but in general this data will need to be protected against access by third parties, have its integrity protected, and have its provenance authenticated.

The following commands on the payShield 9000 used by the P2PE Solution Provider will enable secure data transfers to be made to POI terminals.

payShield 9000 Host Command	
ID	**Command Description**
GM	Hash a block of data
M6	Generate a MAC on a message using the POI terminal's TAK.
EW	Generate a signature on a message, using the P2PE Solution Provider's private key.
M0	Encrypt a block of data using the POI terminal's TEK or IKEY

Chapter 5 – Mobile Payments

Introduction

The preceding chapter discusses mPOS, or Mobile Acceptance – the use of mobile devices such as tablets by ***merchants*** to accept card payments. This chapter discusses the use of mobile devices – in particular smartphones – by ***cardholders*** in order to make payments.

The principle of mobile payment is that the mobile device (which must support NFC – Near Field Communication) appears to the card payment system as though it were a contactless EMV card, such that it can be used to make payments with minimal changes to the card payment infrastructure. The following approaches can be taken to enable the smartphone to be used in this way:

- Secure Element (SE) – where traditional card holder data is stored in a secure storage area in the smartphone. The SE is essentially acting like the EMV chip in an EMV card.
- Host Card Emulation (HCE) – where an app running on the processor (the host) in the smartphone emulates a payment card.
- Tokenization – which can be used as part of the security design of SE or HCE implementations.

This chapter describes the payShield 9000 functions which support the provisioning of cardholder data onto mobile devices and the processing of transactions originating from them. In order to respond quickly to the rapidly changing technology in the mobile payments space, some of the functions described may be available in generally available custom software (e.g. 1425-09xx) in advance of its appearance in standard base software.

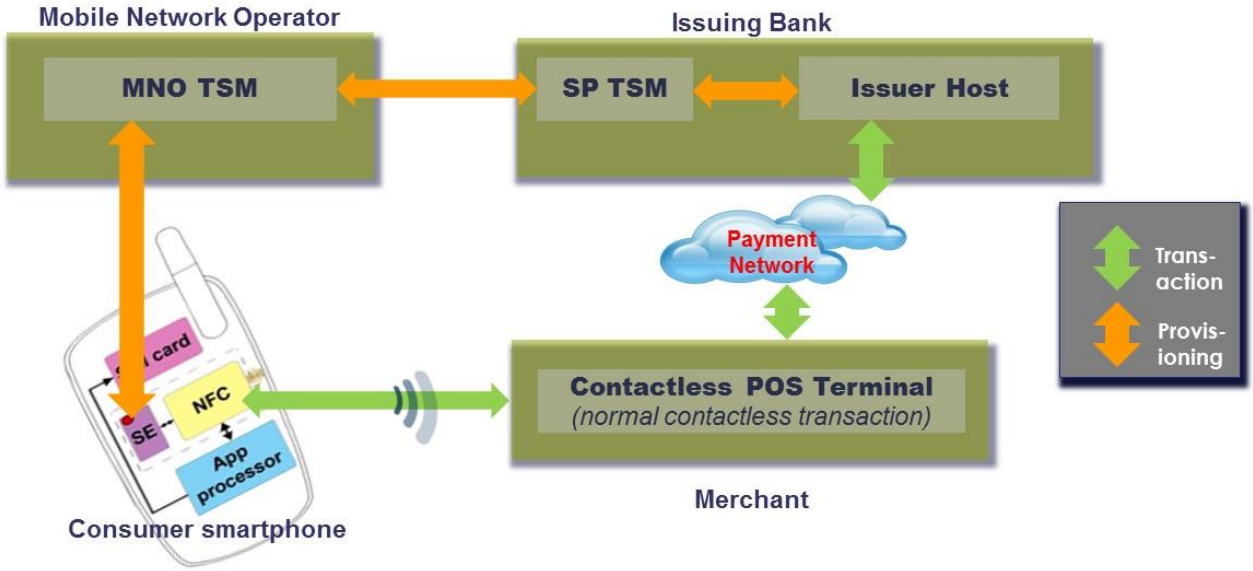
Broader descriptions of SE and HCE technologies and discussions of the associated security considerations can be found in the following White Papers from Thales e-Security:

- *Trust in the cloud or trust in the phone?*
- *Creating a trust infrastructure to support mobile payments.*

Secure Element (SE)

The Secure Element can be a MicroSD card provided by the issuing bank, or it can be a chip embedded in the phone by the manufacturer, or a Universal Integrated Circuit Card (UICC) provided by the Mobile Network Operator. The UICC approach has extensive GlobalPlatform specification support and is the most widely used: it is the mechanism described in this section.

The issuing bank generates cardholder data in the traditional manner and works through the phone's Mobile Network Operator (MNO) to provision this data onto the SE in the cardholder's phone.



The Trusted Service Manager (TSM)

The TSM acts as a broker between the issuer's service provider and the MNO to control the SE devices in the field:

- The issuer-oriented TSM manages the interface to the issuer and controls the loading and personalization of the payment application on the SE. Issuing banks will generally make use of a Service Provider (SP) for the TSM function to avoid having to develop TSM capability for all the MNOs they need to work with.
- The MNO-oriented TSM manages the SE devices themselves on behalf of their owner and grants permission and space on the SE for the issuer to exploit.

Data Preparation

The cardholder data preparation and key management for the SE is essentially the same as for EMV cards, and issuing banks will make use of their payShield 9000s and data preparation applications such as Thales P3 in this process as usual. This results in a secure file that traditionally is passed to a bureau to create the EMV cards: in the case of provisioning the data onto an SE the data goes to a TSM – which is effect a card bureau for "digital cards" using SEs on phones.

Provisioning to the SE

The payShield 9000 base supports secure provisioning and update of keys and data through a TSM using the GSM Secure Channel Protocols (SCP). This requires optional license HSM9-LIC018 see the *payShield 9000 Host Command Reference Manual Addendum for Licenses LIC011, LIC016, LIC018 & LIC023 (Card & Mobile Issuance Commands)* for full details. The following protocols are supported:

- SCP02 i=55 (double-length TDES / triple-length TDES MAC) – for Over the Air (OTA) personalization with Card not present.
- SCP03 (AES/CMAC) – Strong cryptography. Used for Card not present.
- SCP80/81 - OTA Protocol wrapping of SCP02 or SCP03.

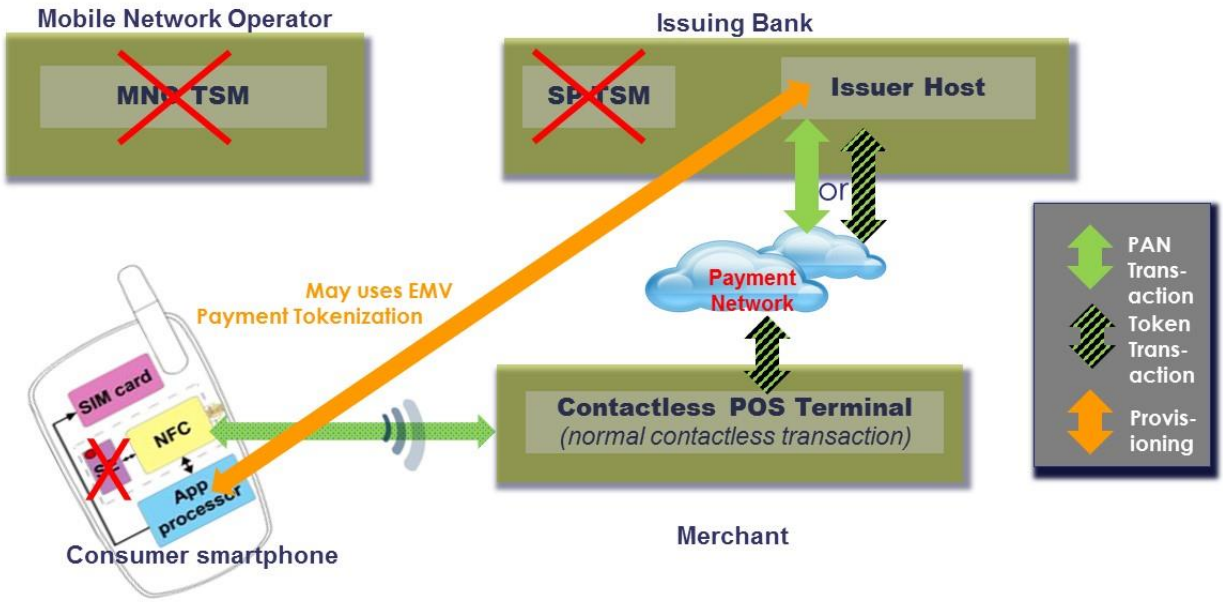
Adoption of the SE approach

Take-up of the SE approach has generally been limited because of the organizational complexity of having to work through a large number of MNOs.

However, the SE approach is used in the deployment of Apple Pay, for which a description of the process is not available. This has the attraction to issuers that they have to work through only one organization – Apple. In existing implementations in the US and UK provisioning is performed by the card schemes, and so issuing banks do not need to be involved. The Apple Pay system makes use of tokenization, and so transaction processing must be able to handle tokenized data: this is described later in this chapter.

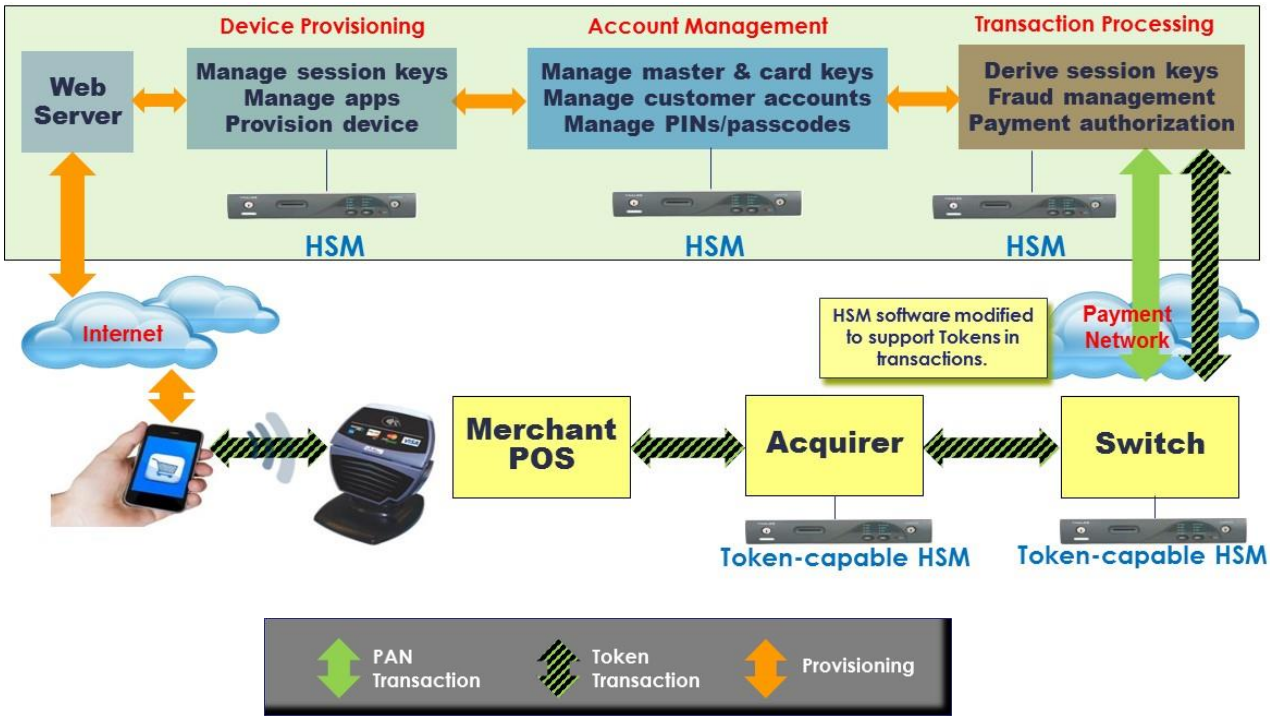
Host Card Emulation (HCE)

In the HCE environment, cardholders download an app from their issuing bank. The app runs in the phone's application processor, and no SE or TSM is involved.



Tokenization may be used as part of the security design of the HCE implementation – see later in this chapter.

The payShield 9000 plays an important role in both provisioning apps to the smartphone and in processing transactions.



Provisioning

The following host commands provide the functionality for the secure provisioning of apps, data, and keys to the smartphone. Full details on the use of the commands and any optional licenses required can be found in the:

- *payShield 9000 Host Command Reference Manual* and
- *payShield 9000 Host Command Reference Manual Addendum for Licenses LIC011, LIC016, LIC018 & LIC023 (Card & Mobile Issuance Commands)*.

Command	Description
KI	Derive Card Unique DES Keys (includes Key derivations for Limited-/Single-Use Keys; key derivation methods for Visa, Discover and AmEx.)
GI	Import key under RSA Public Key (includes import of AES keys under an RSA public key)
IO	Generate Remote Management Session ID and Session Keys
IQ	Validate Authentication Code (Authentication Code is generated by the Mobile Payment Application and sent to the host Credential Management System during the initialization phase allowing the CMS to authenticate the MPA.)
IU	Generate Remote Management Secure Message
IW	Validate and Recover Remote Management Secure Message
IY	Generate Digitized Card Single Use Keys

Transaction processing

The following host commands provide the additional functionality needed to process transactions from HCE-enabled smartphone. Full details on the use of the commands and any optional licenses required can be found in the:

- *payShield 9000 Host Command Reference Manual* and
- *payShield 9000 Host Command Reference Manual Addendum for Licenses LIC011, LIC016, LIC018 & LIC023 (Card & Mobile Issuance Commands)*.

Command	Description
KW	Verify ARQC / ARPC (includes schemes for Visa Cloud Based Payments, MasterCard HCE and AMEX HCE CVN 05, 07 and MPVV)
PM	Verify a Dynamic CVV/CVC (includes option for Version Scheme ID for Visa LUC (Limited Use Cryptogram))

If tokenization is being used, then the host commands in the following section on tokenization are also relevant.

Tokenization

Implementations of SE and HCE technology on smartphones may incorporate tokenization as part of their security design. With tokenization, the cardholder's PAN is replaced in transactions by a token: the token may have the same format as a PAN, but the PAN cannot be derived from the token by a would-be attacker. Tokenization can be implemented in different ways (e.g. a permanent or semi-permanent token resident on the mobile device, or single-use tokens created cryptographically on the device).

A transaction including a token in place of a PAN passes through the secure payments network in the same way as a normal transaction using a PAN. However, additional functionality is required to process a tokenized transaction when it is processed in the payment network and authorized by the issuer. In particular:

- **PIN Block translation** – if the incoming PIN Block was formed using a token rather than the PAN but the outgoing PIN Block is to be formed using the real PAN then the PIN Block translation functionality must include the capability of swapping the real PAN for the token during processing. The following payShield 9000 host commands allow both the token and the real PAN to be included in the command parameters to enable this additional processing to be performed:

Command	Description
CC	Translate PIN block from one ZPK to another ZPK.
G0	Translate PIN Block from a DUKPT BDK to a ZPK
CI	Translate PIN Block from a DUKPT BDK to a ZPK (legacy command)

- **PIN verification** - if the incoming PIN Block was formed using a token rather than the PAN but the PVV must be calculated using the real PAN then the PIN verification functionality must include the capability of swapping the real PAN for the token during processing. The following payShield 9000 host commands allow both the token and the real PAN to be included in the command parameters to enable this additional processing to be performed:

Command	Description
EC	Verify an Interchange PIN Using the Visa Method
GQ	Verify a PIN Using the Visa Method (DUKPT)
CM	Verify a PIN Using the Visa Method (DUKPT) (legacy command)

Full details on the use of the commands and any optional licenses required can be found in the *payShield 9000 Host Command Reference Manual*.

Chapter 6 – 3-D Secure

Introduction

3-D Secure is a mechanism which improves security of online, Cardholder Not Present, purchases by authenticating the cardholder using secrets known only by the cardholder and the card issuer which cannot be deduced from the card or by compromising communications channels. Cryptography is used to secure the process by protecting both stored and transmitted information.

3-D Secure relates specifically to cardholder authentication, and does not itself provide payment authorisation. The results of the 3-D Secure authentication are subsequently used as input to the payment authorisation process, but payments may still be authorised or declined whatever the outcome of the 3-D Secure process.

This chapter explains how the payShield 9000 can be used in the context of 3-D Secure. It focusses on areas of 3-D Secure which are relevant to the use of an HSM. Although it briefly describes the 3-D Secure architecture, readers should consult literature published by the card schemes (see the section at the end of this chapter) for a full description.

Overview of 3-D Secure

The 3-D Secure method was originally developed by Visa, but has since been adopted by other card schemes, using the following brand names:

- American Express "SafeKey"
- JCB "J/Secure"
- Mastercard "SecureCode"
- Visa "Verified by Visa"

It is up to cardholders whether they enrol into their issuer's 3-D Secure scheme, and up to merchants whether they participate in 3-D Secure. Merchants benefit from participation because the liability for a fraudulent transaction shifts from them to the issuer if the transaction has used 3-D Secure - even if the cardholder has not enrolled into 3-D Secure.

The name "3-D Secure" comes from the concept of security spanning 3 domains:

- Issuer Domain
- Acquirer Domain
- Interoperability Domain

The Issuer Domain

This domain covers the card issuer and the cardholder to whom the card is issued. This domain includes the following system components:

- **Cardholder access device** - used by the cardholder to make an online purchase. This is generally an internet-enabled PC running a standard browser, but it could also be a mobile device - in particular a WAP phone.

- **Access Control Server (ACS)** - operated by the issuer, or on behalf of the issuer by the card scheme or a service provider. The ACS is specific to 3-D Secure and has the following main purposes:
  - Enroll cards into 3-D Secure at the request of the cardholder. This enrolment may occur as a discrete activity prior to transactions being made, or may happen during a transaction where the issuer determines that the cardholder is not already enrolled.
  - Confirm whether a specified card is enrolled to use 3-D Secure
  - Provide/refuse payer (i.e. cardholder) authentication to the merchant.

  In some implementations, the enrolment functions may be separated out into a discrete Enrolment Server. For the purposes of this document, it will be assumed that the ACS handles enrolment.
- **Payment Processing system** - operated by the issuer or on behalf of the issuer by a service provider, to authorize transactions to the merchant. This function is not specific to 3-D Secure environments, but will need to use the 3-D Secure payer authentication information when authorizing/declining a transaction.

The Acquirer Domain

This domain covers the merchant that the cardholder is purchasing from, and the acquirer that processes the merchant's card transactions. This domain includes the following system components:

- **Merchant's e-commerce System** - not specific to 3-D Secure.
- **Merchant Plug-In (MPI)** - a software adjunct to the e-commerce system specifically for 3-D Secure. The MPI may be operated by the merchant, by the acquirer, or by a service provider. Its main functions are to:
  - Check with the Directory Server (see later) whether a presented PAN relates to a card which is enrolled into 3-D Secure.
  - If the card is enrolled, to request payer authentication from the ACS.
  - Handle the response to the payer authentication request.
- **Acquirer's system** - to handle transactions from merchants and get payment authorization from the issuer. This is not specific to 3-D Secure, but the transaction will include the results of payer authentication requests.

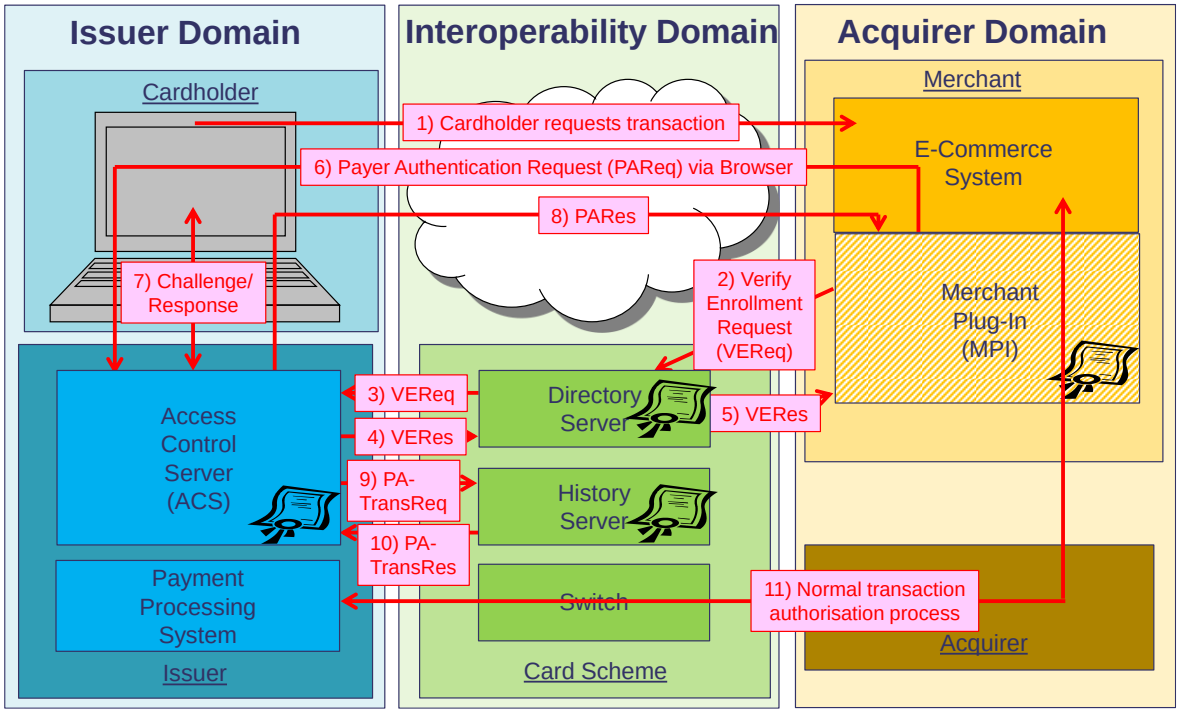
The Interoperability Domain

This domain covers the communication channels between the other two domains, and includes the public internet where the cardholder is involved, and the card scheme for communications between issuer and merchant/acquirer. The following card scheme system components are involved:

- **Directory Server** - to tell the MPI whether a card (i.e. its PAN) is enrolled in 3-D Secure, and if so how to access the issuer's ACS.
- **Authentication History Server** - to hold records of all 3-D Secure authentication activities to help in future dispute resolution.
- **Switch** - for payment authorization. Not specific to 3-D secure.

Message Sequence and Processing

The following text uses the message numbers referenced in the diagram below.



This processes described below are a subset of the possible processes:

- Processes where 3-D Secure payer authentication cannot be performed or is declined are not described here, as these would not involve the HSM.
- Processes and messages relating to caching of card ranges at the MPI are not covered here because they are optional and would not involve the HSM.

1. Cardholder Transaction

The cardholder is using the Internet to make an online purchase form a merchant, and has got to the point where he/she has entered their card details and indicated to the merchant that they wish to confirm the transaction. The merchant's e-commerce system asks its MPI to obtain a 3-D Secure payer authentication.

2. Verification of Enrolment Request (VEReq)

The MPI contacts the card scheme's Directory Server for confirmation that the card is enrolled in 3-D secure by sending a Verification of Enrolment Request (VEReq) message.

3. Verification of Enrolment Request (VEReq)

The Directory Server has lists of card ranges which are potential 3-D Secure members, and if the offered PAN is in one of these range it forwards the VEReq to the issuer's ACS to confirm whether the specific card has been enrolled into 3-D Secure.

4. Verification of Enrolment Response (VERes)

The issuer ACS responds to the Directory Server, identifying whether the specific card is enrolled in 3-D Secure. In this VERes message, the PAN is replaced by an Account Identifier, from which the ACS can later on recover the PAN.

5. Verification of Enrolment Response (VERes)

The Directory Server returns a VERes to the MPI. If the Directory Server knows that the submitted PAN is outside of any ranges which may be enrolled in 3-D Secure, then it generates the VERes itself without reference to the issuer ACS.

If the Directory Server referred the VEReq to the ACS, then it forwards the VERes from the ACS to the MPI. The VEReq includes the URL of the issuer's ACS.

6. Payer Authentication Request (PAReq)

The MPI sends a Payer Authentication Request to the issuer ACS, using the URL provided in the VERes. In fact, this is done via the cardholder's browser or WAP phone (see the section later on Modifications for Mobile Devices).

7. Challenge/response

The issuer ACS provides a challenge to the cardholder, who has to provide the appropriate response.

3-D Secure does not specify the nature of the challenge - that is up to the issuer. Possible methods include:

- Simple password
- Characters x, y, z of the password
- Choice of secret information (place of birth, mother's maiden name, etc.)
- One-time passwords (e.g. CAP or DPA for EMV cards)
- Use of a card reader attached to the cardholder's device (EMV only).

In July 2012, Visa indicated that for some transactions in the UK the user will be authenticated without the need to enter a password. Instead, transactions are risk-assessed, in real time, using a wide range of information and data sources, with cardholders only being asked to enter a password for higher risk transactions. The rest of the 3-D Secure process is unaffected. It is hoped that this change will increase the percentage of transactions covered by Verified by Visa and reduce the number of abandoned transactions.

8. Payer Authentication Response (PARes)

The ACS sends the Payer Authentication Response (PARes) to the MPI via the cardholder's device. This will include the confirmation or refusal of authentication, and include a cryptographic value (the AAV, AEVV, or CAVV) which can later be used to prove whether payer authentication was provided for this transaction.

9. Payer Authentication Transaction Request (PATransReq)

The ACS sends a Payer Authentication Transaction Request (PATransReq - which includes a copy of the PARes) to the card scheme Authentication History Server, where it is stored to assist in any future dispute resolution.

10. Payer Authentication Transaction Response (PATransReqs)

The Payer Authentication Transaction Response (PATransRes) is essentially an acknowledgement to the ACS of receipt by the AHS of the PATransReq.

11. Payment Processing

Normal payment processing now resumes as normal, with the acquirer sending the transaction to the issuer for authorization. The authorization request includes the results of the 3-D Secure payer authentication and the associated AAV/AEVV/CAVV.

Communications Security

All of the communications are protected by TLS or SSL v3 - which for convenience in this chapter will be referred to simply as "SSL". (Note, however, that SSL and TLS v1.0 are not considered secure by organizations such as PCI, and it is recommended that TLS v1.2 or later is used.)

To support this, all the 3-D Secure components (MPI, ACS, AHS, Directory Server) have certificates with the card scheme as the root CA. The MPI will also need a server certificate issued by a public CA which the cardholder device can use.

payShield 9000 and the MPI

This section discusses the payShield 9000 functions that can be used to support the operation of the Merchant Plug-In (MPI). Although use of an HSM is not a requirement for the MPI, an HSM will add security to the implementation, will assist in meeting the needs of security audits, and can reduce development effort.

Card Scheme Root Certificate

The MPI must be able to generate a public/private key pair when generating a certificate request. The following payShield 9000 host commands can be used for this:

payShield 9000 Host Command	
ID	**Command Description**
EI	Generate RSA Key pair

The Merchant server certificate signed by the card scheme root certificate must be encrypted and securely stored in the MPI. This can make use of the payShield 9000 message encryption host commands:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block
M2	Decrypt a Data Block

There is a requirement on the MPI that it stores key materials in a secure manner. This is automatically achieved by using the payShield 9000. The HSM operates by encrypting all working keys using its Local Master Key (LMK) which is not accessible.

Creating the Payer Authentication Request (PAReq)

The PAReq itself does not include any sensitive information, and so an HSM does not have a role to pay. However, the PAReq is included in a form which has an optional Merchant Data field. This field contains whatever information the merchant's systems require to allow it to associate the 3-D Secure messages with the transaction. This data is not required by any of the other 3-D Secure parties, and if it contains cardholder or other sensitive information it must be made secure. This can be achieved by using the payShield 9000 message encryption host commands:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block
M2	Decrypt a Data Block

Processing the Payer Authentication Response (PARes)

The PARes returned to the MPI from the ACS is signed by the ACS. The MPI must verify the signature, which it can do using the following payShield 9000 host command:

payShield 9000 Host Command	
ID	**Command Description**
EY	Verify an RSA Signature

Protection of Stored Data

The MPI may need to store data which includes cardholder and other data which needs to be encrypted. Examples of this data are:

- Merchant password (sent in CRReq and VEReq messages in certain circumstances)
- PAReq and PARes messages with associations to the relevant PANs, for later dispute resolution.
- Other cardholder data.

This can be achieved by using the payShield 9000 message encryption host commands:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block
M2	Decrypt a Data Block

payShield 9000 and the ACS

This chapter discusses the payShield 9000 functions that can be used to support the operation of the issuer's Access Control Server (ACS).

Card Scheme Root Certificate

The ACS must be able to generate a public/private key pair when generating a certificate request. The following payShield 9000 host commands can be used for this:

payShield 9000 Host Command	
ID	**Command Description**
EI	Generate RSA Key pair

The ACS server certificate signed by the card scheme root certificate must be encrypted and securely stored. This can make use of the payShield 9000 message encryption host commands:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block
M2	Decrypt a Data Block

Key Management

The security requirements relating to the ACS place a number of conditions on how keys are managed, such as:

- Keys used by the ACS must exist only within secure cryptographic devices or must be suitably encrypted.
- The generation of keys must meet defined standards
- Key generation generally must be performed using a secure cryptographic device.
- Encipherment of keys must take place within a secure cryptographic device.
- Keys may only be used for the intended purpose and location.
- Keys must never be shared between test and production environments.
- Private keys used for signing or client authentication must be used within secure cryptographic devices.
- Minimum key strength standards must be met.
- Use of split knowledge and dual control.
- Audit logging of key management activities
- If an ACS provides services to multiple issuers, each issuer's keys and data must be kept separate from those of the other issuers.

Meeting of these requirements can be achieved or assisted by using the payShield 9000. The payShield 9000 has a rich set of key management host commands. The most relevant ones to 3-D Secure are:

payShield 9000 Host Command	
ID	**Command Description**
A0	Generate a key
A2	Generate and print a component
A4	Form a key from components
A6	Import a key
A8	Export a key
EI	Generate RSA Key pair
EK	Load a private key
EO	Import a private key
EQ	Validate a public key
GI	Import key under a public key
GK	Export key under a public key

The payShield 9000 also provides a range of key management functions which can be executed manually by security staff using the console or payShield Manager. The following list shows the console commands that are relevant:

payShield 9000 Console Command	
ID	**Command Description**
GC	Generate Key Component
GS	Generate Key and Write Components to Smartcard
EC	Encrypt Clear Component
FK	Form Key from Components
KG	Generate Key
IK	Import Key
KE	Export Key

Note that to comply with the requirements of 3-D Secure:

1. Single DES must not be used. Triple-DES with at least double-length keys is required.

2. The security setting "*Restrict Key Check Values to 6 hex chars?*" must be set (using the *CS* Console command or payShield Manager *Configuration / Security Settings / Initial*).

3. Commands (e.g. the *FI* Host command or the *BK* Console command) which are classified as "Legacy" commands (in the functional lists of Host and Console commands in the appropriate manuals) must not be used.

Data Storage and Transfer

Cardholder authentication data, including the card scheme root certificate, which is stored at the ACS must be encrypted using an algorithm equivalent in strength to Triple-DES with double-length keys and may not appear in the clear outside of a secure cryptographic device.

Cardholder data transferred to or from the ACS also must be encrypted using an algorithm equivalent in strength to Triple-DES with double-length keys.

These requirements can be met using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block
M2	Decrypt a Data Block

Enrolment

Enrolment processes will be different for each issuer, but they typically involve the cardholder:

- authenticating themselves by providing the card's CCID (i.e. CVC2/CVV2/CSC)
- providing one or more secrets (such as a password or personal facts - e.g. place of birth, mother's maiden name, etc. - which will be used later by 3-D Secure to authenticate the user during a transaction).

Verification of CCID

In processing the enrolment, the ACS will need to verify the CCID and encrypt the secret(s). These actions can be achieved using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block
CY	Verify a Mastercard CVC or Visa CVV
RY	Verify an American Express CSC

Hashing of Secrets

For added security, the ACS may store a hash of the secrets entered by the cardholder rather than the values themselves. This could be achieved with the following payShield 9000 host command:

payShield 9000 Host Command	
ID	**Command Description**
GM	Hash a Data Block

Cookie Encryption

An option available to issuers to enhance security is to offer encrypted cookies to the cardholder's device during enrolment to limit access to the Personal Assurance Message (PAM), password hints, etc. The cardholder's browser will return the cookies to the ACS during the payer authentication process when a transaction is being made. This encryption could be performed using the following payShield 9000 command:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block

Processing Verification of Enrolment Requests

Decrypting stored data

The ACS will check whether the PAN included in a VEReq sent by the MPI (via the Directory Server) relates to a card which is enrolled for 3-D Secure. The cardholder data stored in the ACS must be encrypted, and so must be decrypted to allow this check to be made. This can be done using the following payShield 9000 host command:

payShield 9000 Host Command	
ID	**Command Description**
M2	Decrypt a Data Block

Generating the Account Identifier

The response (VERes) sent by the ACS to the MPI (via the Directory Server) does not include the PAN. Instead it includes a one-time Account Identifier, which is a token that does not reveal the PAN (but for which the issuer can subsequently recover the PAN) and which is statistically unique. The method for deriving the Account Identifier is not specified, and one possible method would be encryption or hashing of the PAN (or some other unique cardholder data) together with some data that varied with each transaction (such as a transaction counter). This can be achieved using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
GM	Hash a block of data
M0	Encrypt a Data Block

Processing Payer Authentication Requests

Decoding the Account Identifier

The PAReq received from the MPI will include the Account Identifier provided by the ACS to the MPI in its Verification of Enrolment Response (VERes). The ACS needs to recover

the PAN from the Account Identifier, so if this was generated by hashing or encrypting some cardholder data then the following payShield 9000 host command will be required to decrypt it:

payShield 9000 Host Command	
ID	**Command Description**
GM	Hash a block of data
M2	Decrypt a Data Block

Decrypting stored data

The ACS will need to recover cardholder data from storage. This data will be encrypted, and so will need to be decrypted using:

payShield 9000 Host Command	
ID	**Command Description**
M2	Decrypt a Data Block

Challenge/Response - Hashing of Response

If the ACS stores a hash of the secrets entered by the cardholder during the enrolment process, then the ACS will need to hash the input from the cardholder during the when responding to the challenge. This is achieved with the following payShield 9000 host command:

payShield 9000 Host Command	
ID	**Command Description**
GM	Hash a Data Block

Cookie Decryption

If encrypted cookies were provided to the cardholder's browser during enrolment, these will now need to be decrypted using the following payShield 9000 command:

payShield 9000 Host Command	
ID	**Command Description**
M2	Decrypt a Data Block

AAV / AEVV / CAVV Generation

The PARes sent from the ACS to the MPI (and copied to the AHS) includes a 20-byte field including a cryptogram which can be subsequently used to confirm whether a 3-D Secure payer authentication was performed for the transaction: American Express refer to this as an AEVV, MasterCard refer to it as an AAV, and Visa refer to it as a CAVV.

The AEVV and CAVV use the CSC (American Express) or CVV (Visa) algorithms to calculate the Cryptogram.

The MasterCard AAV is different. It requires 2 cryptographic functions:

- a SHA-1 hash of the merchant name.
- either:
  - a cryptogram using the CVC algorithm (similar to the American Express and Visa cryptograms), or
  - a SHA-1 HMAC across the rest of the AAV.

These cryptographic calculations can be performed using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
CW	Calculate a CVC or CVV
RY	Calculate/Verify a CSC
GM	Hash a Data Block
LQ	Generate an HMAC on a Data Block

Signing the PARes

The PARes sent from the ACS to the MPI (and copied to the AHS) must be digitally signed using the ACS's certificate issued by the card scheme's CA. This can be done using:

payShield 9000 Host Command	
ID	**Command Description**
EW	Generate an RSA Signature

Record keeping

Logs should be maintained of 3-D authentications, including copies of the PARes and other relevant cardholder data. This information must be encrypted, and this can be achieved by using:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block

Dispute Resolution

In the event of a subsequent dispute concerning a transaction, and this will require transaction data held at the ACS to be decrypted. It will generally be necessary to verify the AAV / AEVV / CAVV described previously. These actions can use the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
CY	Verify a CVC or CVV
RY	Calculate/Verify a CSC
GM	Hash a Data Block
LS	Verify an HMAC on a Data Block
M2	Decrypt a Data Block

Modifications to allow for mobile Internet devices

The process described so far is based around a cardholder making a purchase using a browser-equipped computer. The 3-D Secure specification includes extensions to allow for cardholders using mobile Internet devices - specifically WAP 'phones. These devices have limited message lengths and slower communications compared with computers.

The 3-D Secure extensions for mobile Internet devices do not change any of the principles or cryptographic processes described elsewhere in this document. The changes in the extensions relate to how data is represented in messages and which messages data is included in, with the objective of reducing the lengths of those messages which are conveyed using WAP. This results in two new message types:

1. CPRQ - the Condensed PAReq
2. CPRS - the Condensed PARes.

Although the host applications running in the ACS and MPI need to be changed to make use of these new message types, the potential use of the payShield 9000 is not affected.

payShield 9000 and the AHS

The Authentication History Server, operated by the card scheme, is a central repository for payer authentications to facilitate any disputes subsequent to the transactions having been processed. It is populated with copies of the PARes data provided by the ACS in PATransReq messages. This data includes cardholder data, and so must be encrypted when stored, and subsequently decrypted when required for dispute resolution. This can be achieved using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
M0	Encrypt a Data Block
M2	Decrypt a Data Block

payShield 9000 and Payment Authorization

Verification of AAV/AEVV/CAVV

When the 3-D Secure payer authentication process has completed, the MPI provides the results to the merchant's e-commerce system. Assuming that the authentication has been successful, the e-commerce system includes the AAV/AEVV/CAVV in the payment authorisation request it sends via the acquirer to the issuer.

As part of its payment authorisation process, the issuer's systems will need to verify the AAV/AEVV/CAVV. It can do this using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
CY	Verify a CVC or CVV
RY	Calculate/Verify a CSC
GM	Hash a Data Block
LS	Verify an HMAC on a Data Block
M2	Decrypt a Data Block

Key Exchange

The issuer's payment authorisation system will need to exchange some keys with the ACS to enable the AAV/AEVV/CAVV verification to be performed. Although the ACS is also "owned" by the issuer, it may be a completely different system to that used for payment authorisation, and may be operated on behalf of the issuer by a third party. Therefore secure key exchange mechanisms are required. These can be facilitated using the following payShield 9000 host and console commands:

payShield 9000 Host Command	
ID	**Command Description**
A0	Generate a key
A2	Generate and print a component
A4	Form a key from components
A6	Import a key
A8	Export a key
EI	Generate RSA Key pair
EK	Load a private key
EO	Import a private key
EQ	Validate a public key
GI	Import key under a public key
GK	Export key under a public key
NE	Generate a key and print components

payShield 9000 Console Command	
ID	**Command Description**
GC	Generate Key Component
GS	Generate Key and Write Components to Smartcard
EC	Encrypt Clear Component
FK	Form Key from Components
KG	Generate Key
IK	Import Key
KE	Export Key

payShield 9000 and TLS/SSL

TLS or SSL v3 (which, for simplicity, will be referred to just as "SSL" in this section) is used to provide secure communications between the various components of a 3-D Secure system.

Many users have software implementations of SSL. In such an implementation the payShield 9000 can be used to provide some of the crypto functions required and to enhance security. This section described the payShield 9000 host commands that can be used for these purposes.

Asymmetric key generation at the server

Each SSL party which has to authenticate itself to the other party must have an RSA key pair, which is used for secure exchange of key materials. Best practise requires that a key pair is set up specifically for use in SSL. The payShield 9000 provides the capability of generating server RSA key pairs and of securely storing the private key (encrypted using the LMK) by using the following command:

payShield 9000 Host Command	
ID	**Command Description**
EI	Generate RSA Key pair

Optional License LIC019 is required to allow the generated private key to decrypt the Pre-Master secret - see below.

Pre-Master Secret Decryption at the Server

As part of the SSL handshake the process, the client generates a Pre-Master Secret which will be used by both client and server to calculate the Master Secret, from which both parties compute the symmetric keys used to protect the data to be transferred. The client encrypts the Pre-master secret under the server's public key and sends it to the server.

The server must decrypt the Pre-Master Secret using its private key. The server can do this using the following host command. The appropriate type of RSA key pair must have been generated using the *EI* command (see above), which requires use of optional licence LIC019 on the HSM.

payShield 9000 Host Command	
ID	**Command Description**
GI	Import Key or data under an RSA Public Key

Validation of X.509 Certificates

The SSL handshake requires the client to validate the server's X.509 certificate, and may require the server to validate the client's X.509 certificate. This can be done by using the following payShield 9000 host command:

payShield 9000 Host Command	
ID	**Command Description**
ES	Validate a Certificate

Digital Signatures

Where the SSL session includes client authentication as well as server authentication, the client sends a CertificateVerify message which includes a digital signature. This sigbnature must be verified by the server. These actions can be facilitated by using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
EW	Generate an RSA Signature
EY	Verify an RSA Signature

Hashing

The Finished messages sent by the client and server include a hash. The recipient has to compute the same hash, and compare it with the received hash. These actions can be performed using the following payShield 9000 host commands:

payShield 9000 Host Command	
ID	**Command Description**
GM	Hash a block of data

Further literature on 3-D Secure

The following is a list of some of the documentation available to gain a full understanding of 3-D Secure, how it works, and how it should be implemented. Some of these documents can be found on the internet, but others are licenced and need to be obtained from the publisher.

American Express

- American Express SafeKey ACS Functional Requirements
- American Express SafeKey MPI Functional Requirements

JCB

- J/Secure ACS Functional Requirements
- J/Secure MPI Functional Requirements

MasterCard

- MasterCard SecureCode Merchant Implementation Guide
- MasterCard SecureCode Member Enrollment and Implementation Guide
- SPA Algorithm for the MasterCard Implementation of 3-D Secure

UnionPay

- E-Commerce Server-Hosted Solution Implementation Guide

Visa

- 3-D Secure Introduction
- 3-D Secure System Overview
- 3-D Secure Acquirer and Merchant Implementation Guide
- 3-D Secure Functional Requirements Merchant Server Plug-in
- 3-D Secure Functional Requirements Access Control Server (July 2002 or later)
- 3-D Secure Security Requirements – Enrollment and Access Control Servers
- 3-D Secure Protocol Specification Core Functions (July 2002 or later)
- 3-D Secure Protocol Specification Extension for Mobile Internet Devices

Americas
Arboretum Plaza II, 9442 Capital of Texas Highway North,
Suite 100, Austin, TX 78759 USA
Tel: +1 888 343 5773 or +1 512 257 3900
Fax: +1 954 888 6211
E-mail: CPL_Sales_AMS_TG@thalesgroup.com

Asia Pacific
Unit 904-906A, Core D
Cyberport 3, 100 Cyberport Road
Pokfulam, Hong Kong | Tel: +852 3157 7111
Fax: +852 2815 8141 | E-mail: apacsales.cpl@thalesgroup.com

Europe, Middle East, Africa
350 Longwater Ave, Green Park,
Reading, Berkshire, UK RG2 6GF
Tel: +44 (0)1844 201800 | Fax: +44 (0)1844 208550
E-mail: CPL_Sales_EMEA_TG@thalesgroup.com

> cpl.thalesgroup.com <